Natural Language Processing

A Paninian Perspective

[1]

Akshar Bharati
Vineet Chaitanya
Rajeev Sangal

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

With contributions from
K.V. Ramakrishnamacharyulu
Rashtriya Sanskrit Vidyapeetha, Tirupati

Prentice-Hall of India
New Delhi

Contents

List of Figures

Preface

This book presents a Paninian perspective towards natural language processing (NLP). It has the following three objectives:

1. To introduce the reader to NLP,
2. To introduce the reader to Paninian Grammar framework applied to the processing of modern Indian languages,
3. To compare Paninian Grammar framework with modern Western computational grammar frameworks.

Thus, the proposed book meets two goals at the same time: (1) It provides an introduction to NLP for Indian languages to students and faculty who are not initiated to NLP; and (2) It acts as a source book on computational Paninian framework bringing all the material in one place.

The first half of the book presents a computational grammar that has been developed for processing of Indian languages. Indian languages like many other languages of the world have relatively free word order. They also have a rich system of case-endings and post-positions (collectively called *vibhakti*). In contrast to this, the majority of grammar frameworks are designed for English and other positional languages. They are extended as an afterthought to handle free-word order languages, usually paying a price both in elegance as well as in processing efficiency. The unique aspect of the computational grammar described here is that it is designed for free word order languages and makes special use of vibhakti. It takes the concept of vibhakti and karaka relations from Paninian framework, and uses them to give an elegant account of Indian languages. Efficient parsers for the grammar are also described. The computational grammar is likely to be suitable for other free word order languages of the world.

Chapter 1 discusses applications and issues in Natural Language Processing (NLP). A history of relevant ideas in different disciplines is discussed in brief. This is followed by the present symbiosis of ideas from computer science and linguistics, leading to some of the modern grammar formalisms.

Chapter 2 introduces the structure of language and presents an overall architecture of our system for analyzing sentences in Indian languages. Properties desirable in computational grammars are also discussed. Chapter 3 discusses morphological analysis, and how a simple but efficient analyzer can be built. Chapter 4 discusses local word grouping in Indian languages. The aim of this process is to group sequences of words to produce vibhakti for nouns and verbs. These vibhaktis are used in the next stage of parsing.

Chapters 5 and 6 are the main chapters on Paninian Grammar. In Chapter 5, the notions of karaka charts, karaka assignment, etc. are discussed. It is argued and shown that a grammar based on these notions, successfully handles karaka relations, control, active-passives, etc. Chapter 6 discusses constraint based parsing for the Paninian Grammar. It discusses a general purpose parsing algorithm, as well as a more efficient algorithm for a restricted class of grammars. Finally, some applications related to machine translation are discussed in Chapter 7.

Second half of the book presents a comparision of Paninian Grammar (PG) with existing modern Western grammar frameworks. It introduces three Western grammar frameworks using examples from English: Lexical Functional Grammar (LFG), Tree Adjoining Grammar (TAG), and Government and Binding (GB). The presentation does not assume any background on part of the reader regarding these frameworks. Rather than trying to give a comprehensive coverage of each of the frameworks, the presentation brings out saliant points about each. Each presentation is followed by a chapter containing either a discussion on applicability of the framework to Indian languages in particular (and free-word order languages, in general) or a comparision with Paninian Grammar framwork.

The material in this book has been class tested in regular courses on NLP at B.Tech., M.Tech. and Ph.D. level at I.I.T. Kanpur, and during several short-term intensive courses on NLP. The material is suitable for anybody interested in NLP who has one of two backgrounds: computer science or linguistics. Actually, what is really needed is *an aptitude for language* and *an analytical mind*. Somebody who possesses these but not the background will still be able to follow the material.

Vineet Chaitanya
Rajeev Sangal

Kanpur
June 1994

Acknowledgements

Much of this material has been developed over the years by the research done by our group, in which large number of students and project staff have been involved. Some of the names of people including titles of their theses and papers can be found in references at the end. Many others contributed by asking questions and raising issues. These include at least seven batches of students of B.Tech. and M.Tech. who did courses on NLP. Also included are the participants of short term courses on NLP held at IITK during the summers of 1990, 1991, and 1992, and at University of Hyderabad in January of 1992 and 1993.

Many visitors and collaborators have also contributed in development of the theory. Special mention must be made of Dr. K.V. Ramakrishnamacharyulu of Rashtriya Sanskrit Vidyapeetha, Tirupati, who has contributed to the theory in a major way. He is the co-author of Chapter 5 and Appendix A.

We would like to thank Prof. B.N. Patnaik for his support to this enterprise and for discussions during the early stages of the work. We enjoyed our intense discussions and arguments on Government and Binding with Profs. Probal Dasgupta, K.A. Jayaseelan and R. Amritavalli during the memorable short term course on NLP at Hyderabad in January 1992. We are thankful to Prof. U.N. Singh of University of Hyderabad for many interesting discussions and support. We also wish to thank Prof. E. Annamalai of Central Institute of Indian Languages, Mysore; Dr. G. Umamaheshwar Rao, Panchanan Mohanty, P.R. Dadegaonkar, and Gautam Sengupta of Universtiy of Hyderabad; Dr. S. Rajendran of Tamil University, Thanjavur; Drs. A.S. Reddy and Narayanamurthy of A.I. Lab., University of Hyderabad; Prof. Ananthanarayana of Mysore (formerly of Osmania); and Dr. Thakur Dass of Kendriya Hindi Sansthan, Agra.

Many people have been involved in the implementation of Paninian parser and anusaraka among Indian languages. The following people played a major role in the earlier versions of the implementation: Sivasubramanian, B. Srinivas, P.V. Ravisankar, and Brajesh Pande. The current implementation has had major contributions from: Mrs. Amba P. Kulkarni, Vasudeva Verma, and V.N. Narayana. The following people have worked on building and checking lexical databases for anusaraka and Paninian parser for Hindi, Kannada, Telugu, Tamil, etc.: Rekha Srivastava, Neelam Tripathi, Pushpa Saxena, Alka Srivastava, Kamala Gangadharaiya, Rajeshwari Ramaseshan, Suguna Sathyamurthy. Dr. Dhanendra K. Jha has helped us whenever we had a question on the Paninian view towards a language phenomenon. Mrs. Amba P. Kulkarni's contributions need special mention:

developing programs, managing the building of lexical databases, and in providing new ideas at both linguistic and system level. She has also been the sounding board for many ideas.

This manuscript has been prepared with secretarial assistance from Mrs. Azra Shirin.

Support for the research work described here has come from Ministry of Human Resource Development (MHRD), Department of Science and Technology (DST), and Department of Electronics (DOE) of Government of India. DOE supported the 5-week long courses on NLP, in which much of this material was presented in front of linguists from all over the country, for the first time. DST project provided support for carrying out investigations comparing the Paninian Grammar (PG) with the modern Western computational grammars. Prof. R. Narasimhan as chairman of the steering and evaluation commitee of the project made valuable suggestions about what needed to be done at various junctures. Indian Society of Technical Education (ISTE) of MHRD provided support for preparation of course material on Paninian Grammar on which the first seven chapters are based.

Vineet Chaitanya
Rajeev Sangal

Kanpur
June 1994

Chapter 1

Introduction to NLP

1.1 Introduction

The goal of natural language processing (NLP) is to build computational models of natural language for its analysis and generation. First, there is technological motivation of building intelligent computer systems such as machine translation systems, natural language interfaces to databases, man-machine interfaces to computers in general, speech understanding systems, text analysis and understanding systems, computer aided instruction systems, systems that read and understand printed or handwritten text. Second, there is a cognitive and linguistic motivation to gain a better insight into how humans communicate using natural language (NL).

The tools of work in NLP are grammar formalisms, algorithms and data structures, formalism for representing world knowledge, reasoning mechanisms, etc. Many of these have been taken from and inherit results from Computer Science, Artificial Intelligence, Linguistics, Logic, and Philosophy.

1.1.1 Some Example Applications

Let us first look at some example applications of NLP.

Natural language interfaces to databases

Computers have been widely used to store and manage large amounts of data. The data might pertain to railway reservation, library, banking, management information, and so on. Normally, to use these systems, specialized computer knowledge is necessary. The goal of natural language interfaces (NLI) is to remove this barrier. The user is expected to interact in natural

language (by means of a keyboard and a screen). For example, to know whether a book by Patanjali is available in the library, the user would ask in NL. The reply would again be in NL, answering whether it is catalogued but issued to a user, is not among the holdings, is currently misplaced, or is on the shelf and its number is so and so. LIFER by Hendrix (1978) and INTELLECT by Harris (1977) were some of the early systems.

Natural language interface to computers

Some general NL interfaces to computers have also been developed. UC, short for UNIX consultant, developed at Berkeley (Wilensky, 1982) assists a new user to UNIX operating system. In case of a problem the user can seek its assistance. It engages him or her in a dialogue and tries to tell him or her what to do. It uses scripts and knowledge about user's goals and plans.

Question answering systems

Several systems have been built as research vehicles in NLP which answer questions about a domain. LUNAR by Woods (1977) was an early system that answered questions about the moon rocks. It makes a sophisticated analysis of quantification in the NL sentences.

Story understanding

There are question-answering systems which, given a story in a specified domain, answer questions about it; for example, about going to restaurants. Much of this work was carried out at Yale under Schank and Abelson (1977). They firmly established the need for domain knowledge, lots of it, for understanding. They also made interesting models of goals, intentions and plans among humans beings.

Machine translation

There has been much renewed interest in machine translation (MT) since the early 80s. There have been several large efforts: Eurotra for European languages, Mu for Japanese and English, KBMT between English and Japanese at Carnegie Mellon University, anusaraka among Indian languages at I.I.T. Kanpur, etc.

1.1.2 Achievements and Brief History

The field of NLP or computational linguistics has emerged in its own right and a large number of research groups around the world are working on it.

A survey by Association of Computational Linguistics (ACL) (Evens and Karttunen, 1983) listed 85 universities conducting courses in computational linguistics. Since then, the activity has only grown (Dorr, 1994). Vigorous activity exists in several other research centres as well. There have been major advances since the early sixties and since the days of the ALPAC report. Briefly, we will sketch the advances in each of the related disciplines separately, and finally how they have been brought together in recent years.

Linguistics

The generative enterprise on the linguistics front started in the late fifties. It brought about a sea change in the view of linguists towards language. Linguistic theories were now expected to be mathematically precise in their description of language. (See Winograd (1983; pp. 8-13) for an interesting account of the history of linguistics as seen through the eyes of a computer scientist.) Under the generative enterprise, several formal theories of syntax were developed successively over the years and the notion of derivation of a sentence from the theory was formalised.

The field also saw the rejection of many alternative theories, including semantic theories (Fillmore (1968)). But coupled with the formal rejection was the absorbing of semantic ideas in the syntactic theories, most notably, of theta roles in transformational generative theories such as GB.

The need for syntactic and semantic features (Katz and Fodor (1963)) was identified and widely accepted. The need for associated lexicon was also assumed but little attention was paid to actually building it for any of the theories by theoretical linguistics. Work on preparing dictionaries (particularly for English) went unnoticed in the background. Lexicographers started using computers, again quite unnoticed. The area has now come with a bang and is making major contributions to lexicon design, semantic features, automatic analysis of large corpora, and even parsing. Statistical approaches are being tried with great success for limited tasks such as tagging part-of-speech for words in a corpus. These feed into grammar based approaches. Thus, the statistical and the grammatical approaches are complementing each other.

Artificial intelligence (AI)

AI workers attempted to look at the problem of language as that of communication. As a result, they looked at all aspects: syntax, semantics and pragmatics (Narasimhan, 1981). The role of World knowledge and domain knowledge in natural language understanding and communication was recognized and given great importance. Several impressive systems were built, but they operated either in a toy domain (Winograd (1972)) or in a real but artificially narrow domain (Schank and Abelson (1977)). Advances in

syntax in linguistics were largely ignored by the research workers to their detriment.

Knowledge representation and inference emerged as an important area in AI with significant contributions from AI workers in NLP (Findler, 1979). First order logic, semantic nets, frames, etc. play an important role (Sowa, 1985). These can be viewed by the linguist as the generalization of the idea of simple semantic features. However, they have more power, in the sense, that constraints can be expressed in these which cannot be expressed as semantic features alone.

Logic programming has emerged as an important area. Its influence particularly the notion of unification, is evident on the computational grammars of the 1980s.

Formal languages and compilers

Grammar formalisms particularly Chomsky hierarchy which was developed in the early years of the generative enterprise found great use in computer science in an unanticipated area: in language compilers for programming languages (which are translators of sorts from a high-level programming language to machine language). The theory of hierarchy of grammar formalisms was studied and refined. Special cases of context free grammar formalism were discovered which were important to programming languages. But perhaps, the advance most important to NLP is the development of parsing algorithms.

Parsing algorithm (or a parser based on it) takes a grammar and a sentence and answers the question whether the sentence can be derived from the grammar. If the answer is yes, usually, it also shows the derivation or the parse tree. Parsing algorithms have been thoroughly researched for regular and context free grammars. Several results are known for other classes of languages as well. A consequence of all this is that building syntactic parsers for various old or new proposed grammar formalisms is no longer a difficult task.

Other advances in computer science

Other advances in computer science which have made NLP possible today are bigger and faster computers, and the development of database technology.

If we compare the size of main memory of the first digital computer (ENIAC) in 1946 with todays smallest personal computer (PC) that one can buy, we find an almost hundred times increase. ENIAC had a few thousand bytes of memory whereas the PC has 640,000 bytes to a few million bytes. (A byte can roughly store a character.) Larger commercially available computers have ten to hundred times more memory than a PC.

The same story holds for speed of operations performed by the computer (or its CPU or central processing unit). Besides the main memory mentioned above, much larger amount of storage is available on disk (typically 200 million to several thousand million bytes).

To harness this power, major software systems have been developed. They assist the user in writing computer programs (such as parsers) or preparing the data (such as grammars and lexicon related data). The most important among them are high-level languages and operating systems.

Development of database systems allows easy storage on and retrieval from disk of large dictionaries, lexicons, lexicalised grammars, and corpora of text.

The symbiosis

Many of the above mentioned advances across disciplines were integrated in the 1980s. This was due to the fact that people from different disciplines were working together. All this led to the formation of a new field called *Computational Linguistics* or *Natural Language Processing*. The tempo of research in this new field has quickened. The influence of the availability of powerful computer systems at low cost is markedly evident in the early 1990s.

The 1980s saw the birth or maturing of several grammar formalisms which are particularly suited for computation of English or other configurational languages (relatively fixed word order language). Major among them are Lexical Functional Grammar (LFG) by Kaplan and Bresnan (1982), Generalized Phrase Structure Grammar (GPSG) by Gazdar et. al. (1985), Tree Adjoining Grammar (TAG) by Joshi (1985), etc. Because of the influence of logic programming, unification has become a basic operation in some of these grammar formalisms.

Similarly, knowledge representation has had major impact on natural language semantics and vice versa in the 1980s. Any computational linguist working on semantics today freely draws upon logic and other knowledge representation schemes. Similarly, researchers working on knowledge representation problems have been deeply influenced by the perspectives of the language.

In the case of Indian traditional linguistics (Paninian and other grammar formalism) and logic (navya-nyaya), they are beginning to have a major impact on how Indian languages should be handled computationally (Bharati et al. (1990), (1990b), (1993a), and Sangal and Chaitanya (1987)). The results may go beyond Indian languages and apply to other non-configurational languages in the first instance. They might also be applicable to all languages.

Finally, there is also a subtle influence on working methods, which ap-

pears when new tools are used. Large number of linguists today are using computers in preparing sentence corpora, electronic dictionaries and lexicons, in writing scholarly papers using word processors, in doing analysis of concordances, in testing their grammatical theories on a corpora of sentences, and so on. Here the computer is only a tool helping them do better or easier what they were doing earlier. But its subtle influences have already begun to appear.

1.1.3 Open Problems

Although impressive gains have been made in syntax, the areas of semantics and pragmatics have only been scratched as yet. Semantic interpretation involves determination and representation of meaning. Some issues that have to be addressed are: identifying the meaning of a word or a word sense, determining scopes of quantifiers, finding referents of anaphora (including pronouns, reflexives, and definite reference expressions), relation of modifiers to nouns (in the manner a modifier modifies a noun, e.g. in cast iron pump vs. in water pump), and identifying meaning of tenses to temporal objects.

The above raise difficult problems of representation and inference. First-order logic (FOL) or knowledge representation systems of equivalent power have difficulty in representing some of the above (e.g., time and modality). Representing and inferring world knowledge, particularly common sense knowledge, is difficult. Again, FOL has difficulty in dealing with generalized quantifiers and exceptions (e.g., most birds fly).

The above problems have to be faced whether one is dealing with a single sentence or discourse. Some work has been done on the structure of discourse. Semantics of discourse segments is a harder problem. "Literal" semantics of single sentences can be used to identify the discourse segments and, ultimately, their semantics.

When one comes to pragmatics, even the issues are fuzzy. The utterance under consideration, whether a sentence or a discourse segment, serves a communication purpose. A simple declarative sentence stating a fact (e.g., it is raining) is not just a statement of fact, but serves some communicative function (e.g., to inform, to mislead about fact, to mislead about speaker's belief about fact, to draw attention to already known fact, to remind about a previously mentioned event or object related to fact, etc.). As can be seen from the example, the pragmatic interpretation appears to be open ended. This could of course be a reflection of our ignorance. Speech act theory and Schank's work are possible approaches. See Chierchia and McConell-Ginet (1991) for more details.

1.2 Major Goal

In all the applications of NLP mentioned in the last section, language is serving a communicative function. For example, in natural language interfaces to databases, users communicate their information need by means of natural language query. In the context of machine translation, the writer wants to convey something to his readers through text. This is not surprising because the primary function of natural language is communication.

A consequence of the above is that NLP focusses on the study of language as a means of communication. When a speaker (or a writer) intending to communicate something to a hearer (or reader) utters or writes, and the hearer (or reader) on receiving the utterance or text gets that "something", communication has taken place. Of course, the communication is seldom perfect; what the hearer receives is an approximation of what the speaker wants to convey. But usually it is a remarkably good approximation. Also, the communication requires not only a common language but also shared knowledge about the domain in question. Nevertheless communication takes place.

The above transfer can be looked at from the point of view of information. The speaker wants to convey some information to the hearer. Having decided on the information he wants to convey, he must decide how to code it in language. Utterance is the only thing actually received by the hearer, using which he gets the information. It follows, therefore, that the information is contained in the utterance, and the hearer must extract it by decoding it.

The above view is non-controversial in the case of cooperative communication involving information exchange and has been advanced in the past in various forms. What we would like to do is to carry this view to the study of language (especially to what is called syntax or syntactic phenomena). Various phenomena in a language will be analyzed from the viewpoint of how they code information. Thus, word order and case endings turn out to be alternative ways of coding the same information (regarding semantic relationships between a verb and the nouns, etc.) When the same device (or phenomenon) in the language is used to code two different kinds of information, there is a possibility that a conflict may arise. Again the choice made in the language to handle the conflict must be looked at from the point of view of information coding. For example, in control when a noun group is in semantic relation with two different verbs, case ending can be assigned by only one of the two verbs. A choice must be made by language in such a way that the information about semantic relations can be decoded to the extent possible.

The information based approach provides natural connections between what are called syntax, semantics and pragmatics. In fact, one can view

all these as providing a theory of communication, each at a different level. Eventually, they would all be integrated together into a general theory of communication. Division into three levels, therefore, would be mainly due to methodological reasons. Knowledge representation and use would also fit neatly in the framework.

It should be mentioned that when a speaker tries to code information which he wants to express in a language, certain parts become cumbersome to code. For example, the gender of the speaker is not coded in the pronoun 'I' or the verb. Thus, the coding process sometimes loses information. Even in such a situation, the hearer is routinely able to decode the information by using his background knowledge. There are several sources of knowledge that are used in decoding the information from an utterance. These can be classified into:

1. Language Knowledge
   (a) Grammar
   (b) Lexicon
   (c) Pragmatics and discourse

   etc.
2. Background knowledge
   (a) General world knowledge (including common sense knowledge)
   (b) Domain specific knowledge (includes the specialized knowledge of the area about which communication is taking place)
   (c) Context (verbal and non-verbal situation in which communication is to taking place)
   (d) Culture knowledge

Thus, a hearer can use all the sources of knowledge above to extract information from a given utterance.

It is important to keep in mind that the speaker has a model of the listener. Thus, the speaker codes what information he wants to convey, using his beliefs about the sources of knowledge available to the listener among other things (such as, goals, plans, desires, processing capability or intelligence of the listener). A good speaker will code information in such a way that his listener(s) can easily decode it.

As already discussed, the study of language from this aspect tries to look at various language phenomena from the information viewpoint. This is not to say that there are no other factors. The language does try to maintain regularity across constructions for ease of acquisition of language as well as ease of decoding (and coding).

The notion of grammar from the information viewpoint is a system of rules that relates information to its coding in language. Moreover, there is a computational requirement that the grammar should be such that it can be used by the speaker to code information in the language and by the hearer to decode the information. When the system of rules relates information to coding devices at the language level and not at the world knowledge level, it is called *syntax*. But clearly, there are strong influences of world knowledge on the coding. First, it influences the fundamental coding conventions of language. For example, in Indian languages, the post position marker for noun group that is goal or theme or karma can be dropped precisely when it is inanimate (because the agent is normally animate and there is no ambiguity in distinguishing the semantic roles). Similar is the case when the second person pronoun is dropped from a command. This happens in all languages including English, which otherwise has a strong requirement that subject must be present. Second, it affects the coding for particular utterances used by a speaker because he knows that he need not try to explicitly code information that is available (or obvious) to the hearer from world knowledge. Thus, in this view there is no sharp isolation between syntax and semantics. Rather, the separation is mainly because of pragmatic reasons of ease of processing or grammar writing. Syntax uses language coding devices while the semantics also uses world knowledge. Indeed some of the intensively studied phenomena in syntax such as anaphora (covering reflexive pronouns and reciprocals) may turn out to be of minor importance once it is realized that the world knowledge is the major deciding factor. Syntactic rules regarding antecedent of anaphora do not have the kind of importance given to them in the recent years. Thus, in our view, syntax is not studied to identify an innate autonomous level, but rather to relate it to semantics and world knowledge to accomplish the overall tasks of communication of information.

The above ideas lead to a fundamental change in the view of grammaticality. It is known that real life utterances, both spoken and written, are full of what are called "ungrammatical" sentences. It is important to explain them because of both theoretical and practical reasons. Any theory which claims to describe or explain natural language, must explain these utterances as well (Narasimhan, 1981). It will not be acceptable to say that such sentences are grammatical in context but ungrammatical otherwise. That would be bypassing the question. If one considers the communicative aspect, sentences are always in context. A sentence is considered in isolation solely to simplify analysis (or generation). A sentence analyzer must not rule out a sentence as ungrammatical when the sentence can make sense in some context. On the contrary, it must perform an analysis (a plausible analysis, if possible) which is available to another analyzer say, a discourse analyzer that takes the context into account. Such a pre-analysis reduces

the work of the discourse analyzer.

Practical systems must deal with erroneous input as well. Even if an ill-formed utterance (whether within or without context) is given, the system should make a guess to obtain the intended meaning. The system may get the user to verify the meaning, or seek his help in case it is able to obtain only a partial meaning. But in any case, the response to be produced is more than just accept or reject.

If the above arguments are accepted, the conventional notion of grammaticality undergoes a change. Instead, one can shift to a view that a sentence has a meaning with an associated cost regarding acceptability. When the cost is zero, it is a good sentence (or conventionally a grammatical sentence). When the cost is low, the sentence sounds odd but makes sense. As the cost increases, the sentence sounds more and more odd until it stops making sense. More generally then, a sentence has associated meaning-cost pairs. A good parser should produce the pairs with lower cost first. If a low cost pair is rejected because of the context, the next higher cost pair is considered, and so on.

The language universals now acquire a very different meaning when compared with those in typological or generative approaches. The universals relate to information and its coding. Seen from this point of view, some seemingly diverse phenomena or categories turn out to be a manifestation of the same information theoretic concept. For example, case endings, prepositional and postpositional markers, and word order turn out to be alternative ways of coding information about which different languages make different choices. Free word order and positional languages are simply two extremes in the spectrum.

The information theoretic approach also prioritizes information, thus producing a priority among devices used by the language to represent different parts of information. For example, theta roles would be considered more important than topicalization information as the former affects gross meaning, whereas the latter brings out a nuance. This would have ramification for coding wherever a conflict arises between the devices needed for them.

At the computational level too, this approach has important consequences. Since language overloads content words with several meanings, function words with several grammatical roles, and uses other devices so as to code several kinds of information, the problem of natural language parsing can be restated as the problem of resolution of ambiguity. The resolution might pertain to: lexical ambiguity, structural ambiguity, topicalization ambiguity, grammatical role ambiguity, anaphora reference ambiguity, quantifier scope or dependency ambiguity, etc. An information theoretic parser would normally be designed by studying where the information to resolve the ambiguity resides, and how it can be processed.

It is important to understand the differences between goals and approaches of conventional linguistics and computational linguistics. First, conventional generative linguistics is interested in identifying that aspect because of which languages can be learnt by the human child so effortlessly. Thus, this raises questions about the human mind. Second, conventional linguistics is interested in giving a grammar or grammar framework, but does not pay attention to the actual processing of sentences using the grammar. As a result, the grammars or grammar frameworks developed by it often cannot be taken and computed with. On the one hand there is a problem of efficiency of computation, on the other hand there is a problem of effectiveness of computation (or computability). The former implies that the theories are such that if one tries to compute with them, it requires excessive resources and time. The latter says that the theory cannot be used for computation because no mechanical procedure can be defined for it. Third, generative linguistics has concentrated on identifying universals across all natural languages, but has avoided writing exhaustive grammars for any particular language.

Psycholinguistics pertains to comprehension and generation of natural language the way people actually do these activities. For example, if a particular sentence construction takes longer to comprehend, psycholinguistic theory tries to account for it. All this requires careful design of experiments and data collection, and theories to explain them. When the theories being constructed are computational in nature, this activity falls under a new field called *Cognitive Science*.

If the motivation is to build useful NLP systems, theoretical linguistics work might seem to be unnecessary. Any approach that works should be good enough. However, the activity being modeled is so complex that ad hoc solutions are unlikely to work. A sophisticated underlying theory is essential. At the same time, the present state of knowledge is such that our theories are incomplete and there are vast gaps in our understanding. Wherever this happens, per force we will adopt whatever solutions that work. Thus, there will be a compromise.

Computational linguistics (used synonymously with NLP) has much to learn from linguistics. However, the theories developed in the latter field must be adapted or rewritten to suit the needs of NLP. Often, the NLP workers ignore the work done in linguistics; whereas some of the linguists feel that their work is directly suitable for computation. Both these views need to change to find a common meeting point.

Further Reading

Hausser (1989) discusses important issues in computational linguistics and how the concerns are different from those in linguistics. Shieber (1988) analyzes differences between computational linguistics and (Chomskyan)

generative linguistics. Bharati et al. (1992a) discuss computational linguistics and its relation to linguistics. Tennant (1981) describes case studies and provides overviews of several systems. Section 1.1 here is based on Bharati et al. (1990b).

Chapter 2

Language Structure and Language Analyzer

2.1 Introduction to Language Structure

In this section, we introduce the reader to language structure. As discussed in Sec. 1.2, we are primarily concerned with the communication function of natural language; in particular, how information is coded in a natural language string[1] and how it is retrieved from the string by a reader. The purpose of identifying structure is that it will establish a relation between information[2] and the string that codes it. Before we can write grammars and design NLP systems that use the grammars, we need to sharpen our intuitions about language. This, then, is the purpose of this section.

The basic units in a written string are words. A word is a continuous sequence of alphabetic characters (possibly including the hyphen).[3] It stands for some information. Further, appropriate sequence of words in a string can be grouped together in a word group. What this means is that for such a group of words, there is a pattern or a rule which helps define the group. Information contained in a word group usually corresponds in some systematic way to the information contained in the constituent words. In other words, information is compositional in a group. Similarly, a word group can be further combined together with other word groups or words to yield larger groups. Here again, the information contained in the larger group systematically corresponds to the information contained in the

[1] A string is a general term for paragraph, sentence, phrase or word.

[2] Information can also be called as meaning, if it helps the reader to understand it. Information is actually an impoverished part (or mundane part) of meaning.

[3] Words are typically separated by delimiters such as blank space, comma, period, quotation mark, paretheses.

constituents.

When two or more items get grouped into a larger unit, a structure is formed, which contains the items and the relationship among them. The information of the larger unit, usually depends on the structure, that is, the constituent items and their inter-relations.

To keep the structure minimal, we introduce only one kind of structure, call it *modifier-modified structure.* There will usually be two items in a structure called the *head* and the *modifier.* [4] For example, in the following Hindi string:

```
sheitaana  laDake
naughty    boys
```

'naughty' is the modifier and 'boys' is the head. Similarly, in the following:

```
lohe  kaa   pampa
iron  reln. pump
(iron pump.)
```

pump is the head, and iron is the modifier.[5] Again, in the following:

```
paanii  kaa    pampa
water   reln.  pump
(water pump)
```

pump is the head, and water is a modifier. Note that properties of the head are inherited by the group. Thus, 'water pump' is a pump, it is not water.[6] Linguistically, we see that the grammatical properties of the head are inherited by the structure. Thus, in the sentence:

```
Iron pumps are heavy.
```

the structure (or word group) 'iron pumps' is plural as seen from agreement with the verb 'are'. The structure is plural because the head ('pumps') is plural.

In the above examples, we have identified the head and the modifier, but not the nature of modification. We know from our world knowledge that the modifier 'cast iron' in 'cast iron pump' indicates the material out of which the pump has been constructed, whereas 'water' in 'water pump'

[4] In Paninian Grammar (PG), such a structure is called *visheshya-visheshana bhaava.* Visheshya means roughly the head, and visheshana means modifier.

[5] This is sambandha (relation) or visheshya-visheshana bhaava in PG.

[6] There are some modifiers that make the meaning of the overall structure substantially different from the head. For example, in 'a fake gun', 'gun' is the head and 'fake' is the modifier but 'a fake gun' is not a gun at all. Thus, it does not have the properties of the head. There is no space for a detailed discussion here. We will give a computational answer that the properties of the head are inherited by the structure first, which are then modified by the modifier. In this case, even the essential property (of being a gun) gets modified or over-ridden.

indicates the fluid that gets pumped. But this is not coded in the string. We will indicate the nature of modification, only if this information is available from the string.

It should be mentioned again that we have defined a minimal structure. This structure can be extended or enriched, if necessary, according to the linguistic theory being used.

Before we look at some commonly occurring structures, we need to discuss categories of words and structures. Words belong to a small number of lexical categories, of which two are most prominent: nouns and verbs. Correspondingly, there are two types of prominent structures (or groups): nominals and verbals. The head of a nominal is a noun or another nominal. Similarly, the head of a verbal is a verb or another verbal. A sentence will be viewed as a verbal. There are a small number of other lexical categories. They usually occur as a modifier in a structure, not as a head. These are: adjectives, adverbs, etc.

Let us now look at different kinds of modifier-modified structures.

1. *Nominal structure with adjective-noun modification.*[7]
   Here, the noun is the head and the adjective is a nominal. Here are some examples:

   ```
   safeda   caadara
   white    sheet

   moTaa    laDakaa
   fat      boy
   ```

   The nouns 'caadara' (sheet) and 'laDakaa' (boy) are the head in the two sentences, respectively. The nature of modification is not coded in the string.

2. *Verbal structure with noun-verb modification.*[8]

   Here, the verb is the head and the noun is a modifier. A verb denotes an activity (or state), while the noun denotes a participant in the activity (or state). For example, in the following two sentences:

   ```
   laDakaa   douDaa.
   boy       ran

   laDakaa so   rahaa  thaa.
   boy     was  sleeping
   ```

[7] Called *samaanaadhikarana* in PG.
[8] Called karaka realation in PG.

the activity 'douDaa' (ran) and the state 'so' (sleep) have the noun modifier 'laDakaa' (boy).

Similarly, in the following example sentence:

```
laDake   ne       paanii  piyaa
boy   ergative  water   drank
```

the head is 'piyaa' (drank) and the modifiers are 'laDakaa' (boy) and 'paanii' (water). However, in this case the nature of modification is known. The 'boy' drinks and 'water' is what gets drunk.

Different linguistic theories give different accounts of the modification, i.e., they see different kinds of relations between the noun(s) and the verb in the above sentences. For example, GB (Government and Binding) theory or Fillmore's case grammer talks about thematic (or theta) relations between them whereas the Paninian theory analyzes in terms of karaka relations. For instance, a question that needs to be addressed can be stated as follows: Is the relation between 'boy' and 'drink' the same as that between 'boy' and 'ran', and 'boy' and 'was sleeping'. Karaka theory calls all these as *karta karaka* relations. Fillmore (1968) calls the first two as *agent* while the third one as *experiencer*. All these will be discussed in Chap. 5. We will use the neutral term of *verb and its arguments* to cover all these relations.

We can draw the modifier-modified structure pictorially. For the example given above, the structure can be shown as in Figure 2.1. The

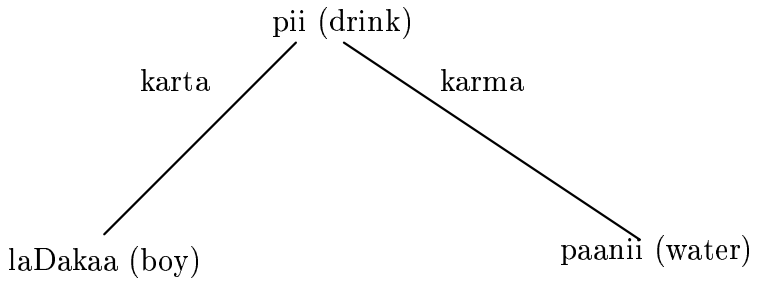


Figure 2.1: Nouns as arguments of verb

root node 'pii' gets modified by its children 'laDakaa' and 'paanii'. The nature of modification is shown by labelling the edges as karta and karma.

3. *Verbal structure with verb as argument of the head verb.*[9]

[9]Called *karaka relation* in PG. Karaka is pronounced as kaaraka.

Arguments of certain verbs are verbs. Consider the sentence:

```
laDake ne kahaa ki usane patanga uDaayii.
boy  nom. said that he   kite    flew.
(The boy said that he flew a kite.)
```

Here the argument of 'say' is an entire sentence 'he flew a kite'.

Diagramatically it is shown in Figure 2.2. As before, a child node is

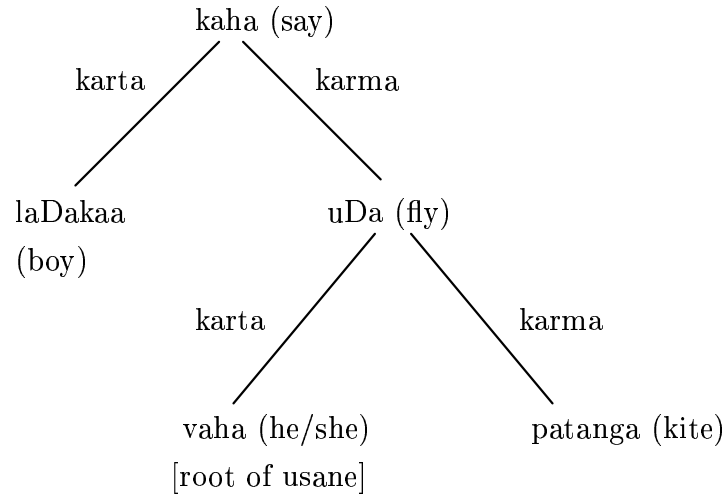


Figure 2.2: A verb (uDa) as argument of another verb (kaha)

a modifier of its parent.

4. *Nominal structure with participle verb as a modifier of a noun.*

   Here, the noun is the head whose modifier is a verb. The verb typically occurs in non-finite form or participle form. As an example, consider the sentence:

   ```
   bhaagataa  huaa  laDakaa
   running          boy
   ```

   where the head is 'laDakaa' (the boy). The verb 'bhaagataa huaa' (running) is modifying a noun (boy). The noun which is modified by the verb, is an argument of the verb. The nature of the participle form determines what the argument is. In the above example, 'bhagataa

huaa' has the 'taa huaa' form which indicates karta relation with the noun it modifies.

This is shown diagrammatically in Figure 2.3. It shows that 'laDakaa'

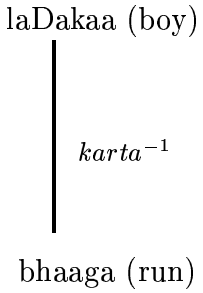


Figure 2.3: Nominal with verbal modifier

(boy) is the head whose modifier is 'bhaaga' (run). The relation between them is inverse karta, in other words, the parent node (laDakaa) is the karta of the child node (bhaaga).

5. *Verbal structure with verb-verb modification.*

   One of the verbs is the head and the other is a modifier but not an argument of the verb. Here is an example:

```
laDakaa aama    khaakara      ghara  gayaa.
boy     mango  having-eaten   home  went
(Having eaten the mango, the boy went home.)
```

   Here, the participle form 'khaakara' (having-eaten) is related to the main verb 'gayaa' (went). The nature of relation is temporal precedence. In other words, the eating action occurred before the going action. Similarly, we have the following example:

```
laDakaa aama    khaataa  huaa ghara  gayaa.
                eating
(The boy went home, eating a mango.)
```

   Here the second action (gayaa) took place while the first one was in progress.

   The modifier-modified relations in the first example sentence above are shown as in Figure 2.4. The verb node 'jaa' (go) is modified by its arguments 'laDakaa' (boy) and 'ghara' (home) as well as the verb

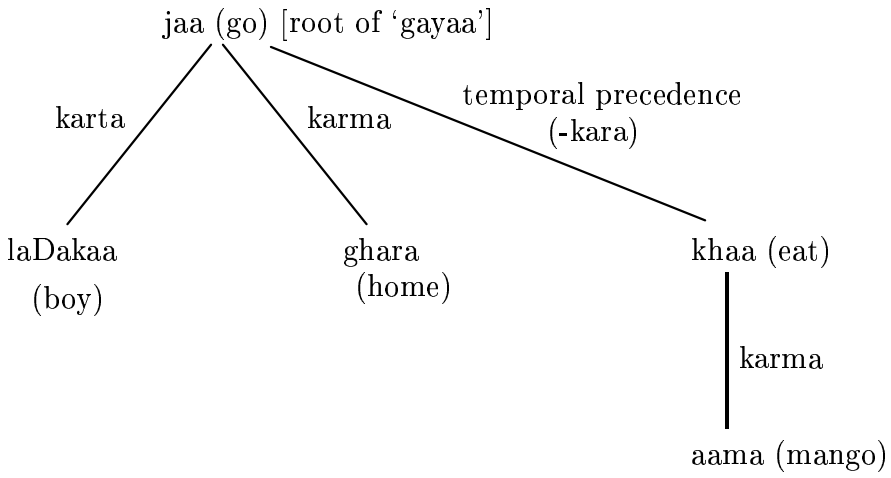


Figure 2.4: A verb-verb modification

'khaa' (eat). Note that since 'laDakaa' occurs only once, it is shown only once even though it happens to be karta of both the verbs. Which verb it modifies, is determined by looking at the sentence construction and such things as agreement. In the sentence under discussion, 'laDakaa' agrees with the verb 'gayaa' (went). If the gender of the noun is changed to feminine 'laDakii' (girl), the gender of the verb also changes to feminine: 'gayii' (went).

6. *Nominal structure with verbal nouns.*

   A verbal noun behaves like a noun but it maintains many of its properties as a verb. In particular, it maintains its relations with its arguments. What this means is that its modifier-modified structures have to be identified as usual for verbs. In the following sentence, for example, even though 'jaanaa' (go) is a verbal noun, it maintains relations with its arguments (Ram and home):

```
rama  kaa ghara jaanaa  mohana ko acchaa lagaa.
Ram   's  home  to-go   Mohan  dat. good  felt
(Mohan felt good at Ram's going home.)
```

   'Jaanaa' itself is an argument of the main verb as shown in Figure 2.5.

   The verbal nouns can usually be identified by their endings or auxiliaries.

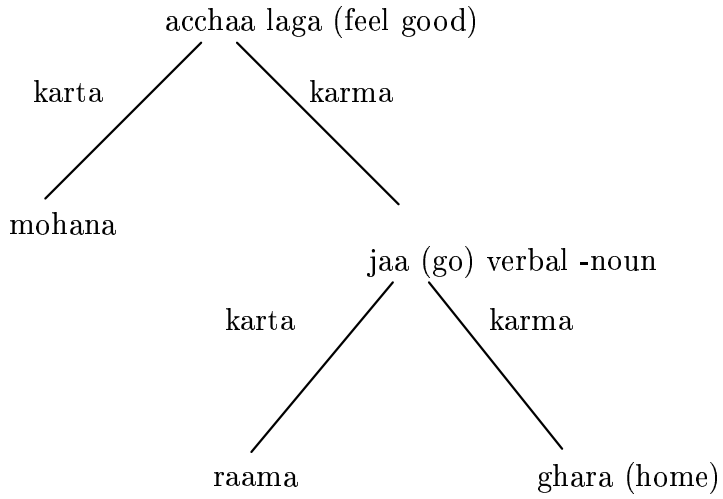


Figure 2.5: Verbal noun

The structures defined here will be added to and enriched in the sections that follow. Building such structures for input sentences will serve as the goal of the NLP systems. In other words, such structures have to be produced automatically after analysis.

2.2 Overview of Language Analyzer

Parsing is a process by which an input sentence is analyzed and assigned a suitable structure. Parsing process makes use of two components: a parser which is a procedural component (a computer program), and a grammar which is declarative. The grammar changes depending on the language to be parsed, while the parser remains unchanged. Thus, by simply changing the grammar, the system would parse a different language. Both the grammar and the parser, however, depend on the grammar formalism. For example, for the Paninian formalism, one can develop a parser which takes any grammar written in the formalism and parses sentences that are specified (or generated) by the grammar. Thus, the parser does not care for the grammar or the language, but only for their inter-relation. Similarly, the grammar for a language depends on the formalism.

One of the key questions that needs to be answered is about the nature of the parse structure. The answer affects the grammar formalism and hence the grammar and the parser in a major way. The choice we have

made is based on the karaka relations. For us the parse structure for a sentence consists primarily of the verbal groups and the nominals in the sentence, and the karaka relations among them. Non-karaka relations may also be present due to adjectives, hetu (purpose) relation, relational words like 'door' (far) and 'paasa' (near), etc. Justification for the choice of parse structure is given in the later chapters (for example, Chap. 5.). Here, we present the parse structure without discussing why it has been chosen.

Take the following sentence as an example:

```
kisaana kheta jotataa hei
farmer farm  ploughs
(Farmer ploughs the farm.)
```

It will be assigned the following parse structure:

```
jota (plough)
  karta: kisaana (farmer)
  karma:  kheta (farm)
```

where 'kisaana' is the karta and 'kheta' is the karma of 'jota'. This is shown diagrammatically in Figure 2.6.

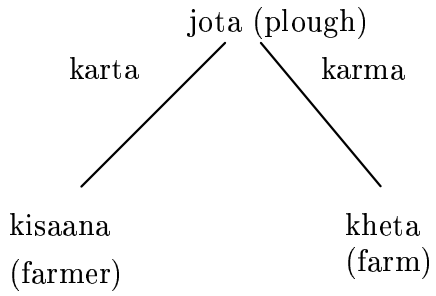


Figure 2.6: A parse structure

Having decided on the parse structure, let us now look at the parser structure. It is fairly obvious that a part of the analyzer, or parser must take care of morphology. For each word in the input sentence, a dictionary or a lexicon needs to be looked up, and associated grammatical information retrieved. The words have to be grouped together yielding nominals, verbal elements, etc. Finally, the modifier-modified relations among the elements have to be identified. This is shown in Figure 2.7. We discuss each part of the parser next.

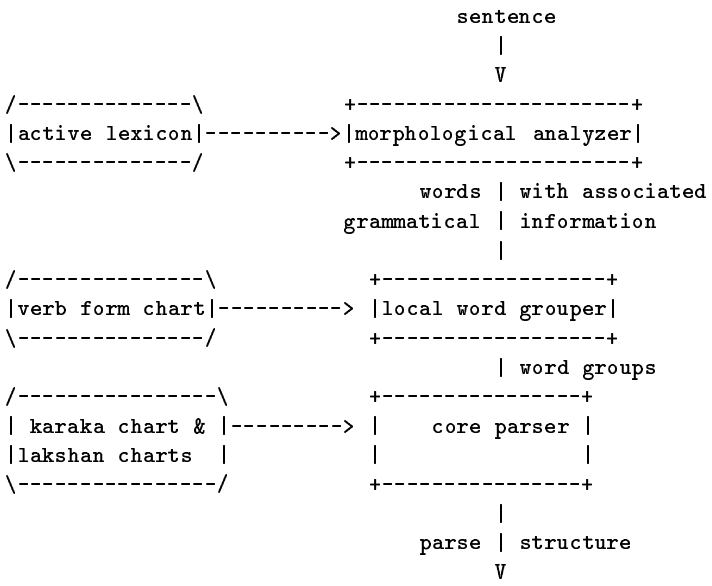


Figure 2.7: Structure of the parser

2.2.1 Morphological Analyzer

The morphological analyzer takes as its input a sentence, that is, a sequence of words. For each of the words, it looks up a lexicon and retrieves such information as the root of the word, its lexical category, gender, number, person, tense, etc. In case, a word has multiple meanings, grammatical information is returned for each of the meanings. For example, the morphological analyzer returns the following information for laDake in Hindi; note that 'laDake' can either be singular-oblique or plural-direct.

```
laDake  -->     [root             : laDakaa
                 lexical category : noun
                 type             : common
                 gender           : masculine
                 number           : plural
                 person           : third
                 case             : direct]

                [root             : laDakaa
                 lexical category : noun
                 type             : common
                 gender           : masculine
                 number           : singular
```

```
                    person           : third
                    case             : oblique]
```

Similarly, the output for 'pahanaa' is:

```
pahanaa  -->    [root             : pahanaa
                 lexical category : verb
                 verb type        : causative
                 gender           : masculine
                 number           : singular
                 person           : first,second or third
                 tam              : 0 ]
```

In the above two examples, each word is either only noun or verb. Word 'haara' which takes multiple lexical categories is shown below.

```
haara -->       [root             : haara
                 lexical category : noun
                 subtype          : common
                 gender           : masculine
                 number           : singular
                 person           : third
                 case             : direct/oblique ]

                [root             : haara
                 lexical category : noun
                 subtype          : common
                 gender           : masculine
                 number           : plural
                 person           : third
                 case             : direct ]

                [root             : haara
                 lexical category : verb
                 gender           : masculine/feminine
                 number           : singular/plural
                 person           : first/second/third
                 tam              : 0 ]

                [root             : haara
                 lexical category : verbal_noun_2
                 gender           : feminine
                 number           : singular
                 person           : third
                 case             : direct/oblique]
```

2.2.2 Local Word Grouper (LWG)

The function of this block is to form the word groups on the basis of the 'local information' (i.e information based on adjacent words).

An example illustrates the job done by the LWG. In the sentence:

```
laDake adhyaapaka ko  haara    pahanaa rahe heiM.
boys   teacher   -ko  garland garland -ing
(Boys are garlanding the teacher.)
```

the output corresponding to the word 'adhyaapaka' and 'ko' from the morphological analysis block will be grouped as one unit by the LWG; similarly, 'pahanaa', 'rahe' and 'heiM' will also form a single unit.

The final output for the word groups below from the LWG will appear as

```
word groups    actual output
[laDake] ---> [noun-group-1        : laDakaa
               noun-group-1-gender : masculine
               noun-group-1-number : plural
               noun-group-1-person : third
               noun-group-1-parsarg: 0 ]
```

(Note that the oblique option of the word 'laDake' has been rejected, because there is no post-position or *parsarg* following 'laDake'.)

```
[adhyaapaka ko] --> [noun-group-2        : adhyaapaka
                     noun-group-2-gender : masculine
                     noun-group-2-number : singular
                     noun-group-2-person : third
                     noun-group-2-parsarg: ko ]
[haara] -->         [noun-group-3        : haara
                     noun-group-3-gender : masculine
                     noun-group-3-number : singular
                     noun-group-3-person : third
                     noun-group-3-parsarg: 0 ]
[pahanaa rahe heiM]-> [verb-root         : pahanaa
                       verb-form         : 0-rahaa-hei
                       verb-gender       : masculine
                       verb-number       : plural
                       verb-person       : third
                       verb-agreement    : karta ]
```

Only those groupings are done by LWG, however, which will need no revision later on. This implies that whenever there is a possibility of more than one grouping for some word, they will not be grouped together by the LWG. For example, because of the two readings of the word 'para' one as parsarg or postposition and the other as a noun, it will not be grouped with 'mora' in either of the sentences below:

```
mora    para     pheilaakara   naaca  rahaa hei.
peacock feathers having-spread dance  -ing  is
(Having spread its feathers, the peacock is dancing.)

tiira mora    para calaanaa aparaadha hei.
arrow peacock -on  fire     crime     is
(It is a crime to give an arrow on the peacock.)
```

This block has been introduced to reduce the load on the core parser resulting in increased efficiency and simplicity of the overall system.

2.2.3 Core Parser

The function of the core parser is to accept the local word groups produced by LWG, and produce the parse structure. The parse structure produced by it is shown at the beginning of this section. Mainly, it identifies karaka relations between the verbs and the nouns.

The core parser makes use of the ideas of 'aakaankshaa and yogyataa ' (demand and merit) from Paninian Grammar. Some words (or word groups) make demands and others satisfy them. For example, verbs (or verb groups) make demands for their karakas, and typically the nouns (or noun groups) satisfy them. However, only certain nouns that have the desired parsarg and semantic properties are eligilble. This is called *yogyataa* or merit of the nouns.

Besides the above, there is another important notion: 'sannidhi' or nearness. This specifies the relations between two constituents when they are close to each other in the sentence.

Aakaankshaa, yogyataa and sanniddhi from PG have been used in building a fast parser which will be described in Chap. 6.

2.3 Requirements of Computational Grammars

To appreciate the properties required of computational grammars, it is important to first understand the needs of natural language processing (NLP) systems which have been built for diverse applications. For example, the applications pertain to retrieving data in response to a user request in natural language, answering questions pertaining to a domain of expertise, following instructions in natural language, machine translation from one natural language to another, and so on. An NLP system must do one or both of two tasks: it must analyze the given natural language utterance, or generate natural language text from the given content.

An NLP system consists of a grammar and ggprocedural components. The grammar is used by the procedural components in performing analysis, generation, etc. For example, a question-answering system would consist of a set of programs (procedural component) and a grammar for the language concerned. The programs would analyze a given question using the grammar and represent the meaning appropriately. A reasoning component produces a representation of the answer. Finally, another part of the procedural component plans what is to be said and then uses the grammar in generating a response in a natural language.

2.3.1 Computational Aspect

This places a new requirement on the grammar, namely, that it should be suitable for processing by the procedural component. An important criterion for judging suitability is how efficiently it can be used in processing. A grammar formalism would be considered superior to another if grammars written in it can be used for processing faster or with lesser computing resources etc. Grammar formalisms that have been designed with processing in mind are called *computational grammar formalisms* or computational grammar models. A grammar written in such a model will be called a computational grammar.

2.3.2 Systems Aspect

System building has its own requirements, which affect among other things, the theory used in building them. First, a working system requires complete detail, nothing can be waived or wished away. This serves to test the theory thoroughly because difficult problems sometimes relegated to details confront us when we try to build a system. Some examples of such details often left unspecified in linguistic theories are choice of features, a detailed lexicon, and world knowledge.

A system forces us to deal with problems which might have been set aside by the field to be dealt with later. For example, a system for question answering will have to deal with pragmatics; it cannot choose to deal with syntax or semantics only. Similarly, such a system may be forced to deal with sentence as well as discourse. Crossing the level boundaries and dealing with hitherto ignored problems produces new ways of segmenting the problem. At times, it produces ad hoc solutions. They are all precursors to a new theory.

On the other hand, there are occasions when a well-developed theory is bypassed in a system because the phenomenon being explained by the theory is not of much interest. For example, most machine translation systems adopt a simple model for morphology for reasons of fast processing

and because derivation of words from morphemes is not very important.

The 80-20 rule also comes into the picture when one tries to have a "reasonable" grammar for a system. This rule states that 20 percent of the grammar covers 80 percent of the language. When one starts with a grammar for 80 percent language and tries to cover the remaining 20 percent of language, a several fold increase in grammar size takes place. This happens with the best of theories. It is a reminder that the world is not as ordered as the theories would like it to be.

2.3.3 Large System Aspect

When the system being built is large and complex, as most NLP systems are, it has implications for theory as well as methodology.

Modularity is a desirable property of large systems. It implies that the system can be divided into several parts in such a manner that the interaction between parts is minimal and clearly specified. This is very important, for example, when large numbers of people have to work as a team in developing the system. The implication for the grammar model is that it should be such that several people can work cooperatively in writing a grammar using the model (or framework).

Extensibility is one of the most important desirable properties of a large system. It means that the system can be extended or changed bit by bit. There are two reasons for this. First, a large real-life system has to keep changing to satisfy changing requirements. It will soon become useless if the system cannot be modified. Second, and more importantly, large systems are not built as finished systems in one shot. They are built in stages from simple to complex. Such a phased construction is possible only if the system built in each phase is extensible.

Extensibility applied to grammars means that when the lexicon is augmented or a new language phenomenon is sought to be covered, it does not lead to extensive rewriting of the grammar.

A system will invariably be faced with situations in which it will fail to perform the tasks it has been designed for. *Dealing with failures*, therefore, must be part of system design. There are several aspects to this problem. When a system detects that it cannot deal with the natural language related task, it must communicate it to the user. The user can then rephrase his request in another manner which the system might be able to handle. Such communication also serves to educate the user about the limitations of the system, so that in future, he might learn to avoid them. The latter implies that for the system to appear friendly, it might be better to avoid handling those natural language phenomena for which there are no adequate solutions. Otherwise it will be difficult for the user to keep track of the limitations.

The system must also provide sufficient *feedback* to the system designers (which includes the grammar writers) regarding its failures. It can help them detect problems in their design (in procedures or grammars) and thus help fix them (or debug them), if possible. Experience with building large systems indicates that *debugging* is an important activity both while building the system and after it starts getting used. This implies for grammar and grammar writing, that the grammar should be so designed that errors and problems can be identified easily. Specialized programming tools are usually necessary to assist in debugging.

Graceful degradation is another desirable property of large systems. When the user requests tasks that approach the limits of the capabilities of the system, the deterioration in system response should be slow. In other words, when a request is given that the system cannot handle properly, it must still respond with partial information; and when more difficult requests are made, the answer gets progressively worse.

Finally, the system should be *tolerant of user errors*. In the case of natural languge interfaces (where natural language is being used to facilitate communication with the computer), such errors might pertain to spellings, sentence constructions, agreement rules, etc. Correction by the system should be implicit without prompting it to the user. Other kinds of errors pertaining to misconceptions ought to be detected and corrected explicitly.

Further Reading

Requirements of computational grammar in Sec. 2.3 are based on Bharati et al. (1992a).

Exercises

2.1 Show the modifier-modified structure for the sentences in Hindi given below. In case of ambiguity show the structures for all the different senses.

```
1. bacce ne apanii kitaaba paDhii.
   child ne self   book    read
   (The child read his/her book.)

2. bacce ko   usakii kitaaba paDhanii paDii.
   child dat. his    book    read     had-to
   (The child had to read his book.)

3. bhaarata kii janataa ne narasiMharaava ko
   India gen.   people -ne NarsimhaRao   acc.  %v2 }

   pradhaanamaMtrii cunaa.
   prime-minister   elected
```

(The people of India elected NarasimhaRao
as the prime minister.)

4. shyaama ko hari kii baatoM para hazsii aa rahii thii.
Shyam dat. Hari gen. talk -par laugh coming
(Shyam felt like laughing at Hari's talk.)

5. mohana shyaama ko buddhimaana maanataa hei.
Mohan Shyam dat. intelligent considers
(Mohan considers Shyam to be intelligent.)

6. adhyaapaka ne vidyaarthii ko apaniii pustaka
teacher -ne student acc. self book

paDhane ke liye kahaa.
to-read asked
(The teacher asked the student to read his book.)

7. raama ne shyaama ko apanii kitaaba dii.
Ram -ne Shyam acc. self book gave
(Ram gave his book to Shyam.)

8. daravaajaa khulane se ThaMDa laga rahii hei.
door to open -se cold feeling
(I am feeling cold because of the opening of the door.)

9. yaha taalaa aasaanii se nahiM khulegaa.
this lock ease with not will open
(This lock will not open easly.)

10. kisa bacce ne kitaaba kaa pannaa phaaDaa thaa.
which child -ne book gen. page did -tear
(Which child tore the page of the book.)

11. raama ne mohana ko kitaaba paDhane ke liye kahaa.
ram -ne mohan acc. book to-read asked
(Ram asked Mohan to read the book.)

12. raama ko ThaMda laga rahii hei.
Ram dat. cold is-feeling
(Ram is feeling cold.)

13. mohana ko yaha kitaaba inaama meM milii.
Mohan dat. this book prize -mein received
(Mohan receive this book as a prize.)

14. tumane bhii likhaa hei kucha.

```
    you-ne also have-written something
    (You have aslo written something.) (marked sentence)

15. aaja  ciTThiyaaz aaii heiM kuch.
    today letters     have-come some
    (Some letters have come today.) (marked sentence)  %v2 }

16. kaanapura ina   dinoM bahuta ThaMDa hei.
    Kanpur    these days  very   cold
    (These days kanpur is very cold.)

17. baccaa douDakara ghara jaanaa caahataa hei.
    child  running   home  go     wants
    (The child wants to go home running.)

18. mohana ko   siradarda ho rahaa hei.
    Mohan  dat. headache occurring
    (Mohan has a headache.)

19. mohana ke   peTa     meM darda hei.
    Mohan  gen. stomach -meM pain  is %v2 }
    (Mohan has stomachache.)

20. mohana ne Shyaama se kahaa ki miThaaii khaanaa
    Mohan -ne shyam dat. said that sweet   to-eat

    acchii baata hei.
    good   thing is
    (Mohana told Shyam that eating sweets is a
     good thing.)
```

2.2 Identify modifier-modified relation and the nature of modification among constituents in each of the following sentences:

```
1.  raama paanii piikara        ghara gayaa
    ram   water  having-drunk   home  went
    (Ram went home having-drunk water)

2.  raama ne phala kaaTane ke liye churii maazgii
    ram  -ne fruit to-cut -ke-liye knife  asked
    (Ram asked for a knife to cut the fruit)
```

2.3 Identify modifier-modified relation and the nature of modification among constituents in each of the following sentences:

```
1.  Ram said that Mohan is crying in the room.
2.  Ram persuaded Mohan to go to the market.
```

2.4 Show the modifier-modified structure for the dominant sense of the following sentences in Hindi. Alternatively, you may translate the Hindi sentence to your mother tongue and show its structure.

```
1.  laDake   ne    phuula   dekhaa.
    boy      erga. flower   saw
    (The boy saw a flower)

2.  laDake  ne   jo    phuula  dekhaa, vaha baaga  meM
    boy    erga. which flower  saw     that garden in

    ugaa  huaa  thaa.
    growing      was
    (The flower that the boy saw was growing in the garden)

3.  laDake ne  phuula jisa  baaga  meM dekhaa, vaha peDoM
    boy  erga. flower which garden in  saw,    that trees  %v2 }

    se   bharaa   thaa.
    with filled
    (The garden in which the boy saw the flower was filled
     with trees.)
```

2.5 Show the modifier-modified structure for the isolated sentences in English translated to your mother tongue given below (In case of ambiguity, show for the dominant sense.):

```
1.  The boy plucked the flower quickly.
2.  The boy plucked a red flower at 5.
3.  The boy plucked the flower which was growing
    in the garden.
4.  The boy plucked the flower growing in the garden
5.  The boy plucked the flower running in the garden.
6.  The boy plucked the flower which was against the
    rules.
7.  The boy who plucked the flower was walking in the
    garden.
8.  The flower that was plucked was growing wild.
```

2.6 Write a grammar for time expressions in your language such as: five o'clock, before five o'clock, half past three, soon after the eruption, one minute earlier than the firing, three days later, tomorrow, day after tomorrow, two days and three hours after the fire, after the second explosion, etc.

Chapter 3

Words and Their Analyzer

3.1 Introduction

We define a word to be a sequence of characters delimited by spaces, punctuation marks, etc. in case of the written text. There is no difficulty in identifying words in the written text entered into the computer because one simply has to look for the delimiters.

A word can be of two types: simple and compound. A simple word[1] consists of a root or stem together with suffixes or prefixes. A compound word (also called a conjoined word) can be broken up into two or more independent words. Each of the constituent words in a compound word is either a compound word or a simple word and may be used independently as a word. On the other hand, the root and the affixes, which are constituents of a simple word, are not all independent words and cannot occur as separate words in the text.

Constituents of a simple word are called morphemes or meaning units. The overall meaning of a simple word comes from the morphemes and their relationships. Similarly, in case of a compound word, its meaning follows from its constituent words and their inter-relationships.

It should be noted that we have taken a pragmatic position regarding words. Anything that is identifiable using the delimiters is a word. This is a convenient position to take from the processing viewpoint. Similar is the case with the definition of compound words.[2]

[1] A simple word is what is usually called the word.

[2] It should be noted that linguists have paid a great amount of attention to the concept of word. They have tried to address the twin issues of: how it is pronounced, and how it gets its meaning. The units of pronunciation are the phonemes, while the units of meaning are morphemes. For the latter, they have tried to explore various issues as illustrated by the following examples: Is "cannot" a separate word? What is its relationship to "Can't"? Clearly both of them share meaning with "can not"; however,

With the above definition, an analyzer of words in a sentence does not have to do much work in identifying a word: it simply has to look for the delimiters.[3] Having identified the word, it must determine whether it is a compound word or simple word. If it is a compound word, it must first break it up into its constituent simple words before proceeding to analyze them. We call the former as Sandhi analyzer and the latter as morphological analyzer, both of which are important parts of a word analyzer.

The detailed linguistic analysis of a word can be useful for NLP. However, most NLP researchers have concentrated on other aspects, e.g., grammatical analysis, semantic interpretation, etc. As a result, NLP systems use rather simple morphological analyzers. Consequently, our discussion too would be limited to rather simple kinds of morphological analyzers.

So far we have talked about analyzers. A generator does the reverse of an analyzer. Given a root and its features (or affixes), a morphological generator generates a word. Similarly, a sandhi generator can take the output of a morphological generator, and group simple words into compound words, where possible.

3.2 Why Morphological Analysis

The first question we need to address is why we need to perform morphological analysis at all. If we had an exhaustive lexicon which listed all the word forms of all the roots, and along with each word form it listed its features values then clearly we do not need a morphological analyzer. Given a word, all we need to do is to look it up in the lexicon and retrieve its feature values. For example, suppose an exhaustive lexicon for Hindi contains the following entries related to the roots laDakaa and kapaDaa:

words can intervene now as in "can certainly not". The position adopted in NLP is not opposed to the linguistic position. NLP has simply taken a short cut. It is not opposed to taking or benefiting from the linguistic analysis in the long run.

[3]It does not have to decide whether "can't" is a word, for example. If it can occur in a sentence separated by spaces, it is a word and must be analyzed.

Word form	*Category*	*Root*	*Gender*	*Number*	*Person*	*Case*
laDakaa	noun	laDakaa	masc.	sg.	3rd	direct
laDake	do.	do.	do.	pl.	do.	do.
laDake	do.	do.	do.	sg.	do.	oblique
laDakoM	do.	do.	do.	pl.	do.	do.
laDaka-pana	do.	laDaka pana	do.	sg.	do.	any
kapaDaa	noun	kapaDaa	masc.	sg.	3rd	direct
kapaDe	do.	do.	do.	pl.	do.	do.
kapaDe	do.	do.	do.	sg.	do.	oblique
kapaDoM	do.	do.	do.	pl.	do.	do.
kapaDaa-pana	do.	kapaDaa pana	do.	sg.	do.	any

Now, given a word, it can be looked up and its feature values returned.

The above method has several problems. First, it is extremely wasteful of memory space. Every form of the word is listed which contributes to the large number of entries in such a lexicon. Even when two roots follow the same rule, the present system stores the same information redundantly.

Second, it does not show relationships among different roots that have similar word forms. Thus, it fails to represent a linguistic generalization. This is necessary if the system is to have the capability of understanding (even guessing) an unknown word. (In fact, human beings routinely deal with word forms they have never heard before when they know the root and the affixes separately.) In the generation process, the linguistic knowledge can be used if the system needs to coin a new word.

Third, some languages have a rich and productive morphology. The number of word forms might well be infinite in such a case. Clearly, the above method cannot deal with such languages.

Morphological analysis with different degrees of sophistication can be carried out. As discussed in the last section, most NLP systems use simple linguistic theories for morphological analysis. The scheme we will describe focuses on the first issue (space requirement). It can deal with the second issue partially. However, when the morphology becomes rich (issue 3 above), the scheme will not be able to handle it.

There is another criterion by which to judge a morphological analyzer or a scheme for morphological analysis. This is the speed with which it performs the analysis. In case of the exhaustive lexicon, the time spent in analysis is zero, the only time needed is in searching and retrieving a word from the lexicon. As the analysis scheme becomes more sophisticated, it is also likely to take more time. A proper balance may, therefore, have to be struck. The schemes popular in NLP have chosen speed over the

requirements of dealing with unknown words etc.[4]

	Case	
Number	*direct*	*oblique*
Singular	laDakaa	laDake
Plural	laDake	laDakoM

Vocative case has not been shown to keep the example simple.

Figure 3.1: Word forms for the root laDakaa

	Case	
Number	*Direct*	*Oblique*
Singular	(0, ϕ)	(1, e)
Plural	(1, e)	(1, oM)

Each entry in the table shows the number of characters to be deleted from the root and the string to be suffixed.

Figure 3.2: Paradigm table for 'laDakaa' class

3.3 Morphological Generation Using Paradigms

The scheme we will describe now is based on paradigms. This or a variant of this scheme has been used widely in NLP.

The linguist or the language expert is asked to provide different tables of word forms covering the words in a language. Each word-forms table covers a set of roots which means that the roots follow the pattern (or paradigm) implicit in the table for generating their word forms. For example, in Hindi the paradigm for laDakaa and other roots in its class can be specified by giving its word forms.[5] Other roots such as 'kapaDaa' (cloth) behave like laDakaa and belong to the same paradigm.

The paradigm can be extracted from the word forms of laDakaa by identifying the number of characters to be deleted from the root and the

[4]It is possible to build a system that uses a fast and simple scheme for normal operation, and a complex and powerful scheme in case the simpler scheme fails or the word is not found.

[5]Note that oblique or direct case refers to the fact whether the word must or must not be followed by the post positional marker. laDakoM, for example, is oblique case because it must be followed by a postpositional marker such as 'ne', 'ko', etc.

characters to be added to obtain the word forms. For example, by looking at Figure 3.1 we can say that if you want plural, oblique case of the root laDakaa, delete the last character ('aa') and add ('oM') at the end, where 'aa' is a single character in Devanagri script and in the internal representation in the computer (which is 'ladakA' as given in the notation in Appendix B, Table 2):
[root = laDakaa, number = plural, case = oblique] → laDakoM

This can be expressed in the form of a table (Figure 3.2) and the reader can verify by looking at Figure 3.1 that it is correct:

Algorithm 3.1 Forming paradigm table

Purpose: To form paradigm table from word forms table for a root

Input: Root r, Word forms table WFT (with labels for rows and coloumns)

Output: Paradigm table PT

Algorithm:

1. Create an empty table PT of the same dimensionality, size and labels as the word forms table WFT.
2. For every entry w in WFT, do

   if w = r
   then store "(0,ϕ)" in the corresponding position in PT.
   else begin

   let i be the position of the first characters in w and r which are different
   store (size(r) - i + 1, suffix(i,w)) at the corresponding position in PT
3. Return PT

End algorithm

Along with the roots can be stored the types and other grammatical information that is common to all the associated endings (that is, word forms). Figure 3.3 shows some example roots together with common gender information.

Note that the endings of type (n,laDakaa) are applicable to laDakaa as well as kapaDaa (cloth), ghoDaa (horse), etc. Similar is the case with roTii (bread), laDakii (girl), lakaDii (wood), etc. The paradigm table can be

Root	*Type*	*Gender*
laDakaa	(n,laDakaa)	m
kapaDaa	(n,laDakaa)	m
bhaaSaa	(n,bhaaSaa)	f
roTii	(n,laDakii)	f
laDakii	(n,laDakii)	f

Figure 3.3: Dictionary of roots

used with any of the roots in the same class to generate its word forms. For example, kapaDoM can be generated from root kapaDaa, number plural, and case oblique, by deletion and addition as specified by the paradigm table. See Algorithm 3.2.

This leads to efficient storage because there is only one paradigm table for a class of roots rather than a separate word forms table for each root.

Algorithm 3.2 Generating a word form

Purpose: To generate a word form given a root and desired feature values

Input: Root r, Feature values FV

Uses : Paradigm tables, Dictionary of roots DR, Dictionary of indeclinable words DI

Output: Word w

Algorithm:

1. If root r belongs to DI
   then return (word stored in DI for r irrespective of FV)
2. let p = paradigm type of r as obtained from DR
3. let PT = paradigm table for p.
4. let (n,s) = entry in PT for feature values FV
5. w := r minus n characters at the end
6. w := w plus suffix s

End algorithm

Note that the roots in the dictionary of roots should be kept sorted. This allows us to obtain the entry for a root (step 2 in Algorithm 3.2) very rapidly. Our own intuition tells us that searching for an item in a sorted list is much faster. We have all experienced that searching for a word in a

dictionary is very rapid because the words are sorted. What would happen if the words were not sorted in a dictionary![6].

We have so far talked of generating a word using paradigms. In fact, the word form table given by the language expert is from the point of view of generation. It is set up so that given a root and the desired features, one can locate the right table and then look up the right entry. It is not surprising, therefore, that the paradigm table is also set up for generation.

3.4 Morphological Analysis Using Paradigms

The important question is how the paradigms (which are specified for generation) can be used for analysis. Analysis and generation are the inverse of each other. Human experts find it easier to specify solution to the generation problem. It is the task of the computational linguist (one whose primary background is in computer science) to solve the indirect or the inverse problem. Solution of the indirect problem requires some amount of search.

There are other instances of similar inverse problem in other domains. Humans find it easier to specify solution to one of the problems, call it the direct problem. For example, how to multiply two integers is a direct problem whose solution is neatly provided. The indirect problem, namely division, between two integers, requires some amount of search, using multiplication (the solution to the direct problem). A moment's thought would reveal that the most commonly used division algorithm for decimal numbers, actually involves a trial and error (search) at each step, to obtain a single digit which is part of the answer.[7] Other examples of inverse problem pairs are tying a knot and untying it, climbing a ladder up and climbing down, differentiation and integration, encryption and decryption, etc.

Let us now outline a method based on search of paradigm tables for doing morphological analysis. Suppose for example, we are given kapaDoM and are asked to find its root and feature values. Assume further that we have only two paradigm tables for laDakaa paradigm and bhaaSaa paradigm (Figure 3.4). The first step is to see whether kapaDoM occurs as an indeclinable word. This check would be performed on a dictionary of indeclinable words, which should be available separately.

The next step is to check all the entries in all the paradigm tables having the last character 'M' in the suffix. There happens to be none.

The next step is to check all the entries in all the tables that have 'oM' as the suffix string. There turn out to be two entries. They are:

[6]The technical name given to a rapid method of searching for an item in a sorted list is *binary search*.

[7]The search step involves multiplying the divisor by a single digit to find the largest such digit without exceeding the appropriately sized leftmost part of the dividend.

Word forms table for bhaaSaa

	Case	
Number	*direct*	*oblique*
Singular	bhaaSaa	bhaaSaa
Plural	bhaaSaaeM	bhaaSaaoM

Paradigm table for bhaaSaa

	Case	
Number	*direct*	*oblique*
Singular	(0, ϕ)	(0, ϕ)
Plural	(0, eM)	(0, oM)

Figure 3.4: Word forms and paradigm table for bhaaSaa

In laDakaa paradigm table : (1, oM)
In bhaaSaa paradigm table : (0, oM)

For each of the entries, add as many characters as are shown from the root of the paradigm table. We get two roots kapaDaa and kapaD respectively. These can now be checked in the dictionary of roots (Figure 3.3). The former occurs in the dictionary. It is now checked whether it has the same paradigm in whose table its suffix has matched. As the check turns out to be true, kapaDaa is identified as the root. Grammatical features associated with 'kapaDaa' in the dictionary of roots and with suffix 'oM' in the paradigm table together constitute an answer (or a lexical entry) for 'kapaDoM'. The system continues searching for additional suffixes such as 'oM', 'adoM' etc. In case additional answers are found (none in this example) they would also be returned. Figure 3.5 shows inputs and outputs to morphological analyzer. Algorithm 3.3 gives the exact procedure.

Algorithm 3.3 Morphological analysis using paradigm tables.

Purpose To identify root and grammatical features of a given word.

Input A word w

Output A set of lexical entries L (where each lexical entry stands for a root and its grammatical features)

Uses Paradigm tables, Dictionary of roots DR, Dictionary of indeclinable words DI.

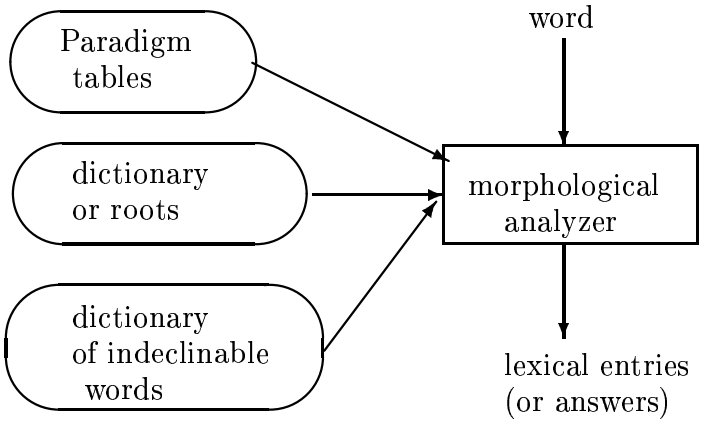


Figure 3.5: Morphological analyzer input-output

Algorithm

1. L := empty set
2. If w is in DI with entry b then add b to L.
3. for i := 0 to length of w do

   let s = suffix of length i in w
   for each paradigm table P
   for each entry b (consisting of a pair) in P do
   if s = suffix in entry b then
   begin
   - r = root of paradigm table P
   - j = number of characters to be deleted as shown in b
   - proposed-root = (w - suffix s) + suffix of r consisting of j characters
   - If (proposed-root is in DR) and (the root has paradigm P)
     then construct a lexical entry l by combining (a) features given in DR with the proposed-root, and (b) features associated with e.
   - Add l to set L

   end of begin
   end for every entry in P
   end for every paradigm

end for every i

4. If L is empty
   then return "unknown word w"
   else return (L)

End algorithm

3.5 Speeding Up Morphological Analysis by Compilation*

The method outlined in Algorithm 3.3 for morphological analysis is quite expensive in time. Each entry in every paradigm table is scanned and compared with every possible suffix for a given word. This search can take a long time. It turns out that the search can be speeded up enormously by analyzing the paradigm tables and the information contained therein beforehand, and generating another table or data structure suitable for this search. This process is called compilation. Once the new data structure is generated, it is used by the morphological analyzer, and the paradigm tables are no longer needed. (See Figure 3.6). For the compilation process for a language to start, all the paradigm tables must be available for the language.

The data structure we shall describe now is based on the sorted list of suffixes from the paradigm tables (call it sorted reverse suffix table). The compiler takes all the entries from all the paradigm tables, and rearranges them after sorting on the reverse of suffixes. Thus, the table in Figure 3.7 would be generated if we had only two paradigm tables for laDakaa and bhaaSaa in our language.

Once the table in Figure 3.7 is generated, the original paradigm tables (of Figure 3.4) are no longer needed. The two tables have exactly the same information. What is different is their organization which is responsible for the speed up. The key idea used here is that the suffixes are ordered. Therefore, given a suffix of a given word, it can be located very quickly without scanning all entries in all the paradigm tables.

It is the same saving in time one obtains in searching for a word in a human-readable dictionary when the entries are in alphabetical order versus in arbitrary order. Algorithm 3.4 gives the exact details, it should be compared with Algorithm 3.3 by the reader.

Algorithm 3.4 Morphological analysis using sorted reverse suffix table. (Compare with Algorithm 3.3)

[7]* This section requires a greater background of computer science and may be optionally skipped.

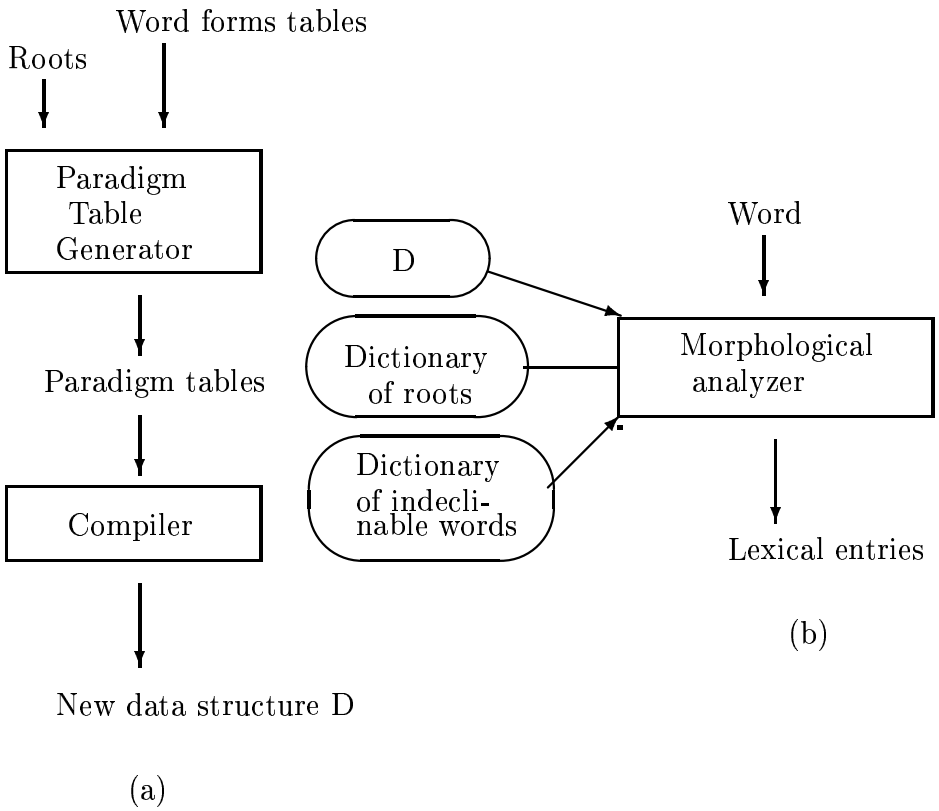


Figure 3.6: (a) Compilation of paradigms; (b) Morphological analysis

Reverse suffix	*Character string to be appended to the prefix*	*Grammatical features (number, case, paradigm type)*	*Example word (for the reader not for the machine)*
ϕ	ϕ	s,d, laDakaa class	laDakaa
ϕ	ϕ	s,d, bhaaSaa	bhaaSaa
ϕ	ϕ	s,o, bhaaSaa	bhaaSaa se
e	A	s,o, laDakaa	laDake se
e	A	p,d, laDakaa	laDake
Me	ϕ	p,d, bhaaSaa	bhaaSaaeM
Mo	A	p,o, laDakaa	laDakoM se
Mo	ϕ	p,o, bhaaSaa	bhaaSaaoM se

Figure 3.7: Sorted reverse suffix table

Purpose To identify root and grammatical features of a given word.

Input A word w

Output A set of lexical entries L (where each lexical entry stands for a root and grammatical features)

Uses Sorted reverse suffix table RST, Dictionary of roots DR, Dictionary of indeclinable words DI.

Algorithm

1. L := empty set;
2. If w is in DI with entry b
   then add b to set L.
3. for i := 0 to length of w do
   let S = suffix of length i in w
   if reverse(s) is in RST
   then for each entry b associated with reverse(s) in RST do
   begin
   proposed-root = (w - suffix s) + string to be added from root
   if (proposed-root is in dictionary of roots) and (the root has paradigm type P in b)
   then
   - construct a lexical entry l by combining (a) features given for the proposed root in DR, and (b) features associated with b.
   - add l to L

   end of begin for each entry
   end for i
4. If L is empty then return ("unknown word w") else return (L)

End algorithm

3.6 Morphological Analyzer – Some Additional Issues*

[7] * This section requires a greater background of computer science and may be optionally skipped.

There are alternatives to the sorted reverse suffix table data structure. A major alternative is the use of tries. *Tries* are trees in which keys are stored in the path starting from root node, and not in the nodes themselves. For example, Figure 3.8 shows tree and trie structures for the reverse suffixes for laDakaa and bhaaSaa paradigms from Figure 3.7. For the four different

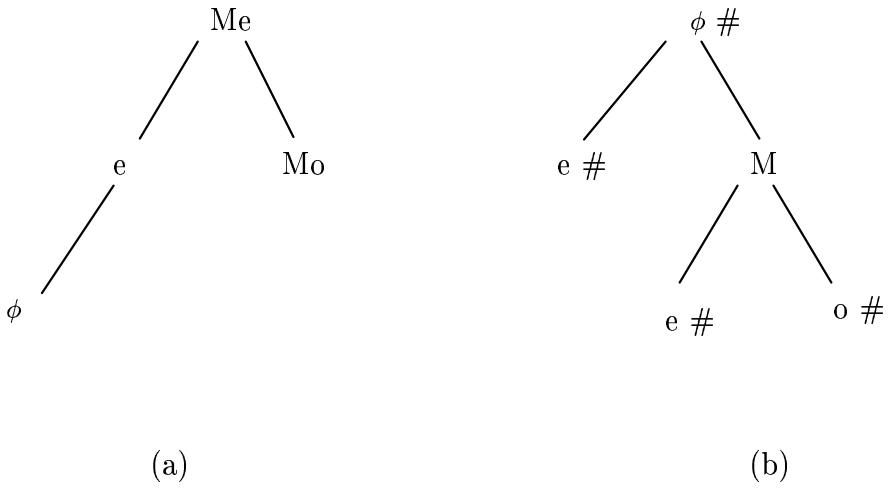


Figure 3.8: (a) Ordered binary tree and (b) Trie structures

suffixes there are four nodes in the binary tree (in Figure 3.8 (a)), that is, one each for a distinct suffix.[8]

In the trie shown in Figure 3.8(b), all possible paths starting from the root node are ϕ, ϕe, ϕM, ϕMe, ϕMo.[9] Of these, only four marked by '#' stand for suffixes. Information regarding paradigm class, grammatical features, etc. (same as in Figure 3.7) can be associated with '#' or the suffixes in the trie.

Tries permit rapid matching of partial suffixes. For example, if 'laDakoM' word is to be analyzed, one can begin backwards from the word, matching character by character in the trie. Each time a '#' mark is encountered, it is a valid suffix for some root. At each such point, a root is to be proposed and checked in the dictionary of roots.

[8] As shown in Figure 3.8(a), an ordered binary tree has the property that all the nodes in the left sub tree of the root node have keys (suffixes) that occur alphabetically before the key in the root. (For example, ϕ and e occur before Me). Similarly all keys in the nodes in the right sub-tree of the root node occur alphabetically after the key in the root. (For example, Mo occurs after Me.) This property holds for roots of the subtrees and so on.

[9] ϕ is the empty string which prefixed or suffixed with any other character or string leaves it unchanged. Thus, 'ϕe' is same as 'e'.

A number of variations on tries are possible. One can build tries from left to right rather than backwards. In such a case, the roots would be in the trie. Suffixes corresponding to each paradigm class would be stored in a separate trie, which would be linked to all the roots belonging to the same paradigm class (see Sangal (1991) for details).

The most important factor determining what choice to make is the availability of main memory. Tries do not work well on secondary storage such as disks. If very large amount of main memory is available (say, tens of megabytes) the prefix tries can be used. On the other hand, if main memory available is limited (say, tens to hundreds of kilobytes) the reverse suffix tries can be kept in main memory while the dictionary of roots can be stored on the disk.

For a language such as Hindi which has a simple morphology, a few tens of kilobytes are sufficient for storing the reverse suffix tries. For a South Indian language such as Telugu, a few hundred kilobytes might be needed. A dictionary of roots for even a medium sized dictionary (say, 30000 roots) is likely to occupy a few megabytes.

Finally, if the main memory available is very small (say, a few kilobytes)[10] both the suffixes and the dictionary of roots have to be stored on the disk. In such a case, it might be preferable to use sorted reverse suffix table over tries.

Indexes can also be built for sorted files on disk, in particular, for dictionary of roots, sorted reverse suffix table, dictionary of indeclinable words, etc. An index speeds up search of sorted files. Also suitable are dynamic data structures over disk, such as B-trees.[11] Since indexes are much smaller than the files containing actual data, they can be loaded and kept in main memory. It should be mentioned that not much work has been done on storing tries on disk.

Finally, there is the issue of how often are new paradigms added. We should recall that paradigms can be compiled only if all the paradigms for a language are available together. This means that having compiled the paradigms, if we come across a new paradigm (say, when the morphological analyzer is in use), there is no easy way to add a new paradigm. It would require a recompilation and regeneration of the data structure (say, trie, sorted table), before the system would start recognizing and using the new paradigm.[12]

[10] Small amount of memory may be available either because the computer has a small amount of physical memory or because another application can spare only a small amount of memory for morphological analysis. An example of the latter would be an editor or a word processor which can spare only a small amount of memory for morphological analyzer which is part of a spelling checker.

[11] B-trees are different from ordered binary trees.

[12] Normally, a (re)compilation takes a large amount of time (a few minutes to a few hours depending on the language) when compared to morphological analysis (a fraction

Thus, there is a need for developing programs for incremental compilation. Such programs can take an additional paradigm and an existing data structure, and incrementally change the data structure without having to recompile everything. It should be clarified that an unknown word but belonging to an existing paradigm does not pose a major problem. It can be added to the dictionary of roots, and a recompilation (of suffixes) is not needed. In any case, it is hoped that once exhaustive paradigms are available, there will seldom be a need to add new paradigms. The issue of recompilation would then cease to be of importance.

Further Reading

Tree data structures and sorted arrays are described in a large number of texts in computer science. See for example, any of the following texts: Horowitz and Sahni (1978), Reingold and Reingold (1989), or Wirth (1973). Knuth (1973) is an encyclopaedic work. Sorted files, B-trees and indexes are described in books on databases. See a textbook such as Ullman (1987) or Date (1987). Wiederhold (1982) goes into elaborate detail analyzing seek and latency times for disk access. Work on signatures not discussed here also has importance in further speeding up access. See Knuth (1973) for a description of signatures.

Exercises

3.1 Choose five different nouns in your mother tongue. By considering their different forms, identify the parameters (such as gender, number etc.) on which the forms depend. Now build a table of word forms for each of the nouns.

3.2 For one of the tables in Exercise 3.1 create a paradigm table using paper and pencil.

3.3 Repeat Exercise 3.1 but for five verbs instead of five nouns.

3.4 How many different forms (such as ghuumataa, ghuumanaa) do verbs in your language take? For each of the forms identify the gender, person, tense etc.

3.5 Comment on the suitability of the scheme described in this chapter for verbs in your language. Can you propose a better alternative?

3.6 Do a comparative study of efficiency of storage and processing time between using tries and indexed sequential file.

3.7 Vowel harmony can simplify the word paradigms and reduce their number. What would need to be done, if we want to incorporate vowel harmony in the morphological analyzer.

---

of a second).

Chapter 4

Local Word Grouping

4.1 Introduction

Indian Languages have relatively free word order; still there are units which occur in fixed order. The most important examples of these are the main verb followed by auxiliary verb sequences and nouns followed by postpositions. We term such units as verb groups and noun groups respectively. It may be noted that verb groups and noun groups will be sub-parts of what are normally called verb phrases and noun phrases, respectively, in English. However, in our formulation we do not use the concepts of noun phrases and verb phrases for the following reasons:

1. The concept of verb phrase does not seem natural for Indian languages.

2. From computational point of view, recognition of noun phrases and verb phrases is neither simple nor efficient.

On the other hand, noun groups and verb groups can be formed using only local (also called surface) information and more importantly they provide sufficient information[1] for further processing of the sentence according to Paninian karaka theory. So the local word grouping provides all the necessary information with minimum computational effort.

Another important point about our local verb grouping is that we do not attempt to distinguish all the fine shades of semantics associated with these verb sequences. Our experience with data from various Indian languages suggests that to a large extent these fine shades are conveyed by

[1] This information consists of 'prayoga' and 'vibhakti transformation rules' to be discussed in Chap. 5.

identical conventions among Indian languages and so for translation purpose we need not disambiguate them. This approach is also consistent with Indian grammatical analysis where meaning is extracted in several layers with increasing precision.

The third important point is about our strategy for grammar design. As noted by Patanjali any practical and comprehensive grammar should be written in 'utsarga apavaada' approach. In this approach rules are arranged in several layers each forming an exception to the previous layer.

4.2 Verb Groups

As described in the last section, there are typically two kinds of word groups consisting of nouns and parsargs (or postpositions), and main verbs and auxiliary verbs, respectively. The exact groups formed, vary from language to language. For example, in Hindi, sequence of verbs, main and auxiliaries, occur as separate words and hence have to be grouped together. In the south Indian languages they occur together, conjoined into a single word; as a result, the analysis is left to morphology rather than to the local word grouper.

4.2.1 Kriya Rupa Charts

The kriya rupa charts specify the groups to be formed out of the sequence of verbs which denote a single action. Take for example 'khaa rahaa hei' in Hindi which consists of three verbs in a sequence, but they together denote a single act of eating. The role of the auxiliary verb is to give information about tense, aspect, modality etc.

To enable the machine to do the job of grouping, the following information will be required

1. possible verbal roots in sequences
2. information about gnp agreement

Rules of gender, number, person (gnp) agreement are comparatively complicated in Hindi. The normal case would be a verb form like 'khaataa rahtaa hei', where the gender and number of each verb coincides with the gnp of the whole sequence. But consider the following verb sequences

1. khaatii rahatii heiM
2. paDhate rahanaa hei

In (1), the plurality of the whole sequence is reflected only in that of the last verb whereas in (2) the plurality is independent of the individual words (Figure 4.1).

Sentence and (gender, number) of individual verbs				*Overall TAM and (g,n) of verb group*
1. laDakaa	paDhataa (m,s)	rahataa (m,s)	hei. (*,s)	taa_raha_taa_hei (m,s)
2. laDakiyaaM	paDhatii (f,s)	rahatii (f,s)	heiM. (*,p)	taa_raha_taa_hei (f,p)
3. laDakiyaaM	paDhatii (f,s)	rahatiiM (f,p)	heiM. (*,p)	incorrect sentence
4. laDake ko	paDhate (m,s)	rahanaa (m,s)	hei. (*,s)	taa_raha_naa_hei (*,*)
5. laDake ko	paDhataa (m,s)	rahanaa (m,s)	hei. (*,s)	incorrect sentence

Legend:
m = masculine, f=feminine, s=singular, p=plural
$*$ = don't care (no constraint on value)

Figure 4.1: Table of some verb sequences

The linguist will have to provide this information. The first task therefore is to describe the above information in some fixed format. Take the three verb-forms in Hindi,

1. khaataa rahataa hei
2. khaatii rahatii heiM
3. khaate rahanaa hei

The linguistic information for (1) and (2) will be expressed as follows:

```
label          : taa_raha_taa_hei
vg-gnp         : [-2, -1, -1]
seq-agree-spec: taa [agr,agr,-] rahataa [agr,agr,-]
                hei[-,agr,agr]
prayog         : karta
karaka transformation rule: normal
```

The label field above indicates the raw TAM label (TAM stands for tense aspect modality) for the (entire) verb group. It consists of concatenation of the tam of the main verb, followed by the roots and raw tams of the remaining verbs, separated by '_'. In the case of Hindi, the raw tam of a verb is simply the ending of the verb. The label is unique for the verb sequence (root of the main verb not included), and can be used for getting the above specification out of all such specifications that might be there. Seq-agree-spec is a specification giving the sequence of verb roots and their

tams, and the rules of agreement. Vg-gnp gives the specification for filling gender, number, and person of the verb group. Prayog indicates the kind of sentence in which this sequence can occur.

In the example above, the square brackets give the specifications regarding gender, number, and person, respectively. Vg-gnp indicates how the gnp of the verb group is to be arrived at from the gnp of its constituent verbs. First number in square brackets, -2 indicates that gender of the verb group is to be obtained from the gender of the second last verb in the sequence, and the following -1's indicate that number and person, respectively, of the verb group take their values from the last verb (i.e., -1 when counted backwards). seq-agree-spec shows that if the first (main) verb ends in taa, and is followed by 'rahataa' and 'hei', the sequence is possibly grammatical. (Actually this information has already been used in constructing the label and retrieving the specification.)

In the square brackets after taa (tam of the main verb), the first agr indicates that the gender of the main verb should agree with the gender of the verb group, the second agr indicates that the number of the main verb should agree with the number of the verb group, and '-' indicates don't care for person of the verb.

Normally, the agreement rules described above are followed. There is an exception in Hindi, however. Whenever the verb group is feminine plural, the specified agreement rules are not used. Instead, the last verb must be plural and all else singular. The gender and person of the verb group is still obtained using vg-gnp.

The above suggests the method or the algorithm to be followed by the local word grouper. Given a sequence of verbs, V_1 to V_n , the label is formed by taking the tam of V_1 and concatenating with the roots and tams of the verbs V_2 to V_n . Using the label, the specification is obtained. Vg-gnp is now used to obtain the gnp of the verb group. If an exception condition occurs (e.g., feminine plural in Hindi), the agreement rules associated with the exception are tested. Otherwise, the agreement rules with the specification (associated with the label) are applied. In case of success, the verb group is formed and processing continues at the word after V_n . In case of failure of agreement rules, the above process is repeated with V_1 to V_{n-1}. The process of forming a verb group starting from V_1 terminates when either a verb group is formed using more than one verb besides V_1, or a verb group consisting of single verb V_1 is formed. In the latter case, gnp of the verb group is the same as that of V_1 . (More specialized methods can be used to avoid forming labels and checking for those sequences which cannot possibly occur. Such methods are being used for Hindi. For example, intermediate verb groups contain at most two verbs; so we test for them directly. These can again be generalized after more data is available from other Indian languages.)

4.3 Noun Groups

Noun groups are formed out of nouns and parsargs. In Hindi, each of the parsargs gets grouped with the preceding nouns. For example, the parsargs in each of the lines below, get grouped with the preceding noun laDake (boy).

```
laDake ne
laDake ke liye
laDake kii
```

For the noun to participate in the grouping, it must be in oblique form. For example, laDake has two possible lexical entries, one corresponding to singular-oblique and the other to plural-direct. (The oblique case for Hindi implies that noun takes a parsarg.) Only the singular-oblique lexical entry takes part in grouping with the parsarg. The number of the noun group is set to singular.

The actual parsarg encountered (possibly empty) is stored in the noun group as vibhakti. In morphologically rich languages, the noun itself gets declined depending on its relationship with the verb. In languages less rich in morphology, the parsarg serves a similar purpose. By incorporating both these in the group and calling it vibhakti, we are able to deal with it in a uniform way during core parsing.

In the sentence,

```
laDake phala   khaa rahe heiM.
boys   fruits  eat -ing  are
(Boys are eating fruits.)
```

the first noun group is formed out of laDake alone. Only the plural-direct lexical entry takes part, and the number of resulting noun group is plural.

4.4 Strategy for Grammar Development

While the approach outlined above works quite well for most of the sentences, there are some problem cases. The problems usually arises due to ambiguity in lexical category of words, which produces conflict in word grouping. Depending on what lexical category is actually present, it results in a different word group.

To handle the problem cases we need a flexible approach which can make use of special rules and possibly, case by case analysis. We follow the approach in which grammar rules are arranged in layers, each layer containing rules and forming an exception to the higher layer. This approach, called utsarga-apavaada (default-exception), is advocated by Patanjali (Kielhorn, 1880) for writing a practical and comprehensive grammar.

Let us first look at some of the conflicts and special rules (for some of the conflicts). Later we will look at the approach.

1. Conflict between taa form of verbs and the corresponding nouns: 'sotaa' is a verb (meaning 'to sleep') as well as a noun (meaning underground water). Similarly 'khaataa' verb and 'khaataa' noun (ledger or account).

2. Conflict between yaa form of verb and the corresponding nouns: 'diyaa' in Hindi can occur as a verb (to give) or as a noun (lamp).

3. Conflict between naa form of verb and nouns: 'sonaa' can occur as a verb (to sleep), a verbal noun (the act of sleeping), karma of verb, or as a noun unrelated to sleep (gold). The first three senses are related and the conflicts are systematic. Similar conflicts are likely to occur in many other cases. The last conflict for the sense (gold) is 'random' however. So also are the conflicts for cases (1) and (2) above. It should be possible to deal with systematic conflicts by general rules discovered by linguists; for the random conflicts, however, no rules are expected to work and a case by case treatment is necessary.

4. Conflict of verbal root with noun: 'samajha' can occur as a verb (to understand) as well as a noun (the result of understanding).

5. Conflicts of parsargs with nouns, verbs, relational words, etc. For example, se, kii, ke liye, and para normally occur as parsargs, but sometimes they act as other category of words.

The above kind of conflicts can potentially be taken care of by the utsarga-apavaada approach in which there are multiple layers of rules. Normal rules are in layer 1. The problem cases, like the ones outlined above, would be declared as exceptions and there would be rules in layer 2 for them. Here is a sketch of the layers for Hindi:

Layer 1 (General conflict resolver):

1. If word is followed by pure parsarg $\rightarrow$ noun. (where '$\rightarrow$' shows 'word resolves to').
2. If word is followed by verb sequence at the end of the sentence $\rightarrow$ final verb goup (most likely).

Layer 2 (Specific exceptions): In case of tam taa or yaa:

1. If followed by 'huaa' $\rightarrow$ verb (most probably). (Exception to be handled at layer 3 for khaataa as mentioned above.)
2. If reduplication (e.g., khaate khaate) $\rightarrow$ verb.

3. If immediate previous word is kaa → noun (most likely). (Exception to be handled at layer 3 for sotaa as mentioned above.)

Note that exceptions that lead to layer 3 are identified, but suitable rules at that layer have not been worked out. That task remains for the future. There is a need to take up systematic studies on this and related issues.

4.5 Semantics in Stages

Another significant aspect of our approach is that we do not try to get the full semantics immediately, rather it is extracted in stages depending on when it is most appropriate to do so. For example, in verb grouping only TAM label is created, detailed time and modality information is not extracted. In fact, examples suggest that such information is rather complex and requires further processing and context etc. A particular TAM like "taa_hei" (written as "taa hei", "tii hei" depending on gnp) captures various shades of meanings as shown below

1. raama chaaya pitaa hei.
   Ram drinks tea.
   (or Ram has no objection to drinking tea)

2. suurya puurva me nikaltaa hei.
   The sun rises in the east.
   (refers to periodicity in a natural law like sunrise)

3. prithvii suurya ke caaroM tarafa ghuumtii hei.
   Earth revolves around the sun.
   (refers to continuity in a natural law like the earth moving round the sun)

4. usake baada raama ghara jaataa hei aura use ghara banda miltaa hei.
   After that Ram goes home and he finds the house locked.
   (in the context of story telling, it indicates one instance of the activity of going and finding)

5. yadi vaha aataa hei to tuma jaanaa.
   If he comes, you go.
   (in a conditional sentence it signifies a one time activity)

6. raama ko teirnaa aataa hei.
   Ram knows swimming.
   (indicates capability)

7. bicchuu danka maarataa hei.
   Scorpion stings.
   (shows the scorpion's inherent nature which is situation dependent)

8. raama isa samaya duudha piitaa hei.
   At this time, Ram drinks milk.
   (shows habit)

9. raama in dinoM[2] duudha piitaa hei.
   These days Ram drinks milk.
   (signifies an activity which is localised in recent time)

Thus, a single TAM label 'taa_hei' produces all the above different meanings.

In machine translation for Indian languages, it would be sufficient to keep TAM labels without doing any semantic analysis provided the mapping of TAM label from source language to TAM label in the target language covers the senses spanned. Preliminary analysis among Hindi, Telugu, Kannada and Tamil indicates that it is indeed so.

4.6 Some Open Problems

We list below some problems relating to conflicts between parsargs and other lexical categories, for which the solutions in terms of suitable rules remain to be worked out.

1. 'se' can be used as a comparision word. For example,

   ```
   phoola  se   komala raama ne dhanusha toda diya.
   flower like  soft   Ram  -ne bow      broke
   (Ram who was as soft as a flower broke the bow.)
   ```

   If 'se' is taken as a parsarg here, the sentence would indicate flower as an instrument:

   ```
   Soft Ram broke the bow with a flower
   ```

   Perhaps when 'se' is used as a comparision word, there are only a limited number of words that can follow it. Linguists can make a catalogue of such words.

2. 'kii' can occur as a verb. For example,

[2]Note that 'dinoM' is the oblique form of 'dina' (day), yet it is not followed by a postposition. Such time phrases are exceptions to the rule described in Sec. 4.3. (These can be handled by declaring them as exceptions.)

```
raama ne mohana se baata kii.
Ram  -ne Mohan -se talk -ed.
(Ram talked to Mohan.)
```

It might appear that one way to distinguish between the two usages is that the verb occurs at the end of the sentence unlike the parsarg. When 'kii' occurs at the end, it can only be a verb. But would the following sentence be considered an incorrect sentence:

```
raama ne  kitaaba padhi mohana kii.
Ram  -ne  book    read  Mohan -kii
(Ram read the book of Mohan.)
```

3. 'para' can occur as a noun. For example,

```
chitrakaara para    se  chitra  banaataa hei .
artist      feather -se picture make -s
(Artist makes a picture with a feather.)
```

Note that 'par se' together can act as parsarg. For example,

```
vayuyaana  sira  para se gujaraa.
aeroplane  head -par -se passed
(The aeroplane passed overhead.)
```

4. 'ke liye' can occur as a parsarg 'ke' followed by the verb 'liye'. For example,

```
meine ye   aama   dasa rupaye  ke  liye.
I   these mangoes ten  rupees -ke bought
(I bought these magoes for ten rupees.)
```

We have mentioned the conflict in 'taa' forms of sotaa and khaataa verbs which also correspond to nouns. The LWG must make the right choice. Normally, depending on whether a postposition marker or an auxiliary verb follows 'khaataa' (or 'sotaa', for that matter) it can be decided whether it is used in the verb sense or the noun sense. However, there are some problem cases. In the sentence below, for example, a verb 'hua' follows 'khaataa' which occurs in a noun sense.

```
jaba  se   khaataa  hua  hei, neeMda nahiM aatii.
Since when account  being,    sleep  not   come
(Ever since I have an account, I cannot sleep.)
```

4.7 Conclusions

In this chapter, we have argued for the need for local word groups in Indian languages to handle those units in which word order is important. These word groups seem to be at the right level because firstly, they allow us to deal with issues of kaaraka role assignment (in core parsing) uniformly (hopefully) for all Indian languages. Secondly, they make exactly the right kind of information available to the core parser. We have also outlined the utsargs-apavaada approach to grammar design that is particularly suited for grammar development.

Further Reading

The concept of local word grouping for Indian languages discussed here is based on Bharati et al. (1991), and Bhanumati (1989). For utsarga-apvada approach of Patanjali, see Kielhorn (1880).

Exercises

4.1 Does your mother tongue have local word groups for verbs? If yes, choose a verb and identify all possible sequences of auxiliary verbs that can follow it.

4.2 In the verb and its auxiliaries you have identified in Exercise 4.1, what are the agreement rules.

4.3 Write a local verb grouping program yourself or use an existing program (e.g., local word grouper package or finite state machine package such as LEX) that uses the rules obtained in Exercise 4.2 to form verb groups in an input sentence.

4.4 Try to make a catalog of exceptions where the post position marker can be used as a verb or noun etc. Can you suggest a heuristic to recognize when it is a marker and when not?

4.5 Work on one of the open problems in Sec. 4.6. Can you come up with some heuristics for resolving the conflict.

Chapter 5

Paninian Grammar

[1]

5.1 Introduction

How is it that natural language is used by speakers to convey information to the hearers? How is it that on hearing an utterance, the hearers are able to get the intended information? These are the questions that have intrigued Paninians. The goal of the Paninian approach is to construct a theory of human natural language communication that answers these questions. Grammar, a part of such a theory of communication, is a system of rules that establishes a relation between what the speaker decides to say and his utterance, and similarly, what the hearer hears and the meaning he extracts.[2]

The main problem the Paninian approach addresses is how to extract karaka relations (which are syntactico-semantic relations) from a sentence. As it is inspired by an inflectionally rich language, it emphasizes the roles of case endings and markers such as post-positions (or prepositions). Positions or word order is brought into consideration only when necessary.

A majority of human languages including Indian and other languages have relatively free word order. In free word order languages, order of words contains only secondary information such as emphasis etc. Primary information relating to 'gross' meaning (e.g., one that includes semantic

[1] K.V. Ramakrishnamacharyulu of Rashtriya Sanskrit Vidyapeetha, Tirupati is the co-author of this chapter.

[2] There is a difference between the goals of Paninian approach on the one hand and (Chomskyan) generative enterprise on the other. The latter is interested in identifying the innate universal grammar in the mind of every human child by which he or she so effortlessly and without explicit tutoring acquires language that he or she is exposed to.

relationships) is contained elsewhere. In contrast to the languages, most existing computational grammars are based on context free grammars which are basically positional grammars. These are designed for languages in which position of a constituent in a sentence contains key information. It is important to develop a suitable computational grammar formalism for free word order languages for two reasons:

1. A suitably designed formalism will be more efficient because it will be able to make use of primary sources of information directly.

2. Such a formalism is also likely to be linguistically more elegant and satisfying. Since it will be able to relate to primary sources of information, the grammar is likely to be more economical and easier to write.

In this chapter, we describe such a formalism, called the Paninian framework, that has been successfully applied to Indian languages.

5.2 The Semantic Model

In this section, we describe the elements of the semantic model in the Paninian framework.

Every verbal root (dhaatu) denotes an action consisting of:

1. an activity (a vyaapaara), and

2. a result (a phala)

Result is the state which when reached, the action is complete. *Activity* consists of actions carried out by different participants or *karakas*[3] involved in the action.

An action is usually a complex which can be broken into sub-actions. For example, in the opening of a lock, a person inserts a key in the lock and turns it, the key on its part presses the levers and moves them, the latch in turn moves and the lock opens (Figure 5.1). Each of the sub-actions (e.g., inserting and turning a key, the key pressing and moving the levers, the latch moving and the lock opening) has its own semantic relations with associated objects.

Another important concept is that of vivaksha. *Vivaksha* refers to the speaker's viewpoint or attitude towards the activity. A sentence is not only a statement of an objective activity but also contains information regarding the speaker's viewpoint. (Vivaksha should not be confused with the

[3]Karaka pronounced as 'kaarak' is defined later.

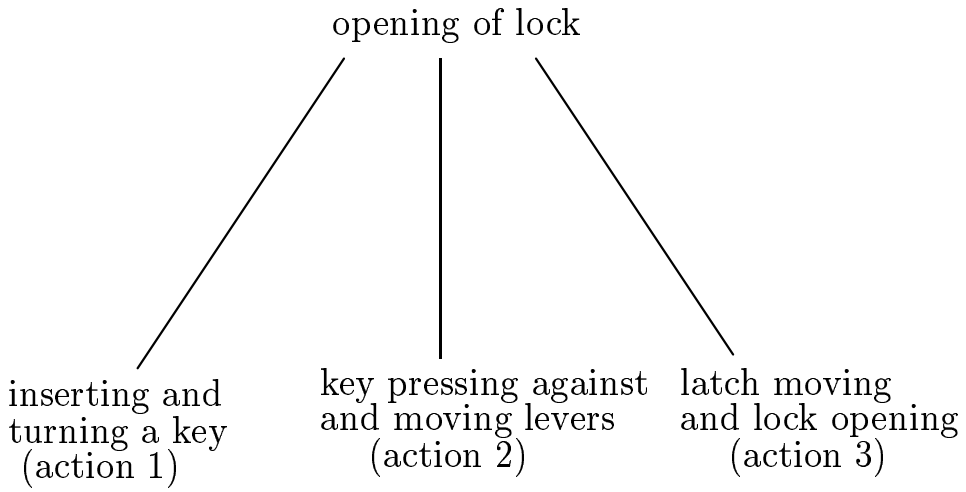


Figure 5.1: Structure of an action

speaker's intentions which come under pragmatics.) Usually, vivaksha affects the choice of verb form which, in turn, affects the choice of participants and their relation with or role in the action.

Out of all the participants in an action, there is one who is 'swatantra' or most independent, and is called *karta*[4] karaka. Thus, in the following sentences:

```
A.1 The boy opened the lock.
A.2 The key opened the lock.
A.3 The lock opened.
```

the karta is, respectively, the boy, the key, and the lock.

The Paninian theory chooses to talk in terms of karta instead of agent. It is not just a change in nomenclature, but as the above example shows, karta and agent are different in general.

According to Kaundbhatta, who further elaborates on the concept of swatantra, every verb in a sentence refers to a complex of activity. Frequently, however, the same verb can be used to refer to not only the main activity but also to sub-parts of the complex.[5] Karta of a verb in a sentence is one which is the 'aashraya' or locus of the activity. Thus, in A.1 the activity referred to is the act of opening of the lock by the boy by inserting a key and turning it, etc. (action 1 in Figure 5.1). Thus, the boy is the

[4]Karta is pronounced 'kartaa'.

[5]Sometimes the verb can be ambiguous, in which case it may refer to one of several actions. When used in a sentence, it may refer to any one of its possible actions.

karta of this activity. In A.2, the activity referred to is the pressing and moving of the levers by the key causing the lock to open, etc. (action 2 in Figure 5.1). The 'key' is the karta of this activity. In A.3, the activity is that of motion of the latch and opening of the lock (action 3 in Figure 5.1). Here, the karta is the lock. In A.1 the speaker decides to give importance to the role of the boy, whereas in A.2 the speaker wishes to emphasize the role of the key. ("It is *this* key which opened the lock!") In A.3 the speaker emphasizes the fact that the lock has been opened; which key opened it and who opened is unimportant as far as that utterance is concerned. As further evidence that the same verb 'open' refers to distinct actions (howsoever related they might be), note that in a language like Hindi there is a different word for 'open' in A.3 as compared to A.1 and A.2. See B.1, B.2 and B.3 below for Hindi sentences.

```
B.1 mohana  ne   taalaa   kholaa.
    Mohan  -ne   lock     opened
    (Mohan opened the lock.)

B.2 isa   caabii    ne    taalaa   kholaa.
    This  key       -ne    lock    opened
    (This key opened the lock)

B.3 taalaa  khula gayaa.
    lock    opened
    (The lock opened.)
```

The notion of karaka relations is central to the Paninian model. The karaka relations are syntactico-semantic (or semantico-syntactic) relations between the verbals and other related constituents in a sentence. They capture a certain level of semantics. But this is the level of semantics that is important syntactically and is reflected in the surface form of the sentence(s). Figure 5.2 shows the relationship.

In Figure 5.2, the surface level is the uttered or written sentence. The vibhakti level is the level at which there are local word groups based on case endings, preposition or postposition markers.[6] Vibhakti for nouns has already been defined earlier. A noun group is a unit containing a noun (or a pronoun, proper name, etc.), its vibhakti and possibly some adjectives.

Vibhakti for verbs can be defined similar to that for the nouns. A head verb may be followed by auxiliary verbs (which may remain as separate words or may combine with the head verb). Such information is collectively called vibhakti for the verb. The vibhakti for a verb gives information about tense, aspect and modality (TAM), and is, therefore, also called the TAM label. TAM labels are purely syntactic determined from the verb form and the auxiliaries.

[6]For positional languages such as English, it would also include position or word order information.

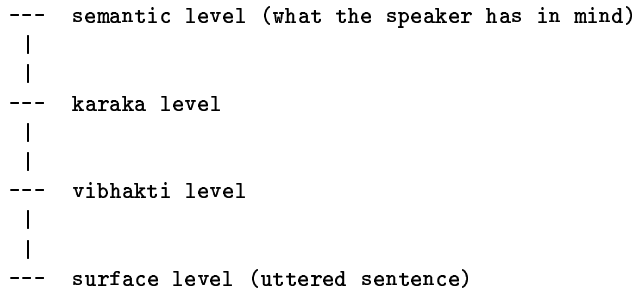


Figure 5.2: Levels in the Paninian model

The vibhakti level abstracts away from many minor (including orthographic and idiosyncratic) differences among languages. The topmost level relates to what the speaker has in his mind. This may be considered to be the ultimate meaning level. Between this level and vibhakti level is the karaka level. It includes karaka relations and a few additional relations such as tadarthya (or purpose). One can imagine several levels between the karaka and the ultimate level, each containing more semantic information. Thus, karaka level is one in a series of levels, but one which has relationship to semantics on the one hand and syntax on the other.

There are only around six different types of karaka relations. It is clearly not possible for them to capture the innumerable types of semantic relations among all possible actions or states (written as verbs) and all possible objects (written as nominals) in the world. What the karaka relations do specify with respect to a verb, however, are six or so relations of nominals (or verbals) that participate in the action specified by that particular verb. These are sufficient for providing a mapping from karaka relations to semantic relations. Thus, the karakas provide the maximum necessary information relative to a verb. As an example, the karta karaka may get mapped to agent for one verb, and experiencer for another, etc.

We have already seen the definition of karta karaka. Here, we informally discuss the remaining ones. We saw earlier that karta, the swatantra or the most independent participant in an action, is the ashraya (or the *locus*) of activity. In other words, the activity resides in or springs forth from the karta. Ashraya of the result is called karma, if it can be different from karta. A verb in which ashraya of activity and result can be different, is called *sakarmaka* (loosely, *transitive)*. For example, in the following sentence:

raama ne raavana ko tiira se maaraa.

```
Ram   ergative Ravan accus.  arrow instr. killed
(Ram killed Ravan with an arrow.)
```

Ram is the karta, and Ravan is karma.

Similarly karana karaka is the instrument. With the vyapara (or activity) of the karana, phala (or result) is immediately achieved. Thus, in the above example, with the vyapara of the arrow (i.e., motion of the arrow and striking Ravan) the result of the verb (kill) is immediately achieved.

Adhikarana karaka is the locus of karta or karma. It is what supports, in space or time, the karta or the karma.

Apadan [7] karaka is that participant in an action involving separation which remains stationary (or is the reference) when the separation takes place. Note that vivaksha or speaker viewpoint towards a separation event is what is important in determining what is stationary or the reference point. For example, in the following sentence:

```
peDa  se  baaga  meiM  patte  gire.
tree -se  garden -meiM leaves fell
(Leaves fell from the tree in the garden.)
```

the tree is the apadana, and garden is adhikarana.

Sampradana[8] is the beneficiary. For example, in the following sentence:

```
laDakaa laDakii  ko  phoola detaa hei.
boy      girl   acc. flower gives
(The boy gives a flower to the girl.)
```

the 'girl' is the sampradana.

There is another relation called tadarthya (roughly translated as purpose), which connects two verbs by the 'purpose' relation. For example:

```
aadami ne  naava banane   ke  liye lakaDii kaaTii.
man    -ne boat  to-make -ke -liye  wood   cut
(The man cut wood to make a boat.)
```

Even though taadaarthya is not a karaka relation, it is included here because of the important role it plays.

It should be re-emphasized that other than karta, we have had no detailed discussion on any other karaka. For more details on other karakas see work by Kaundabhatta (Shastri, 1973).

5.3 Free Word Order and Vibhakti

Indian languages have a relatively free word order. Many of the constituents of a simple sentence can occur in any order without affecting the gross

[7] Pronounced 'apaadaana'

[8] Pronounced 'sampradaana'

meaning of the sentence; what is affected is perhaps the emphasis, etc. For instance, noun groups in a sentence can come in any order without affecting the karaka relationship (or theta relationships) between the verb and the noun groups. Since position or order of occurrence of a noun group does not contain information about the karaka or theta roles in a simple sentence, a question can be asked regarding what carries this information. The answer seems to be that post-position markers after nouns (in North Indian languages) or surface case endings of nouns (in South Indian languages) or a mixture of the two at times, play a key role in specifying semantic relationships. We will collectively refer to the post position markers and surface case endings of nouns as vibhakti of nouns. Consider the following sentences of Hindi (Bharati et al. (1991a)).

```
C.1  raama mohana  ko  piiTataa  hei.
     Ram   Mohan  -ko  beat      is
     (Ram beats Mohan.)

C.2  mohana  ko  raama  piiTataa  hei.
     Mohan -ko   Ram    beat      is
     (Ram beats Mohan.)

C.3  mohana  raama  ko   piiTataa hei.
     Mohan   Ram   -ko   beat     is
     (Mohan beats Ram.)

C.4  raama ko  mohana piiTataa  hei.
     Ram  -ko  Mohan  beat      is
     (Mohan beats Ram.)
```

In C.1 and C.2, Mohan has the same vibhakti (i.e., parsarg or postposition 'ko') and semantic relation with beating. Even though the positions of 'raama' and 'mohana ko' are interchanged in C.2, it does not alter the respective semantic relations of Ram and Mohan with the verb. C.3 and C.4 show that semantic relation of Ram is interchanged with that of Mohan by interchanging their vibhakti. So the vibhaktis are crucial in determining the semantic roles. Relative position of the nominal seems to be not very important for determining semantic relations.

However, things are not always straightforward and the following need to be accounted for:

1. A different vibhakti can be used for the same semantic relation with a given verb in a different sentence. For example, in the Hindi sentences D.1 to D.4, although Ram has the same semantic relation with eat (namely, the agent of eat), a different vibhakti is used each time (ϕ, ne, ko, se which represent nominative, ergative, accusative, ablative, respectively).

```
D.1  raama phala   ko   khaataa  hei.
     Ram   fruit  -ko   eats     is
     (Ram eats the fruit.)

D.2  raama  ne  phala   khaayaa.
     Ram   -ne  fruit   ate
     (Ram ate the fruit.)

D.3  raama   ko   phala  khaanaa  paDaa.
     Ram    -ko   fruit  eat      had-to
     (Ram had to eat the fruit.)

D.4  raama se   phala  nahii   khaayaa  gayaa.
     Ram  -se   fruit  not     eat      could
     (Ram could not eat the fruit.)
```

2. Same vibhakti can be used with the same verb for two different semantic relations. In E.1 and E.2, the same vibhakti (that is, 'ne' parsarg) is used for Mohan and the key, whereas their semantic relation to 'open' are quite different. Similarly for 'se' in E.3 and E.5, (E.1, E.2 and E.4 are same as B.1, B.2, B.3, respectively).

```
E.1  mohana  ne   taalaa    kholaa.
     Mohan  -ne   lock   opened
     (Mohan opened the lock.)

E.2  isa   caabii    ne    taalaa  kholaa.
     this  key      -ne    lock    opened
     (This key opened the lock)

E.3  caabii   se    taalaa    khula     gayaa.
     key      by    lock      opened
     (The lock opened 'unexpectedly' by the key.)

E.4  taalaa   khula     gayaa.
     lock     opened
     (The lock opened.)

E.5  bacce      se    taalaa    khula     gayaa.
     child      by    lock      opened
     (The lock was opened unintentionally by the child)
     OR (The child was able to open the lock.)
```

5.4 Paninian Theory

Paninian grammar is particularly suited to free word order languages. It makes use of vibhakti information for mapping to semantic relations, and uses position information only secondarily. As the Indian languages have (relatively) free word order and vibhakti, they are eminently suited to be described by Paninian grammar. The Paninian framework was originally designed more than two millennia ago for writing a grammar of Sanskrit; it has been adapted by us to deal with modern Indian languages.

5.4.1 Karaka Relations

Example sentences in Hindi from the previous section (in C, D and E) indicate that there is no straightforward mapping from vibhakti to semantic relation between noun groups and verbs. The key to arriving at an answer is to identify appropriate relations, i.e., karaka relations.

The most important insight regarding the ggkaraka-vibhakti mapping is that it depends on the verb and its tense aspect modality (TAM) label. The mapping is represented by two structures: default karaka chart and karaka chart transformation. The default karaka chart for a verb or a class of verbs gives the mapping for the TAM label known as basic. It specifies the vibhakti permitted for the applicable karaka relations[9] for the nouns etc. when the verb has the basic TAM label. The basic TAM label chosen for Hindi is 'taa_hei' and roughly corresponds to present indefinite tense. Any other TAM label could have been chosen as basic without any problem. The TAM labels are purely syntactic in nature and can be determined by looking at the verb form and the associated auxiliary verbs, etc. For other TAM labels, there are karaka chart transformation rules. Thus, for a given verb with some TAM label in a sentence, appropriate karaka-vibhakti mapping can be obtained using its default karaka chart and the transformation rule depending on its TAM label.

The default karaka chart for three of the karakas is given in Figure 5.3. This explains the vibhaktis in sentences C.1 to C.4. As Ram is the agent in C.1 and C.2, and Mohan in C.3 and C.4, and the agent is the most independent for the action (beat), it is expressed by means of the karta karaka; and the remaining nominal by karma karaka. For the karta karaka, ϕ vibhakti is used with taa_hei TAM label in C.1 to C.4 as explained in Figure 5.3.

Figure 5.4 gives some transformation rules for the default mapping for Hindi. It explains the vibhakti in sentences D.1 to D.4 (assuming that Ram is the karta and phala (fruit) is the karma). As explained by Figure 5.4,

[9] What karaka relations are permissible for a verb, obviously depends on the particular verb. Not all verbs will take all possible karaka relations.

Karaka	*Vibhakti*	*Presence*
Karta	ϕ	mandatory
Karma	ko or ϕ	mandatory
Karana	se or dvaaraa	optional

Figure 5.3: Default karaka chart

karta takes 'ne' vibhakti in D.2 because of TAM label 'yaa' (in the main verb group khaayaa), 'ko' in D.3 because of TAM label naa_paDaa (in khaanaa paDaa), and 'se' in D.4 because of TAM label yaa_gayaa (in khaayaa gayaa). (See Appendix for how Paninian Grammar handles similar phenomena in Sanskrit.)

TAM label	*Transformed vibhakti for karta*
yaa	ne
naa_paDaa	ko
yaa_gayaa	se or dvaaraa (and karta is optional)

Figure 5.4: Transformation rules

The default mapping and transformation rules also explain E.1 to E.5. Note that by our definition of swatantra or independent, Mohan is the karta in E.1, caabii (or key) in E.2, and taalaa (or lock) in E.3, E.4 and E.5. After this the vibhaktis of karta in E.1 to E.5 are as explained in Figure 5.4.[10]

In general, the transformations affect not only the vibhakti of karta but also that of other karakas. They also 'delete' karaka roles at times, that is, the 'deleted' karaka roles do not occur in the sentence.

It is important to re-emphasize that the transformation depends on TAM label which is purely syntactic, and not on tense, aspect and modality which are semantic. The TAM label can be determined for Hindi and other Indian languages by syntactic forms of the verb and its auxiliaries without the need to refer to any semantic aspects. The specification for obtaining TAM labels is given by a finite state machine as described in Chap. 4.

The Paninian theory outlined above (i.e., karaka charts and karaka chart transformations) can be used to generate (or analyze) the sentences given above as we have seen. However, there are additional constraints that would disallow the following sentences to be generated, for example[11]:

[10]The choice of nominals to be used in the sentence is made by the speaker using vivaksha while mapping from semantic level to the karaka level.

[11]A '*' before a sentence indicates that it is not a good sentence.

```
F.1   *ladake  ne  raama  ne laDakii  ko  kitaaba dii.
       boy    -ne  Ram   -ne girl    -ko  book    gave
       (*The Ram the boy gave a book to the girl.)

F.2   *laDake  ne  laDakii  ko kitaaba  phoola  dii.
       boy    -ne  girl    -ko book     flower gave
       (*The boy gave a book a flower to the girl.)
```

The constraints are given below:

1. Each mandatory karaka in the karaka chart for each verb group, is expressed *exactly once*. (In other words, a given mandatory karaka generates only one noun group with the specified vibhakti in its karaka chart unlike in F.1.)
2. Each optional karaka in the karaka chart for each verb group, is expressed *at most once*.
3. Each source word group satisfies some karaka relation with some verb (or some other relation). In other words, there should no unconnected source word group in a sentence, otherwise, the sentence becomes bad as in F.2.

Karaka charts are based on the idea of aakaankshaa and yogyataa mentioned earlier in Chap. 2. A karaka chart for a verb expresses its aakaankshaas or demands, and specifies the vibhaktis that must be used (i.e., yogyataa) with word groups that satisfy the demands.

The same ideas can be used to handle noun-adjectives, noun-noun relations, verb-verb relations etc.

To show that the above, though neat, is not just an ad hoc mechanism that explains the isolated phenomena of semantic roles mapping to vibhaktis, we discuss two other phenomena: active-passive and control.

5.5 Active Passive

No separate theory is needed to explain active-passives. Active and passive turn out to be special cases of certain TAM labels, namely, those used to mark active and passive. Again consider the following example in Hindi.

```
F.3 raama   mohana ko piiTataa  hei. (active)
    Ram     Mohan -ko beats    pres.
    (Ram beats Mohan.)

F.4 raama dvaaraa mohana  ko  piiTaa gayaa. (passive)
    Ram   by      Mohan  -ko  beaten  was
    (Mohan was beaten by Ram.)
```

Verb in F.4 has TAM label as yaa_gayaa. Consequently, the vibhakti 'dvaaraa' for karta (Ram) follows from the transformation already given earlier in Figure 5.4.

5.6 Control

A major support for the theory comes from complex sentences, that is, sentences containing more than one verb group. We first introduce the problem and then describe how the theory provides an answer. Consider the following sentences in Hindi:

```
G.1  raama phala  khaakara     mohana ko  bulaataa hei .
     Ram   fruit  having-eaten Mohan -ko  calls    is
     (Having eaten  fruit, Ram calls Mohan.)

G.2  raama ne  phala    kaaTakara   khaayaa  .
     Ram   ne  fruit    having-cut  ate
     (Ram ate having cut the fruit.)

G.3  phala   kaaTane  ke liye usane caakuu liyaa .
     fruit   to-cut  -ke-liye he-ne knife took
     (To cut fruit, he took a knife.)
```

In G.1, Ram is the karta of both the verbs: khaa (eat) and bulaa (call). However, it occurs only once. The problem is to identify which verb will control its vibhakti. In G.2, karta Ram and the karma phala (fruit) both are shared by the two verbs kaaTa (cut) and khaa (eat). In G.3, the karta 'usa' (he) is shared between the two verbs, and 'caakuu' (knife) the karma karaka of 'lii' (take) is the karana (instrumental) karaka of 'kaaTa' (cut).

The observation that the matrix or main verb rather than the intermediate verb controls the vibhakti of the shared nominal is true in the above sentences. The theory we will outline to elaborate on this theme will have two parts. The first part gives the karaka to vibhakti mapping as usual, the second part identifies shared karakas.

5.6.1 Karaka to Vibhakti Mapping

The intermediate verbs have their TAM labels just like other verbs. For example, kara is the TAM label of khaa (eat) in G.1 and G.2, and naa is the TAM label of kaaTa (cut) in G.3. As usual, these TAM labels have transformation rules that operate and modify the default karaka chart. In particular, the suggested transformation rules for the two labels are given in Figure 5.5. The transformation rule with kara in Figure 5.5 says that

TAM label	*Transformation*
kara	Karta must not be present. Karma is optional.
naa	Karta and karma are optional.
taa_huaa	Karta and karma are optional.

Figure 5.5: More transformation rules (for complex sentences)

karta of the verb with TAM label kara must not be present in the sentence and the karma is optionally present.

By these rules, the intermediate verb khaa (eat) in G.1 and kaaTa (cut) in G.2 do not have (independent) karta karaka present in the sentence. Ram is the karta of the main verb. phala (fruit) is the karma of khaa in G.1 but not of kaaTa in G.2. In the latter, phala is the karma of the main verb. All these are accommodated by the above transformation rule for 'kara'.[12] The modifier-modified tree structures produced are shown in Figure 5.6 (ignore dotted lines).

5.6.2 Karaka Sharing

Now we give rules for obtaining the shared karakas.

Karta of the intermediate verb khaa in G.1 and G.2 can be obtained by the sharing rule S1'.

Rule S1': Karta of a verb with TAM label 'kara' is the same as the karta of the main verb.

The sharing rule(s) are applied after the tentative karaka assignment is over using karaka to vibhakti mapping.

For karma of intermediate verb with TAM label kara, we have the sharing rule S2'.

Rule S2': If an intermediate verb with the TAM label 'kara' takes a karma (as specified in its default karaka chart) while none has been obtained using karaka vibhakti mapping, then it shares its karma with the karta or the karma of the main verb.

[12] Bhartrahari formulated the original rule that if a nominal has a karaka relationship with two verbs, then the main verb will control its vibhakti.

Figure 5.6: Modifier-modified relations for sentences G.1, G.2 and G.3, respectively. (Shared karakas shown by dotted lines.)

With the above rules, the shared karakas for G.1, G.2 and G.3 are shown in Figure 5.6 by dotted lines. In the following example,

```
H.1 khariidakara   raama  ne phala  khaayaa.
    having-bought  Ram   -ne fruit  ate
    (Ram ate fruit having bought it.)
```

karta and karma of khariida (buy) are the same as the karta and karma, respectively, of khaa (eat). Figure 5.7 shows the relationships pictorially.

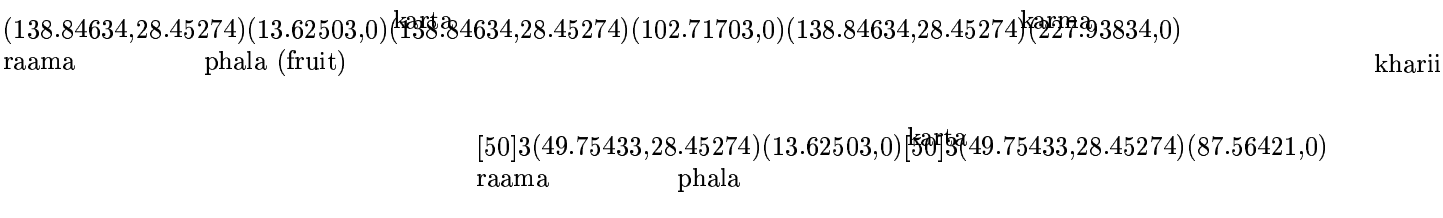


Figure 5.7: Parse structure for sentence H.1

Similarly, the sharing rule for taa_huaa can be stated as S3'.

Rule S3': An intermediate verb with TAM label 'taa_huaa' shares its karta with the karta of the main verb.

As example, consider:

```
H.2 shikaarii ne  bhaagate  hue  shera  ko dekhaa .
    hunter   -ne  while-running  lion  -ko saw
    (The hunter saw a lion while running.)
```

The above is ambiguous between whether the hunter was running or the lion was running. Now, karaka vibhakti mapping yields the following relation:

```
dekha (see)
  karta: shikaarii  (hunter)
  karma: shera  (lion).
bhaaga (run)
  no karaka relation expressed explicitly
```

The above are shown pictorially in Figure 5.8 (ignore dotted lines for now).

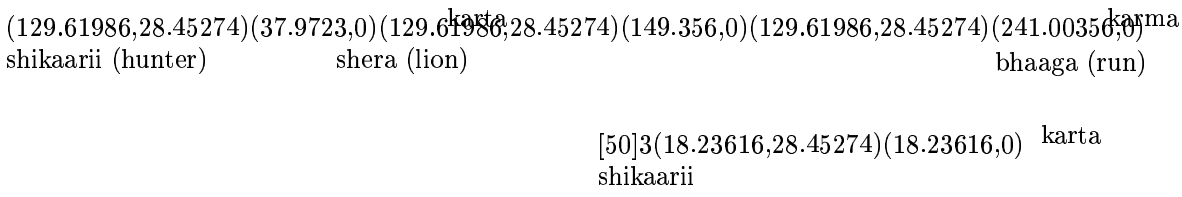


Figure 5.8: Parse structure for sentence H.2

By Rule S3', we can identify the shared karakas of run. Here, the karta of run would be the hunter. This is shown by dotted lines in Figure 5.8.

There is another parse for the sentence corresponding to verb-noun modification with verbal 'bhaagate hue' (running) modifying noun 'shera' (lion). Karaka relations of dekha (see) as obtained by karaka vibhakti mapping are the same. But since bhaaga modifies shera , the karta of bhaaga is shera by rule S4 given below. The parse structure is shown in Figure 5.9.

Rule S4: If a verb with TAM label taa_huaa modifies a noun, then that noun is its karta.

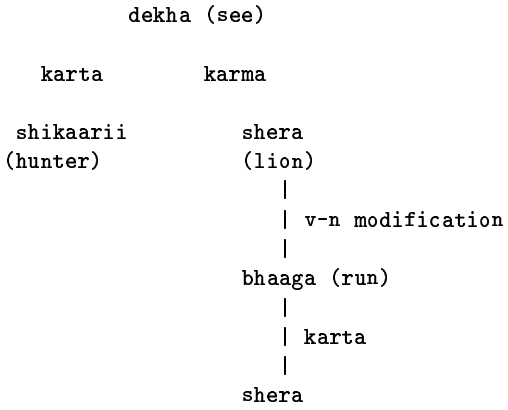


Figure 5.9: Another parse structure for sentence H.2

The part of parse structure shown by solid lines is obtained by karaka vibhakti mapping. The part shown in dotted lines (the karta of bhaaga) is obtained by Rule S4.

When there are more than one intermediate verb groups in a sentence, there is a need to generalize the rules given earlier. To obtain the shared karaka of an intermediate verb, instead of getting it from the main verb we must obtain it from the verb (action) modified by the intermediate verb. The modified verb could be the main verb or another intermediate verb. The new generalized Rules are S1, S2 and S3.

Rule S1: Karta of intermediate verb with TAM label 'kara' is the same as the karta of the verb modified by the intermediate verb.

Rule S2: If an intermediate verb with the TAM label 'kara' takes a karma (as specified in its default karaka chart) while none has been obtained using karaka vibhakti mapping, then it shares its karma with the karta or the karma of the verb modified by the intermediate verb.

Rule S3: An intermediate verb with TAM label 'taa_huaa' shares its karta with the karta of the verb modified by the intermediate verb.

As an illustration, consider the following sentence from Hindi:

```
H.3 raama ne haatha se  chiilkara    kelaa   khaate hue
    Ram -ne hands -se having-peeled  banana  eating

    bandara ko dekhaa.
    monkey -ko saw.
    (Ram saw a monkey eating a banana having peeled it
    with his hands.)
```

The above is ambiguous as to who peeled and ate the banana. Karaka to vibhakti mapping produces the following relations:

```
dekha (see)
  karta: raama
  karma: bandara (monkey)
chiila (peel)
  karana: haatha  (hand)
khaa (eat)
  karma: kelaa  (banana)
```

The above relations are shown in solid lines in Figure 5.10. Now, on applying the karaka sharing rule we get the following four alternative parses:

1. Assume chiila (peel) modifies khaa (eat).[13] By Rule S1, karta of chiila is the same as the karta of khaa. Similarly, karma of chiila is the same

[13] 'chiila' has the TAM label 'kara', which means that it precedes the action it modifies. Thus, 'chiila' modifies 'khaa' means that the action 'chiila' (peel) was performed before performing the action 'khaa' (eat).

as the karma of khaa, by Rule S2. Now, there are two alternatives for what khaa modifies.

(a) khaa (eat) modifies the verb dekha (see). By Rule S3, karta of khaa is the same as the karta of dekha (see). Therefore:

```
khaa (eat)
  karta: raama
chiila (peel)
  karta: raama
  karma: kelaa (banana)
```

Figure 5.10 (a) shows the parse structure.

(b) khaa (eat) modifies the noun bandara (monkey). By Rule S4, bandara is the karta of khaa (eat).

```
khaa  (eat)
  karta: bandara (monkey)
chiila  (peel)
  karta: bandara (monkey)
  karma: kelaa (banana)
```

Figure 5.10 (b) shows the parse structure.

2. Assume chiila (peel) modifies dekha (see). By Rule S1, karta of chiila is same as karta of dekha, namely raama. By Rule S2, the karma of chiila (peel) is the same as karma of dekha (see), namely, bandara (monkey). This is semantically anomalous as monkeys cannot be peeled. If world knowledge shows this to be an anomaly, further processing would be blocked. However, if such world knowledge is not available to the system, it would continue processing. There would be two alternatives. Karta of khaa can have two possible choices like in 1(a) and 1(b).

(c) khaa (eat) modifies dekha (see). Therefore, karta of khaa is the same as the karta of dekha. (See Figure 5.10(c).)

```
khaa (eat)
   karta: raama
chiila (peel)
   karta: raama
   karma: bandara
```

The kartas happen to be the same as in 1(a).

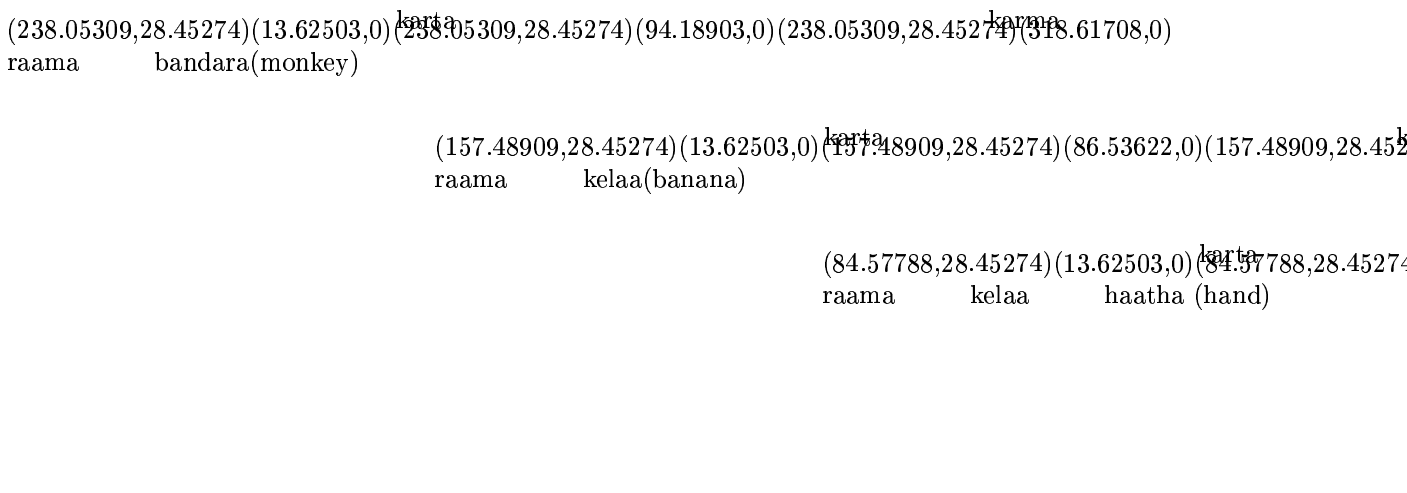
(d) khaa (eat) modifies bandara (monkey), which by rule S4 makes it the karta of khaa. (See Figure 5.10(d).)

```
khaa  (eat)
   karta: bandara (monkey)
```

```
chiila  (peel)
   karta: raama
   karma: bandara
```

5.7 Summary

In this chapter, we have discussed the linguistic aspect of Paninian framework as applied to modern Indian languages.



(a)

(b)

Figure 5.10: Alternative parses (a) and (b) for the sentence H.3

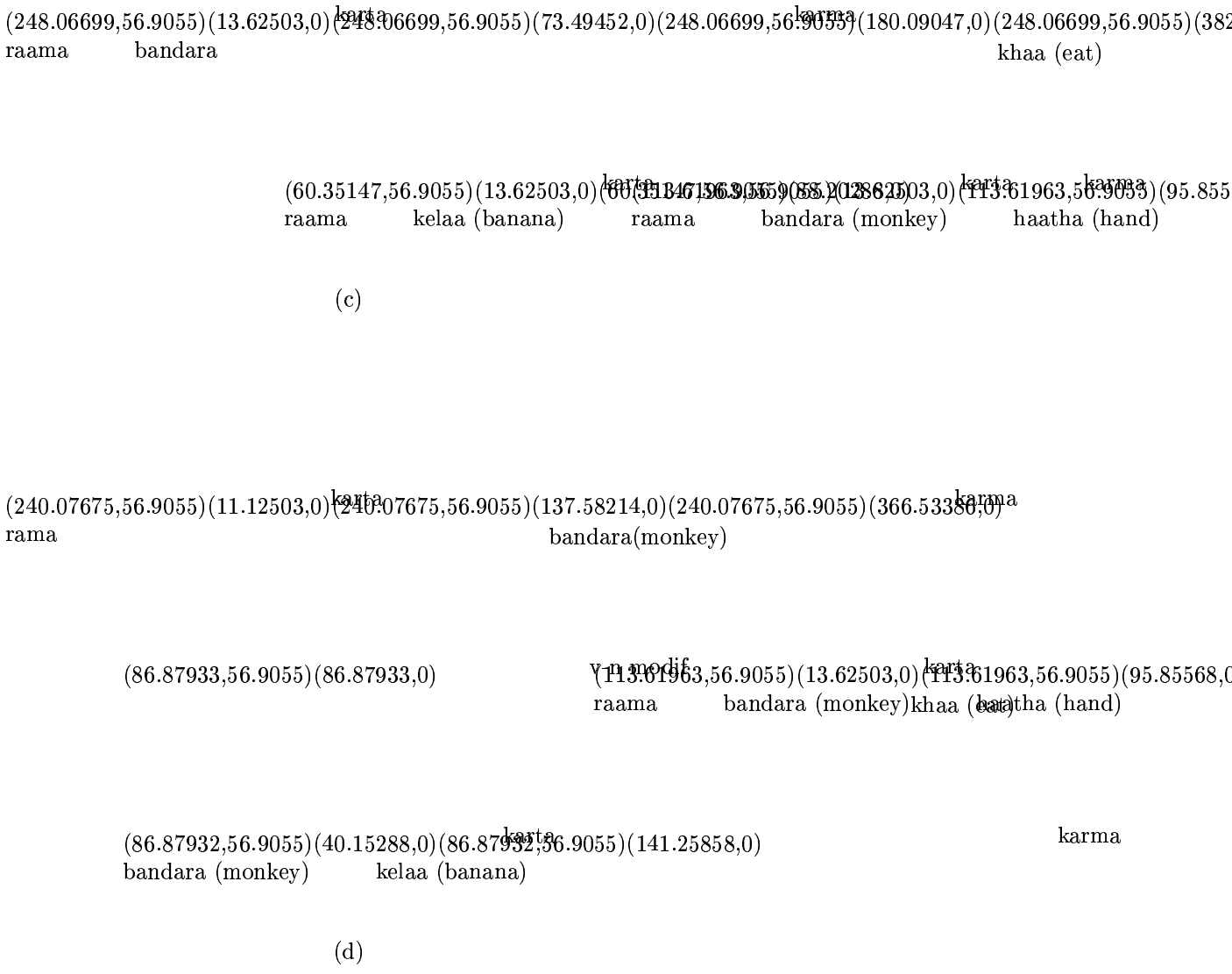


Figure 5.10: Alternative parses (c) and (d) for the sentence H.3

The original framework was applied to Sanskrit; here the concepts and spirit of the framework have been used in explaining various language phenomena.

Further Reading

Ideas presented in this chapter have evolved over the years, earlier presented in Bhanumati (1989), Bharati et al. (1990), (1990b), (1991a), (1993a), etc.

See Appendix A for a discussion on how Paninian Grammar is applied to Sanskrit. References to traditional Paninian grammarians including Panini and Bhartrahari are also given there. Iyer() discusses Bhartrahari's work on means.

Exercises

5.1 In the Paninian grammar, why do you need karaka chart transformation? Illustrate with example sentences from, say, Hindi.

5.2 Show karaka assignments for the following sentences with and without shared karakas. Show the rules by which shared kaarakas are identified. (In case of ambiguity show more than one assignment)

```
(a) raama khaanaa khaakara    paanii piitaa huaa ghara gayaa.
    Ram   food    having-eaten water drinking    home  went
    (Having eaten food, Ram went home drinking water.)

(b) raama dauDataa huaa murgaa  khaakara     ghara gayaa.
    Ram   running       chicken havind-eaten home  went
    (Having eaten a chicken, Ram went home running. OR
    Having eaten a running chicken, Ram went home.)
```

5.3
(a) Show the modifier-modified tree with karaka relations, verb-verb relations, etc. for the sentence given below:

```
raama ne beiThakara caavala pakaane ke liya
Ram -ne after-sitting rice  to-cook

rasoiye ko patiilaa diyaa.
cook   -ko vessel   gave
(Ram gave a vessel to the cook for cooking rice
after sitting down.)
```

Show all parses for the sentence. A parse which cannot be ruled out on the basis of syntax should be shown. (English translation is shown for convenience. It might, however, not show all parses.)

(b) Show the karaka chart for 'pakaanaa' (cook), before and after transformation.
(c) What is the sharing rule for karta of 'pakaanaa'.

5.4
(a) Repeat Exercise 5.3(a) for the following sentence:

```
raama ne mohana se kahaa ki shyaama socataa hei ki ravii
Ram  -ne Mohan -se said that Shyam  thinks     that Ravi

khaanaa khaate hue aadamii kii tasviire banataa hei
food    eating     man     of  picture  makes
(Ram told Mohan that Shyam thinks that Ravi
makes a picture of the man while eating.)
```

(b) Show the karaka chart for 'kahaa' (said).
(c) 'ki' is a pre-position marker. What changes will you make in the parser so that constraints are set up properly. (Show briefly and precisely what rule(s) will get changed.)

5.5 What is the agreement rule between verb and noun in Hindi and how will you handle it in the Paninian framework.

5.6 Why do we separate gnp from TAM for verbs?

5.7
(a) Show karaka assignment for the following sentence:

```
raama ne seva  khaane  ke liya paDosii   se  caakuu
Ram      apple to-eat          neighbour -se knife

lekara       chilakaa utaaraa.
having-taken peel     removed
(Having taken a knife from the neighbour, Ram
peeled the apple to eat.)
```

(b) Also show the shared karakas.

5.8 State the noun-verb agreement rule in Hindi for:

(a) Simple sentences

(b) Complex sentences

(Hint: You will have to state it in terms of karakas and post-positional markers.)

5.9 For the sentences given below

(a) Show the karaka relations that can be obtained by vibhakti information, i.e., karaka charts.

(b) Show by dotted lines the relations that can be obtained by karaka sharing rules, verb-verb relation rule, etc.

(c) List the rules that you have used in (b) above.

```
1. usane bacce ko   pustaka paDhane  ke liye  kahaa.
   he-ne child acc. book    to-read -ke-liye  asked
   (He asked the child to read the book.)

2. usane beiThe hue bacce ko   pustaka paDhane  ke liye
   he-ne seated     child acc. book    to-read -ke-liye

   kahaa.
   asked
   (He asked the seated  child to read the book.)

3. usane beiThe hue bacce ko   pustaka paDhane  ke liye
   he-ne seated     child acc. book    to-read -ke-liye

   kahanaa caahaa.
   to-ask  wanted
   (He wanted to ask the seated child to read the book.)

4. usane beiThate hue bacce ko   pustaka paDhane  ke liye
   he-ne was sitting  child acc. book    to-read -ke-liye

   kahaa.
   asked
   (He asked the child who was in the process of sitting
    to read a book.)

5. usane bacce ko   pustaka paDhane  ke liye biThaayaa
   he-ne child acc. book    to-read -ke-liye cause to sit
   (He made the child sit to read the book.) %v2 }

6. use beiThakara     bacce  kaa  pustaka paDhanaa acchaa
   he  having-sat down child  gen. book    reading  good

   nahiiM  lagaa.
   not like
   (He did not like the child reading a book having sat
    down.)

7. use bacce kaa  beiThakara     pustaka paDhanaa acchaa
   he  cild  gen. having-satdown book    reading  good
```

```
nahiiM  lagaa.
not like
(He did not like the child reading a book having
 sat down.)
```

5.10 In case of multiple parses (obtained by your rules) in the above exercise, mark the preferred parse, if any. In each case indicate what information can be used to obtain the preferred parse mechanically.

5.11 Define karaka charts for the verbs in sentences in Exercise 2.1. Are you able to derive the structures for the sentences using karaka charts and karaka transformation rules?

5.12 Karaka charts described here specify for each karaka, the vibhakti required and whether mandatory or not. If we consider verbs like 'kaha' (say), their argument (karma karaka) is a sentence and not just a nominal. Moreover, the argument occurs to the right of the verb and not to the left.

Provide for specification of these informations in the karaka chart. Give the karaka chart for 'kaha' (say) in the extended form that includes these informations.

Chapter 6

Paninian Parser

6.1 Introduction

In this chapter, we discuss how a parser can be built using the Paninian framework. It turns out that the Paninian theory is extremely suitable from the computational viewpoint. This is so partly because the goal of the Paninian theory matches with the goal of NLP of extracting meaning from an utterance. The theory can be used in a natural manner for structuring a parser which is extremely efficient.

The first part of the parser must take care of morphology. For each word in the input sentence, a dictionary or a lexicon needs to be looked up, and associated grammatical information needs to be retrieved. The words have to be grouped together yielding nominals, verbals etc. Finally, the karaka relations among the elements have to be identified. This is shown in Figure 6.1 (repeated from Chap. 2).

The parser is based on information theoretic considerations where at each stage of processing, just the right amount of information is extracted. At the morphological analysis stage, information available from word forms is obtained. For example, information about gender number, person is obtained (wherever possible) from the nouns. On the other hand, for verbs in Hindi, gender number person (gnp) and part of TAM is obtained from the words. In case of Telugu verbs, the complete TAM label, besides gnp is obtained as well.

In the local word grouping stage, words combine into groups (noun groups and verb groups) based on local information. The groups are formed with minimal computational effort (finite state machine model) as only local information is used. On the other hand, local word grouping brings together just the right information (vibhakti for nouns, TAM labels for verbs) that is needed for the next stage of processing (that is, for karaka assignment). It

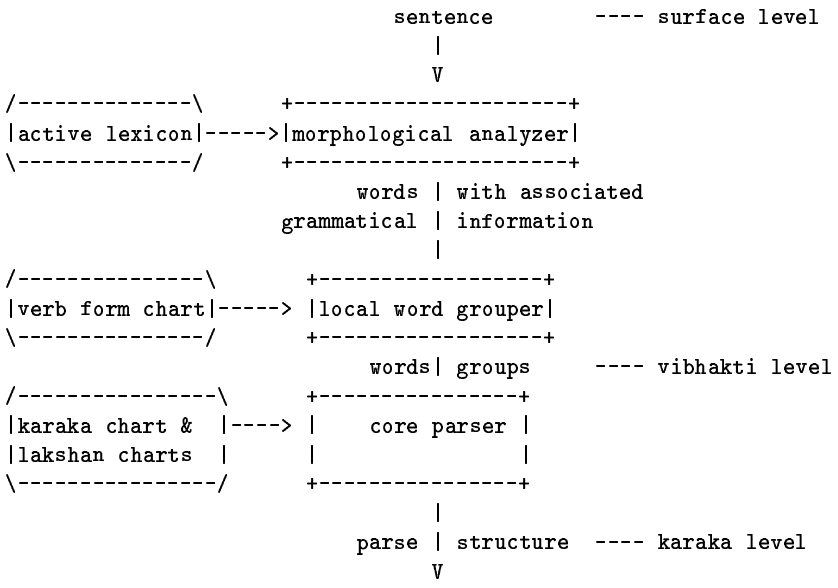


Figure 6.1: Structure of the Parser

does not attempt to distinguish all the fine shades of meaning, for example, temporal structure or modality. For one thing, such information cannot easily be determined at this stage of processing. Secondly, it is not needed for the next stage.

As mentioned earlier, the output of the local word grouper roughly corresponds to the vibhakti level. However, in those few cases where there is ambiguity in identifying the local word groups and the ambiguity cannot be resolved at that level, the decision is postponed. For example, in Hindi, a word that can be both a noun and an adjective, causes ambiguity in forming a local word group with its succeeding noun. The choice can only be made later during karaka assignment using karaka charts. As a result, in our information theoretic parser, such a choice is delayed to the point when it can be made. Similar is the case with a noun that is followed by a marker kaa , ke or kii and succeeded by a noun as in:

```
ladake kii  kitaaba
  boy 's  book
```

After the local word grouping stage, there is karaka assignment and lexical disambiguation stage. This is done next because the necessary information (vibhakti) for doing it is available. Other phenomena such as quantifiers and anaphora are not handled at this stage, because information for resolving them is not available.

This approach is consistent with the Indian grammatical analysis where meaning is extracted in several layers with increasing precision.

6.2 Core Parser

Given the local word groups in a sentence, the task of the core parser is two-fold:

1. To identify karaka relations among word groups,

2. To identify senses of words.

The first task requires knowledge of karaka-vibhakti mapping, optionality of karakas, and transformation rules. The second task requires lakshan charts for nouns and verbs, which will be discussed later.

A data structure called karaka chart stores information about karaka-vibhakti mapping and optionality of karakas for each of the verb groups in a sentence. Initially, the default karaka chart is loaded into a karaka chart for a given verb group in the sentence. Transformation is performed using the TAM label. There is a separate karaka chart for each verb group in the sentence being processed.

Karaka	*Vibhakti*	*Presence*
Karta	ϕ	mandatory
Karma	ko or ϕ	mandatory
Karana	se or dvaaraa	optional

Figure 6.2: Default karaka chart for 'khaa rakhaa'

An example default karaka chart for khaa (eat) is given in Figure 6.2 (repeated from chapter 5). It shows for each of the karakas its necessity (mandatory, desirable, or optional), and vibhakti.

For a given sentence after the word groups have been formed, karaka charts for the verb groups are created and each of the noun groups is tested against the karaka restrictions in each karaka chart (provided the noun group is to the left of the verb group whose karaka chart is being tested). When testing a noun group against a karaka restriction of a verb group, vibhakti information is checked, and if found satisfactory, the noun group becomes a candidate for the karaka of the verb group. This can be shown in the form of a constraint graph. Nodes of the graph are the word groups and there is an arc from a verb group to a noun group labelled by a karaka, if the noun group satisfies the karaka restriction in the karaka chart of the verb group. (There is an arc from one verb group to another, if the karaka chart of the former has a karaka restriction with lexical type as verb or sentence.) The verb groups are called demand groups as they make demands about their karakas, and the noun groups are called source groups because they satisfy demands. (A verb group can be a source group as well when it satisfies the demand of another verb group. This, however, does not affect its status as a demand group as well.)

As an example, consider a sentence containing the verb khaa (eat):

```
S.1  baccaa   haatha  se   kelaa    khaataa  hei.
     child hand -se  banana  eats
     (The child eats the banana with his hand.)
```

Its word groups are marked and khaa (eat) has the same karaka chart as in Figure 6.1. Its constraint graph is shown in Figure 6.3.

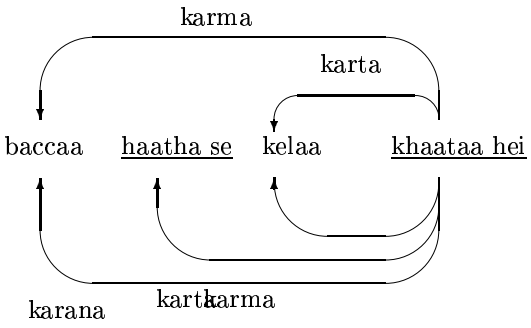


Figure 6.3: Constraint graph for sentence S.1

6.2.1 Constraints

A parse is a sub-graph of the constraint graph containing all the nodes of the constraint graph and satisfying the following conditions (corresponding to those in Sec. 5.4.):

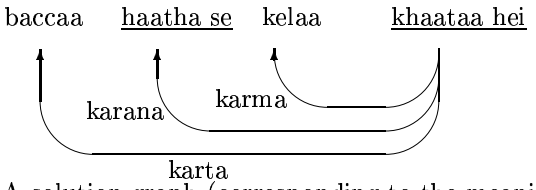
C1. For each of the mandatory karakas in a karaka chart for each demand group, there should be *exactly one* outgoing edge labelled by the karaka from the demand group.

C2. For each of the desirable or optional karakas in a karaka chart for each demand group, there should be *at most one* outgoing edge labelled by the karaka from the demand group.

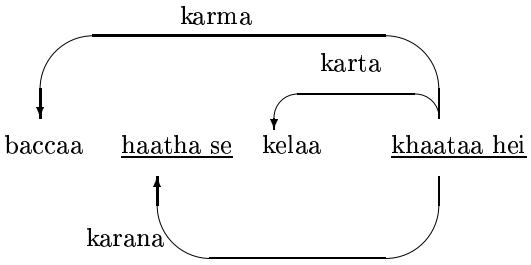
C3. There should be *exactly one* incoming arc into each source group.

If several sub-graphs of a constraint graph satisfy the above conditions, it means that there are multiple parses and the sentence is ambiguous. If no sub-graph satisfies the above constraints, the sentence does not have a parse, and is probably ill-formed. A parse is also called a *solution graph*.

Consider the example constraint graph in Figure 6.3. There are two outgoing arcs labelled by karta from 'khaataa hei'. Constraint C1 says that exactly one of the arcs must be selected to be in a solution graph. Similarly, for arcs labelled by karma karaka. However, constraint C3 states that arcs labelled by karta and karma must not be incident on the same noun. They must be incident differently on baccaa (boy) and kelaa (banana). The arc labelled by karana is optional and is to be retained only if necessary because of constraint C3. In this case, it turns out to be necessary because



(a) A solution graph (corresponding to the meaning: child eats banana)



(b) Another solution graph (corresponding to the meaning: banana eats child)

Figure 6.4: Solution graphs for sentence S.1

otherwise 'haatha se' would have no incoming arc, a violation of constraint C3. Figure 6.4 shows the two possible parses for the constraint graph.

There are similarities between dependency grammars and Paninian grammar (PG) because such constraint graphs are also produced by dependency grammars (Covington, 1990) (Kashket, 1986) (Hudson, 1994). Paninian grammar differs from them in two ways. First, the Paninian framework uses the linguistic insight regarding karaka relations to identify relations between constituents in a sentence. Second, the constraints are sufficiently restricted that they reduce to well known bipartite graph matching problems for which efficient solutions are known. These will be described later.

6.3 Constraint Parser Using Integer Programming

A parse can be obtained from the constraint graph using integer programming. A constraint graph is converted into an integer programming problem by introducing a variable x for an arc from node i to j labelled by karaka k in the constraint graph such that for every arc there is a variable. The variables take their values as 0 or 1. A parse is an assignment of 1 to those variables whose corresponding arcs are in the parse sub-graph, and 0 to those that are not. Equality and inequality constraints in integer programming problem can be obtained from the conditions (C1, C2, and C3) listed earlier, as follows respectively:

1. For each demand group i, for each of its mandatory karakas k, the following equalities must hold:

$$M_{i,k} : \sum_{j} x_{i,k,j} = 1$$

   Note that $M_{i,k}$ stands for the equation formed, given a demand word i and karaka k. Thus, there will be as many equations as combinations of i and k.

2. For each demand group i, for each of its optional or desirable karakas k, the following inequalities must hold:

$$O_{i,k} : \sum_{j} x_{i,k,j} \leq 1$$

3. For each of the source groups j, the following equalities must hold:

$$S_j : \sum_{i,k} x_{i,k,j} = 1$$

   Thus, there will be as many equations as there are source words.

The cost function to be minimized is given as the sum of all the variables.

We illustrate this by forming inequalities etc. for the example sentence S.1 (using Figure 6.3). To keep the notation compact, we introduce abbreviations for karakas given in Fig. 6.5. Word groups in sentence S.1 are referred to by (a, b, c, A):

```
baccaa haatha  se   kelaa    khaataa  hei.
child  hand   -se  banana    eats
  a     b            c        A
```

Abbreviation	*Karaka*
k1	Karta
k2	Karma
k3	Karana

Figure 6.5: Abbreviations for the karakas

Constraint C1 generates the following equalities (for mandatory karakas k1 and k2):

$$M_{A,k1} : x_{A,k1,a} + x_{A,k1,c} = 1$$
$$M_{A,k2} : x_{A,k2,a} + x_{A,k2,c} = 1$$

Constraint C2 generates:

$$O_{A,k3} : x_{A,k3,b} \leq 1$$

Finally, for C3 we have:

$$S_a : x_{A,k1,a} + x_{A,k2,a} = 1$$
$$S_b : x_{A,k3,b} = 1$$
$$S_c : x_{A,k1,c} + x_{A,k2,c} = 1$$

For ease of readability, we rename x's as follows:

$$x_{A,k1,a} = y_1$$
$$x_{A,k1,c} = y_2$$
$$x_{A,k2,a} = y_3$$
$$x_{A,k2,c} = y_4$$
$$x_{A,k3,b} = y_5$$

Now, we get

$$M_{A,k1} : y_1 + y_2 = 1$$
$$M_{A,k2} : y_3 + y_4 = 1$$
$$O_{A,k3} : y_5 \leq 1$$
$$S_a : y_1 + y_3 = 1$$
$$S_b : y_5 = 1$$

$$S_c : y_2 + y_4 = 1$$

The cost function to be minimised is:

$$Cost = y_1 + y_2 + y_3 + y_4 + y_5$$

If we solve the above, we get the following solutions:

$$y_1 = 1; y_4 = 1; y_5 = 1$$

(same as in Figure 6.4(a)), and

$$y_2 = 1; y_3 = 1; y_5 = 1$$

(same as in Figure 6.4(b)).

6.4 Constraint Parser Using Matching and Assignment

With the constraints (C1, C2 and C3) specified above, the parsing problem reduces to bipartite graph matching and assignment problems. These have efficient solutions even in the worst case.

First, let us consider a constraint graph which does not have any optional karaka. We can reduce the constraint graph to a bipartite graph. Finding a solution graph to the constraint graph reduces to finding a maximal matching in the bipartite graph.

6.4.1 Reduction to Bipartite Graph Matching

A *bipartite graph* G(V,U,E) is defined as:

U: set of nodes $u_1, u_2, ..., u_n$
V: set of nodes $v_1, v_2, ..., v_m$
E: edges between U and V, and $U \bigcap V = \phi$.

To perform the reduction of the problem of finding a solution graph to finding a matching, we first construct the bipartite graph. The bipartite graph is constructed in three stages:

1. For every source node s in the constraint graph, form a node s in U.
2. For every demand node d in the constraint graph and every mandatory karaka k in the karaka chart for d, form a node v in V. (Thus, for every pair (d,k) there is a node in V.)

3. For every edge (d,s) labelled by karaka k in the constraint graph, create an edge between node (d,k) in V to s in U

For example, for the constraint graph in Figure 6.3 (but assuming that the optional karana karaka is mandatory) we have the bipartite graph in Figure 6.6.

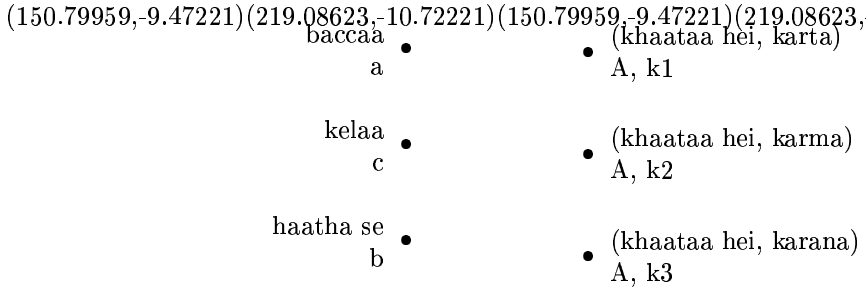
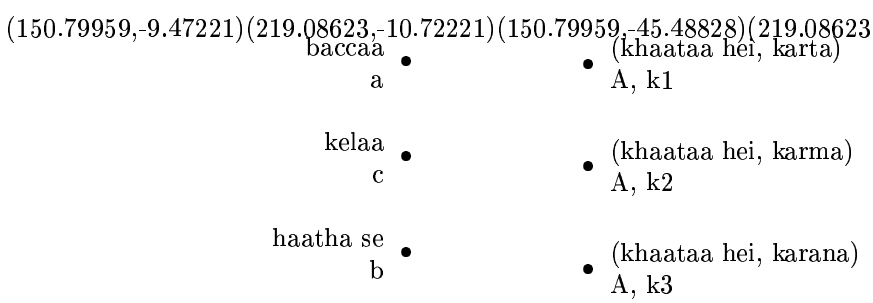


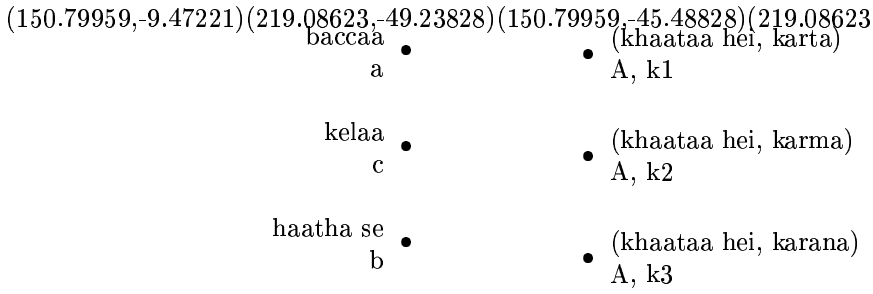
Figure 6.6: Bipartite graph for constraint graph in Figure 6.3

A *matching* M of a bipartite graph $G = (U, V, E)$ is a subset of edges with the property that no edges of M share the same node. The matching problem is to find a *maximal* matching of G, that is, a matching with the largest number of edges. A maximal matching is called a *complete* matching, if every node in U and V has an edge.

There are two maximal complete matchings of the graph in Figure 6.6. They are shown in Figure 6.7. They correspond to the parses shown in Figure 6.4.



(a) Solution corresponds to parse in Figure 6.4(a)



(b) Solution corresponds to parse in Figure 6.4(b)

Figure 6.7: Maximal (complete) matchings of bipartite graph of Figure 6.6.

Now we show that finding a maximal matching in a bipartite graph is the same as finding a parse in a constraint graph. Let M be a maximal matching of a bipartite graph G. If M is complete, it represents a parse. The proof is easy. For an edge between a node (d,k) in V and a node S in U, it represents the demand for karaka k of the demand node d being satisfied by the source node s. Since all the nodes in V have exactly one edge in M, the original constraint C1 is satisfied. Since all the nodes in U have exactly one edge in M, constraint C3 is satisfied. (And since there are no optional karakas, C2 is satisfied trivially.)

If M is not complete, that is, it does not have an edge on at least one node in U or V, then G does not have a parse. If a node in V does not have an edge, it is a violation of constraint C1, otherwise, it is a violation of constraint C3. Therefore, M is not a parse. Let the cardinality of M be m. Clearly $|U| > m$ or $|V| > m$. Since M is a maximal matching, there is no matching on G with larger number of edges. Therefore, any other maximal matching (which might choose a different set of edges) will cover exactly m nodes in U and m nodes in V. Consequently, some node in U or V would again not have an edge. Thus, any other maximal matching would also not give us a parse.

The converse has been left as an exercise.

To find a maximal matching of a bipartite graph, there is the well-known augmenting path algorithm. (See Papadimitrou (1982) or Ahuja et al. (1993) for a description.) [1]

[1]The fastest known algorithm has asymptotic time complexity of $O(|V|^{1/2}.|E|)$ and is based on max flow problem (Hopcroft and Karp (1973)). The reduction itself can be carried out in linear time on number of nodes and edges.

6.4.2 Reduction to Assignment Problem

In the last section, we saw that if all our karakas are mandatory karakas, the problem of finding a parse reduces to finding a maximal matching in a bipartite graph.

If we permit optional karakas, the problem still has an efficient solution. It now reduces to finding a matching which has the maximal weight in the weighted matching problem. To perform the reduction, we need to form a weighted bipartite graph. We first form a bipartite graph exactly as before. Next the edges are weighted by assigning a weight of 1 if the edge is from a node in V representing a mandatory karaka and 0 if optional karaka. The problem now is to find the maximal matching (or assignment) that has the maximum weight (called the *maximum bipartite matching* problem or *assignment* problem). The resulting matching represents a valid parse if the matching covers all nodes in U and covers those nodes in V that are for mandatory karakas. (The maximal weight condition ensures that all edges from nodes in V representing mandatory karakas are selected first, if possible.) This problem has a known solution by the Hungarian method of time complexity $O(n^3)$ arithmetic operations (Kuhn, 1955).

Note that in the above theory we have made the following assumptions: (a) Each word group is uniquely identifiable before the core parser executes, (b) Each demand word has only one karaka chart, and (c) There are no ambiguities between source word and demand word. Empirical data for Indian languages shows that, conditions (a) and (b) hold. Condition (c), however, does not always hold for certain Indian languages, as shown by a corpus. Even though there are many exceptions to this condition, they still produce only a *small* number of such ambiguities or clashes. Therefore, for each possible demand group and source group clash, a new constraint graph can be produced and solved, leaving the polynomial time complexity unchanged.

6.5 Preferences over Parses

World knowledge can also be used during parsing. But the question is, what kind of world knowledge should be used? Such knowledge explodes in size very fast, and is difficult to use during parsing (or processing). One answer is to use semantic types of fillers of karaka roles, but to limit it to that necessary for removing ambiguity, if any, in karaka assignment. In other words, for a given verb, when karaka-vibhakti mapping is not sufficient for producing an unambiguous parse, semantic types are included. The semantic types so included have the sole-purpose of karaka disambiguation. This keeps the number of semantic types under control, and serves as a guiding philosophy for what semantic types to include. Figure 6.8 shows

the starting semantic type hierarchy which is sufficient for a major part of language.

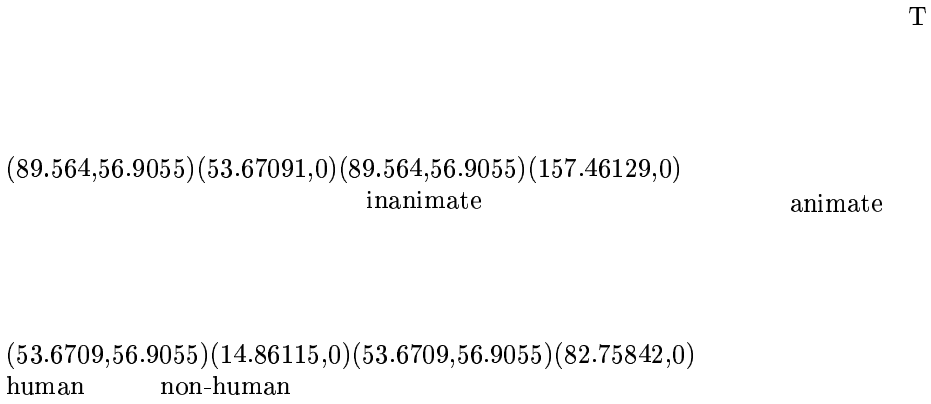


Figure 6.8: Semantic type hierarchy

Sometimes we have constraints which normally hold but sometimes can be over-ridden. These preference constraints can be used to order the parses produced by the parser. If we do so, we see those parses first which satisfy the preference constraint. However, if a parse does not satisfy the preference constraints, it is still produced, but only later.

Preferences over parses can be specified by suitably varying the cost function in the integer programming problem. If we use such a cost function, we will get a parse that has a minimum cost with respect to the cost function. The integer programming system can be so set up that we can ask for the next solution, in which case, we will get another parse with the same or higher cost. This can be repeated to obtain all the possible parses.

Some reasonable preferences which can be incorporated in the cost function are as follows: All else being equal,

1. Karta has the following preferences in descending order: human, non-human, inanimate (animacy preference).

2. A source group is close to the demand group with which it has a relationship. (closeness preference)

3. Karta occurs before karma in a sentence (leftness preference).

To the above list can be added a host of conditions dealing with anaphora, movement, garden-path sentences, etc.

6.6 Lakshan Charts for Sense Disambiguation

The second major task to be accomplished by the core parser is disambiguation of word senses. This requires the preparation of lakshan charts (or discrimination nets) for nouns and verbs.

A lakshan chart for a verb allows us to identify the sense of the verb in a sentence given its parse. Lakshan charts make use of the karakas of the verb in the sentence, for determining the verb sense. For example, in sentences J.1 and J.2 the verb 'jotataa hei' is used in two different senses: plough and attach.

```
J.1 kisaana  kheta  ko jotataa hei.
    farmer  land   -ko  ploughs
    (The farmer ploughs the land.)

J.2 kisaana  gaadii  ko jotataa hei.
    farmer   cart   -ko attache
    (The farmer attaches the cart.)

J.3 kisaana  kheta  ko kaaTataa hei.
    farmer   crops -ko harvests
    (The farmer harvests the crops.)
```

Lakshan chart for 'jota' would allow us to select the appropriate sense of jota by testing whether its karma is land or cart etc. Again, it is designed from an information theoretic point of view. The available information is used in an economical and efficient manner in deducing the right amount of new information.

A verb lakshan chart for a verb is prepared by linguists and language experts by looking up different senses of the verb with the help of conventional dictionaries. The chart builder must carefully select features that would allow verb sense to be obtained.

Noun lakshan charts help disambiguate senses of nouns in a sentence. They make use of the parse structure (i.e., karaka relations) and the verb sense. For example, in sentences J.1 and J.3, the sense of kheta is land and crop, respectively. However, depending on the verb of which it is a karma, the appropriate sense is selected.

Preparation of lakshan charts is a laborious process but is essential for machine translation. Linguistic theories are generally silent on the issue of word sense disambiguation.

6.7 Summary

The Paninian framework turns out to be elegant as well as natural and efficient. It is elegant because it is able to handle diverse phenomena like karaka assignment, active-passives, and control in a unified manner. It is efficient because the constraint parser which arises very naturally using the framework is extremely fast.

Further Reading

The parsing scheme by integer programming described here was earlier discussed in Bharati et al. (1990). Reduction to matching problem in bipartite graph is discussed in Bharati et al. (1993a).

For further details on complexity analysis of parsing algorithms of the constraint parser described here see Perraju (1992). Ravisankar (1991) discusses different ways that the integer programming problem can be formulated to obtain various parses in preferential order. Bharati et al. (1990a) discuss relationship between NLP, complexity theory and mathematical logic.

Exercises

6.1 Show the constraint graph and solution graph(s) for each of the following sentences in Hindi:

```
raama  phala  khaakara     mohana ko  bulaataa  hei.
Ram    fruit  having-eaten Mohan -ko  calls     is
(Having eaten  fruit, Ram calls Mohan.)

raama ne phala   kaaTakara   khaayaa.
Ram   ne fruit   having-cut  ate
(Ram ate having cut the fruit.)

phala   kaaTane  ke liye usane caakuu  liyaa.
fruit   to-cut  -ke-liye he-ne knife   took
(To cut fruit, he took a knife.)
```

6.2 Show the constraint graph and solution graph(s) for each of the following sentences in Hindi:

```
shikaarii ne  bhaagaate hue  shera  ko dekhaa.
hunter   -ne while-running   lion  -ko saw
(The hunter saw a lion while running.)

raama ne haatha se    chiilkara   kelaa  khaate hue
Ram  -ne hands -se having-peeled  banana eating
```

```
bandara ko dekhaa.
monkey -ko saw.
(Ram saw a monkey eating a banana having peeled it with
his hand.)
```

6.3 Form integer programming problem for the sentences in Exercise 6.1.

6.4 Form integer programming problem for the sentences in Exercise 6.2.

6.5 Prove the converse (of the theorem proved in Sec. 6.4.1) that if a constraint graph (with no optional karaka) has a parse, there is a maximal matching covering all the nodes in its bipartite graph, and if it does not have a parse there is no matching covering all the nodes in the bipartite graph.

6.6 Incorporate the preferences described in Sec. 6.5 in the cost function of integer programming problem. In other words, design the cost function in such a way that a solution satisfying the above preferences is produced before another which does not satisfy the preferences.

6.7 (Open problem) Suppose the solution graph must satisfy an additional constraint, called *nesting constraint* defined as follows:

Definition (Nesting constraint): If the nodes of the solution graph are placed on a straight edge of a semi-infinite plane in the order of their occurrence in the sentence, the edges can be drawn without crossing each other.

What is the complexity of finding a solution graph that satisfies the above constraint besides the usual constraints (C.1, C.2 and C.3).

Chapter 7

Machine Translation

7.1 Survey

7.1.1 Introduction

You feed a story written in Oriya into a computer system and out comes its translation in Hindi, Tamil, English and other languages. It is inexpensive, immediate and simultaneous. The language barriers melt away. The richness of other literatures opens up to everyone. The world is intellectually and culturally united into one. This is the dream of people working in a fascinating area of research called Machine Translation (MT).

Although the above goal is far from realization, the first signs of limited success are apparent. In fact, the first faltering steps in MT were taken in the 1950s and 1960s. But at that time, formal liguistics and artificial intelligence had barely been born, and computer science was in its infancy. As a result, the efforts "failed" to achieve success.

7.1.2 Problems of Machine Translation

What makes the MT task so dificult? It was believed at one time that all that was required for MT was a bilingual dictionary and rules for reordering words in a sentence. For example, to translate the following sentence from Hindi to English:

```
raama    ne     kheta    jotaa.
(noun)          (noun)  (verb)
Ram  ploughed  the field
(n)    (v)        (n)
```

the Hindi words are replaced by English equivalents and reordered to follow the sequence non-verb-noun instead of noun-noun-verb. Unfortunately,

this naive method fails to work generally. A number of difficult problems come in the way, like word sense selection, choice of sentence structure, pronoun reference, noun-noun modification, conjunctions like 'and' and 'or', identification of tense and modality, etc. We will see some of these below.

Each word has many different meanings or senses. Selection of the appropriate sense is necessary for translation. For example, consider another sentence in Hindi in which 'jotaa' has a different sense than 'plough':

```
raama ne gaaDii jotii.
Ram  prepared the cart.
```

or

```
raama   ne  ghoDaa  jotaa
Ram  harnessed  the horse.
```

Thus 'jotaa' can be replaced by 'plough', 'prepare' or 'harness' depending on the sense implied in the sentence. The correct sense must first be identified for each of the words before selecting the appropriate replacement.

The sentence structure must also be interpreted correctly for translation. For example, in

```
I saw Ramesh on the hill with the telescope.
```

The telescope could have been the instrument of seeing, or Ramesh could have been carrying the telescope, or it is the hill with the telescope. The three translation are different in Hindi:

```
Meine duurbiina dwaaraa Ramesh ko  pahaaDii par dekhaa.
Meine Ramesh ko duurbiina ke saath pahaaDii par dekhaa.
Meine Ramesh ko duurbiina waalii   pahaaDii par dekhaa.
```

Hence, it would be important to identify the relationship of the telescope correctly (called the sentence structure). Such identifications might require following a paragraph and maintaining a context.

Frequently, it is important to find the referent of the pronoun. Consider the following sentences for example:

```
A dog saw a cow on the road.
It started barking on seeing it.
```

The first 'it' can refer to a dog, cow or road. If one were translating into Hindi, the gender of the verb would depend on the gender of the referent:

```
vaha use dekhakara bhounkane lagaa.
```

If 'vaha' was referring to the cow or the road, feminine would be used.

It may sound ridiculous to even consider that the road can bark, or it may seem obvious that it is the dog which barks. However, reaching such conclusions requires more than just word replacement ability.

The problems described above point to a common theme: A sentence must be "understood" (at least partially) before it can be translated.

Natural language understanding is a hard task because it requires formulating not only a grammar for the language but also using background knowledge including common sense knowledge. All this must be used in complex and as yet unknown ways to process a given text.

7.1.3 Is MT Possible?

The difficulty of the task makes it clear that literature and poetry are beyond MT in the foreseeable future. Legal texts which are carefully drafted to avoid ambiguity and unwanted implications might also be beyond the ambit of MT. Jokes and other material which rely on conveying double meaning are also not good candidates for MT at present.

Are there any task domains where MT is applicable? While definitive statement cannot be made, it seems that the task domains are rather narrow. At first sight it appears that circulars, official communication, minutes of meetings, technical literature including scientific papers, manuals etc. are applications that can be handled by MT. These texts have limited vocabulory and fixed styles. Their audience is not very large. Many of such texts have a short life. Circulars and minutes of meetings are used intensively for a short period of time, and then filed and forgotten. They are also considered uninteresting by human translators. Because of the above reasons, these texts are ideal candidates for MT. However, there are no practical automatic systems for translation other than one system in the domain of meteorological forecasts.

So far we have discussed MT as a fully automatic system requiring no human intervention. There are other possibilities as well. One can have human aided machine translation. The human (translator) could pre-edit the text in the source language paraphrasing the difficult sentence constructions so that the computer can understand it, or aid the computer during the translation process when it has a difficulty it cannot handle, or post-edit the generated text in the target language. The MT system should be viewed as a tool provided to the human translator.

Most MT systems present today require post-editing. Note that if the MT systems become reasonably good, the post editor needs to know only one language.

7.1.4 Brief History

The history of MT can be traced starting from the early 50s when it was realized that computers could be used for translation. In the US, a large number of research groups sprang up to work on the task (usually Russian

to English), with funding from defence and intelligence establishments. In the USSR, there was a similar effort to translate from English and French to Russian.

As mentioned earlier, most of this work based itself on bilingual dictionary lookup. The developers quickly started realizing that far more was needed. But unfortunately, in their enthusiasm and optimism during the early days, they had proclaimed that MT systems were around the corner, and that the MT systems would be capable of producing high-quality translations for general texts without any human intervention. Thus in the US, when a committee called ALPAC was set up to evaluate the MT research, it easily came to the conclusion that research had failed to live up to its promises. It said in its report in 1966 that basic research was needed and MT was not feasible in the foreseeable future.

The ALPAC report rang the death knell of MT efforts in the US at that time. All funding ceased, the research groups disintegrated, and the field went in disrepute. The fate of MT in Europe and the USSR did not change so dramatically. It was generally recognized, however, that it was a field whose time had not yet come. Only a few research groups continued to remain active.

The field revived in the late 70s after the successful completion of the TAUM-METEO system in Canada in 1977. It translates the Canadian weather forecasts from English to French. Around the same time other systems like Titus (English to French for textile technology), CULT (Chinese to English for Mathematics and Physics journals), etc. were also developed.

In the 80s, the Japanese successfully completed a national project (Mu) on MT between English and Japanese. The European Community has also undertaken an ambitious project called Eurotra covering all the languages of the Community. Work has also been undertaken by groups in France, Germany, Switzerland, the US and India.

7.1.5 Possible Approaches

There are three major types of MT systems that one can imagine based on their dependence on languages. The first type of MT systems are designed for a particular pair of languages. If translation capability is needed between another pair of languages, a new system must be constructed. Because of its dependence on the languages, the system has the advantage that special features and similarities between the concerned languages can be made use of. It is called the direct approach.

The second type of system is based on the concept of interlingua. A sentence in a source language is first analyzed and is represented in an intermediate language. The intermediate language need not be a human "natural" language and is typically a formal "mathematical" language. Next a

generator takes the intermediate representation and generates a sentence in the target language. The major advantage of this approach is that the analyzer of parser for the source language is independent of the generator for the target language. As a result, if one wants to build a system with translation capability amonge, say, 15 languages, only 15 parsers and generators need to be constructed. Contrast it with 210 (=15x14) systems that need to be constructed in the first approach.

The difficulty with the second approach, however, is that it is difficult to define the interlingua. Also, it is not possible to take advantage of similarities between languages. As a result, many research groups use a third approach called the transfer approach. In this approach, the parser produces the representation for a source language, which is then transfered to the representation for the target language. The generator takes over from there. This approach is intermediate between the first two. For 15 languages, in this approach only 15 parsers and generators are needed, however, the number of transfer components needed are 210.

In the Indian context where there are a large number of languages (officially 15 major ones) which are also very close, the interlingua approach would seem preferable, but the nature of interlingua is open.

7.1.6 Current Status

Currently the field is going through a phase of vigorous activity. The Japanese National Project (Mu) successfully completed resulting in an industrial prototype. Work has already started on a major development effort. Eurotra project in Europe has hundreds of researchers working on it. It uses the transfer approach in which a tree representation of a source languages sentence is obtained by a series of steps involving formal grammars and well formedness fiters. This representation is transformed to another tree structure using the transfer component, and forms the basis for generation. A number of other research groups are also active in Europe.

India too has active groups in MT. The earliest published work was undertaken by Chakraborty in 1966. More recently, work has been undertaken in Tamil University Thanjavur, National Centre for Software Technology (NCST) Bombay, Centre for Development of Advanced Computing (C-DAC) Pune, and IIT Kanpur. Research group at Thanjavur attempted Russian to Tamil translation based on the direct approach in 1985 and the first system translated simple sentences. A group at NCST did some work on English to Hindi translation but did not develop a working system. The group at C-DAC did some preliminary work on MT, but has lately been concentrating on processing of Sanskrit. The IIT Kanpur effort by (Akshar Bharati group) is the most comprehensive and is focusing on Indian languages. It has applied the principles of Indian traditional grammar (San-

skrit vyakarana) to modern Indian languages. University of Hyderabad is an important centre starting work on computational linguistics and MT.

The MT systems will have to fit in an organization. Instead of replacing people, they will complement them. There can be other fallouts from the MT effort. At an intellectual level, it can give a big boost to the traditional Sanskrit vyakaran studies, linguistics and artificial intelligence disciplines. These will grow to new heights borrowing from one another. At the practical level, one can imagine a number of computer based products. For example, language assistant (a system that corrects grammatical mistakes), computer assisted language instruction (which includes various lessons including comprehension tests for language teaching), natural language interfaces to computers (so that one can communicate with computers using language) etc. The marriage of speech processing work with natural language processing has the potential of producing computer systems that can listen and talk. If the optical character recognition technology is also combined, it can lead to systems that can read out books (to the blind, for example).

Work for Indian languages will have to be done mainly by Indians. We are also the best equipped to do it. The problems are challenging and the potential immense. What is needed is a national will and a national effort.

7.2 Anusaraka or Language Accessor

[1]

It is possible to overcome the language barrier in India today using anusaraka. Anusaraka tries to take advantage of the relative strengths of the computer and the human reader, where the computer takes the language load and leaves the world knowledge load on the reader. It is particularly effective when the languages are close, as is the case with Indian languages. Keeping in line with the anusaraka philosophy, it bridges the gap between languages by chosing the most appropriate or nearest construction available in the target language together with suitable additional notation.

It is argued in this section that there are only three major differences in the south Indian languages and Hindi. All these can be bridged by simple additional notation in Hindi. The resulting language can be viewed as a southern dialect of Hindi. Anusaraka uses this dialect to make source text in southern languages accessible to Hindi readers.

[1] +Pronounced as anusaaraka.

7.2.1 Background

The problem being addressed here is how to overcome the language barrier in India. Fully-automatic general-purpose high-quality machine translation systems (FGH-MT for short) are extremely difficult to build. There are no existing translation systems for any pair of languages in the world that qualify to be called FGH-MT. The difficulty arises from the following reasons:

1. In any natural language text, only part of the information to be conveyed is explicitly expressed and it is the human mind which fills up the details by using the world knowledge. The basic reason for this state of affairs is that the concepts and shades which a natural language purports to describe form a continuum and the number of lexical items available are finite, so necessarily they have to be overloaded and it is only because of the total context and shared background understanding that we are able to disambiguate them and are able to communicate.

2. Different natural languages adopt different conventions about the type and amount of information to be used. The reasons for this may be: the history of language development (differing tastes, arbitrary choices made in the long history of language, presence of other languages and mixing with them, etc.) and the envisaged primary function of the language etc.

The net result of this is that unless we provide machines with knowledge and inferring capability comparable to human beings FGH-MT will not be feasible. It will not be an exaggeration to say that inspite of tremendous progress in computer technology (mainly in terms of speed, memory and programming environments) FGH-MT remains a distant dream.

7.2.2 Cutting the Gordian Knot

In spite of the difficulty of FGH-MT, it can be claimed that with the help of machines, the language barrier can be overcome today. If this sounds paradoxical let us consider an analogy. Scientists even today are struggling to build a machine that can walk like a human, avoiding obstacles to reach a destination. At the same time, we have been successfully using rail transport for more than a century. The distance barrier has been overcome even without the machines learning to walk.

True, for this we have had to lay railway tracks all over the country; build bridges across rivers; dig tunnels through mountains and build a huge infra-structure to make the whole thing feasible. Even then it delivers the goods and passengers at only the railway stations; we need separate

arrangements to get the things and persons at home! But all said and done, it does enable us to overcome the distance barrier.

If FGH-MT is like the walking machine, what is the counterpart to the railway locomotive? The answer lies in separating language-based analysis of text, from knowledge and inference-based analysis. The former task is left to the machine, and the latter task to the human reader because of their proficiency for the respective tasks. We also relax the requirement that the output be grammatical. We do require, however, that the output be comprehensible. Apart from the effort to build appropriate multilingual databases from a computational viewpoint, creative ideas are needed to establish language bridges. Though this is challenging, it is definitely feasible. With the appropriate infra-structure, the language barrier can be overcome with today's technology at least among the Indian languages.

7.2.3 The Problem

The practical aspect of this problem is to divide the load between man and machine in such a way that the aspects which are difficult for the human being are handled by the machine, and aspects which are easy for the human being are left to him. The aim is to minimize the effort of the human being.

The approach, however, can not be a heavy-handed method because the problem is very complex. A brute force solution will either blow up as a result of the large amount of resources it will need, or the large amount of time it will take. Another desirable feature of the approach would be the ease with which the system can be extended to other related languages. The system is likely to possess efficiency, extensibility, etc. only if it is based on sound principles and theory.

Theoretical issues of interest relate to information and how it is coded in language. How is the information extracted? A related question is why the same amount of surface information in one language is clearly understood while in another language, it is not clear to the reader. What are the sources of knowledge that are used in information extraction? How should the knowledge be organized?

Although the concept of anusaraka is general, we will discuss it in the context of Indian languages: with respect to Kannada-Hindi anusaraka in particular.

7.2.4 Structure of Anusaraka System

Structure of the anusaraka is shown in Figure 7.1. A source language sentence is first processed by morphological analyzer (morph). The morph considers a word at a time, and for each word it checks whether the word is

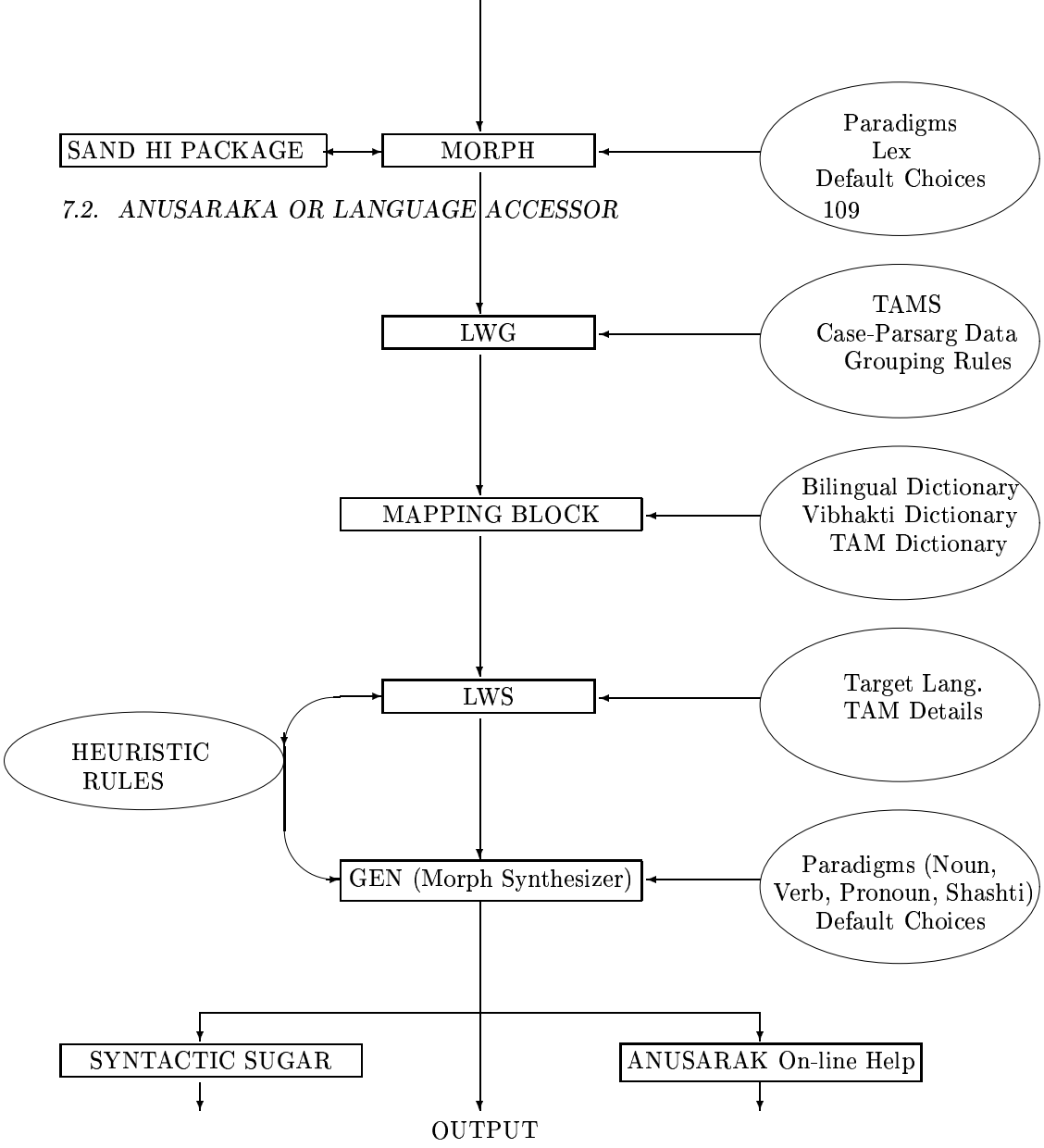


Figure 7.1: : Block Schematic of Anusaraka

in the dictionary of indeclinable words. If found, it returns its grammatical features. It also uses the word paradigms to see whether the input word can be derived from a root and its paradigm. If the derivation is possible, it returns the grammatical features associated with the word form (obtained from the root and the paradigm). In case, the input word cannot be derived, it is possibly a compound word and is given to the sandhi package to split it into two or more words, which are then again analyzed by morph.

The output of morph is given as input to the local word grouper. Its main task is to group function words with the content words based on local information such as postposition markers that follow a noun, or auxiliary verbs following a main verb. This grouping (or case endings in case of inflectional languages), identifies vibhakti of nouns and verbs. The vibhakti of verbs is also called TAM (tense-aspect-modality) label.

After the above stage, sentential analysis can be done. Current anusaraka does not do this analysis because it requires a large amount of linguistic data to be prepared. Also, since the Indian languages are close, the 80-20 rule applies to vibhakti. Use of vibhakti produces 80% "correct" output with only 20% effort.[2] Sentential parser can be incorporated when large lexical databases are ready.

The next stage of processing is that of the mapping block. This stage uses a noun vibhakti dictionary, a TAM dictionary, and a bilingual dictionary. For each word group, the system finds a suitable root and vibhakti in the target language. Thus, it generates a local word group in the target language.

The local word groups in the target language are passed on to a local word splitter (LWS) followed by a morphological synthesizer (GEN). LWS splits the local word groups into elements consisting of root and features. Finally, GEN generates the words from a root and the associated grammatical features.

7.2.5 User Interface

Anusaraka output is usually not the target language, but close to it. Thus, the Kannada-Hindi anusaraka produces a dialect of Hindi, that does not have agreement etc. It can be called a sort of Dakshini (southern) Hindi. Some additional notation may also be used in the output. Certain amount of training is needed for a user to get used to the anusaraka output language.

The role of the anusaraka interface (or on-line help in Figure 7.2) is to facilitate the reading of output by the reader. It should keep track of what concepts have been introduced to the user, and also provide on-line help when the user faces a problem.

[2]It is our estimate that the sentential parser will improve the performance only marginally for translating from south Indian languages to Hindi.

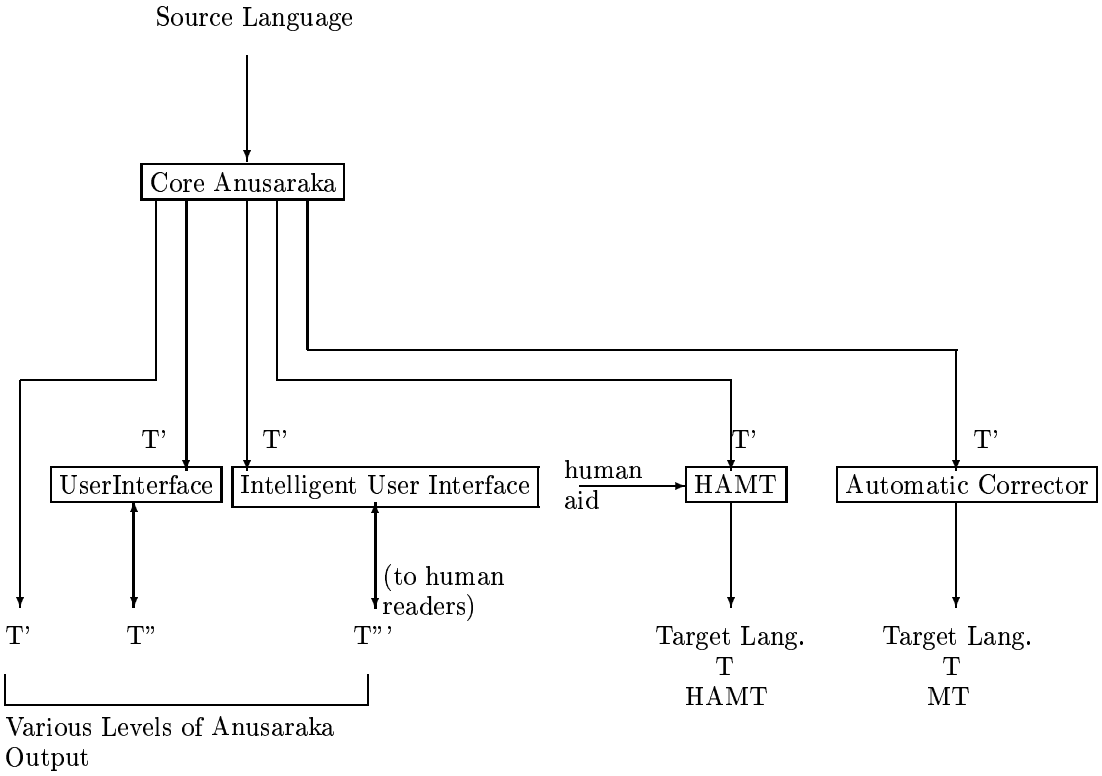


Figure 7.2: : Different Interfaces for Anusaraka

Depending on the nature of interface, one can have an ordinary user interface, intelligent user interface, ggHAMT (human aided machine translation), and machine translation (see Figure 7.2).

7.2.6 Linguistic Area

The reason the above approach works for Indian languages even without a full fledged parser is that the source and target languages are similar. At mapping level, it is possible to find roots, TAMs, and noun vibhakti in Hindi from an equivalent Kannada text without too many clashes.

Based on our experience, we can say that apart from agreement, there are only 3 major differences in the south Indian languages with Hindi. All these can in fact be bridged. They are described below for Kannada-Hindi

anusaraka. But first let us look at agreement.

7.2.7 Giving up Agreement in Anusaraka Output

Hindi has an agreement rule which can be stated as follows:

Gender number person (gnp) of the verb agrees with the gnp of the karta if it has ϕ vibhakti[3], otherwise the gnp of the verb agrees with the karma if it has ϕ vibhakti; otherwise the verb takes masculine, singular, third person form.

When the input Kannada sentence has an ambiguity in identifying karta (or karma whichever is relevant for agreement) and the gnp of the two candidate kartas (or karmas) is different, an ambiguity will appear in the gnp of the verb.

Similarly, possessive modifiers of nouns need information about gnp of the related nouns. There are a number of cases where it is not easy to compute. Sometimes, the relevant information may not even be available.[4]

Kannada has three genders (masculine, feminine and neuter), whereas Hindi has only two (masculine and feminine). Consequently, some information loss is bound to occur in going from Kannada to Hindi. To avoid this loss, it will be necessary to provide additional notation to mark the neuter gender. Here is an example to give its significance:

In Hindi, for karma-kartr prayoga, typically separate verb forms are available; but in Kannada, frequently a single verb stem is employed. Thus, on the surface, it seems that it does not distinguish between "daravaajaa khulaa" and "daravaajaa kholaa". However, in practice one can distinguish between the two usages by paying attention to the gender marking on the verb; in the first case it will be neuter gender while in the second, typically, it will be a non-neuter gender[5] (where K specifies Kannada sentence, @H specifies Hindi produced by anusaraka).

```
  K: baagilu    tereyitu.
@H: daravaajaa khulaa [ kholaa].
    (The door opened.)
  K: baagilu    teredanu.
@H: daravaajaa khulaa [ kholaa].
    ( (He) opened the door.)
```

[3]Karta having ϕ vibhakti means that it is not followed by a postposition marker (a function word).

[4]There is a temptation to provide correct agreement according to standard Hindi grammar, wherever it is possible without much effort and not bother about agreement when it is not possible to do so easily; but such a policy will be potentially confusing to the user. From the utility point of view, it always helps to keep the working of the system simple and to provide a faithful picture of the working of the system to the user.

[5]"daravaajaa kholaa" (opened the door.) is an incomplete sentence in isolation but it can occur in discourse in response to the question "raama ne kyaa kiyaa?" (What did Ram do?)

Loss of agreement is not expected to be too jarring to Hindi speakers in view of their exposure to various varieties of non-native Hindi heard everyday as propagated by television, radio and films.

7.2.8 Language Bridges

Apart from agreement there are only three major syntactic differences between Hindi and Kannada. Surprisingly all of these can be taken care of by enriching Hindi with a few additional functional particles or suffixes as shown below. Thus, they can be viewed as lexical gaps or function word gaps. language bridges

"ki" construction

In case of embedded sentences in Hindi, these are put after the main verb unlike in Kannada. For example (where K, H, and E specify the language of the sentence as Kannada, Hindi and English, respectively; '!H' stands for gloss in Hindi, '!E' for gloss in English, @H for anusaraka output in Hindi, etc.):

```
(7.1) H: raama ne kahaa ki   mohana kala     aayegaa.
     !E: Rama     said  that mohana tomorrow come-fut
      E: (Rama said that mohana will come tomorrow.)

      K:*raama heLidanu eneMdare   naanu manege   hoguttene.
     !H: raam  kahaa    ki         meiM  ghara ko jaauuMgaa.
         (Ram said that he will go home.)
```

Note, that the above Kannada sentence sounds odd to a Kannada person, because of the order of the constituents. However by using "eisaa" instead of "ki" we can easily avoid this problem, as illustrated below:

```
(7.2)  K: mohana naaLe    baruvanu  eMdu raama heLidanu.
      !H: mohana kala     aayegaa   eisaa raama ne kahaa.
      !E: Mohana tomorrow come-fut that  Rama  said.
```

'eisaa' construction is a proper construction in Hindi; only it is used less frequently. In the dialect of Hindi produced by anusaraka, however, this will be the normal construction used.

"jo" construction

Kannada has a large number of adjectival participial phrases or clauses which convey information about tense, aspect etc. but they do not have information about karaka relations. In sentences (7.3) and (7.4), Kannada codes information about tense in eating and making.

```
(7.3) K:  raama tiMda camacavannu toLe.
      @H: raam khaayaa thaa cammaca dho Daalo.

(7.4) K:  raama tayaarisida camacavannu toLe.
      @H: raama banaayaa thaa cammaca dho Daalo.
```

It does not contain information about the relation eat has to spoon or make has to spoon. It is the background knowledge that provides the relation between them. For example, the general world knowledge may be used to say that spoon is the instrument of eat (and is not eaten) in (7.3). In the next sentence (7.4) spoon could be the instrument or theme of make.[6]

Hindi, on the other hand, has only two participial phrases viz. yaa_huaa and taa_huaa which code perfective and continuous aspects only, e.g.,

```
(7.5) H: khaayaa huaa phala
         eaten       fruit

(7.6) H: khaataa huaa laDakaa
         eating       boy
```

Thus, anusaraka would be able to use these constructions in Hindi only when the tense information in Kannada is appropriate. But what about the other tense, aspect and modality? There is a syntactic hole in Hindi!.

There is another problem, however. The two participial phrases in Hindi have coding for karaka relations which is absent in Kannada. yaa_huaa codes karma[7], while taa_huaa codes karta. Participial phrase (7.5) says that fruit that was eaten, while (7.6) says the boy who is eating. Thus, Hindi is poorer than Kannada in coding tense, aspect, modality information, while richer in coding karaka information in case of adjectival participles. But this compounds the problem for anusaraka. Using these constructions in Hindi would mean putting in something that is not contained in the source language sentence.

The answer lies in identifying another construction in Hindi and creating a correspondence between it and the constructions under consideration in Kannada.

Hindi has another construction-the 'jo' construction-which allows both tense and karaka information to be specified. For example, to say 'wash the spoon with which Ram has eaten' we can write any of the following:

```
(7.7a) raama ne  jisa cammaca se khaayaa thaa usako dho Daalo.
```

[6]Most native speakers are surprised to learn of this fact. The information dynamic view, therefore, raises questions very different than those raised and studied in conventional linguistics.

[7]More correctly, yaa_huaa codes karma in case of sakarmaka or transitive verbs, and karta in case of intransitive verbs.

```
       raama erg. which spoon instr. eaten    that wash
       (Wash the spoon with which Ram has eaten)
(7.7b) raama ne khaayaa thaa jisa cammaca se use dho Daalo.
(7.7c) raama ne khaayaa thaa jisase usa cammaca ko dho Daalo.
```

To express the same information as in the Kannada sentences (7.3) and (7.4), we can invent a notation along with the jo-construction as follows:

```
(7.8')  raama ne khaayaa thaa jo_* vaha cammaca ko dho Daalo.
```

The vibhakti markers (i.e., the functional words se, ko etc.) are replaced by '*'. 'jo_*' could even be replaced by 'jouna' to produce a kind of colloquial Hindi in some region.

```
(7.8")  raama ne khaayaa  thaa jouna vaha cammaca ko dho Daalo.
```

Unlike the first case, this idea takes some time and effort for the Hindi reader to get used to.

"ne" construction

This "ne" construction is a peculiarity of only the Western belt languages in India. In case of the present or past perfective aspect of the main verb in Hindi sentence, "ne" is used with the karta:

```
(7.9) H: raama ne  phala khaayaa.
         Ram  erg. fruit ate.
         (Ram ate the fruit.)
```

In case of yaa_gayaa TAM label, 'ne' is not used.

```
(7.9') H: raama  phala khaa gayaa.
```

Therefore, we can postulate a new word TAM "yaa'" with same semantics as "yaa", but which does not use "ne" construction; with this TAM we can express the corresponding Kannada sentence more faithfully as:
`raama phala khaayaa.`[8]

7.3 Summary

We have argued that it is possible to overcome the language barrier in India using anusaraka. Anusaraka tries to take advantage of the relative strengths of the computer and the human reader, where the computer takes the language load and leaves the world knowledge load on the reader. It is particularly effective when the languages are close, as is the case with Indian

[8]It may be of interest to note that the "yaa" pratyaya in Hindi corresponds to "kta" pratyaya in Panini's grammar and so the new proposed pratyaya "yaa" will be natural counterpart of the "ktavatu" pratyaya in the Sanskrit grammar.

languages. Keeping in line with the anusaraka philosophy, it bridges the gap between languages by chosing the most appropriate or nearest construction available in the target language together with suitable additional notation.

It needs to be re-stated that even without a sentential parser, the anusaraka for Indian languages delivers something practical today. More detailed sentential or text analysis requires preparation of karaka charts and other lexical databases. These can be gradually built and incorporated in the system. Thus, the system is designed to grow modularly.

Anusaraka can be viewed from the following different points of view.

1. Anusaraka as an evolutionary system. Anusaraka delivers something practical today without waiting for several years and has the potential to keep pace with developments in technology; the work to be done for building it, needs to be done in any case for high-quality fully automatic machine translation systems; and its availability will help in accelerating the work towards development of machine translation systems of the future.

2. A practical approach to develop intermediate language (or interlingua) for a group of languages. Designers can get a first hand view of various source language phenomena in terms of the language the designer knows.

3. A modular system. By adding suitable modules, it can be converted into:

   - A device to understand source language text.
   - Human aided machine translation system.
   - Fully automated high quality domain specific machine translation system.

4. A way to factorize language part from world knowledge part. Anusaraka suggests a clean way to separate the language knowledge from world knowledge, and how to use them in construction of a system in a systematic way.

There are some common misconceptions about anusaraka. We describe them now.

1. Anusaraka is a "rough" translation system: Strictly speaking, this is not true. Conceptually, anusaraka is different from translation. It provides exactly the information contained in the source text, while translation involves interpretation of the source text before expressing it in the target language. In case of legal documents, therefore, translation seldom suffices. Anusaraka on the other hand will only

say what is explicitly stated. When one is willing to put in extra effort and time then online output of anusaraka can be superior to any translation. Because anusaraka makes full surface information, as well as complete language knowledge available to the reader, if the reader is willing to take some pains, he can get full appreciation of the original text. The effort on the part of a reader can be minimized by:

(a) the proper design of the intelligent user interface;

(b) proper training on the part of reader;

(c) practice in using the system.

2. Ad hoc improvements such as partial agreement can improve the performance of the system: This is not correct, because then the user will not be sure about what to expect and what not to expect from the machine. It pays to keep the working of the machine simple. Syntactic sugar is to be added at personal risk. It may sometimes be injurious to health. Ad hoc improvements which work for one text might cause grave problems with other texts.

3. Anusaraka, in principle, is against having a sentential parser: Anusaraka is not against sentential parsers. A sentential parser may be included depending on the availability of technology and linguistic databases. As the Indian languages are close, vibhakti mapping is quite effective. Therefore, anusaraka for Indian languages can be built without waiting for the parser to be available. Later, when large computational lexical databases are available, parser can be incorporated. In fact, for an anusaraka from English to Indian languages, a parser will be absolutely necessary.

Further Reading

Section 7.1 originally appeared in the magazine 2001 (formerly Science Today) in the January 1989 issue. For a tutorial on Machine Translation, see King (1987). Nirenberg (1987) contains several articles on what is possible and what is not. The journal of Machine Translation is exclusively devoted to this area. For a discussion on an appropriate strategy for MT in India, see Bharati et al. (1994a).

Section 7.2 is same as Bharati et al. (1993c). For a discussion on anusaraka as a measuring tool see Bharati et al. (1993d). Anusaraka is discussed in detail in the Ph.D. thesis of Narayana (1994).

Natural language interface to databases is another important application area. Such an interface for Indian languages is discussed in Bhargava (1992) and Bharati et al. (1993).

Chapter 8

Lexical Functional Grammar

Lexical Functional Grammar (LFG) is a strong computational formalism that addresses how to extract grammatical relations from a sentence in a positional language such as English. Its major strength is that it gives explicit algorithms. Its weakness is that it does not offer any theory regarding lexical ambiguity, adjuncts, optional theta-roles, and mapping from grammatical relations to theta-roles. Here, we describe how LFG handles active-passive and dative constructions, and wh-movement in questions. We also discuss features and feature structures, and unification.

8.1 Introduction

LFG has been designed by Kaplan and Bresnan (1982) with a view to provide a computational formalism for analyzing sentences in natural language. The main problem it addresses is how to extract grammatical relations from a sentence. It postulates two levels of representation: one based on constituent structure and the other on grammatical functions such as subject, object. A particular source of difficulty in English is the fact that positions are used for coding both theta relations as well as topicalization etc. Considerable effort has gone into design of LFG so that it can deal with and separate these two kinds of information.

LFG also indicates how the grammatical functions can be mapped onto theta roles. But here, it offers no theory, the mapping must be enumerated exhaustively in the lexicon.

A major strength of LFG is that it gives explicit algorithms for extracting grammatical functions. It uses context free grammar (CFG) for speci-

fying constituent structure. Efficient parsing algorithms for CFG are well known (e.g., Early (1970), Cocke, Younger, Kassami algorithm (in Younger (1967)), Tomita (1986)). LFG uses the powerful unification mechanism for specifying mapping to grammatical relations. The same mechanism uniformly handles constraints across constituents in the constituent structure. More importantly, algorithms that solve these constraints are completely specified.

A weakness of LFG is that it does not offer any theory regarding lexical ambiguity, adjuncts and optional theta roles, and mapping from grammatical relations to theta roles. These tasks are left for the lexicon with LFG offering no linguistic insight as to how to do them. In fact, if one were to exhaustively enumerate the possibilities, in the lexicon the solution would be computationally expensive too.

In this chapter we will focus on some selected aspects of English, namely, two types of movements, and see how LFG handles them. The following are the movements of interest here:

1. active-passive and dative constructions
2. wh-movement in wh-questions

Before going into the LFG formalism, it is appropriate to make a few observations on the phenomena to be explained.

8.1.1 Active-Passive and Dative Constructions

As examples of active-passive consider the sentences

```
(1)  A boy gave a book to the girl.
(2)  A book was given to the girl by a boy.
```

Sentence (2) can be thought of as a sentence that has been obtained from its active counterpart (1) by moving the object 'a book' to the subject position, and converting the subject 'a boy' as a prepositional phrase and moving it to after verb. Moreover, optionally, the latter can be dropped altogether as shown by sentence (3). In fact, in the passive sentence, both object and the 'by' prepositional phrase can be dropped.

```
(3)  A book was given to the girl.
(4)  A book was given, yesterday.
```

In dative construction, there are two objects.

```
(5)  A boy gave the girl a book.
```

This can again be considered to be obtained by movement of 'to' prepositional phrase as the object, and earlier object as object2. Another passive form is possible on dativization. Example sentences with the passive form are given in (6) and (7).

(6) The girl was given a book by a boy.
(7) The girl was given a book.

8.1.2 Wh-movement in Questions

In a wh-question, the wh-phrase containing the questioned element moves from its normal position to the front of the sentence.

Examples are the following sentences[1]

(8) *Ram killed who ?
(9) Who did Ram kill ?
(10) *Mohan said Ram killed who ?
(11) Who did Mohan say Ram killed ?
(12) Who did Mohan say Shyam believed Ram killed ?

As sentence (12) shows, wh-element can move (arbitrarily) long distance in case of (arbitrarily deep) embedded sentences.

8.2 Overview of LFG

LFG assigns two representations (at two different levels) to a sentence. They are called c-structure (for constituent structure) and f-structure (for functional structure). The former is a tree structure which shows word order and hierarchical structure of constituents. The latter is a structure containing a set of attribute value pairs, and it may also be hierarchical. Consider sentence (5) above, as an example.

(5) A boy gave the girl a book.

Its c-structure is shown in Figure 8.1(a). It has the usual syntactic and lexical categories. The tree encodes hierarchical as well as word order information. The corresponding f-structure is given in Figure 8.1(b).

It has the attributes subj, pred, tense, obj, obj2, spec, etc. The values of the attributes are shown next to them. Subj, obj, and obj2 are called grammatical functions because they are specified by the grammar and not by the lexicon. Pred is a special attribute that maps to semantic representation from the f-structure. Its value is the predicate argument combination for the f-structure. From the above f-structure, we can obtain the following predicate argument representation (ignoring spec and tense):

```
give (boy, book, girl)
```

If we take the passive sentence (6)

(6) The girl was given a book by a boy.

[1] Echo questions are not included. Some of the questions that are marked as bad are alright as echo questions.

S

(163.30849,56.9055)(46.72644,0)(163.30849,56.9055)(224.2613,0)

NP

(46.72644,56.9055)(16.25003,0)(46.72644,56.9055)(177.14185,0)(102.35568,56.9055)(92.56398,0)(102.35568,56.9055)(177.14185,0
det (A) n (boy) v (gave) NP (the girl) NP (a book)

(a): c-structure

$$\begin{bmatrix} subj. & \begin{bmatrix} spec & a \\ pred & 'boy' \end{bmatrix} \\ pred & 'give <\uparrow subj, \uparrow obj2, \uparrow obj >' \\ tense & past \\ obj2 & \begin{bmatrix} spec & a \\ pred & 'book' \end{bmatrix} \\ obj & \begin{bmatrix} spec & the \\ pred & 'girl' \end{bmatrix} \end{bmatrix}$$

(b): f-structure

Figure 8.1: Representation of sentence (5)

the final predicate-argument representation must remain the same as before:

```
give (boy, book, girl)
```

indicating that the 'gross' meaning of active and passive remains the same. This holds inspite of the different c-structure and f-structure from the active sentence:

As the f-structure in Figure 8.2 indicates (see values of pred) there would be at least two lexical entries for give: one corresponding to the active use and the other to passive use in dative construction. (In fact, there will be two more lexical entries for normal active and passive, and several more to take care of optional object and prepositional phrases in case of the passives.)

8.3 LFG Formalism

LFG formalism has two major components, a ggcontext free grammar and a functional specification. The former gives the c-structure for a sentence, and the latter gives the f-structure. The two components are interrelated, however, and the f-structure is produced by using functional specification together with the c-structure.

The functional specifications usually consist of equalities associated with each non-terminal on the right-hand side of the context free (CF) rule. In the example grammar from Kaplan and Bresnan (1982) given below, there are two special symbols: up-arrow and down-arrow (called meta-variables). The down-arrow in a functional specification associated with a non-terminal refers to the f-structure with the non-terminal, while the up-arrow refers to the f-structure associated with the symbol on the left-hand side of the CF rule.

(R1) S →	NP	VP		
	↑ subj = ↓	↑ = ↓		
(R2) VP → V	{ NP }	{ NP }	PP*	
	↑ obj = ↓	↑ obj2 = ↓	↑ (↓ pcase) = ↓ obj	
(R3) PP →	prep	NP		
		↑ obj = ↓		
(R4) NP →	det	noun |	pronoun	

Rule (R1) says that a sentence (S) consists of a noun phrase (NP) followed by a verb phrase (VP). The functional specification associated with the NP says that the f-structure for S has an attribute subj whose value is the f-structure for NP. Thus, the f-structure for NP is the subject in the f-structure for S. The second specification says that the f-structures for VP and S are equal.

In rule (R2), both NPs are optional, which are followed by prepositional phrase (PP) repeating zero or more times. The NPs contribute to object or object2, in the f-structure of the sentence, PPs are stored as adjunct.

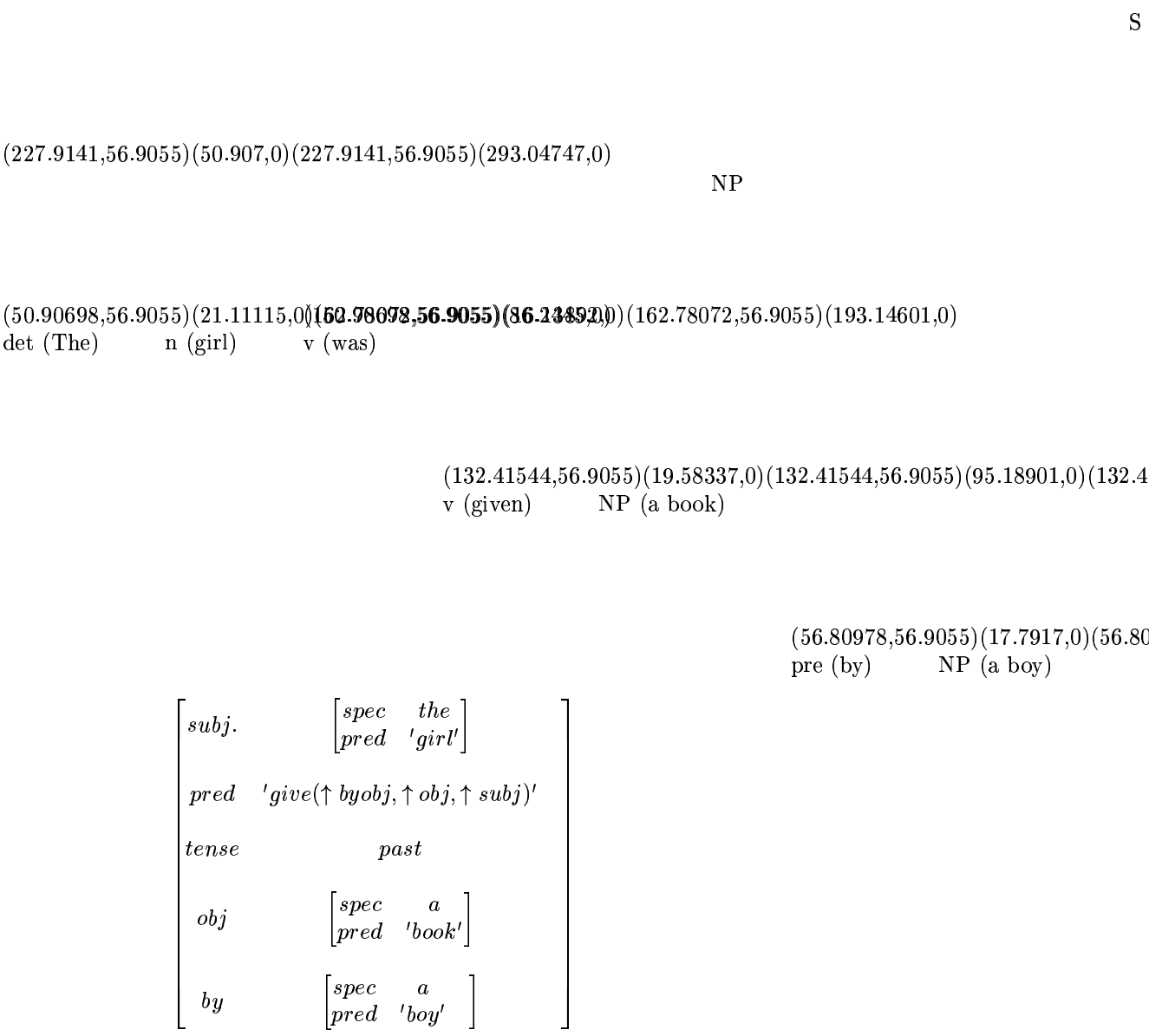


Figure 8.2: C-structure and f-structure for passive sentence (6)

To obtain the f-structure, we must use the functional specification along with the c-structure. Consider as an example, sentence (5) whose c-structure is given in Figure 8.1 (a). Let f1 be the f-structure of the sentence and f2 that of the NP. Therefore, on using the subject specification of the c-structure we get:

```
f1  subj = f2
```

Similarly, other equations can be written down. The terminals also yield equations, using the lexicon. Solution to the equations is an f-structure shown in Figure 8.1 (b). It is associated with the root node S in the c-structure.

Some example lexicon entries are:

the, det
↑ spec = the
boy, noun
↑ pred = 'boy'
↑ num = singular

8.4 Well-formedness Conditions

The f-structure assigned to a sentence must satisfy certain well-formedness conditions. If any of these conditions are violated the assignment is rejected and an alternative structure is explored.

The first condition that must be satisfied by an f-structure is that of *uniqueness*. It says that an attribute in an f-structure can have at most one value. For example, if it is required that the following f-structures (corresponding to 'boys' and 'a') be made equal.

$$\begin{bmatrix} num & plural \\ pred & boy \end{bmatrix} \begin{bmatrix} num & singular \\ spec & a \end{bmatrix}$$

it will fail to yield a consistent f-structure because of the clash in the value of attribute num. Note on the other hand that the f-structures in Fig. 8.3 (a) can be made equal to yield the f-structure in Fig. 8.3 (b).

$$\begin{bmatrix} num & singular \\ pred & boy \end{bmatrix} \begin{bmatrix} num & singular \\ spec & a \end{bmatrix}$$

Figure 8.3 (a): F-structures for 'boy' and 'a'

Figure 8.3 (b): Resulting f-structure on unification of f-structures of 'boy' and 'a'

Figure 8.3: An example of unification of f-structures

$$\begin{bmatrix} num & singular \\ pred & boy \\ spec & a \end{bmatrix}$$

This process of making two f-structures equal is called unification. It will be discussed in detail later. The uniqueness condition is a general method for specifying co-occurrence restrictions including agreement.

The second condition is that of *completeness*. Approximately, an f-structure is complete if it contains all the attributes named in the arguments of its predicate. For example, if we have the attribute pred with the value:

give $<$ ($\uparrow$ subj, $\uparrow$ obj2, $\uparrow$ obj) $>$

the f-structure must contain values of the attributes subj, obj and obj2. This condition would cause the following sentences to be rejected for example:

```
*A boy gave the girl.
*A boy gave.
```

Note that these would not otherwise be rejected by the grammar rules.

The third and final condition relates to *coherence*. It states that if there is a grammatical function in the f-structure, it must also occur in the predicate-argument combination. For example, if there is a grammatical function obj2, then it must also occur in the value of pred. This would cause the following sentence to be rejected:

```
*The boy slept the book.
*The boy ate the apple the girl.
```

because predicate for sleep has only one argument (relating to subject), and that for eat has only two arguments.

The second and third conditions correspond to theta-criterion of GB (see Chap. 12) or aakaankshaa-yogyataa principle of Panini (see Chap. 5.). They capture sub-categorization and theta-role assignment.

8.5 Handling Wh-movement in Questions

To handle long distance movement, there are bounded metavariables notated as down and up arrows with double edged tails ($\Downarrow$, $\Uparrow$). They are

respectively called controller and controllee. Here, we will consider wh-movement related to wh-questions only.

To handle the fronting of wh-phrases we will change rule R1 to R1'. Also, the gap left behind by the movement will be handled by allowing empty noun phrase (see R5).

(R1') S →	{ NP }	NP	VP
	↑ wh = +	↑ subj = ↓	↑ = ↓
	↑ Quest = ↓		
	↑ = ⇓		
(R5) NP →	e		
	↑ = ⇑		

By defining the f-structure of the moved wh-phrase as a controller, and that of the empty NP as a controllee, we indicate that the f-structures are the same.

Thus, controller and controllee have a special meaning in LFG. The closest nested matching controller and controllee are made equal or unified. (This is equivalent to coindexing in other grammars.) Other rules about agreement etc. continue to hold. As an illustration consider the following sentences:

```
*Which boys  e comes to school in time.
 Which boys  e come to school in time.
```

The former sentence is bad because of agreement violation of the empty element with the verb.

8.6 Computational Aspects

As mentioned in the introduction, a major strength of LFG is that it gives explicit algorithms by which c-structure and f-structure can be obtained for a sentence. As it uses CFG, the parsing problem namely, arriving at a c-structure from a sentence is a solved problem. Several efficient algorithms are known and their implementations are available "off-the-shelf."

The f-structure of a sentence can be obtained by using its c-structure and functional specifications in LFG. Here, use is made of unification, a powerful operation. We discuss the notion of feature structures and unification below. The discussion is based on Reyle and Rohrer (1988; Introduction).

8.6.1 Features and Feature Structures

Features have been used in linguistics since long at nearly all levels of linguistic description. A simple example of features associated with a noun phrase has been given in Figure 8.3 in the form of matrix representation.

It can be expressed in the form of a graph representation in Figure 8.4.

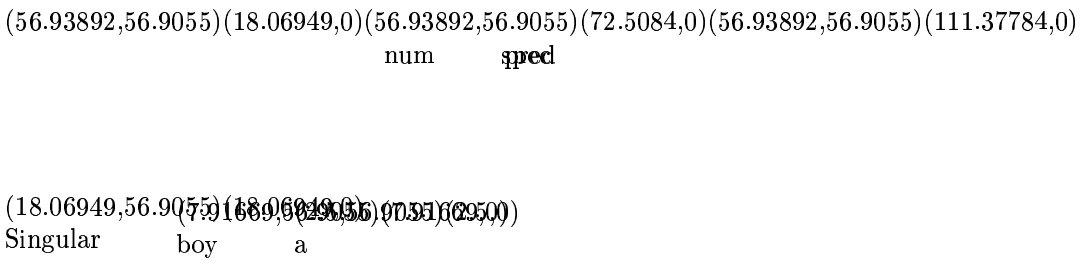


Figure 8.4: Graph representation of a f-structure

It is simple in the sense that each attribute (or feature) has an atomic value that it can take out of a fixed finite set. If we relax atomicity, the value of an attribute can be a feature structure and we can build hierarchical structures. Suppose we were to group num and person into one feature called agreement, then its value would be non-atomic (See Figure 8.5). In f-structures in LFG, we have been using such complex structures.

Feature structures can simply be viewed as complex symbols or complex categories. GPSG by Gazdar et al. (1985) which uses features extensively, in fact, can be shown to be equivalent to a CFG by this (and other) argument(s). To illustrate the argument, suppose we want to say, using features, that subject must agree with the verb in number:

S →	NP	VP
	↑ subj = ↓	↑ subj num = ↓ num
		↑ = ↓

Suppose further that num can take one of two values s(ingular) and pl(ural). Now number agreement can be expressed without features by suitably adding new categories and rules:
S → NP-S VP-S | NP-PL VP-PL
where NP-S stands for singular noun phrase etc.

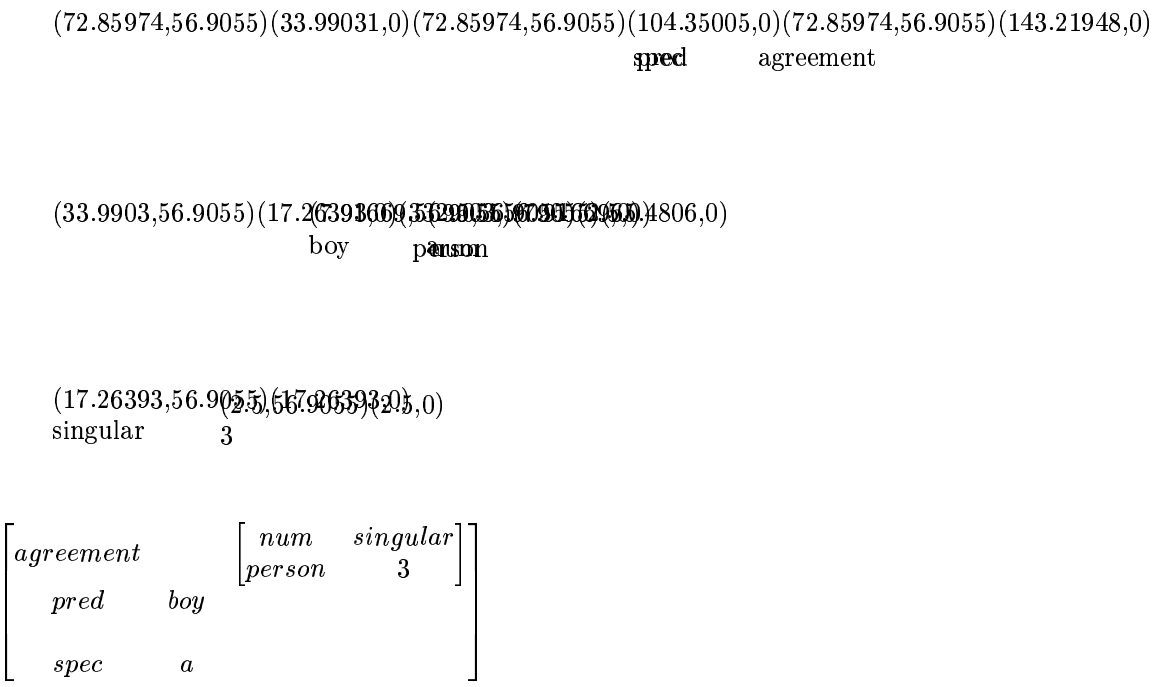


Figure 8.5: Graph and matrix notation for a complex feature structure

8.6.2 Unification

Feature structures or complex categories form a lattice based on subsumption also called extension (notated as $\subseteq$). It can be defined as follows:
Definition: A $\subseteq$ B iff
Case (i) If A and B are atomic values, then A = B
Case (ii) If A and B are feature structures, then if there is a pair (a,u) in A, there is a pair (a,v) in B such that u $\subseteq$ v.

For every attribute value, the bottom element of the lattice is the empty feature structure and the top element T is an artificially introduced element for which X $\subseteq$ T for all X. Subsumption is a partial order which, roughly speaking, expresses whether two structures are consistent and whether one contains more specific information than the other.

Two feature structures FS1 and FS2 are *consistent* iff there is a feature structure FS (other than T) such that FS $\subseteq$ FS1 and FS $\subseteq$ FS2.

Unification of FS1 and FS2 is the least upper bound of FS1 and FS2 over the subsumption lattice. (Unification is said to succeed if a least upper bound (other than T) exists.)

The notion of unification, while it has been borrowed from logic, is more general here. A term in logic has fixed number of arguments, whereas in feature graphs the number of features is open. For example, in Prolog we may express the NP 'a boy' from Figure 8.3(b) as:

```
np ( boy, a, singular)
```

Here, the arguments correspond to pred, spec and num, respectively. It can be obtained by unification from the following two terms for 'boy' and 'a' corresponding to Figure 8.3(a):

```
np (boy,  ___ , singular )
np ( ___, a,  ___ )
```

If we now wanted to add some additional information with the term for 'boy' say, about height, it is not easy as changes have to be made for all kinds of other np terms.

As opposed to above, it is a relatively minor matter in feature structures to add a new feature. Absence of a feature in a feature structure simply means that the structure does not restrict the value of the feature in any way. It can be said that the feature structures assume an open world while the terms assume a closed world.

Thus, feature structures and their unification gives flexibility to the linguists in that they do not have to specify in advance how many features can show up and in what order they should appear.

In unification grammar of Kay (1979), LFG, GPSG etc. the context free rules specify how the constituents are to be built up, and the constraints on features indicate restrictions that hold across constituents (between daughter nodes or between mother and daughter nodes). Informally

speaking, constraints provide the possibility to transport information along the branches of a syntactic tree. (If we get rid of the constraints and encode them into complex categories in context free rules, the information transport across categories is "compiled" by increasing the number of categories.)

If we compare with attribute grammars we find that in those grammars, synthesized features percolate from bottom to top of the tree, whereas inherited attributes flow down the tree. Unification, on the other hand, is order free. The same result is obtained irrespective of top-down or bottom-up processing. Of course, completeness and coherence can only be checked at the end after the final f-structure is obtained.

Unification over feature structures has the advantage that it gives considerable freedom to the linguist in specifying restrictions across constituents. It also gives freedom to the implementor to perform the checks at whatever stage in whatever order he wishes. But there is a price to be paid. It results in an inefficient operation for two reasons:

1. No order is implied, and because there is no way to assert that something, say, some important feature should be checked first, it results in inefficiencies. Structures are built and only much later an important feature may cause failure and undoing of work.

2. Because of the flexibility it provides, unification is an expensive operation.

8.6.3 Other Constraints

LFG also permits existential constraints. It is used to express that an attribute must have a value without specifying the value. For example,
(↑ tense)

is an existential constraint in the CF rule R6 for yes-no questions:

(R6) S →	V	NP	V	
	↑ aux $=_c$ +	↑ subj = ↓	↑	
			(	

It asserts that the attribute tense must have a value in VP. This constraint can only be checked at the end after the f-structure has been built.

A negative existential constraint is also available, which in contrast to above, asserts that the named attribute must not be present.

Finally, there is another type of constraint that expresses necessity of existence of a particular value. For example,
↑ aux $=_c$ +
in rule R6, expresses the necessity that the attribute aux must have the value '+'. Unlike unification (for normal equality) this constraint only checks the value, it does not add it to the f- structure if not already present.

The purpose of this device is to avoid putting default value of an attribute everywhere when the non-default value is needed only at a few places. In the example above, if we did not have the necessity constraint, we would have to say with every non-auxiliary verb, that the value of attribute aux is '-'. Now, that is not necessary; when nothing is stated about aux, the default value is '-'. It should be noted that if for an attribute only this kind of constraint is used, it overrides the open world assumption.

8.7 Conclusions

In this chapter, we have tried to show that LFG is a strong computational formalism for obtaining grammatical relations from a sentence. It has been especially designed for a positional language such as English which uses positions or word order to encode theta relations as well as topicalization. By using grammatical relations (in f-structure), it is able to extract appropriate information from an English sentence.

Further Reading

Bresnan (1982) contains several papers on LFG from the linguistic viewpoint. Kaplan and Bresnan (1982) introduces the formal LFG system. Material in this chapter first appeared as part of a tutorial (Bharati et al., 1992b). See Ramesh and Sangal (1989), Block and Haugeneder (1988) for a discussion on implementation aspects of LFG.

Generalized Phrase Structure Grammar (GPSG) mentioned in this chapter is another well known grammar formalism proposed by Gazdar et al. (1985). Lately, Head-driven Phrase Structure Grammar (HPSG) has become popular in Europe (Pollard et al. (1985), (1987), Copeland et al. (1991)).

Exercises

8.1 Show the c-structure and f-structure for the following sentences

(a) The farmer saw a big fireball in the sky.

(b) The boy went to the school.

(c) The child climbed a tree.

(d) Divers searched the ocean bed for pearls.

8.2
(a) Show lexical entries for 'went' and 'searched' to handle the above.
(b) To handle the following sentences as well, what will you have to do?

(i) The boy went home.

(ii) Divers searched for pearls on the ocean bed.

8.3 Consider the following sentences:

(a) We want cakes.

(b) *Cakes are wanted by us.

(c) Police wants you.

(d) You are wanted by the police.

State what special propreties of LFG grammar can make (a),(c) and (d) acceptable whereas (b) is rejected.

8.4
(a) Show the c-structure and f-structure for:

The secretary persuaded the boss to give her a long leave.

(b) Show the lexical entry for 'persuade' so that the implicit subject of 'give' is identified.
(c) What is the difference between lexical entry for 'promise' and 'persuade'

8.5 The following sentences are ungrammatical. How does LFG account for ungrammatility in each case.

(a) Ram promised Sita go.

(b) Ram promised.

(c) Ram promised that Sita to go.

8.6 Write LFG grammar to handle the following kinds of sentences:

(a) A expected to go.

(b) A expected B to go.

Also show the lexicon entry for 'expect'. (While writing grammar you should keep in mind other sentence types in English or simply begin from the grammar given in the chapter and modify it to handle the above type of sentences.)

8.7 Show the c-structure and f-structure for the following sentences. Also mark the nodes in c-structure where controller and controllee occur.

(a) A believed that B wondered what did the girl give to the baby.

(b) The girl wondered who Ram believed Sita claimed that the baby saw.

8.8 Analyze the following sentences and show their c-structure and f-structure. What is the modified grammar that is needed.

(a) Who did Ram give a book to?

(b) To whom did Ram give a book?

(c) The girl wondered who I believed that the baby saw?

(d) The girl wondered who the baby persuaded the boy to see.

8.9 How will you handle the following sentences:

(a) The gardener believes that the boys plucked the fruits.

(b) It is believed that the boys plucked the fruits.

(c) The boys are believed to have plucked the fruits. (Ignore "have". Assume that the sentence is OK without it.)

Build the necessary grammar rules and lexical entries.

Chapter 9

LFG and Indian Languages

A grammar in LFG formalism has two basic components: an underlying context free grammar and the associated functional specification. If we try to use LFG formalism to write a grammar for Indian languages (or any free-word order languages), the problems begin with CFG and go all the way to functional specification.

9.1 CFG and Indian languages

Indian languages are basically free word-order languages whereas CFG formalism is designed to handle order or position elegantly. As a result, it is a misfit.

Let us understand the problem through examples. We know that for simple sentences in Hindi, the order of occurrence of noun groups (NGs) is unimportant.[1] Consider, for example, the following sentences:

```
(9.1)  laDake    ne    laDakii  ko    phoola  diyaa.
       boy   ergative  girl   dative flower  gave
       (The boy gave a flower to the girl.)

(9.2)  laDakii   ko    laDake  ne    phoola  diyaa.
(9.3)  phool    laDakii   ko   laDake  ne    diyaa.
```

Even though the three noun groups occur in different order the gross meaning of the three sentences is the same. In all three, the boy is the giver,

[1] Noun groups are discussed in Chap. 4. A noun group consists optionally of adjectives followed by a noun and an optional vibhakti marker, or it contains a pronoun or a proper noun, etc.

the recipient is the girl, and the object that is given is a flower. How would these be captured by a CFG? Since CFGs express position information readily, we can write the following grammar:

(CF.1)	S	→	NG* VG
(CF.2)	NG	→	adj* n pp-marker
(CF.3)	NG	→	pronoun | proper-noun
(CF.4)	VG	→	dekhaa
(CF.5)	pp-marker	→	ne | ko | se | ϕ

Note that NG and VG (verb group) require their constituents to occur in certain order which can be expressed by CFG (rules (CF.2) to (CF.4)). In the case of a sentence, however, all that the context free rule (CF.1) expresses is that in a sentence S, noun groups (NGs) can come in any order followed by a verb group (VG). This is actually saying precious little! Nothing has been stated about the number of NGs which can occur. This is controlled by the verb in VG, but there does not appear to be any compact way to state this in CFG. Nothing has been stated about what post-position markers (pp-marker) may occur. This is determined by the verb group as discussed in Chap. 5. But again there is no compact way to state this. If we try to express this in CFG, see what happens to the resulting grammar for say 'dekha' (see).

S	→	N' ne	N' ko	N'	VG |	
		N' ko	N' ne	N'	VG |	
		...				
N'	→	{adj}* n				

For the three noun groups (N' followed by post-position marker) we have six rules. Thus, to express that *order is not important* we have had to increase the number of grammar rules.

If we take into account the fact that depending on the TAM label (tense-aspect-modality label) of the verb, the post-position marker would be different, the number of rules increases even further:

S	→	N' ne	N' ko	N'	diyaa
S	→	N' ko	N' ne	N'	diyaa
		...			
S	→	N'	N' ko	N'	detaa hei
		...			
		etc.			

One can see that CFG is not designed to handle free word-order. Two problems occur: First, to capture free word-order we have to increase the number of rules. Second, free word-order languages have a rich system of case endings. Far from capturing that richness, it leads to yet greater increase in the number of rules. Finally, what it does capture compactly in (CF.1) (that VG occurs at the end), can be captured by regular grammar

or finite state machines. The power of CFGs is not needed.

9.2 Functional Specification

Let us try to solve the problems that were faced in trying to use CFG, by using functional specification. For example, if we work with rule (CF.1), we can associate a functional specification that pulls out post-positions into the functional structure. Thus, we might have the following:

(CF.1)	S →	NG *	VG
(FS.1)		↑ (↓ pp) = ↓	↑ = ↓
(CF.2)	NG →	adj * n	pp-marker
(FS.2)			↑ pp = ↓ ppm
(CF.5)	pp-marker →	ne |	ko |
(FS.5)		↑ ppm = erg.	↑ ppm = dat.
		ko |	ϕ
		↑ ppm = acc	↑ ppm = nom

It is not difficult to see that the f-structure for the sentence (9.1) would be:

pred	" GIVE (↑ erg, ↑ nom, ↑ dat) "
erg	[pred "BOY"]
dat	[pred "GIRL"]
nom	[pred "FLOWER"]

The same f-structure would be produced for sentences (9.2), (9.3) etc. Thus, the problem of word order is taken care of using functional specification resulting in a compact grammar.

It is important, however, to look critically and see what has been achieved. All that has been done is that the post-positions have been separated and listed next to the noun groups. In fact, all this could have been done trivially in linear time using a finite state machine together with an ability to list things. The power of CFG and functional specification is not used. [2]

How would this system handle the dependence of the post-position markers of noun groups on TAM label of groups? These would be handled by creating multiple entries in the lexicon. For example, we might have the following kinds of entries in the lexicon:

diyaa (gave)

$$\uparrow pred = \text{“}GIVE(\uparrow erg, \uparrow nom, \uparrow dat)\text{”}$$

[2] We have ignored the ambiguity of 'ko' here. Each ambiguity in a post-position, marker would increase the number of possible f-structures.

detaa hei (gives)

$$\uparrow pred = \text{“}GIVE(\uparrow nom, \uparrow nom, \uparrow dat)\text{”}$$

diyaa gayaa (could be given)

$$\uparrow pred = \text{“}GIVE(\uparrow instr, \uparrow nom, \uparrow dat)\text{”}$$

Well-formedness condition on f-structure would cause all but one f-structure to be rejected. But this might turn out to be very expensive. There would be overhead of repeated construction of f-structures and then their rejection.

There would be another problematic aspect. A pointer to it appears in the lexical entry for ‘detaa hei’. There are two nominatives in a sentence with this form of the verb. How would those be handled? If a post-position can repeat, we will have to form sets and check for membership in (FS.1) rather than equality.

(FS.1′) $\quad \downarrow \in \uparrow (\downarrow \text{pp})$

Similarly, optional post-positions would introduce additional ambiguity. The solution no longer looks as neat.

The most important point is that all this could have been done much more efficiently by a much simpler grammar. The power of CFG and functional specification, that is, the power of LFG is neither needed, nor used.

Further Reading

Mohanan (1982) discusses the need for the functional level for Malayalam (a major Indian language). He rejects the need for postulating subject for it in the configurational sense. Gupta et al. (1988) discuss some issues in writing a grammar in LFG for Hindi. Sengupta (1993) extends LFG to handle Indian languages. But still the new extended LFG grammar has a much greater power than needed, that is, it accomplishes something that can be done by a much simpler grammar.

Chapter 10

Tree Adjoining Grammar

10.1 Lexicalized Grammars and Locality

There are two properties of grammars that are desirable: lexicalization and locality. It has been shown by recent work in linguistics and NLP that lexicon plays a very important role in grammar. Properties associated with lexical items are important, and must play a central role in the grammar. Lexicalization gives expression to the central role of lexicon.

Definition (Lexicalized Grammar): A grammar is said to be *lexicalized* if

- every finite structure of the grammar is associated with one or more lexical items. The associated lexical item(s) is (are) called the *anchor* of the corresponding structure, which must be realized overtly in a sentence that is derived using the structure.
- There are one or more operations for composing the structures. These operations on structures are used in deriving sentences.

A typical context free grammar consists of rules (which are referred to as finite structures in the above definition), and the operation is that of rewriting or substitution. Such a grammar will be termed as lexicalized if, and only if every, rule is associated with an anchor. LFG grammar for English that we have seen in Sec. 8.3 is not lexicalized because no lexical item is associated, for example, with the rule :

$$S \longrightarrow \quad NP \qquad VP$$

The second important property is locality. By *locality* is meant that any constraint or property in the grammar is specified over a single finite structure. Such constraints should not span over multiple finite structures. Thus, if we have agreement between subject NP and verb in a sentence,

it should be expressible in terms of a single finite structure (i.e., rule). Similarly, if one is giving semantics to the derived structures, it is given in terms of units corresponding to the finite structures used in the derivation, and their composition.

10.2 Lexicalized Tree Substitution Grammar

A Tree Substitution Grammar (TSG) consists of initial trees and the substitution operation. Some example initial trees are given in Figure 10.1. An initial tree is a tree structure having nodes labelled by non-terminal

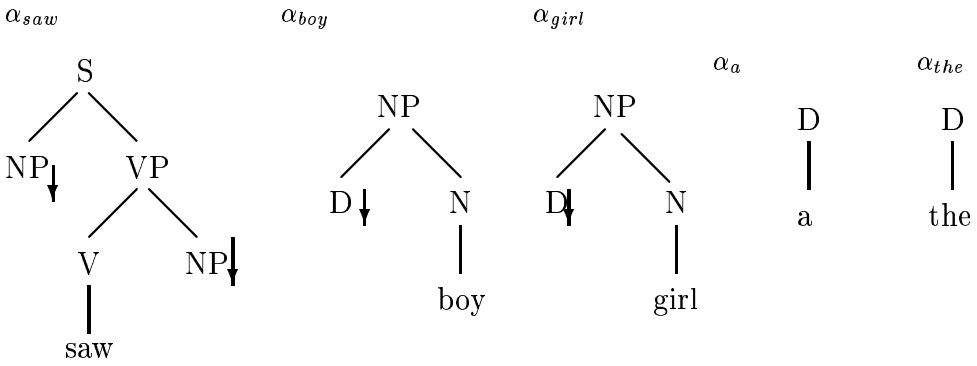


Figure 10.1: Some example initial trees

and terminal symbols but with some restrictions: All the leaf nodes of the tree are labelled by either terminals or non-terminals; in the latter case, the leaf nodes are called substitution nodes and are marked by a down arrow. In case every initial tree has at least one terminal symbol, the grammar is called lexicalized TSG.

The substitution operation allows us to derive new trees by replacing the substitution nodes in initial trees by other trees. For example, Figure 10.2 shows the result of substituting α_{the} in α_{boy}. Informally, a tree obtained by such sequence of substitutions is called a *derived* tree. A tree is called *completed* if all its leaf nodes are labelled by terminal symbols. Some initial trees are completed, for example, α_a and α_{the}. Completed trees can also be derived from initial trees by substitution operation, for example, in Figure 10.2.

Nodes in a tree can be assigned addresses. We will use Gorn (1965) addressing which assigns address 0 at the root node of a tree. Each of the k children of the root node is assigned a number from 1 to k in left to right

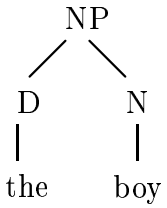


Figure 10.2: Result of substituting α_{the} in α_{boy}

order. For any other node if it is the ith child of its parent, and the parent has an address p, then its address is $p.i$. For example, The first NP in α_{saw} has the address 1, while the second NP has the address 2.2.

Now we are ready to define the substitution operation.

Definition (Substitution Operation): Substitution operation takes an (initial or a derived) tree, an address in it, and a completed tree, and replaces the node at the specified address in the first tree by the completed tree provided the following two conditions are satisfied: the addressed node is a substitution node and its label is the same as that of the root node of the completed tree.

Definition (Derived Tree): A derived tree is obtained by substituting a completed initial tree in an initial tree, or by substituting a completed initial tree or completed derived tree in an initial tree or a derived tree.

Figure 10.3 shows various substitutions necessary to obtain a derived tree for:

```
The boy saw a girl.
```

Derived tree γ_1 is the same as the tree in Figure 10.2. γ_2 is obtained by substituting the completed tree α_1 at address 1 in α_{saw}. γ_3 is obtained by substituting α_a in α_{girl} at address 1. Finally, $\gamma_4$is the result of substituting the completed tree γ_3 in the derived γ_2 tree at address 2.2.

In the example above, note that we are making use of lexicalized TAGs. Every initial tree has a lexical item. The structures of the grammar also follow the principle of locality because each structure carries its predicate arguments combination and the corresponding constraints. For example, α_{saw} is anchored on the lexical item 'saw', which specifies a dyadic predicate 'see', and the (NP) slots for its arguments. If any restrictions are placed by the predicate 'see' on its arguments, they appear as constraints on the single initial tree. Thus, constraints would not be spread over several structures of the grammar. In contrast, the following rules in CFG:

$$\text{S} \longrightarrow \text{NP VP}$$
$$\text{VP} \longrightarrow \text{V NP}$$

do not possess locality. The predicate (associated with V) appears in the second rule, whereas one of its arguments appears in the first rule. Thus,

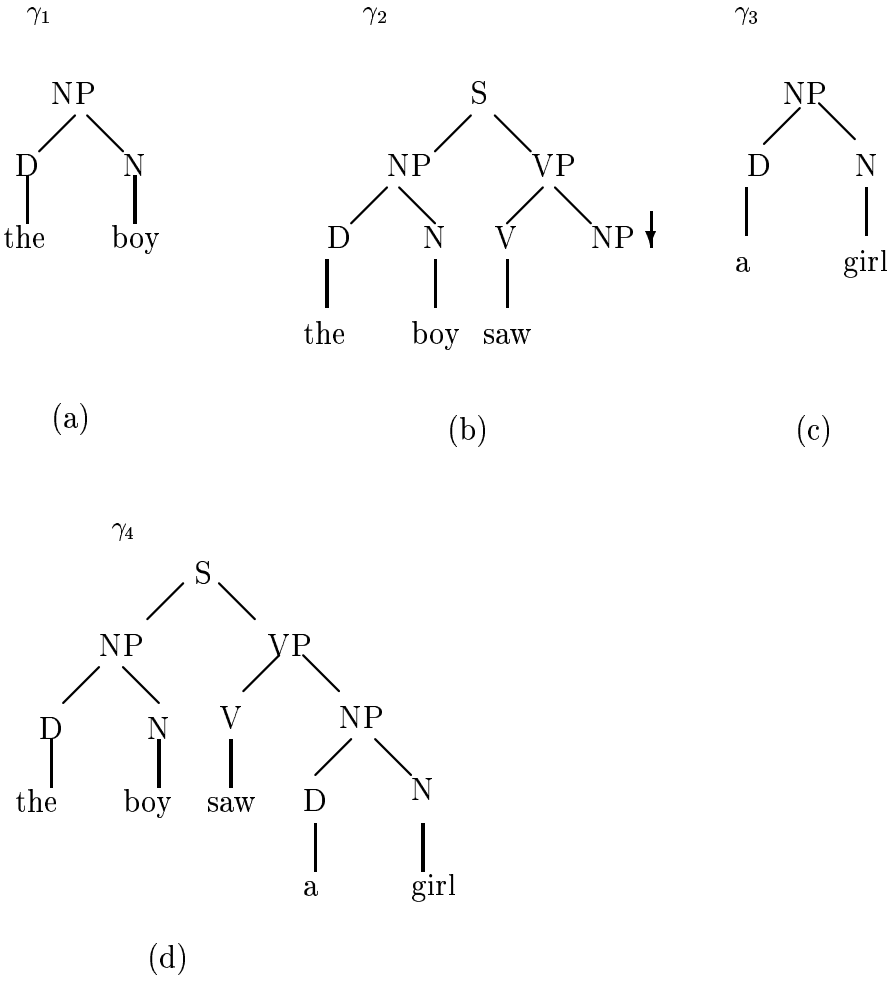


Figure 10.3: Derived tree $\gamma_4$for 'the boy saw a girl'

even though the second rule is lexicalized it does not satisfy the locality property. The first rule is neither lexicalized nor has locality.

As another example, consider an initial tree with 'kicked' as its anchor, and another initial tree with 'kicked the bucket' as its anchor, shown in Figure 10.4.

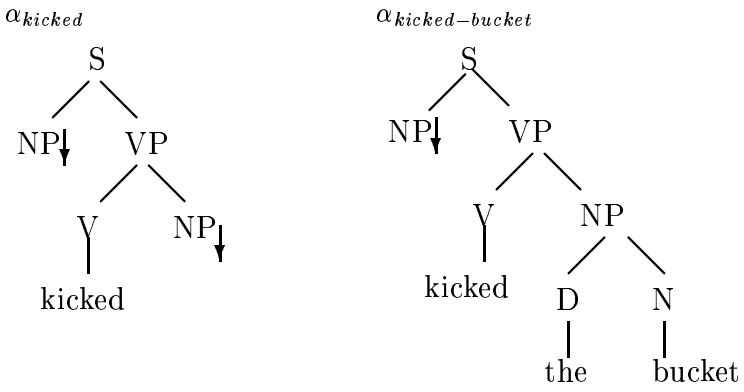


Figure 10.4: Initial trees associated with 'kicked' and 'kicked the bucket'

The predicate associated with α_{kicked} is KICKED (i.e., hit by foot), and associated with $\alpha_{kicked-bucket}$ is the predicate DIE. Again, the locality property is satisfied with these structures.

Consider the sentence :

```
The boy kicked the bucket.
```

Its derived tree is given in Figure 10.5.

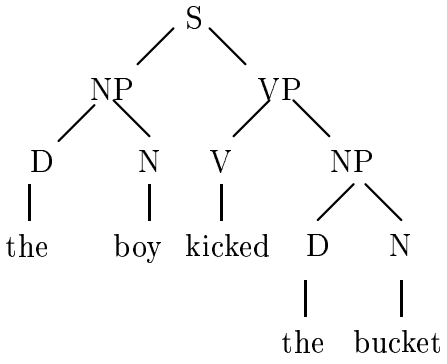


Figure 10.5: Derived tree for 'the boy kicked the bucket'

However, there are two possible derivations: one from α_{kicked} and the other from $\alpha_{kicked-bucket}$. The derived structure, therefore, does not make explicit the derivation.

There is another structure called the derivation tree which is more fundamental. It shows derivations to be performed to obtain the derived structure. For example, for the sentence 'the boy kicked the bucket' there are two derivation trees shown in Figure 10.6.

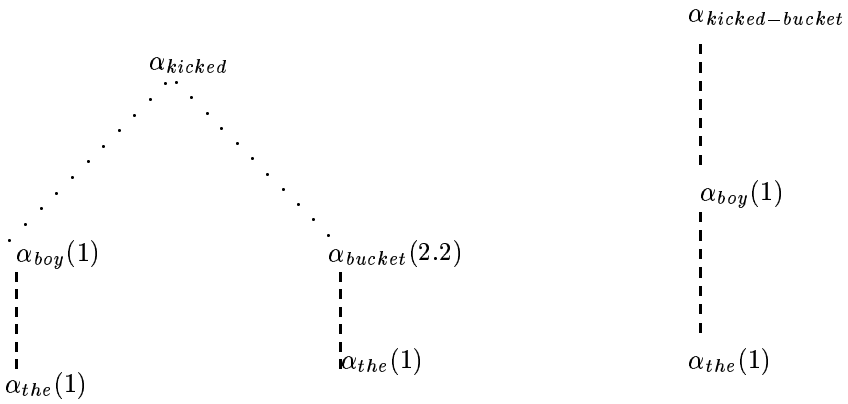
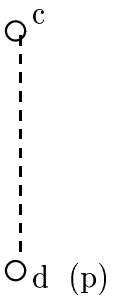


Figure 10.6: Derivation trees for 'the boy kicked the bucket'

In a derivation tree, a leaf node d is a child of a node c (connected by dotted line arc) if d is substituted in c. The address p in c where d is to be substituted to obtain derived tree is shown in parentheses next to d:



In general, if d, a child of a node c, is a non-leaf node in a derivation tree, then first a derived tree is obtained for the tree rooted at d and then it is substituted in c as before.

Thus, given a derivation tree, the derived tree can be obtained by performing the substitutions in a bottom up manner starting from the leaf nodes. However, there is no real need to obtain the derived tree. The meaning of a sentence can be constructed from the derivation tree(s). For example, semantic objects corresponding to the ambiguous sentence 'the boy kicked the bucket' can be constructed from the two derivation trees in Figure 10.6.

Here is a sample construction of the semantic object. α_{kicked} has an associated semantic form as shown:

KICKED (1, 2.2)

It says that the predicate is KICKED, its second argument is what gets substituted at address 2.2, and so on for first argument. Similarly, α_{boy} has the semantic form BOY, and α_{bucket} has the semantic form BUCKET. From the first derivation tree we get (by substituting at 1 and 2.2 in 'KICKED (1, 2.2)):

KICKED (BOY, BUCKET)

(Here, α_{the} and α_a have been ignored for simplicity.) The semantic form associated with $\alpha_{kicked-bucket}$ is DIE(1) and the second derivation tree yields:

DIE (BOY).

10.3 Lexicalized Tree Adjoining Grammar

TSG is not powerful enough to handle the so called 'wh-movement' in English. For this purpose, we will need to augment TSG with additional structures and operation. The augmented grammar is Tree Adjoining Grammar (TAG).

TAG consists of initial trees and auxiliary trees, and two operations substitution and adjoining (or adjunction). Initial trees and substitution operation have already been defined as part of TSG. *Auxiliary* trees have leaf nodes labelled by terminal symbols and nonterminal symbols; exactly one of the leaf nodes with a non-terminal label same as the root of the auxiliary tree is the *foot node* (marked by '*') and all other leaf nodes with non-terminal labels are substitution nodes (marked by '↓'). If every auxiliary tree (besides initial trees) has at least one lexical item (i.e., terminal symbol) at a leaf node, the grammar is called lexicalised TAG.

Adjoining operation takes a tree and an auxiliary tree and performs an operation as shown in Figure 10.7. In other words, the auxiliary tree

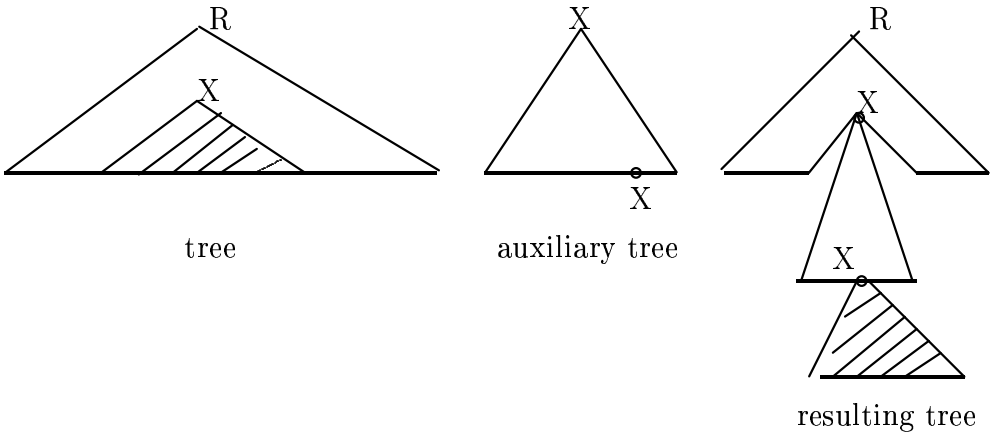


Figure 10.7: (a) Pictorial rendering of adjoining operation

at a non-terminal node is inserted in the tree. For example, if we have

an auxilary tree for 'quickly'; adjoining operation can be performed on the derived tree for 'He spoke' resulting in a tree for 'He spoke quickly' as shown in Figure 10.8.

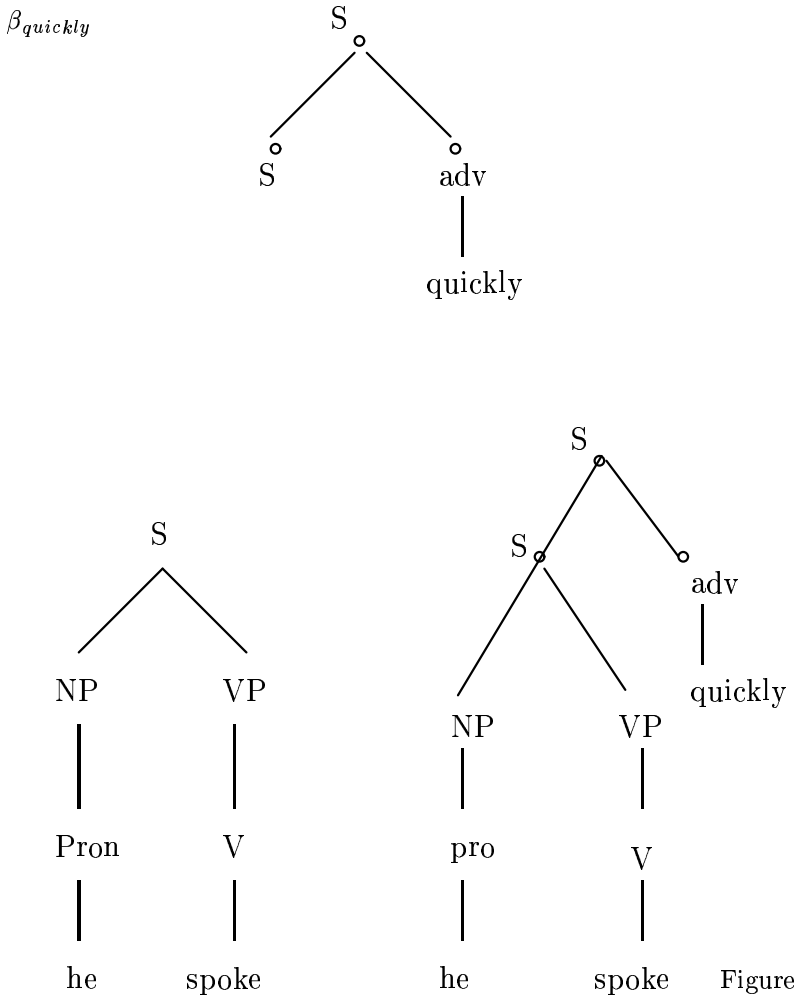


Figure 10.7(b) A derived tree and after adjoining with $\beta_{quickly}$

Definition (Adjoining Operation): To perform adjoining operation on a tree t at address p using a tree u with a footnode (see Fig. 10.7(a)), remove the subtree s of t rooted at p leaving a copy of the root node at p, substitute u with the node at address p in t, and finally substitute s at the footnode of u. Adjoining is disallowed if node at address p in t is a substitution node. Also clearly, this operation can be carried out only if the

Figure 10.8: Arc in derivation tree

label of the node at address p in t is the same as the label of the root node of u. The adjoining operation can be shown by an arc in the derivation tree in Fig. 10.7(b). A *completed tree* is now defined as one which has no substitution node. Thus, an auxiliary tree with no substitution node is a completed tree (even though it does have a non-terminal at its foot node).

Definition (Derived Tree): A derived tree is any of the following:

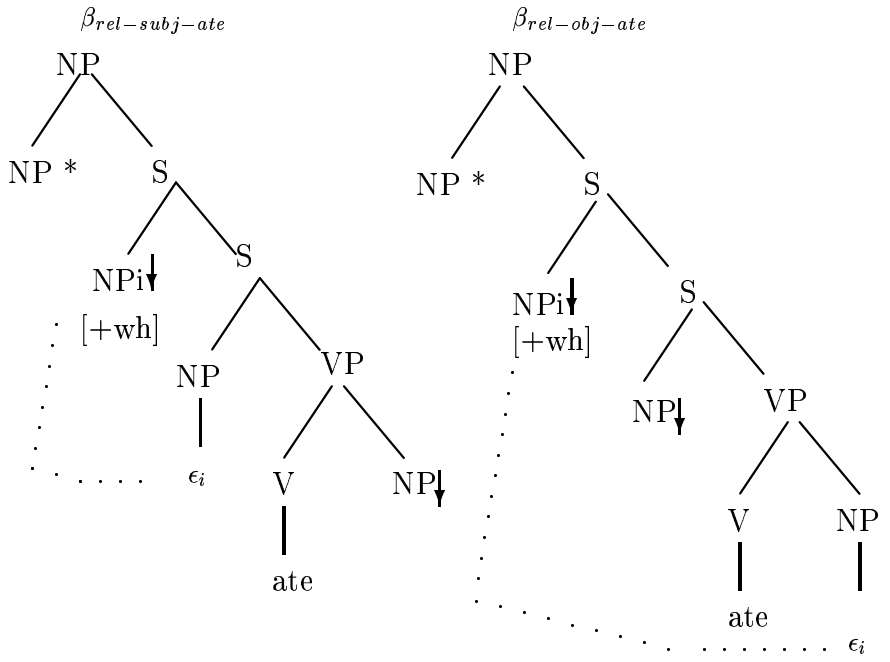
1. An initial tree or an auxiliary tree.
2. A tree obtained by substituting a completed derived tree (without a footnode) in an initial tree.
3. A tree obtained by adjoining a completed derived tree (with a foot node) in a completed derived tree.

As an example, consider the auxiliary trees for relative clause for 'ate' in Figure 10.9. The first is a relative clause with subject relativization and the second is with object relativization. The NP dominating the empty string (ϵ) indicates that the argument of V is not in its position, and coindexing ϵ and wh-NP by i (or shown by linking by dotted line) indicates that they are the same elements. The rest of the tree is a familiar declarative structure. Figure 10.10 shows the derived tree for 'The boy who ate a banana saw a girl'. It is obtained by substituting α_a in α_{banana}, then the result in $\beta_{rel-subj-ate}$. This completed auxiliary tree is adjoined in γ_1 (for 'the boy') at address 0, which is then substituted in α_{S-decl}. The final resultant tree is called γ_5.

The derivation trees for γ_4 and γ_5 are given in Figure 10.11.

It should be emphasized that the auxiliary trees are lexicalized and have locality. Auxiliary trees for ggrelative clause for verb 'ate' have both arguments of the predicate for ate in the structure.

Consider now an auxiliary tree for 'thought' in Figure 10.12.



(a) Subject relativization (b) Object relativization

Figure 10.9: Auxiliary trees for relative clause

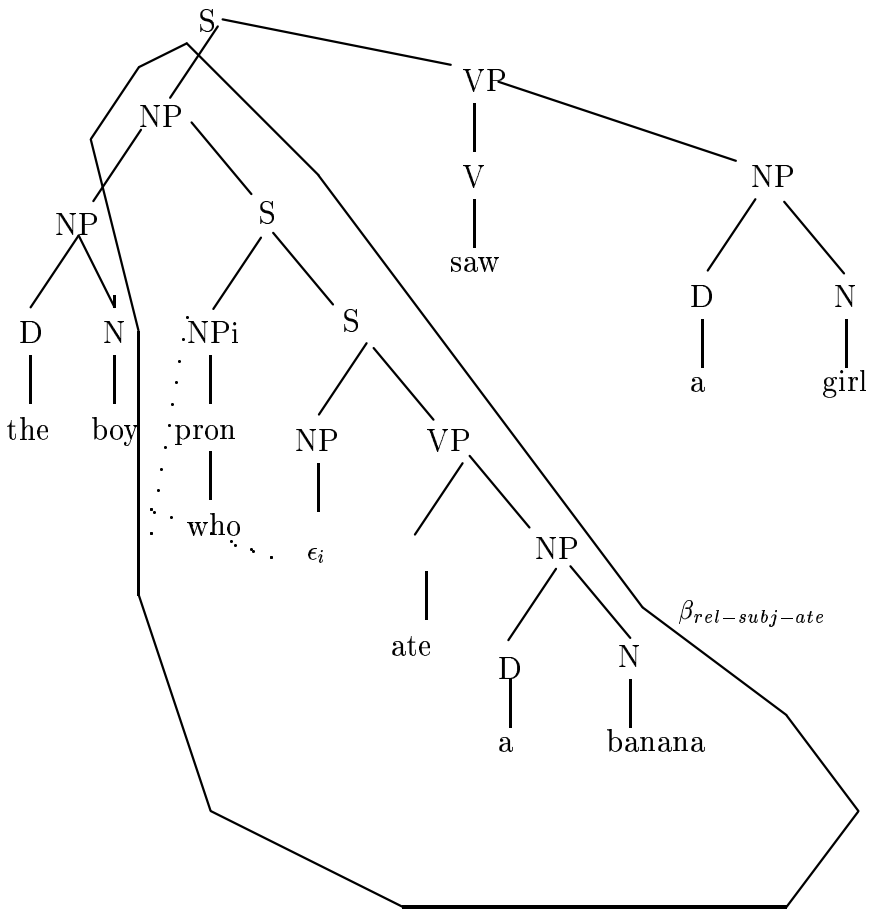
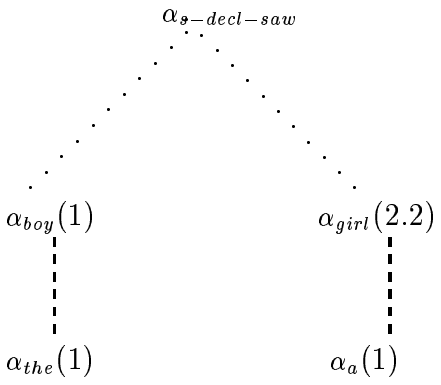
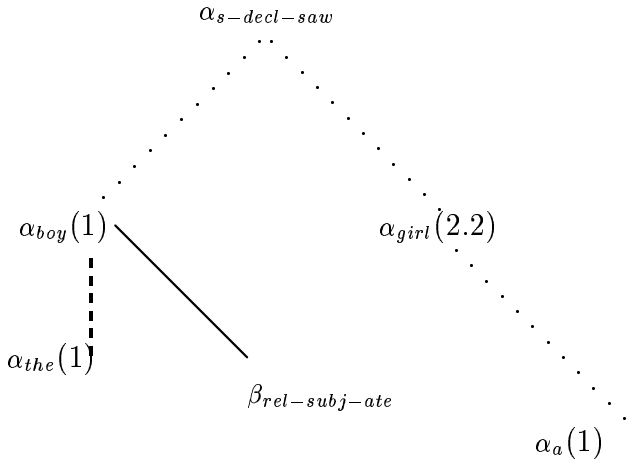


Figure 10.10: Result of adjoining an auxiliary tree on γ_4



(a) for γ_4



(b) for γ_5

Figure 10.11: Derivation trees for γ_4 and γ_5

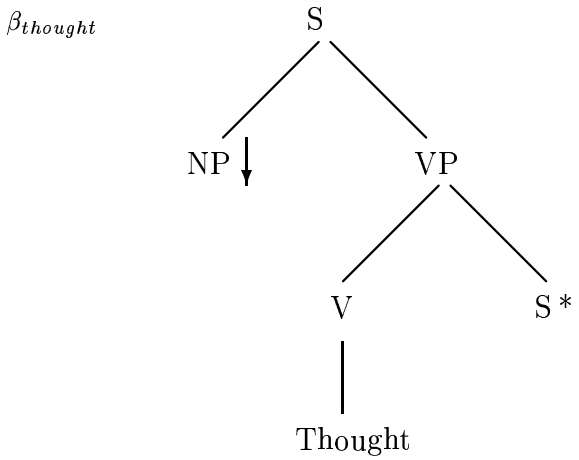


Figure 10.12: Auxiliary tree for 'thought'.

If $\beta_{thought}$ is adjoined in $\beta_{rel-subj-ate}$ at address 2.2 before it is adjoined to γ_4 at 1 we get the derived tree γ_6 shown in Figure 10.13 for the sentence

```
The boy who Ram thought ate a banana  saw a girl.
```

Note that even though 'who' in γ_6 has apparently moved away from ϵ when compared with γ_5, this has been achieved without any movement rule. The long-distance dependency is simply a consequence of inserting some lexical items by means of the adjoining operation. The important point is that auxiliary trees for the relative clause have locality, and yet it does not affect long-distance dependency. It can thus be argued that TAGs have an extended domain of locality of just the right amount.

10.4 Feature Structures

First, we discuss how feature structures can be used with TSG to handle agreement. Later we will extend the notion to TAG.

To express that gender, number and person etc. of the verb must agree with the subject NP, we introduce constraints. These are expressed by structures which look like feature structures but have variables.[1] If the same variable name occurs at two places, it indicates that the same value must occur at both places.

Figure 10.14 shows the agreement constraint on an initial tree α_{S-decl}. It says that the value of the feature agr at V, VP and NP is the same as

[1] Feature structures and their unification has been discussed along with LFG in Sec. 8.6.

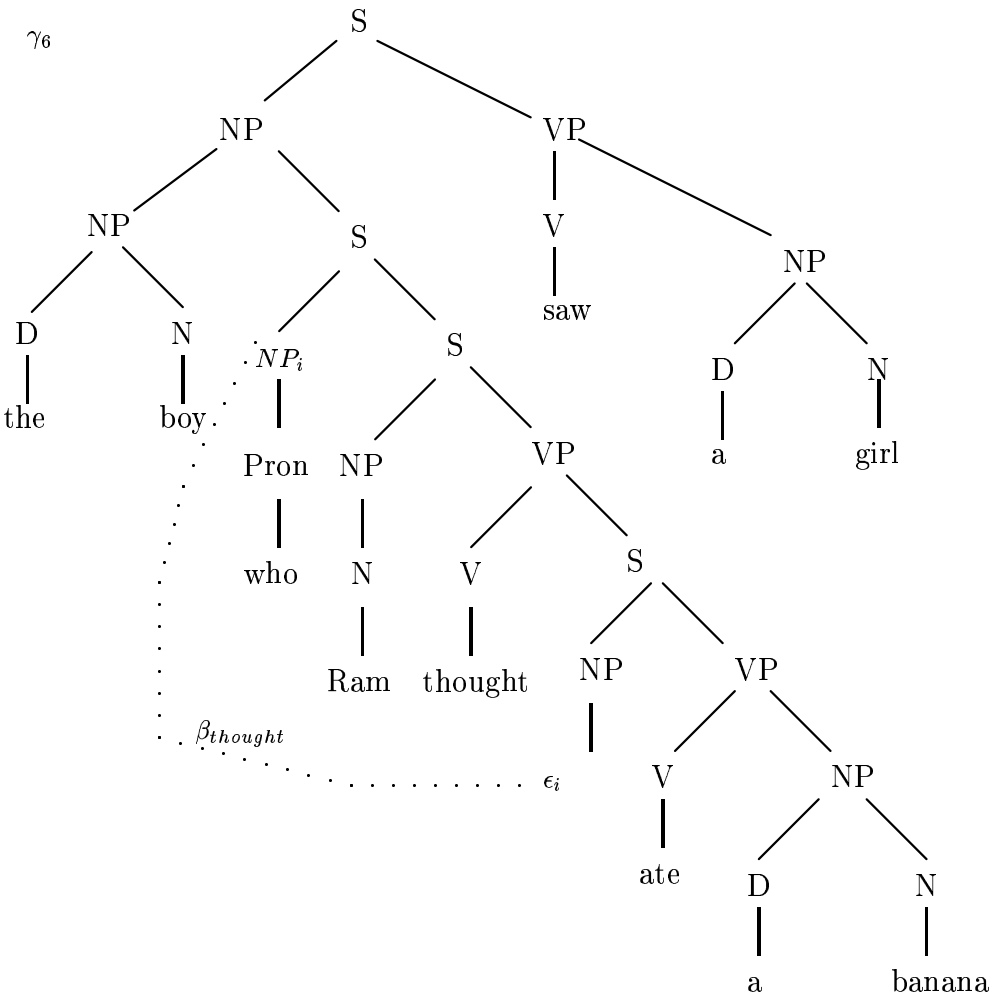


Figure 10.13: After adjoining $\beta_{thought}$ in $\beta_{rel-subj-ate}$, and the result in γ_4 we get γ_6

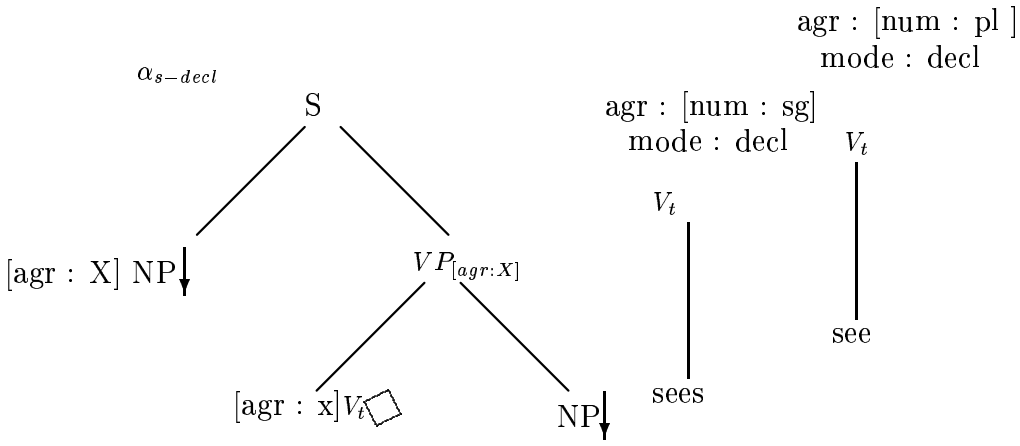


Figure 10.14: Feature constraints for agreement

variable X, where X can take any value. (Note that the initial tree α_{S-decl} is anchored on a transitive verb. The anchor is not shown explicitly but its pre-terminal V_t is marked by a diamond. The anchors 'sees' and 'see' are shown separately with V_t as root node having feature singular and plural, respectively. The use of the diamond mark does not affect the power of the grammar but is only a convenience device.)

When a substitution operation is performed, the substitution can be performed only if the features unify. Thus, if we attach the tree for 'see' at the diamond node (like a substitution operation) the feature structures unify, and X gets bound to:

$$X = [num : pl]$$

Now, if we try to generate the sentence:

```
* He see a movie every day.
```

by substituting the following tree:

```
   NP     [agr : [num : sg]]
   |
pronoun [agr : [num : sg]]
   |
   HE
```

in α_{S-decl} at address 1, the substitution fails (or is blocked) because the unification of X with value [num : pl] fails with [num : sg].

In feature-based TAGs, two structures (called top and bottom) are attached to each of the root node and foot node in an auxiliary tree. Similarly,

two structures are attached at every node in a tree at which adjunction might take place. Figure 10.15 shows how adjunction and substitution affect the structures.

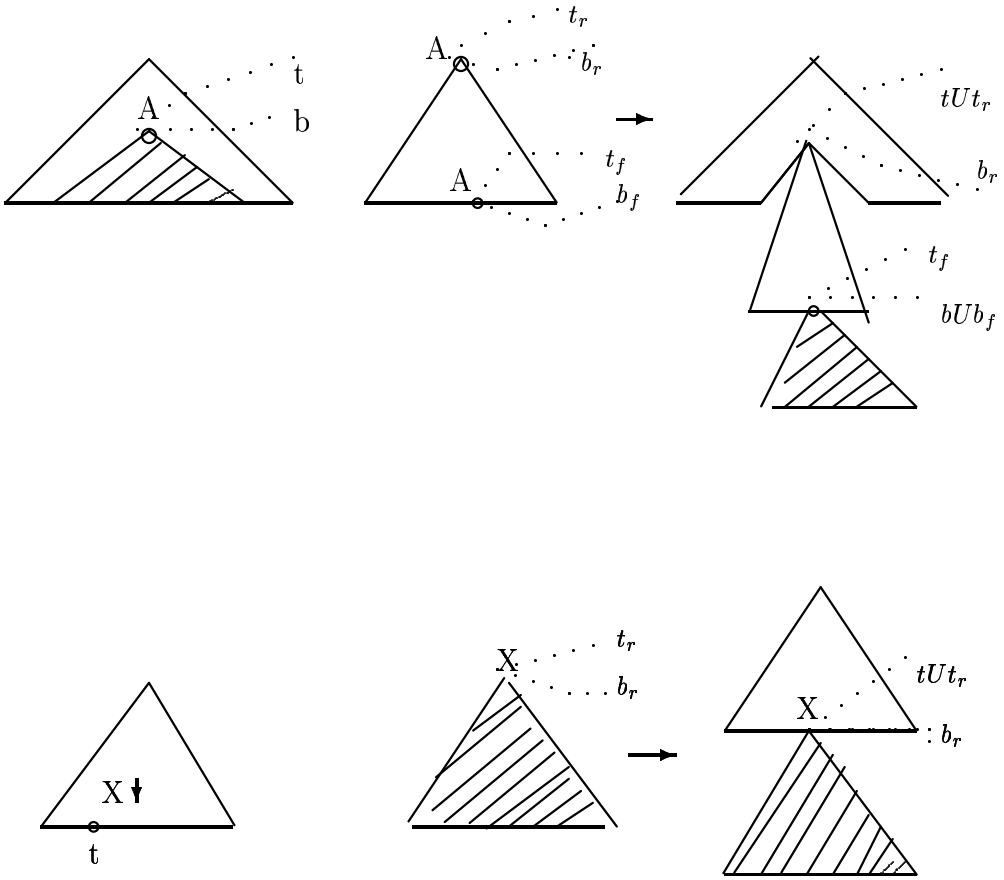


Figure 10.15: Structures after adjoining and substitution operations (U stands for unification)

Note that an adjoining operation may be performed at an adjoining node if its top structure t unifies with the top structure t_r of the root, and bottom structure b unifies with the bottom structure b_f of the foot node.

Finally, after the derived tree is obtained, the top and bottom structures at every node are unified. If unification fails at a node, the derivation is rejected. Further adjunction(s) may be performed at the nodes at which

unification failed, leading to a new derived tree. The unification of top and bottom structures would be carried out to check whether the new derived tree is accepeable.

As an example, consider agreement between agr of subject NP and VP, and modality when adjoining operation is present. In Figure 10.16, variables W, X, Y, and Z as values of features, specify constraints. Note

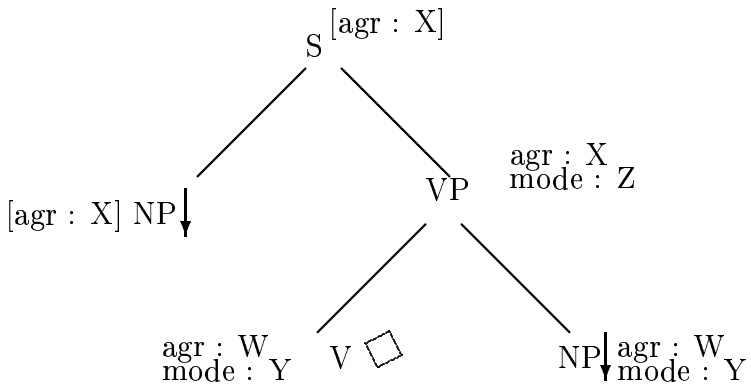


Figure 10.16: Feature constraints for agreement and modality

that at node VP the value of agr in bottom structure and top structure, is W and X, respectively because adjunction at VP node may give different values to W and X.

To generate the sentence

```
The boy could see a girl.
```

the auxiliary tree for β_{could} needs to be adjoined to t_1 resulting in t_2 (Figure 10.17). In tree t_2, since the final structure has been reached, the top and bottom structures in each of the VP node are unified. Note that in the final structure, the two VP nodes have different number and modality. The lower VP node gets its value from 'see' while the higher one gets its value from 'could'.

10.5 Some Mathematical Aspects

We have emphasized the notions of lexicalization and locality. Some readers might wonder whether a given CFG G1 can be replaced by a lexicalized CFG. G1 can always be converted to a weakly equivalent lexicalized grammar G2 (in Greibach Normal Form), however, there might be no strongly equivalent lexicalized CFG (Schabes, 1990). As a result, the parse structures for sentences would become different and linguistically unmotivated.

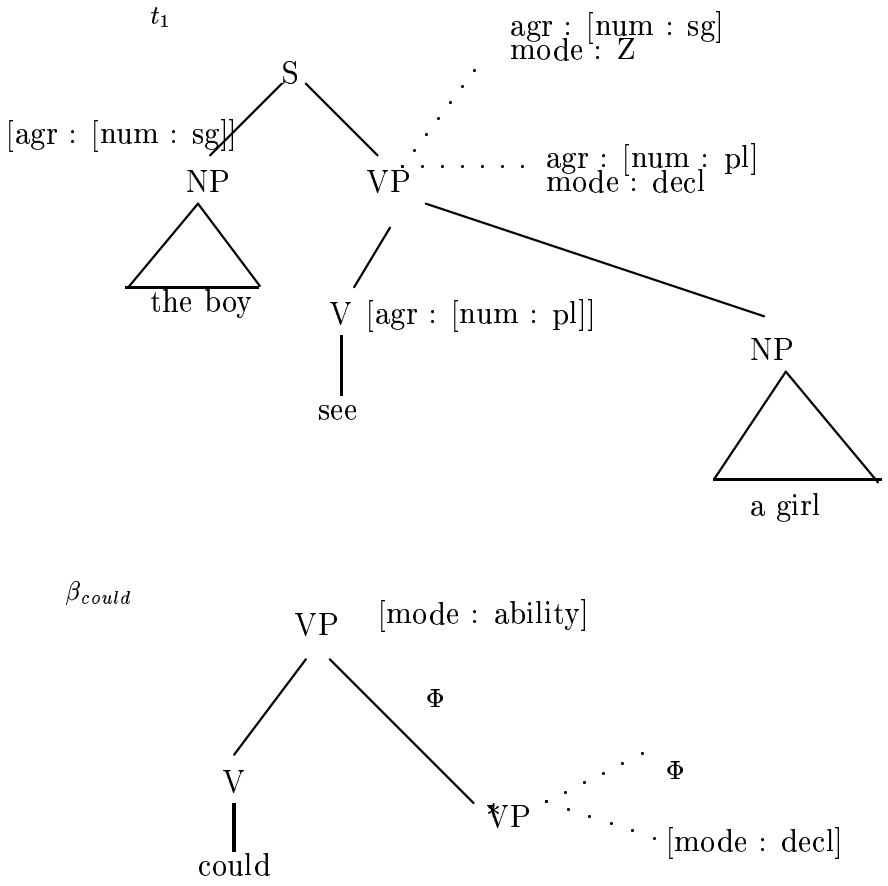


Figure 10.17 (a): Some initial and auxiliary trees with feature constraints

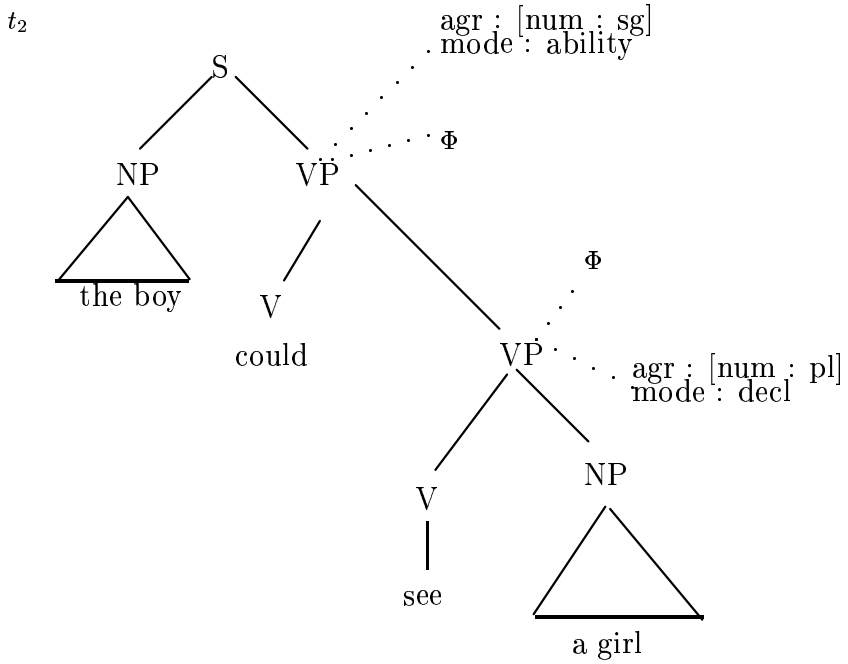


Figure 10.17 (b): The final derived tree t_2 by adjoining β_{could} in t_1

Figure 10.17: Adjoining with feature constraints

In case of TAGs, it has been shown that a strongly equivalent lexicalized TAG exists for any given TAG grammar. Thus, lexicalized TAGs have the same power as TAGs.

TAGs are more powerful than CFGs. It is known that CFGs do not have adequate power to handle some natural language phenomena. TAGs define a class of languages called mildly context sensitive because they fall in-between CFG and Context Sensitive Grammar (CSG). It has been forcefully argued that TAGs have just the right amount of power to capture and explain language phenomena. Dutch ggcross serial dependencies, for example, can be handled by it.

Moreover, TAGs have an extended domain of locality when compared to other frameworks having greater power (e.g, LFG). Long distance dependencies of wh-elements with empty elements in English, for example, occur within a single grammer structure in TAGs, normally auxiliary trees. There is also no need to talk about unrestrained movement. Therefore, TAGs have superior locality property than other more powerful grammar formalisms, and they are able to account for language phenomena without resorting to unrestrained movement. They may also have greater psychological validity.

Further Reading

A large amount of literature is available on TAGs. See for example Joshi (1985), (1987), Vijay-Shankar and Joshi (1991).

The Greibach normal form is described in any book on formal grammars such as Hopcroft and Ullamn (1979). See Schabes (1990) for a discussion on lexicalization of CFG.

Exercises

10.1 Define initial trees to handle passives in the following sentences:

```
A girl was seen.
A girl was seen by the boy.
```

10.2 Show initial trees for the verb 'give'. You should be able to derive the following sentences:

```
The boy gave the girl a flower.
The boy gave a flower to the girl.
A flower was given to the girl by the boy.
A girl was given a flower by the boy.
```

Show the derivation trees.

10.3 Handle the following sentences in TSG by defining suitable initial trees:

```
It is raining.
He is an engineer.
The table is brown.
```

10.4 Show the derived and derivation trees for the following sentences :

```
The boy who Ram beat saw a girl.
The boy saw a girl who beat Ram.
The boy saw a banana which Ram bought.
```

Create any initial and auxiliary trees that you need.

10.5 How will you handle wh-questions? Build the requisite initial and auxiliary trees needed to handle the following sentences :

```
Who gave Sita a book?
What did Ram give Sita?
Whom did Ram give a book?
```

Show derived and derivation trees. (Ignore 'did' for the time being.)

10.6 Show derivation trees for the following sentences :

```
The boy who Mohan thought Ram beat saw a girl.
The boy who Mohan thought beat Ram saw a girl.
The boy saw a banana which Mohan thought Ram bought.
The boy saw a banana which Mohan thought Shyam said Ram
    bought.
```

10.7 Show derived and derivation trees for the following sentences:

```
Who did Mohan believe gave the book to Sita?
Who did Mohan believe Shyam said gave the book to Sita?
```

10.8 How can tough-movement be handled? Give your solution for the following sentences:

```
It is tough to believe Ram beat Mohan.
Ram is tough to believe beat Mohan.
```

10.9 Show derived and derivation trees. (Ignore 'did' for the time being.)

```
Who did the boy who ate a banana give the book?
Whom did the boy who Mohan believed ate a banana give
     the book?
Whom did the boy who Mohan believed ate a banana did
     Shyam believe gave the book?
```

10.10 Use feature constraints to generate the good sentences and block the bad sentences of the kind given below.

```
  They have gone to the city.
* He have gone to the city.
* He gone to the city
  They could have gone to the city.
  He could have gone to the city.
* He could has gone to the city
  He has gone to the city
```

10.11 Extract modality information in the good sentences in the previous exercise.

10.12 Handle number agreement in TAG between 'boy' and 'know' in the following phrases:

```
  the boy who knows swimming
* the boys who knows swimming
  the boy who they say knows swimming
* the boys who they say knows swimming
```

10.13 Handle number agreement in TAG between 'Ram' and 'run' below.

```
  Ram is tough to believe runs a mile
* Ram is tough to believe run a mile
* Ram are tough to believe runs a mile
```

Chapter 11

Comparing TAG with PG

There are remarkable similarities between the tree adjoining grammar (TAG) and the Paninian grammar (PG). Derivation trees in TAG and modifier-modified trees in PG have a one-to-one correspondence.

The differences relate to handling of optional arguments and sentential arguments of verbs, and long distance dependencies. For Indian languages which do not have long distance dependencies, PG should perform better than TAG computationally because of the price paid for adjunction.

Finally, it should be noted that the existing TAG and PG are both lexicalized and have locality. PG has superior locality because optional arguments and sentential arguments of verbs are a part of its karaka chart.

11.1 Introduction

There are remarkable similarities between the tree adjoining grammar (TAG) and the Paninian grammar (PG). Even though the two formalisms were developed in two different traditions (Western and Indian, respectively) for two different languages originally (English and Sanskrit), they both have similar structures. Here, we will look at the basic similarities and also some differences.

11.2 Similarities Between TAG and PG

TAG considers derivation trees to be important rather than the derived trees (Joshi, 1985). PG considers modifier-modified tree to be the important structure (see Chap. 5). There is a direct correspondence between derivation trees on the one hand and modifier-modified trees on the other. As an example, the TAG derivation tree for the sentence:

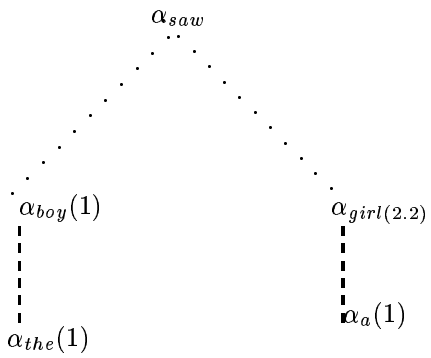


Figure 11.1: A TAG derivation tree.

```
The boy saw a girl.
```

is shown in Figure 11.1.

Similarly, the modifier-modified tree in PG for the following sentence in Hindi:

```
usa ladake ne eka ladakii ko dekhaa.
that boy ergative a girl accusative saw
(That boy saw a girl.)
```

is shown in Figure 11.2.

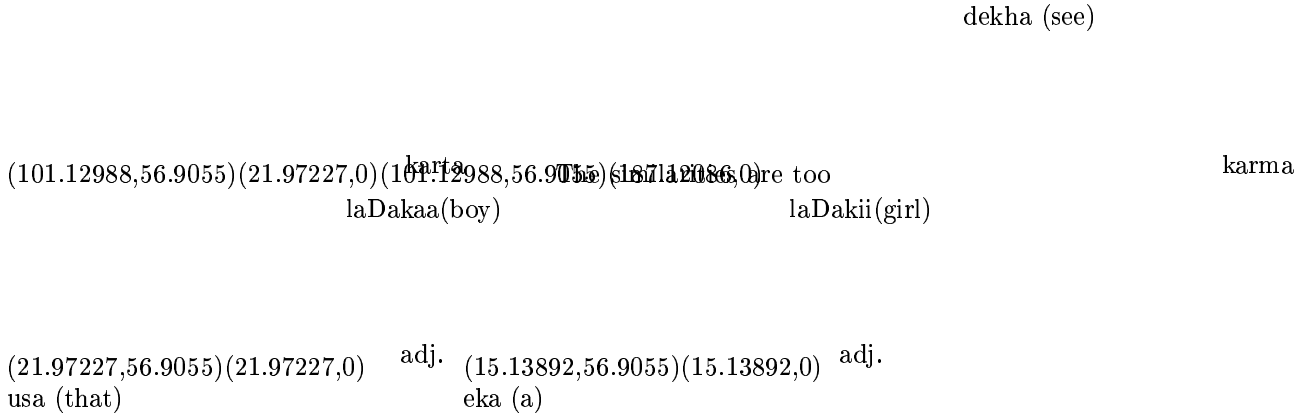


Figure 11.2: A PG modifier-modified tree

obvious to be missed. Note also the correspondence between tree addresses in elementary trees of TAG (e.g., address (1) and (2.2) in α_{saw}) and the karaka label of demand word in PG (e.g., karta and karma of dekhaa (saw)).

Addresses where substitution takes place in an elementary tree indicate its arguments, similarly the karakas indicate the "arguments" of a verb demand word. For example, address 1 of α_{saw}, a formal object at this level of analysis, can be mapped to an appropriate theta role such as agent at the next level of analysis. In the case of PG, the label karta, again a formal object at this level of analysis, can be mapped to agent theta role at the next level of analysis. [1]

It should be noted, however, that the tree addresses carry information about word order (e.g., address (1) occurs before address (2.2)) whereas the karakas do not. Karakas, in fact, relate to vibhakti [2] for free word order languages, but that is part of the karaka charts and vibhakti does not explicitly appear in the modifier-modified tree. Since vibhakti has been used and abstracted away in obtaining the modifier-modified tree, it is only appropriate that it be so. Tree addresses in TAG, while performing similar abstraction, continue to have order information as a left-over. (This does not cause any problem, however.)

If one compares the elementary trees (i.e., initial trees and auxiliary trees) in TAG with karaka charts in PG, one again finds a similarity. An

[1] There is a difference here in that TAG would not be obliged to defend similarity in meaning of address 1 when it occurs in two different trees. It would simply be a tree address that occurs in two different trees! PG on the other hand would be obliged to, and does, defend the use of the same label karta in two different trees.

[2] Vibhakti stands collectively for case endings, postpositions and prepositions, or a combination thereof, depending on the language.

initial tree for a verb anchor, (or a demand word) is like a karaka chart (or a demand chart). The initial tree for a verb specifies (along with initial trees for noun etc.) how the nouns "fit" in with the verb anchor. This "fitting" is primarily in terms of word order. The karaka chart in PG, on the other hand, specifies what vibhaktis must occur with the nouns, for them to "fit" in with the verb. This fitting is in terms of constraints on vibhaktis. All this is perfectly reasonable: Position works for positional languages and vibhakti works for free word order languages. With the recent proposed extensions to TAG by Vijay-Shankar (1992) and Rogers and Vijay-Shankar (1992) and (1993), the concept of quasi-trees comes even closer to karaka charts because the notion of tree is made more free and constraints are introduced.

These basic similarities occur inspite of the fact that the formalisms were developed independently and in two extremely different scenarios. This suggests that the similarities are a sign of a discovery of some deep underlying principles of language.

11.3 Differences between TAG and PG

The differences arise mainly because TAG uses adjunction whereas PG uses karaka chart constraint satisfaction to handle the same language phenomena. In particular, we look at the handling of optional arguments, sentential arguments, and long distance dependency.

11.3.1 Optional Arguments

Verbs take optional "arguments" which in TAGs are handled by the adjoining operation. Thus, to generate the sentence:

```
The  boy  saw a girl by the river
```

the optional argument of location ('by the river') is inserted by the adjoining operation. PG handles optional arguments by karaka charts and vibhakti constraint satisfaction, the same mechanism that is used for mandatory arguments. This has a better locality property where we use the same definition of locality that is used by research workers in TAG (Schabes,1990). Various constraints on feature structures of optional orguments, such as selectional restrictions, follow from the verb. In TAG, they become non-local whereas in PG they remain local to the karaka chart for the verb.

11.3.2 Sentential or Verbal Arguments

Adjunction is also used in TAG to handle those arguments of a verb that take a sentence (or a verb phrase) as an argument. For example, the auxiliary tree for 'thought' (Figure 11.3) permits 'thought' to be inserted by

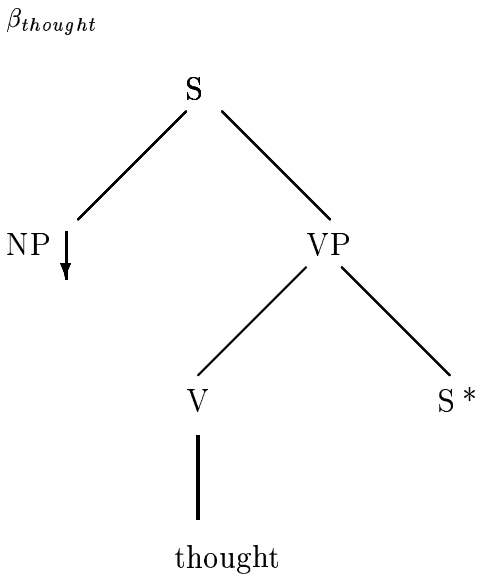


Figure 11.3: Auxiliary tree for 'thought'

adjunction in a tree for its sentential argument. This is already different from earlier use of adjoining because now the initial tree corresponds to an argument, whereas the auxiliary tree is the "predicate". PG handles verbs such as above through the karaka chart mechanism. For such verbs, the appropriate karaka is marked to have a verb as its filler. After this the normal method of karaka chart constraint satisfaction takes over.

It should be pointed out that the use of adjunction to handle the phenomena above is not without a price. The worst case asymptotic complexity result for a TAG parser is $O(n^6)$, while for the restricted karaka chart satisfaction it is $O(n^3)$ (see Chap. 6).

11.3.3 Some Important Phenomena

Finally, adjoining operation is also used to handle unbounded movement (or long distance dependency) such as movement of a wh-element in a wh-question. TAGs handle these very cleanly which is a major strength of TAG. Such movements are not known to occur in Indian languages. Hence, this is not an issue for Indian languages. (Research has been conducted on how PG can be applied to English to handle long distance dependency (Bhatt,1993) and more work needs to be done.)

The reverse question which can be asked: Can TAGs handle Indian languages? The answer given by Joshi et al. (1991) is that by relaxing the ordering constraint on children of a node in an elementary or derived tree, free word order languages can be handled. But here one might end up paying a price in efficiency because parsing of these may turn out to have the same problem as parsing with ID/LP grammars (Barton et al.,1987).

11.4 Discussion

For Indian languages which do not have long distance dependencies, PG should perform better than TAG (because of the price paid for adjunction). To handle free word order, TAGs have to relax ordering which may lead to a further price in efficiency. PG, on the other hand, utilizes the vibhakti constraints to build an efficient system. It remains an open question as to how well does PG handle long distance dependencies.

With the recent proposals to change TAG, the new quasi-trees are beginning to look even more like the karaka charts and other structures in PG.

Finally, it should be noted that the existing TAG and PG are both lexicalized and have locality. PG has superior locality because optional arguments of verbs are a part of its karaka chart.

Further Reading

Recent proposals by Vijay-Shanker and his group (Vijay-Shanker (1992), Rogers and Vijay-Shanker (1992), (1993)) are noteworthy. With the suggested changes, the new TAGs with quasi-trees begin to look even more like the Paninian grammar.

This chapter is based on Bharati et al. (1994b). There is a need to explore the application of PG to English (Bhatt, 1993). Many interesting new results are likely to come out of such an exercise.

Chapter 12

Government and Binding

Government and Binding (GB) is the dominant linguistic theory, yet it has not been very popular with computational linguists. There are two reasons for this. First, GB does not address the problem of either parsing or generation. As a result, it proposes its formalism in a form which is not amenable to computation directly. Second, GB keeps changing so much and so rapidly, that it is difficult to know what GB is at any given time and implement it. In this chapter, we sketch GB theory and give some tentative suggestions for implementing it for an English like language.

12.1 Introduction

We begin by posing a question: Why inspite of being the dominant linguistic theory, has Government and Binding (GB) been comparatively less popular with computational linguists? Basically there are two reasons:

1. GB does not address itself to either the problem of natural language parsing or of generation. Its goal is to identify the innate structure in human mind which enables a child to acquire language so effortlessly. As a result it proposes its formalism in a form for which neither existing parsing tools developed by computer scientists can readily be used, nor is it clear how new efficient parsing techniques can be designed in a straightforward manner. In other words one has to begin in some sense, ab-initio.

2. The GB theory has not yet reached a stable enough form where one would invest the effort in building a complete GB parser. It has been commented "TG is a field whose very foundations shift as remorselessly as quicksand" (Radford, 1988).

The next question that comes up naturally is: If the above reasons hold why should NLP workers invest their time and effort in studying GB? The answer is as follows:

1. If one is interested in the task of accepting only grammatical sentences then at least for languages like English where movement plays a crucial role, examples are known, for which, "there is no natural way to capture their effects in any of the well-known logic grammars or extensions of them" (Stabler, 1990). It is not being claimed here that GB has satisfactory solutions to these problems but at least a very large number of leading linguists are working on these problems in the GB framework and one would like to get the benefit of their efforts and insights.

2. Whereas many other grammatical formalism either do not address the problems of anaphora resolution and quantifier scoping or assume that they are best handled by pragmatics, GB linguists invest most of their efforts in attempting to solve these problems using the same machinery which GB has developed for handling wh-movements. Of course it must be admitted here that they address a very small fraction of the anaphora resolution problem.[1]

3. Because for implementing GB parser one has to start ab-initio, one tends to tailor the parser according to the need of the natural language, e.g., due weightage is given to the lexical properties of heads of constituents early enough in the parser along with principles for case assignment and thematic role assignment, which in turn enables one to properly address the problem of ambiguity in an appropriate manner (see Wehrli (1988)). In other words, because the theory is silent or neutral to parsing or generation, particularites of the natural language concerned can be made use of while designing a parser without coming into conflict with the theory.

Having seen the goal and the concerns of GB and why NLP researchers should look at GB, at least while working on English like languages, we will first give an overview of the GB model. In the next section we will look at GB in more detail, and in particular, at each of the principles and modules. In Section 12.3, we will suggest how GB can be used computationally for parsing English.

The basic organization of GB grammar can be seen in Figure 12.1. It has three levels of representations of a sentence: D-structure, S-structure and LF-representation. PF-representation is the sentence itself.

[1] By anaphora, linguists mean reflexive pronouns like 'himself' etc. and reciprocals such as 'each other'.

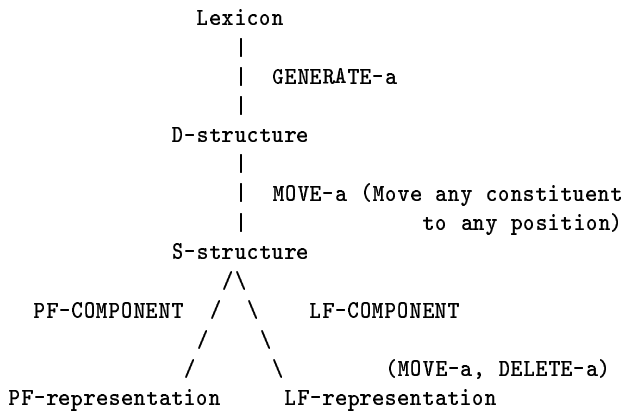


Figure 12.1: GB Model

In the GB model a crucial role is played by interacting systems of principles which are listed in Figure 12.2. These systems of principles place constraints thus filtering out ungrammatical representations.

```
X-bar theory
Thematic theory (theta  theory)
Government (Definitions)
Case theory
Bounding theory
Binding theory
Control theory
```

Figure 12.2: GB Principles

Typically, various principles have some parameters associated with them. These parameters are meant to make the grammar flexible enough to account for all the different languages.

12.2 The GB Modules

In what follows is a very brief summary of GB principles. It is based, primarily, on Duarte (1990).

12.2.1 X-bar theory

The X-bar theory gives the structure of phrases in GB. It replaces the phrase structure rules of earlier transformational grammars. It says that if there is a head Xo, it has a maximal projection termed as X. The maximal projection is a phrase optionally containing a complement and a specifier as determined by the head and also possibly adjuncts. This can be stated in terms of rule schema as follows, where curly brackets indicate the optionality of the constituent:

math

$$
\begin{array}{lll}
\text{(CF.1)} & \bar{\bar{X}} \rightarrow & \text{adjunct } \bar{\bar{X}} \\
\text{(CF.2)} & \bar{\bar{X}} \rightarrow & \bar{\bar{X}} \text{ adjunct} \\
\text{(CF.3)} & \bar{\bar{X}} \rightarrow & \text{Spec } \bar{X} \\
\text{(CF.4)} & \bar{X} \rightarrow & \text{Xo Compl} \\
\text{(CF.5)} & \bar{X} \rightarrow & \text{adjunct } \bar{X} \\
\text{(CF.6)} & \bar{X} \rightarrow & \bar{X} \text{ adjunct}
\end{array}
$$

The order of constituents on the right hand side of rules (CF.3) and (CF.4) may be language dependent, e.g., in Hindi Compl will precede Xo. This is a simple example of a choice of parameter. The order shown here is for English.

Xo is termed head and $\bar{\bar{X}}$ is its maximal projection. Xo can be a lexical category like N (noun), V (verb), A (adjective) or P (prepostion) or it can be a functional category like infl, COMP etc. Spec, Compl and adjunct must be maximal projections. Head controls the choice of Spec and Compl both semantically as well as syntactically.

Projection Principle: Lexical properties must be represented by categorial structure at every level of syntactic representation (i.e., D- and S-structure, and LF representation).

This principle which is not a part of any one of the seven systems of principles listed in Figure 12.2 ensures that lexical requirement information is present at each level of representation and, moreover, it also insists that the information must be coded in terms of configurational positions, a choice which is natural for English-like languages.

12.2.2 Theta Theory

θ-Criterion:

1. If α is an argument of β then the position P occupied by α is assigned one and only one θ-role R;

2. If β has a θ-role R to assign, then this θ-role is assigned to a position P and P is occupied by one and only one argument.

Note that in GB theory θ-roles are assigned to syntactic positions and not directly to arguments.

One of the main consequences of the θ-criterion is that constituents are allowed to move to non-thematic positions only. (This is a simple example illustrating constraints on movement imposed by system of principles.)

12.2.3 Government

Before we define government, we will define some structural relations.

Definition C-command: α *c-commands* β iff
(a) α does not dominate β, and
(b) the first branching node that dominates α dominates β.

Definition M-command: α *m-commands* β iff
(a) α does not dominate β, and
(b) every maximal projection that dominates α also dominates β.

Definition Intervene: S *intervenes* between α and β iff
S dominates α and does not dominate β.

Definition (preliminary) Government: α *governs* β iff:
(a) α is a head;
(b) α m-commands β, and
(c) there is no intervening maximal projection between α and β.

A revised definition and one we will use is as follows:

Definition Government: α *governs* β iff:
(a) α is a head;
(b) α m-commands β; and
(c) there is no intervening barrier between α and β.

Where barrier is defined as follows:

Definition Barrier: *Barrier* is a maximal projection such that

(a) it is non-θ-marked;
(b) is fully specified (i.e. its spec position is filled); and
(c) its head is a functional category.

Remark: The crucial idea behind government is to define the domain of influence for a head.

Definition Proper Government: α *properly governs* β iff

1. Either it is a lexical government, that is:
   - α governs β; and
   - α is a lexical head
2. or, it is an antecedent government, in which α governs β except it does not require α to be a head, it can be a maximal projection. However, α must be co-indexed with β.

12.2.4 Case Theory

Case Filter:

(i) Every NP with phonetic content must be case marked.
(ii) Every argument NP (distinct from PRO) must be case marked.

Case Assignment: Case is assigned by certain heads e.g.

1. verb assigns accusative case.
2. preposition assigns oblique case.
3. infl [+AGR] assigns nominative case.

Principle: Case is assigned under government.

12.2.5 Bounding theory

Subjacency: No instance of Move-α can cross more than one barrier.

12.2.6 Empty Category Principle (ECP)

Principle: All traces must be properly governed.

12.2.7 Binding theory

Definition Binding: α *binds* β iff:
(a) α c-commands β; and
(b) α and β are co-indexed

Binding Principles:

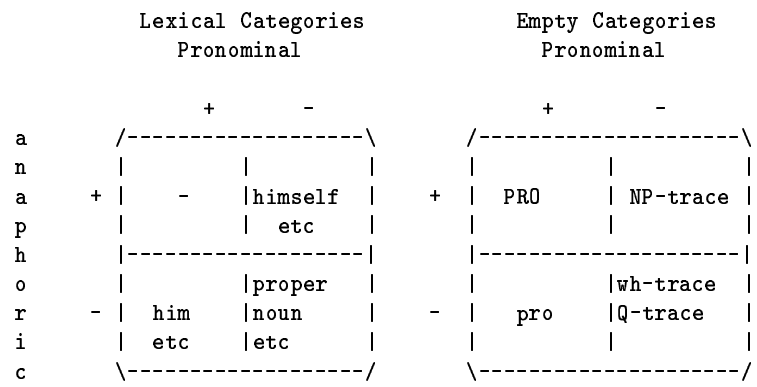
(a) An anaphore must be bound in its governing category.
(b) A pronominal must be free in its governing category.
(c) An R-expression must be free everywhere.

Note: Here, Free means it should not be bound by a potential argument position.

Definition Governing Category:
α is the *governing category* for β iff
α is the minimal maximal projection containing:
(a) β
(b) the governor of β, and
(c) a SUBJECT

Definition SUBJECT: A *SUBJECT* is:
either AGR with respect to infl,
or a subject NP with respect to NPs and small clauses.

The tables given below indicate what items are anaphore and what are pronominal. They include lexical categories as well as empty categories.



12.2.8 Constraints on movement

Based on the above principles, this is what we can say about movements (Duarte, 1990):

1. What can be moved?
   - Only heads and maximal projections are free to move.
2. Where can these be moved?

- The landing site has to be a non-thematic position.
- A head can only be moved to another head position and a maximal projection can only be moved to a maximal projection position.

3. By what process can a movement occur?
   - If the target position is an empty spec or head position then the process is of substitution else it will be by adjunction.
4. How far apart can a moved constituent and its trace be? This is answered indirectly by the following conditions:
   - Subjacency: In a single step no movement can cross more than one barrier.
   - ECP: All the traces must be properly governed.
   - Binding theory: An NP-trace must be bound in its governing category and a wh-trace (in general, a variable) must not be bound in its governing category.

12.3 How Can GB Help in Parsing?

In an English-like language where primary information about thematic roles is coded in terms of relative position and yet it also permits movements for topicalization etc., it is not difficult to see the importance of various constraints on movement which directly follow from various principles of GB theory.

To begin with, one might suggest the following steps for parsing an English like language.

1. Using X-bar system of principles to form phrasal categories on the basis of lexical heads. This step may again be broken up into the following sub-steps:
   - Recognize lexical heads.
   - Search for the longest well-formed sequence of specifiers and attach it (i.e., make it a sister) to the phrasal category dominating the lexical head.
   - Non-deterministically propose possible Xs as complements.
2. Using the θ-criterion and projection principle, identify the needed empty categories in the s-structure.

   To decide the type of empty-category one may use the facts of following kind.

- If the position is not properly governed it must be a PRO (from ECP).
- If the position has a case assigned to it, it must be a variable trace. (from non NP-movement).

3. Using binding theory (in case of PRO, using control theory) find the co-indexed phrase.

   In case of a trace, determine its governing-category, and then depending on the fact whether it is an NP-trace or a variable look for the antecedent in the governing-category or outside it, respectively.

Note that the domain of search need not cross more than one barrier.

Following are some examples from Saint-Dizier (1990) with some-what simplified S-structure which illustrate the approach.

1. **Relative Clause construction:** Take the example:

   The boy who Mohan met yesterday is coming tomorrow

   S-structure of relative clause in the above is:

   $[_{COMP}\ who_i\ Mohan\ met\ [_{NP}\ t_i]\ yesterday]$

   Note the empty category is not a PRO because it is properly governed and it is not an NP-trace because it has accusative case assigned by the verb. So it cannot be bound by an argument position co-indexing it with 'who' satisfies all the constraints.

2. **Passive Construction:**

   In case of passives, following additional assumptions are made:

   - A passivized verb cannot assign case to its object NP.
   - θ-grid requirement of a passive verb differs in the following manner from that of an active verb.
     (a) No θ-role is assigned by a passive verb to its subject-NP in d-structure
     (b) A 'by-complement' with the θ-role of the subject NP of the active verb can be optionally present at d-structure.

   Consider as an example:

   A book was thrown.

   $[[_{NP}\ a\ book]_i\ [was\ thrown\ [_{NP}\ t_i]]$

   Here, the trace is an NP-trace because it is in a position which is properly governed but does not have a case. So it must be bound in its governing category which is the sentence itself and the NP in subject position binds it appropriately.

3. **Subject-to-subject raising**

   Consider the example sentence below with its S-structure:

   Mohan seems to be on time.

   $$[_{infl}\ Mohan_i\ seems[_{COMP}\ trace_i\ [_{VP}\ to\ be\ on\ time]]]$$

   Again this is a case of NP-movement because it can not get case from an infl with -AGR feature, so it moves to a non-θ-marked position. Note 'seem' does not θ-mark its subject position. (It seems Mohan is on time.)

12.4 Conclusion

We have briefly sketched the GB theory in this chapter. Finally, we have given some tentative suggestions about how it can be implemented for an English like language. It should be mentioned that recently GB theory has been superseded by the new minimalist programme of Chomsky.

Further Reading

Duarte (1990) is an excellent summary of GB theory; we strongly recommend it to NLP people wanting to learn more about the GB theory. Chomsky (1981) is the original reference, but very difficult to read unless one is already familiar with the issues being addressed. Haegeman (1991) is a better source for reading about GB theory. Van Riemsdijk et al. (1986) and Radford (1988) are excellent and highly readable sources on grammar viewed from the perspective of Chomskyan Generative Enterprise. Of course, they take their examples mainly from English.

Journal of *Linguistic Inquiry* published by MIT Press, Cambridge is the primary source where researchers in GB publish their work.

Work on GB parsing has been published in Saint-dizier and Szpakowicz (1990), Wehrli (1988), Berwick et al. (1991), etc.

This chapter is based on Duarte (1990) and Saint-Dizier and Szpakowicz (1990). It was originally prepared for a tutorial on NLP at UNESCO 2nd Regional Workshop on Computer Processing of Asian Languages (Bharati et al., 1992b).

Chapter 13

Comparing GB with PG

It is argued in this chapter that many of the differences between GB and the Paninian approach stem from differences in their goals. While the former tries to answer why children are able to acquire language so effortlessly, the Paninian approach addresses how information is conveyed using language by a speaker to a listener.

13.1 Introduction

The problem that has intrigued generative linguists is how a human child is able to acquire natural language without any formal training simply by exposure to a miniscule amount of 'positive' language data. They postulate that there must be a universal grammar every child is born with, and which gets instantiated to a particular grammar for the language to which the child is exposed. The grammar allows the child to determine the grammaticality of sentences.

The goal of the generative enterprise is to characterize the initial state of knowledge of language that allows a human child to acquire the language so effortlessly (by his or her intimate association with a speech community). This is assumed to be distinct from other cognitive structures in the mind.

In contrast to above, the question that has intrigued Paninians is, how is it that a speaker is able to convey information to a hearer by means of natural language. How is the information that a speaker wants to convey, represented or coded in language, and how is the hearer able to extract the information. The goal of the Paninian enterprise is to construct a theory of human communication using natural language. Grammar, a part of such a theory of communication, is a system of rules that establishes a relation between what the speaker decides to say and his utterance, and similarly, the utterance and the meaning a hearer extracts form it.

The major task for the theory in generative enterprise is to correctly classify sentences as grammatical or ungrammatical. Meaning comes in through coindexing or theta assignment but at times, meaning and grammaticality may be at variance. For example, in anaphora, coindexing shows what the grammar permits; on the other hand, there are known contexts when a coindexing not permitted by the theory is accepted by native speakers. (See Zribi-Hertz (1989) for examples.) Such instances are neither interesting nor relevant for the theory according to Wasow (1979). The implication of such a position is that a parser based on the theory may reject and thus block further processing of a sentence which might be acceptable in the given context.

The main task for the Paninian theory is to assign a meaning to a sentence which is the same as that assigned by the hearer. Sub-division into separate levels is a theory internal matter as long as the meaning assignment remains the same. There is certainly no separate autonomous syntactic level postulated by the theory. In fact, the Paninian Theory views the range from the sentence to the meaning as consisting of levels of meaning, each level more refined than the previous one. The early levels, call them vyakaran levels, make minimal use of world knowledge and greater use of morphological or karaka knowledge. The later levels make greater use of world knowledge, intentions of the speaker and hearer, etc. It is important to mention however, that even the vyakaran levels use the notion of vivaksha or speaker viewpoint which has a relation to intentions or pragmatics. Figure 13.1 shows the vyakaran levels in detail, and besides these, shows only the final meaning level as perceived by the hearer/speaker.

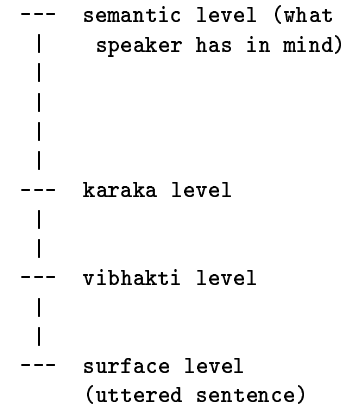


Figure 13.1: Levels in the Paninian model

The GB model, on the other hand, is shown in Figure 13.2. Here, there

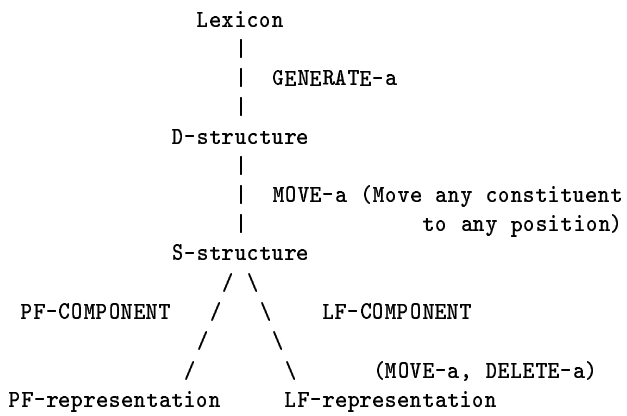


Figure 13.2: GB Model

are three levels of representation for a sentence: D-structure, S-structure and LF-representation. The PF-representation is the sentence. There is also a production component that generates representations at D-structure.

There is a significant difference between the two models which is sometimes missed in the details of levels. In the GB model, it seems that it is tacitly assumed that meaning is an 'objective' event or entity out-there in the world which is represented by means of the three levels of representations. In the Paninian model, the meaning is a mental object in the mind of the speaker relating the objective event or entity with the speaker's viewpoint it. This comes about with the way theta roles and karaka roles are conceived in the two models.

It is instructive to compare the levels across the two models. To do so it is important to identify relationships between some key categories and relations across the models. The task is harder than it seems because the terminology and the model for the generative enterprise is inspired by a positional language such as English, whereas that for the Paninian enterprise is inspired by a free word order language. In fact, incorrect equivalences between terms has been a major source of confusion while comparing the two models. First, karaka roles are different from thematic roles. For example, karta karaka gets mapped to agent, instrument, and patient respectively in sentences (1), (2) and (3), below.

(1) The boy opened the lock.
(2) The key opened the lock.
(3) The lock opened.

Karaka relations combine the notion of vivaksha or speaker viewpoint with theta roles.

Further, according to us the so-called grammatical functions subject, object etc. are at vibhakti level in the Paninian model. They have been frequently sought to be defined as a distinct level between case and theta roles. It should be noted that subject, object have arisen out of positional languages like English where they have an intuitive appeal as well as natural and simple definitions and tests. In free word order languages like Indian languages, the same information is contained in case endings or post-positional or prepositional markers. Concepts of subject, object as a distinct level are not only unintuitive but also it is very difficult to come up with criteria or tests for deciding when something is a subject. Most attempts try to define them configurationally for free word order languages, at which point the grammar building runs into serious difficulties.

If we draw a picture incorporating the above insights, we have the picture shown in Figure 13.3. It should be mentioned here that many GB

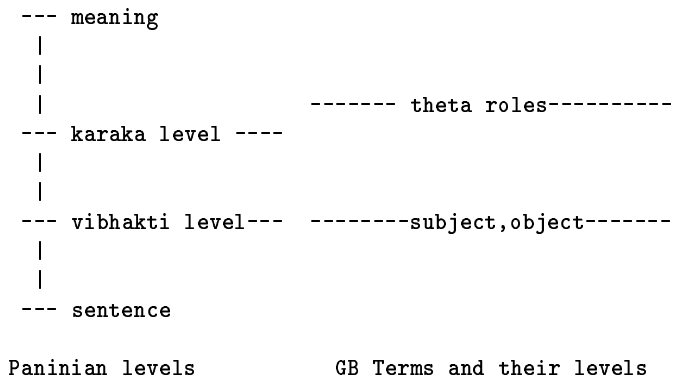


Figure 13.3: Relationships between terms in the two models

theorists working on Indian languages define subject in such a way that it turns out to be more or less the same as karta.

There is another difference in the concerns the two models have towards formal power of grammars. Since the generative enterprise wants to characterize the universal grammar that every human is born with, there is an attempt to use a grammar formalism with as little formal power as possible. This would ensure that it generates only the grammatical sentences in natural languages, and no more. The Paninian approach is neutral to such concerns.

The Paninian model focuses on karaka roles, while major research effort in GB is on anaphora and quantifiers. So they both have something to offer. It should be noted, however, that GB includes only reflexives and reciprocals out of all the possible anaphora. These are a tiny fraction of the total. Moreover, the theory does not give the referent of this restricted class of anaphora but rather identifies a domain within which the referent has to be found.

13.2 Summary

We give below a comparision of GB and Paninian approach in tabular format. It summarizes the arguments given above.

Topic	*Generative Enterprise*	*Paninian Approach*
Motivating question	How is the child able to acquire language?	How is the speaker able to convey information to the hearer using language?
Goal	Study of innate and autonomous structures in the human mind relating to language faculty as distinct from other cognitive structures	Study of language as a means of communication
Task	Theory should classify sentences as grammatical and ungrammatical	Theory should explain the process of going from meaning to sentences and vice versa
Focus	Isolates and studies syntax	Combines syntax, semantics and pragmatics in an overall framework
Overall model	Indirect connection between surface form (PF-representation) and meaning (LF-representation)	Direct connection between surface form and meaning
Language types	Principles and terminology inspired by positional languages	inspired by free word order languages

Appendix A

Panini's Grammar and Sanskrit

[1]

In this appendix, we outline how the original Panini's grammar applies to sentences in Sanskrit. The original text presenting the grammar system is called Ashtadhyayi and was written by Panini somewhere around 500 BC. This presentation is based on Kiparsky (1982) and focusses on: (1) the relation between the syntax of actives, passives and statives, (2) the connection between sentences and nominals, and (3) the relation between cases and meanings.

A.1 Karaka Theory

The karaka [2] system of Panini is the crucial element in accounting for the issues mentioned above. Consider the following from Kiparsky (1982) where the A's are sentences and the Bs are nominals for active, passive and stative as marked:

```
A.1  devadattah        odanam       pacati.
     Devadatta (nom.)  rice (acc.)  cooking (active)
     (Devadatta is cooking rice.)
```

[1] K.V. Ramakrishnamacharyulu of Rashtriya Sanskrit Vidyapeetha, Tirupati is the co-author of this appendix.

[2] Pronounced as kaaraka. Let us also outline, here, our spelling scheme: Whenever a sentence in Hindi is transliterated in roman, it is written using the scheme given in Appendix B. This can be seen in the examples given in displayed material. However, whenever a term from Indian language is adopted in English, the spelling used is based on previous usage in literature. Thus, 'karaka' is not spelt as 'kaarak', or similarly, 'karta' is not spelt as 'kartaa'. They continue to use the prevalent spelling in literature.

```
A.2  devadattena odanah pacyate.
       (instr.)  (nom.) (passive)
     (Rice is being cooked by Devadatta.)

A.1b devadattah svapiti.
        (nom.)  (active)
     (Devadatta is sleeping.)

A.3b devadattena supyate.
     (instr.)    (stative)
     'Devadatta is being slept'
     (Devadatta is sleeping.)

B.1  devadattah  odanasya   paacaka.
       (nom.)    (genitive) (nom.)
     (Devadatta cooker of rice.)

B.2  devadattena  odanah  pakva
       (instr.)   (nom.)  (nom.)
     (Rice cooked by Devadatta.)

B.3  devadattena  odanasya  paktir.
        (instr.)   (gen.)    (nom.)
     (The cooking of rice by Devadatta.)

B.1b devadattah     svapitaa.
        (nom.)       (nom.)
     (Devadatta a sleeper.)

B.3a devadattasya    paktih.
       (gen.)       (nom.)
     (Devadatta's  cooking.)
```

These can be shown as follows in a table (where the sentences numbers suffixed by 'b' are for the verb 'sleep'):

Prayoga	*Sentences*	*Nominals*
active	A.1, A.1b	B.1, B.1b
passive	A.2	B.2
stative	A.3b	B.3, B.3a

In Sanskrit, the nominal can occur all by itself (as an independent sentence with an assumed implicit 'be' verb).

Panini accounts for these sentences (and nominals) as alternative realizations of the same underlying structure. The active is no more basic than

passive, just as the sentences are no more basic than the nominals. In this, it introduces a set of grammatical functions called karakas, which specify relations between nominals and the verbal root. In words of Kiparsky (1982; p.4) "They are neither semantic nor morphological categories in themselves, but they correspond to semantics according to rules specified in the grammar and to morphology according to other rules specified in the grammar."

In the examples involving the verb cooking above, the nominals represent two karakas: karta and karma. A karaka is a participant in the action denoted by the verb. Semantically the *karta* is the most independent of all the participants[3], and *karma* is the "principal goal of the karta"[4]. In the examples involving the verb 'sleep', there is only karta karaka.

The derivation of the sentence proceeds as follows. The "speaker" selects (or the grammar *freely generates*) verbs and nouns from the lexicon, corresponding to an action and the participants in it. Karaka relation between the nouns and verb are selected depending on the semantic relation between them, and time reference is associated with the verb. This is at the karaka level representation (See Fig. 5.2).

From the karaka level representation, vibhakti level representation is generated using Panini's sutras (dealing with karakas). In this representation, vibhakti (or case) is identified for each noun and verb. The sutras are rules which apply on a representation to produce another representation at another level. Panini's sutras dealing with morphology apply on the vibhakti level representation to generate the actual words (and thus the sentence).

The most important principle of the karaka theory is (Kiparsky, 1982):

Principle: Every karaka must be "expressed" by morphology, but no karaka may be expressed more than once.

Here "express" is used in a technical sense.

Let us look at the derivation of the sentence A.1 (devadattah odanam pacati). The derivation is initiated by the speaker wanting to express that 'Devadatta is cooking rice.' At the karaka level, this results in a structure consisting of verbal root 'pac' and noun roots 'devadatta' and 'odana', with 'devadatta' as karta, and 'odana' as karma. Further, the verbal root has ongoing time (vartamaana) and active prayog (loosely, active voice) associated with it.

From the above structure at karaka level, we generate the structure at the vibhakti level. The verb gets the abstract present tense marker 'laT'[5]

[3]Ashtadhyayi sutra no. 1.4.54: svatantrah kartaa.
[4]Sutra 1.4.49 kartur iipsitatamam karma.
[5]Sutra 3.2.123 vartamaane laTa.

By another rule, the tense laTa "expresses" (in the technical sense) either (i) karta, (ii) karma or (iii) state[6]. The active prayog (active voice) selects (i). In other words, laTa expresses karta. (Note that choices (ii) and (iii) produce sentences A.2 and A.3, respectively.)

The generation of vibhaktis of the nominals is as follows. Karma karaka causes an accusative ending (dvitiiyaa or second vibhakti) to be assigned to 'odana'[7]. Karta karaka normally causes an instrumental ending (tritiiyaa or third vibhakti) to be assigned to the nominal bearing the karta karaka relation[8]. But here the leading principle that nothing is expressed more than once (anabhihite) comes into play. Since karta has already been expressed by laT, it should not be expressed again. Thus, the instrumental case is not assigned to 'devadatta'. Instead, nominative case (prathamaa or first vibhakti) is assigned to 'devadatta', when only the nominal stem notion, gender and number remain to be expressed[9].

Finally, the representation at the vibhakti level is taken and the word forms are generated.

The derivation processes of nominals and sentences are completely parallel. They are derived from identical karaka relations between the nouns and the verb.

The verbal noun can again be in three prayog expressing karta, karma or the process. This gives us three kinds of nominals shown in B.1, B.2 and B.3, respectively. In B.1, the derivation is like A.1 except that the karma of verbal noun is expressed by the genitive case[10].

Thus, the actives, passives, and statives on the one hand and sentences and nominals on the other hand, can be analyzed elegantly in a parallel manner by the karaka theory.

A.2 Control

Karaka system is also important in formulating control of non-finite verbs in complex sentences. Consider the following sentences, for example:

```
C.1  graamam    aagatya   devdattah    odanam   pacati.
     village-Acc.        Devdatt-Nom. rice-Acc.  cooks
               having-arrived
    (Having arrived at the village,Devdatta cooks rice.)

C.2  graamam  aagatya   devdattena   odanah   pacyate.
       Acc.                instr.             nom.
```

[6]Sutra 3.4.69 lah karmani ca bhaave caakarmakebhyah.
[7]Sutra 2.3.2 karmani dvitiiyaa.
[8]Sutra 2.3.18 kartrikaranyoa tritiiyaa.
[9]Sutra 2.3.46 praatipadikaarthaling-aparimaana vacanamaatre prathamaa.
[10]Sutra 2.3.65 kartrikarmonoh kriti.

```
((Devdatta) having arrived at the village, rice was
 cooked by Devdatta.)
```

Here, the agent of 'arrive' remains unchanged in C.1 and C.2 because the overtly specified karta of 'cook' remains the same. Panini gives the rule which says that:

> Karta of the gerund suffix (ktva in Sanskrit) is the same as the karta of the main verb[11].

Note that this is unlike English, where control is formulated in terms of subject, object etc.

Panini has given great importance to control in non-finite verbs. A number of sutras give rules for control of such verbs. For example, Sutra 3.3.158 for verbal roots with tumun (infinitive of purpose, such as, to go)[12].

Further Reading

This appendix is based on Kiparsky (1982). Original sources are Panini's Ashtadhyayi written somewhere between 2400 to 2600 years ago, Patanjali's Mahabhashya (some 2100 years ago), Bhartrahari's Vakyapadiya (roughly 1500 years ago), Kaundbhatta's Vaiyakarana- bhushanasara (about 350 years ago), Nagesh Bhatt's Laghumanjusha (about 250 years ago). All these were written in Sanskrit.

The above are part of the Paninian tradition and are available with commentary (in English except as noted): Jigyasu (1979) (in Hindi) on Ashtadhyayi; Joshi (1968) and Kielhorn (1970) on Mahabhashya; Shastri and Tripathi () (in Sanskrit) on Vaiyakarana- bhushanasara; Abhyankar and Limaye (1965) and Helaraja on Vakyapadiya (Bhartrahari()); etc.

Shastri (1973) (in Hindi) and Cardona (1988) are good materials on the Paninian tradition. For recent debates on Paninian tradition see Deshpande (1985), Kiparsky (1982), Matilal (1982), Cardona (1976), among others.

It is important to mention that there is a view that Tolkappiyam's grammar for Tamil predates Panini. Similarly, there were a number of pre-Paninian grammars for Sanskrit. All these are similar in spirit. There is a need to study Tolkappiyam's grammar for Tamil from computational viewpoint. Our conjecture is that it will be instructive and will offer similar but interesting insights.

There is a vast amount of literature on the Paninian and related traditions. See Cardona (1976) for a survey of research.

There are many discussions related to the Paninian tradition, in the literature of Navya-Nyaya (Indian system of logic and inference) and that of Mimamsa (Indian system of discourse analysis).

[11] Sutra 3.4.21 samanakartrikayoh puurvakaale.

[12] Sutra 3.3.158 samaanakartrikesu tumun.

Appendix B

Roman Notation for Devanagri

Table 1: Notation used in this book

अ	आ	इ	ई	उ	ऊ	ऋ	ए	
a	aa	i	ii	u	uu	R	e	
ऐ	ओ	औ	ं	ः	ँ			
ei	o	ou	M	H	z			
क	ख	ग	घ	ङ				
k	kh	g	gh	n				
च	छ	ज	झ	ञ				
c	ch	j	jh	n				
ट	ठ	ड	ढ	ण				
T	Th	D	Dh	N				
त	थ	द	ध	न				
t	th	d	dh	n				
प	फ	ब	म	म				
p	ph	b	bh	m				
य	र	ल	व	श	ष	स	ह	
y	r	l	v	sh	S	s	h	

Table 2: Internal representation in the computer

अ	आ	इ	ई	उ	ऊ	ऋ	ए
a	A	i	I	u	U	q	e
ऐ	ओ	औ	ं	ः	अँ		
E	o	O	M	H	z		
क	ख	ग	घ	ङ			
k	K	g	G	f			
च	छ	ज	झ	ञ			
c	C	j	J	F			
ट	ठ	ड	ढ	ण			
t	T	d	D	N			
त	थ	द	ध	न			
w	W	x	X	n			
प	फ	ब	भ	म			
p	P	b	B	m			
य	र	ल	व	श	ष	स	ह
y	r	l	v	S	R	s	h

Conversion programs between this notation and ISCII standard are available free of charge from the authors.

Bibliography

[1] Abhayankar, K.V., and V.P. Limaye, *Vakyapadiya of Bhartrhari*, Sanskrit and Prakrit Series, Vol. 2, University of Poona, 1965.

[2] Ahuja, R.K., Thomas L. Magnanti, and James B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[3] Barton, G. Edward, Robert C. Berwick, and Eric S. Ristad, *Computational Complexity and Natural Language*, MIT Press, Cambridge, MA, 1987.

[4] Berwick, Robert C., S.P. Abney, and C. Tenney, (eds.), Principle-Based Parsing: Computation and Psycholinguistics, Kluwer Academic, Boston, 1991.

[5] Bhanumati, B., *An Approach to Machine Translation among Indian Languages*, Tech. Report TRCS-89-90, Dept. of CSE, IIT Kanpur, Dec. 1989.

[6] Bharati, Akshar and Rajeev Sangal, A Karaka Based Approach to Parsing of Indian Languages, In *COLING90: Proc. of Int. Conf. on Computational Linguistics (Vol. 3), Helsinki*, Association for Computational Liguistics, NY, August 1990, pp. 25-29.

[7] Bharati, Akshar, Rajeev Sangal, and Vineet Chaitanya, Natural Language Processing, Complexity Theory and Logic, In *Foundations of Software Technology and Theoretical Computer Science 10, Lecture Notes in Computer Science 472*, Springer Verlag Berlin, 1990a, pp.410-420.

[8] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Computational Framework for Indian Languages, Technical Report TRCS-90-100, Dept. of CSE, IIT Kanpur, July 1990b. (Course notes for Intensive Course on NLP for Linguists, Vol. 1).

[9] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Local Word Grouping and Its Relevance to Indian Languages, In Vijay P. Bhatkar and Kiran M. Rege, editors, *Frontiers in Knowledge Based Computing*, pages 277–296. Narosa Publishing House, New Delhi, 1991.

[10] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Computational Grammar for Indian Languages Processing, *Indian Linguistics journal*, 52(1–4):91–103, Mar.–Dec. 1991a.

[11] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Machine Translation and the Language Barrier, In V V S Sarma, N Viswanadham, B Yegnanarayana, and B L Deekshatulu, editors, *Artificial Intelligence and Expert System Technologies in the Indian context*, volume 2, pages 58–66. Tata–McGraw Hill, New Delhi, 1991b.

[12] Bharati, Akshar, S.A. Nawathe, Vineet Chaitanya, and Rajeev Sangal, A New Inference Procedure for Conceptual Graphs, In *4th University of New Brunswick Artificial Intelligence Symposium*, Frederiction, N.B. Canada., 19–21 Sept. 1991d.

[13] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Computational Linguistics and Its Relation to Linguistics, *International Journal of Dravidian Linguistics*, XXI(2):106–114, June 1992a.

[14] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, LFG, GB and Paninian Frameworks: An NLP Viewpoint, In *Tutorial on NLP at CPAL–2: UNESCO 2nd Regional Workshop on Computer Processing of Asian Languages*, Dept. of CSE, IIT Kanpur, 12–16 March 1992b, pp.1-42. (Also available as TRCS-92-140, Dept. of CSE, IIT Kanpur.).

[15] Bharati, Akshar, Y, Krishna Bhargava, and Rajeev Sangal, Reference and Ellipsis in an Indian Languages Interface to Databases, *Computer Science and Informatics Journal 23*, 3, Sept. 1993, pp. 60–82.

[16] Bharati, Akshar and Rajeev Sangal, Parsing Free Word Order Languages using the Paninian Framework, In *ACL93: Proc. of Annual Meeting of Association for Computational Linguistics*. Association for Computational Linguistics , NY., 1993a, pp105-111.

[17] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, *Course material on A Computational Grammar Based on Paninian Framework*, Indian Society for Technical Education, Ministry of Human Resource Development, New Delhi, October 1993b.

[18] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Anusaraka or Language Accessor: A Short Introduction, In *Presented at Seminar*

on Automatic Translation, Thiruvananthpuram, October 1993c. Int. school of Dravidian Linguistics. (Also available as TRCS-93-205 Dept. of CSE, IIT Kanpur).

[19] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Anusaraka as a Measuring Device for the Linguist, In *Presented at the Dialogue on Future Linguistics in India, 22-24 Dec. 1993*. Central Institute fo Indian Languages, Mysore, December 1993d. (Also available as TRCS-93-210 Dept. of CSE, IIT Kanpur.).

[20] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, An appropriate strategy for machine translation in india, In S.S. Agarwal and Subhash Pani, editors, *Information Technology Applications in Language, Script & Speech*, New Delhi, 1994a. BPB Publications.

[21] Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Tree Adjoining Grammar and Paninian Grammar, Technical Report TRCS-94-219, Dept. of CSE, IIT Kanpur, March 1994b.

[22] Bhargava, Y.K., Reference and Ellipsis in a NL Interface to Databases, M.Tech. thesis, Dept. of CSE, IIT Kanpur, 1992.

[23] Bhartrahari, Sadhana Sumuddesha (III kanda), in Bhartrahari, *Vakyapadiya* with the commentry of Helaraja, Poona. (Translation in English by K. Subramanya Iyer, Poona)

[24] Bhatt, Rajesh, *Paninian Theory for English*, B.Tech. thesis, Dept. of CSE, IIT Kanpur, 1993.

[25] Block, Hans-Ulrich, and Hans Haugeneder, An Efficiency-Oriented LFG Parser, In Reyle and Rohrer (1988), pp.149-176.

[26] Bresnan, Joan, (ed.), *The Mental Representation of Grammatical Relations*, M.I.T. Press, Cambridge, MA, 1982.

[27] Budhiraja, Navin, Subrata Mitra, Harish Karnick, and Rajeev Sangal, Parsing Generalized Phrase structure Grammar with Dynamic Expansion, *Proc. of Int. Workshop on Parsing Technologies*. Center for Machine Translation, CMU, Pittsburg, Aug. 1989, pp. 458–467.

[28] Cardona, George, *Panini: A Survey of Research*, Mouton, Hague-Paris, 1976.

[29] Cardona, George, *Panini: His Work and Its Tradition, (Vol. 1: Background and Introduction)*, Motilal Banarsidas, Delhi, 1988.

[30] Chierchia, Gennaro, and Sally McConell-Ginet, *Meaning and Grammar*, MIT Press, Cambridge, Mass., 1991.

[31] Chomsky, Noam. *Lectures on Government and Binding*, Foris, Dordrecht, 1981.

[32] Chomsky, Noam. *Knowledge of Language: Its Nature, Origin and Use.* Praeger Publishers, New York, 1986.

[33] Copeland, C., J. Durand, S. Krauwer, and B. Maegaard, *The Eurotra Linguistic Specification*, Studies in Machine Translation and Natural Language Processing, Vol. 1, Office for Official Publications of Commission of the European Communities, Luxemburg, 1991.

[34] Covington, Michael A., Parsing Discontinuous Constituents in Dependency Grammar, (Technical Correspondence), *Computational Linguistics*, 16,4 (Dec. 1990), p.234.

[35] Dash, K.C., (ed.), *Indian Semantics*, Agamakala Publications, Delhi, 1994.

[36] Date, C.J. *Introduction to Database Systems*, Addison-Wesley, Reading, MA, 1987.

[37] Deshpande, Madhav M., *Ellipses and Syntactic Overlapping: Current Issues in Paninian Syntactic Theory*, Bhandarkar Oriental Research Institute, Poona 411004 India, 1985.

[38] Deo, Narsingh. *Graph Theory.* Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

[39] Dorr, Bonnie, A Survey of Courses in Computational Linguistics, Assoc. of Computational Linguistics, New Jersey, 1994.

[40] Duarte, Ines. X-Bar Theory: Its Role in GB Theory. In *Natural Language Processing*, Lecture Notes in Aritificial Intelligence, Springer Verlag, Berlin, 1990.

[41] Earley, Jay, An Efficient Context-Free Parsing Algorithm, *Communications of ACM 6*, 8, 1970, pp.451-455.

[42] Evens and Larry Karttunen, Survey on Computational Linguistics Courses, *Association of Computational Linguistics*, New Jersey, 1983.

[43] Findler, Nicholas V. (ed.), *Associative Networks: Representation and Use of Knowledge by Computer*, Academic Press, New York, 1979.

[44] Fillmore, Charles J., The Case for Case, In E. Bach and R.T. Harms editors, *Universals of Linguistic Theory*, Holt Rinehart and Winston, New York, 1968, pp.1-88.

[45] Gazdar, G., E. Kleine, G.K. Pullum, and I.A. Sag, *Generalized Phrase Structure Grammar*, Basil Blackwell, 1985.

[46] Gupta, Ashish, S. Melhotra, and M. Saxena *A Parser for LFG and a Grammar for Hindi*, B.Tech. thesis, Dept. of CSE IIT Kanpur, 1988.

[47] Gupta, Sandeep K.S., Vineet Chaitanya, Harish Karnick, and Rajeev Sangal, *An Architecture for Parallel and Distributed Parsing of Indian Languages*, Technical Report, TRCS-89-82, Dept. of CSE IIT Kanpur, 1989.

[48] Haegeman, L., Introduction to Government and Binding Theory, Basil Blackwell, 1991.

[49] Harris, Larry R, User-Oriented Database Query with ROBOT Natural Language Query System, *Int. J. of Man-Machine Studies*, 1977.

[50] Hausser, Roland, *Computation of Language*, Springer Verlag, Berlin, 1989.

[51] Hendrix, G.G., Earl Sacerdoti, D. Sagalowicz, and J. Slocum, Developing a Natural Language Interface to Complex Data, *ACM Trans. on Database Systems 3, 2*, 1978, pp.105-147.

[52] Hopcroft, J.E. and R.M. Karp, "A $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *J. SIAM Comp. 2* (1973), pp.225-231.

[53] Hopcroft, J.E. and Jeffrey D. Ullman, *Intro. to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[54] Horowitz, Ellis, and Sartaj Sahni, *Computer Algorithms*, Computer Science Press, 1978.

[55] Iyer, K. Subramanya, *Bhartrahari on Means*, University of Poona, pp. 283-343.

[56] Jigyasu, Brahmdatt. *Ashtadhyayi (Bhashya) Prathamavrati, three volumes*, Ramlal Kapoor Trust Bahalgadh, (Sonepat, Haryana, India), 1979. (In Hindi)

[57] Joshi, Aravind K., Tree Adjoining Grammar: How much Context-Sensitivity is Required to provide reasonable Structural Description. In D. Dowty, L. Karttunen, and A. Zwicky (eds.), *Natural Language Parsing*, Cambridge University Press, Cambridge, UK, 1985, pp 206-250.

[58] Joshi, Aravind K., An Introduction to Tree Adjoining Grammars. In A. Manaster-Ramer (ed.), *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[59] Joshi, Aravind K., Processing Crossed and Serial Dependencies, *J. of Language and Cognitive Processes* 5, 1, 1990, pp.1-27.

[60] Joshi, S.D. (editor). *Patanjali's Vyakarana Mahabhashya*, (several volumes), Univ. of Poona, Pune, 1968.

[61] Kaplan, R.M., and Joan Bresnan, Lexical Functional Grammar: A Formal System for Grammatical Representation, In *The Mental Representation of Grammatical Relations*, Joan Bresnan (ed.), MIT Press, Cambridge, 1982, pp. 173-281.

[62] Kashket, Michael B., Parsing a free-word-order language: Warlpiri, *Proc. of 24th Annual Meeting of ACL*, 1986, pp. 60-66.

[63] Katz, J.J., and J.A. Fodor, The Structure of Semantic Theory, *Language 39*, 1963, pp.170-210.

[64] Kay, Martin, Functional Grammar, *Proc. 5th Annual Meeting of the Berkeley Linguistics Society*, 1979, pp.142-158.

[65] Kielhorn, F., (ed.), *The Vyakarana-Mahabhashya of Patanjali*, Otto Zeller Verlag Osnabruck, 1970, (Reprint of edition 1880).p. 6 (m 1.1.1) (In Sanskrit).

[66] King, Margaret, *A Tutorial on Machine Translation*, Working Paper 53, ISSCO, Geneva, 1987.

[67] Kiparsky, P., *Some Theoretical Problems in Panini's Grammar*, Bhandarkar Oriental Research Institute, Poona 411004 India, 1982.

[68] Knuth, Donald E., *Fundamental Algorithms*, Addison-Wesley, 1973.

[69] Kuhn, H.W., "The Hungarian Method for the Assignment Problem", *Naval Research Logistics Quarterly, 2* (1955), pp.83-97.

[70] Matilal, B.K., Some Comments of Patanjali under P.1.2.64, *Proc. of the International Seminar on Panini*, Centre of Advanced Study in Sanskrit, University of Poona, 1982, pp. 119-126.

[71] Mohanan, K.P., Grammatical Relations in Malayalam, In *The Mental Representation of Grammatical Relations*, Joan Bresnan (ed.), MIT Press, Cambridge, 1982, pp. 504-589.

[72] Narasimhan, R., *Modeling Language Behaviour*, Springer-Verlag, Berlin, 1981.

[73] Narayana, V.N., *Anusarak: A Device to Overcome the Language Barrier*, Ph.D. thesis, Dept. of CSE, IIT Kanpur, January 1994. Submitted.

[74] Nirenberg, Sergie, *Machine Translation: Theoretical and Methodological Issues*, Cambridge University Press, Cambridge, UK, 1987.

[75] Papadimitrou, Christos H., and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[76] Patnaik, B.N., Lexical Functional Grammar, *Course Notes for NLP for Linguists, Vol. 2: Some Computational Models of Language*, 1990. (Available as Technical Report TRCS-90-101, Dept. of CSE IIT Kanpur.)

[77] Perraju, Bendapudi V.S., *Algorithmic Aspects of Natural Language Parsing using Paninian Framework*, M.Tech. thesis, Dept. of CSE, IIT Kanpur, Dec. 1992.

[78] Pollard, C., *Lecture Notes on Head-Driven Phrase Structure Grammars*, Center for the Study of Language and Information (CSLI), University of Chicago Press, 1985.

[79] Pollard, C., and Ivan Sag, *An Information-Based Approach to Syntax and Semantics*, Vol. 1, CSLI Lecture Notes 13, Stanford University, 1987.

[80] Radford, Andrew. *Transformational Grammar*, Cambridge University Press, UK, 1988.

[81] Ramakrishnamacharyulu, K.V., Paninian Linguistics and Computational Linguistics, *Samvit*, Series no. 27, Academy of Sanskrit Research, Melkote, Karnataka, 1993, pp. 52-62.

[82] Ramesh, P.V. and Rajeev Sangal, Issues in Implementation of LFG, *Proc. of National Seminar on NLP*, Vishakhapatnam, India, Dec. 1989. (Paper available as Technical Report TRCS-89-87, Dept. of CSE IIT Kanpur.)

[83] Ravisankar, P.V., *Enhancements to the Linguist's Workbench*, M.Tech. thesis, Dept. of CSE, IIT Kanpur, 1992.

[84] Reingold, E.M., and R.N. Reingold, *PascAlgorthims*, Scott, Foreman and Co., Glenview, Illinois, 1988.

[85] Reyle, Uwe, and Christian Rohrer (eds.), *Natural Language Parsing and Linguistic Theories*, D. Reidel Publishing Company, Dordrecht, 1988.

[86] Rogers, James, and K. Vijay-Shanker, A Formalization of Partial Description of Trees, Draft, June 1992.

[87] Rogers, James, and K. Vijay-Shanker, Reasoning with Descriptions of Trees, *ACL92: Proc. of Annual Meeting of Assoc. of Computational Linguistics*, ACL, 1992.

[88] Rogers, James, and K. Vijay-Shanker, Obtaining Trees from Their Descriptions: An Application to Tree-Adjoining Grammars, *Computational Intelligence journal*, 1993 (Submitted).

[89] Saint-Dizier, Patrick, Yannick Toussaint, Christophe Delaunay, and Pascale Sebillot, A Natural Language Processing System Based on the Government and Binding Theory, in Saint-Dizier and Szpakowicz (1990), pp. 108-140.

[90] Saint-Dizier, Patrick, and S. Szpakowicz, (eds.), *Logic and Logic Grammars for Language Processing*, Ellis Horwood, Chicester, 1990.

[91] Sangal, Rajeev, Machine Translation, *2001* (formerly *Science Today*), Jan. 1989.

[92] Sangal, Rajeev, *Programming Paradigms in LISP*, McGraw Hill, New York, 1991.

[93] Sangal, Rajeev and Vineet Chaitanya, An Intermediate Language for Machine Translation: An Approach based on Sanskrit Using Conceptual Graph Notation, *Computer Science & Informatics*, Journal of Computer Society of India, 17, 1, pp. 9-21, 1987.

[94] Sangal, Rajeev, Vineet Chaitanya and Harish Karnick, An Approach to Machine Translation in Indian Languages, *Proc. of Indo-US Workshop on Systems and Signal Processing*, Indian Institute of Science, Bangalore, Jan. 1988.

[95] Schabes, Yves, *Mathematical and Computational Aspects of Lexicalized Grammars*, Ph.D. thesis, Univ. of Pennsylvania, 1990.

[96] Schank, Roger P., and R.P. Abelson, *Scripts, Goals, Plans and Understanding*, Lawrence Earlbaum Associates, Hillsdale, NJ, 1977.

[97] Sengupta, Probal, *On Lexical and Syntactic Procesing of Bangla Language by Computer*, Ph.D. thesis, Indian statistical Institute, Calcutta, Sept. 1993 (submitted).

[98] Shastri, Charudev, *Vyakarana Chandrodaya (Vols. 1 to V)*. Delhi: Motilal Banarsidass. 1973. (In Hindi)

[99] Shastri, Gopal, and Ramprasad Tripathi, *Vaiyakarana Bhushansara: Sarala Subodhini Vyakhyadvayopeta*, Chaukhambha Sanskrit Series Office, Varanasi. (In Sanskrit)

[100] Shieber, Stuart M., Separating Linguistic Analysis from Linguistic Theories, In *Natural Language Parsing and Linguistic Theories*, U. Reyle and C. Rohrer, (eds.), D. Reidel, Dordrecht, 1988, pp. 33-68.

[101] Sowa, John, Conceptual Structures, Addison-Wesley, Reading, 1985.

[102] Srinivas, B., *Linguist's Workbench: A Grammar Development Tool for Indian Languages*, M.Tech. thesis, Dept. of CSE, IIT Kanpur, May 1991.

[103] Stabler, Jr., Edward P., Parsing as Logical Constraint Satisfaction, in Saint-Dizier and Szpakowicz (1990), pp. 72-93.

[104] Tennant, Harry, *Natural Language Processing*, Petrocelli Books, New York, 1981.

[105] Tomita, Masaru, *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, 1985.

[106] Ullman, Jeffrey D., *Principles of Database Systems*, Computer Science Press, 1987.

[107] van Riemsdijk, Henk C., and Edwin Williams, Introduction to the Theory of Grammar, MIT Press, 1986.

[108] Vijay-Shanker, K., Using Descriptions of Trees in a Tree Adjoining Grammar, *Computational Linguistics* 18, 4 (Dec. 1992), pp. 481-517.

[109] Vijay-Shanker, K., and Aravind K. Joshi, Unification Based Tree Adjoining Grammer. In J. Wedekind (eds.), *Unification-Based Grammars*, M.I.T. Press, Cambridge, MA, 1991 (to appear).

[110] Wasow, T., *Anaphora in Generative Grammar*, E. Story-Scientica P.V.B.A. Scientific Publishers, Brussels, 1979.

[111] Wehrli, Eric, Parsing with GB-Grammar, In *Natural Language Parsing and Linguistic Theories*, U. Reyle and C. Rohrer, (eds.), D. Reidel Publishing Co., Dordrecht, 1988, pp. 177-201.

[112] Wiederhold, Gio, *Database Design*, McGraw-Hill, New York, 1982.

[113] Winograd, Terry, *Understanding Natural Language*, Academic Press, New York, 1972.

[114] Winograd, Terry, *Language as a Cognitive Process, Vol. 1: Syntax*, Addison-Wesley, Reading, MA, 1983.

[115] Wirth, Niklaus, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[116] Wilensky, Robert, *Planning and Understanding*, Addison-Wesley, Reading, MA, 1983.

[117] Woods, William A., Lunar Rocks in Natural English: Explorations in Natural Language Question Answering, In A. Zampolli (ed.), *Linguistic Structures Processing*, Elsevier North-Holland, 1977.

[118] Younger, D.H., Recognition and Parsing of Context Free Languages in Time n^3, *Information and Control 10*, 1967, pp. 189-208.

[119] Zribi-Hertz, Anne, Anaphora, Binding and Narrative Point of View: English Reflexive Pronouns in Sentence and Discourse, *Language 65, 4*, 1989, pp. 695-727.

Index

——

Glossary of Paninian and Other Terms

*		A '*' before a sentence indicates that it is a bad sentence.
@H		Indicates output Hindi produced by anusaraka.
!E		Indicates gloss in English.
karaka		Syntactico-semantic relation between a verb and its arguments.
	karta karaka	The most independent of the karakas (roughly, corresponds to agent, but not always.)
	karma karaka	The goal argument of the verb.
	karana karaka	The instrument.
	sampradana-karaka	The beneficiary of the action.
	apadana- karaka	The fixed or reference object from which the separation action takes place.
	adhikarana-karaka	Time and location of the action.
lakshan-chart		Discrimination net.
LWG		Local word grouper.
parsarg		Post position marker.
	ϕ parsarg	Nominative marker in Hindi.
	ko parsarg	Accusative or dative marker in Hindi.
	ne parsarg	Ergative marker in Hindi.
	se parsarg	Instrumental or ablative marker in Hindi.
	kaa parsarg	Genitive marker in Hindi.
PG		Paninian Grammar (framework)
tadarthya		Purpose for which the action is carried out.
TAM		Tense aspect modality.
vibhakti		Collectively: case ending, post position or preposition of a noun, or TAM label of a verb.