ON LEXICAL AND SYNTACTIC PROCESSING OF BANGLA LANGUAGE BY COMPUTER

Probal Sengupta
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
203 B. T. Road,
Calcutta 700 035,
INDIA.

A thesis submitted to the *Indian Statistical Institute*
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Acknowledgments

At the very beginning, I would like to thank my supervisor Prof. B. B. Chaudhuri for introducing me to the topic of NLP, his advise, guidance and his insistence on being meticulous in different aspects. This being a sort of pioneering work in the field of NLP on Bangla, naturally, I could not get many authorities to fall back upon. Prof. Chaudhuri filled the void to the best of my advantage, advising, discussing, providing counter-examples to my earlier and less mature hypotheses. He was always alert regarding the quality of our publications and even scrutinized minute typographical mistakes with utmost care.

During the early days of my courtship with Bangla-NLP, Prof. D. Dutta Majumder lent me considerable moral and technical support. He would occasionally summon me to his chambers, only to point out yet another new book or journal or another call for paper from some conference including NLP in its purview. Under his stewardship, the Electronics and Communication Sciences Unit (ECSU) of the ISI reached the pinnacle of glory, predominantly in vision research. Nevertheless, he was for ever ready with encouragement for me working in virtually a perpendicular line of study.

The room adjacent to the computer lab I used to work in houses the speech and music processing lab of the ECSU. Prof. A. K. Dutta has been diligently working on speech mainly and on music of late, in this lab. For speech analysis and synthesis, he and his colleagues mostly considers Bangla as the target language. Naturally, there are quite a few areas of mutual interest between my type of work and that of Prof. Dutta and his colleagues. We have had frequent discussions, debates and fights on these areas to the benefit of everybody. I have endeavored to pick up from Prof. Dutta his knack of visualizing cognition related problem in a top-down manner..

Many colleagues, friends, students helped me in many ways in the course of my work. A few years ago, Mr. Paresh Banerjee helped me a lot with initial development of the morphological processor. Prof. A. M. Ghosh, the previous head of the computer science and technology department of B. E. College, my earlier place of work, supported me with valuable suggestions and adjusted my teaching routines in such a way as to provide me with enough working time for my research interests. I thank the CST department of B. E. College for loaning me important reference books for long duration from the departmental library. I thank my friends Mr. Amarnath Gupta and Dr. Aditya Bagchi and Mr. Sabyasachi Basu for their encouragement and advise. I thank Dr. Partha Pramanik for scrutinizing my research papers. Ms. Maya Dey has occasionally helped me typing the present manuscript. I thank Mr. Debaprasad Bandyopadhyaya and Mr. Abhijit

Majumder for providing me with the required linguistic insight. Being linguists by background, they helped removing some of the obvious linguistic omissions that otherwise would have escaped my notice. I thank other members of the language group, that includes Ujjwal Maulik, for periodic discussions.

I reserve special thanks for Dr. Mary Dalrymple of Xerox PARC, California, without whose timely help, the survey of LFG and related works carried out in the present thesis would have been delayed considerably. Upon request through e-mail, Dr. Dalrymple not only sent me a thick package of LFG related materials but also took the pains to buy and send me the book *The Mental Representation of Grammatical Relations*, that too by courier post and advised me to "forget about money".

I express gratitude towards every member of my family — my mother, aunt, brother and others, for their encouragement. I thank my uncle Dr. S. Sengupta, a liguist of repute, for constant encouragement. I reserve special acknowledgment for my wife Chandreyee for enduring the last phase of the research, especially the thesis preparation stage. The same goes for my two-year old daughter Jhimli. However, I note with great regret that I could not make it possible for my father to enjoy the thrills of knowing his son has received his doctorate. My father left this world for his heavenly abode on the 25th of October, 1992 and with due respect, I dedicate this thesis towards the memory of my late father Anil Chandra Sengupta.

August 30, 1993

PROBAL SENGUPTA
ECSU, ISI,
Calcutta.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A distinctive intelligent trait of human beings is the ability to carry out meaningful communication through language. The communication may be direct as in spoken conversation or indirect as in written form, through the audio-visual media, etc. Linguistic ability in humans have fascinated scholars ever since man first learnt to use language. Linguistics, the branch of study involved in studying the nature of human linguistic communication, is perhaps as old as language itself. The invention of the computer added a new dimension to linguistics. Making the computer emulate human linguistic behaviour was taken up as a challenge by computer scientists. Branches of study like Cybernetics, Machine Translation (MT) from one language to another, Natural Language Generation, Natural Language Processing/ Understanding (NLP/ NLU) developed as a result. During the mid-1950s, a branch of modern linguistics known as Computational Linguistics (CL) came into existence. CL is dedicated mostly to develop more realistic models of cognition. MT, NLP, NLU, etc. were adopted by the artificial intelligence (AI) community of computer scientists as domains of research towards making computers more intelligent. AI workers in the above lines had to work in close association with computational linguists in this regard. There is however one major distinction in the viewpoint of CL and a NLP/ NLU. CL is more concerned about gaining insight into why and how human beings display cognitive behaviour as seen around us. NLP is more concerned in making the computer emulate such behaviours with implementation (as computer programs) of the linguistic theories, albeit only for a workable sub-set of the target language. The present work is primarily implementation motivated. However, at times certain linguistic conjectures have also been suggested.

ELIZA of Weizenbaum [174] was an early attempt towards making computer behave intelligently. However the treatment was quite superficial. Chomsky [37, 38] first proposed a formal mathematical theory of human cognitive behaviour. Subse-

quent refinement in the original Chomskian principles led to a major sub-discipline called the Government and Binding (GB). At present, there are many other important computational linguistic theories, some quite close and some quite distant from GB-theories. The Generalized Phrase Structured Grammar (GPSG) of Gazder et al [70], the Tree Adjoint Grammar (TAG) of Joshi [85], Functional Unification Grammar (FUG) of Kay [100, 109] and the Lexical Functional Grammar (LFG) of Kaplan et al [88] as some of the modern formalisms. In the implementation scenario, the Augmented Transition Network of Woods [181, 183], Marcus Parser of Marcus [121, 122], Slot and Modifier approach of McCord [126] and Definite Clause Grammar of Pereira [138] are a few important syntactic techniques. The GPSG, the FUG, the TAG and the LFG formalisms also come with associated implementation semantics. The mathematical properties of linguistic theories have been analyzed by Perrault [140]. Kay [110], Kaplan et al [87] and Koskenniemi [116, 117, 118] have proposed important formalisms for morphological processing at the lexical level. There are quite a few reported works on application of th above principles in lexicon design, as discussed later.

History of Indian linguistics dates back to more than a thousand years before Christ. Grammarians like Panini, Katyayana, Patanjali, etc. have left behind minutely detailed treatise on the Sanskrit language. Modern linguistic works in Indian languages have been mostly on prosody and comparative philology. Pioneering work on tracing the origin and development of Bangla has been carried out by S. K. Chatterjee [34] in the earlier part of the present century. More recently, a few CL based works have highlighted on certain special aspects of Bangla. However, no large scale work on Bangla, either in the CL domain or in the field of NLP/ NLU has come to our notice. NLP research appears to be in the formative stage throughout the country. Some isolated works in the line have been reported in the last few years (see later). Hence, to the best of our knowledge, the present work is the pioneering concerted effort in NLU with Bangla as the primary target. Our goal is to bridge the gap between existing linguistic knowledge and NLU vis-a-vis Indian languages, Bangla in particular. We would like to unify available knowledge on Indian languages with some state-of-the-art NLP techniques.

Different Levels of Natural Language Processing

NLP analysis of a natural language is conceived of in few different levels which have considerable mutual overlap.

Lexical Level: Responsible for extracting meanings and lexical categories (like Noun, Verb, etc., which are terminal symbols in syntax analysis) of words

considering them as primitive vocabulary items. For inflectional languages constituent morphemes of a word are extracted and the meaning and category of the word is derived from them.

Syntactic Level: Responsible for compartmentalization of the different meaning components of the sentence. Usually the analysis is based on a context-free grammar (CFG), or, as in the modern formalisms, on more powerful versions of grammar like GPSG, LFG, etc. Syntax analyzers are called parsers. The output of a parser is in a form amenable for semantic analysis.

Semantic Level: Responsible for extracting meanings of sentences where "meaning" or "semantics" of a sentence could be logical forms, formulae of higher order predicate calculus, graphical representations like semantic nets, conceptual graphs, etc. A model of the knowledge about the world of speakers should be incorporated.

Discourse Level: This level is an extension of the semantic level where temporal, causational, anaphoric, etc. relationships between different clauses of a discourse are modeled.

The ideas described here span across the lexical and syntactic levels. At the lexical level we are more concerned with morpho-syntactic analysis of words. At the syntax level, we would suitably augment and modify some existing techniques (mostly LFG) to suit Indian languages in general and Bangla in particular.

Review of Related Works

The review is categorized according to the different levels mentioned above. However, the domains of immediate interest, namely, lexical projection through morpho-syntactic analysis and unification based syntactic parsing, have been further reviewed in greater detail later in this section. Towards the end of this section, certain aspects of NLP and CL research in India relevant to the ideas introduced here, would be reviewed.

Most Natural Language Understanding systems are built around some linguistic formalism. Presently, quite a number of linguistic theories are available. Consequently, as noted in [10], there has been considerable duplication of work. Many of the existing formalisms cover only a few aspects of the entire problem of NLU. Some even focus upon specific areas of interest within a sub-domain of NLU. For example, the present thesis describes formalisms to cover the lexical and syntactic

aspects of NLP with Indian languages, especially Bangla. We would concern ourselves mostly on the aspects of morphological processing in the lexical sub-domain and syntax specification for certain types of sentences of Bangla everyday use. It is therefore appropriate to carry out the present review under a few topics of interest of the present thesis.

A. Lexicon Related Works.

"Lexicon" is the component of a NLU system which contains lexical entries associated with phonological, morphological, syntactic and semantic information. The lexicon has assumed the role of a major player in the recent years, such that "Lexical Representation" is presently a fundamental issue of most computational linguistic formalisms. The lexical analysis stage of an NLP system must provide systematic encoding of a variety of information about words that must minimally include:

i. The basic part-of-speech of the word.

ii. Other co-occurring and complement words.

iii. Certain syntactically inherent properties like tense, person (of verbs), etc.

Earlier NLP systems looked at the lexicon with little theoretical interest. These systems, for example, SHRDLU of Winograd [177], Sager's system [146], etc., generally used a CFG as the central syntactic formalism with the lexicon implemented as a collection of relatively unprincipled programs. Words were considered to contain monadic lexical knowledge and none other than the essential syntactic information was stored in the lexicon. However, since the 1980s, there has been an upsurge in interest in lexicon organization, lexical analysis/ representation. This coincided with the advent of feature based syntactic formalisms which conceived lexical knowledge encoded in the lexicon as a complex set of polyadic features covering domains even beyond the immediate needs of parsing. These modern systems, which includes the GPSG, the FUG, the Categorical Unification Grammars (CUG), the LFG, etc., use simple and compact syntactic rule-bases, pushing considerable grammatical facts into the lexicon. The importance of the lexicon has been given the proper stress even while choosing a nomenclature for the LFG. As a result of delegating a portion of the burden of recognition to a more complex lexical component, the overall system becomes more powerful without complicated syntactic manipulations being necessary. Modern textbooks on NLP like the ones

by Gazder and Mellish have provided detailed tutorial material on lexical representation [71, 72][Chap-7], providing analytical justification for the same. The issue of the nature of lexical representation was taken up in [53, 66] etc., while organization of lexical databases was discussed in [10, 56, 83], etc. Machine-readable dictionaries have been the concern of [21, 27, 30] etc. A phrasal lexicon with self-extending properties was taken up in [186].

Morphology and Lexical Analysis

We would concentrate mostly on inflectional morphology as a means of obtaining lexical projections of words as a result of unification of lexical knowledge of constituent morphemes. Gazder has pointerd out [69][pp604], that until recently, there has been relatively little work on morphology compared to syntax. The reason for this, according to Gazder, is that most CL based works dealt with English whose inflectional morphology is "impoverished". Formal aspects of morphological processing was first taken up by Kaplan and Kay [87] in 1981. Here, the suggestion was to have a 'cascade' of finite state transducers (FSTs), each taking care of one aspect of morpho-phonemic, morpho-graphemic irregularity. The cascaded FST networks can next be converted into an "equivalent" network using well-defined principles of formal language theory [1, 2]. Subsequently, Kay [110] has worked on non-concatenative morphology where the above principles do not hold, i.e. the processor can not be formulated as a FST network. Carden [31] has argued about general inadequacies of finite-state devices in word formation. Koskenniemi [116] has argued that the resultant equivalent automata in Kaplan and Kay may become prohibitively large. He suggested a mechanism where the transducers are simulated in "parallel". The central theme of Koskenniemi's work, generally known as the "two-level" morphological analysis technique, is formulated as:

```
A       X _ Y
   <=
a       x   y
```

or

```
A       X _ Y
   =>
a       x   y
```

The first form says that a lexical 'A' in the context 'XAY' ('X', 'Y', are lexical) may be realized in the surface as 'a' when 'X' is realized as 'x' and 'Y' is realized

as 'y'. The second form suggests 'A' in the given context must be realized as 'a'. Other abbreviative devices have also been used by Koskenniemi. Strength of the formalism is in the fact that the rules may be realized as FST networks as has been demonstrated by Koskenniemi. The size of the overall network incorporating all the two-level rules for a language is also not expected to be very large. The notion of the boundary between morphemes is expressed through a diacritic symbol '+'. This way, it is logically possible to have one entry for each word in the lexicon, even though the stem may be subject to various deformities during inflection. This property leads to considerable reduction in the space required to store the lexicon of an inflectional language.

Two-level morphology was originally proposed for Finnish, Koskenniemi's native language. It has received wide acceptance in the following years. Koskenniemi later proposed techniques for automatic compilation of a two-level "specification" into a finite-state automaton. The *KIMMO* system of Kartunnen et al, Khan et al [98, 112] is a complete development system for two-level morphology, including a compiler for two-level morphological description. Two-level morphological analyzers for various languages are available in literature — [120, 73, 111, 4, 18, 119, 105]. Two-level rules with negative rule features has been described in [14]. Ritchie et al applied the principles in the design of a rugged front-end lexical processor for a GPSG based NLP system in the form of a tool that may be applied to any language, rather than being language specific [144]. Our ideas on morphological analysis have been considerably influenced by Ritchie et al. The LILOG system [81] also talks about two-level morphological description [149]. A good review of finite-state morphology in general and Koskenniemi's formalism in particular may be found in [69]. Textbooks or reference books dealing with the two-level approaches in depth have not come to our notice. However, Gazder et al [71, 72] have made passing references to the two-level formalism .

We would like to make a particular reference on morphological analysis related work carried out in the LILOG project that have come to our notice quite recently. Various aspects of the project are described in [81], of which, we refer to the paper [149] here. This work uses two-level specifications (called *LILOG/2LM*) for morphological analysis as part of the linguistic development environment (called LEU/2). It is mostly based on the two-level analyzer of [13]. The interesting point of note is the similarity between the concepts described in the above paper and lexicon handling principles discussed in the present work. *LILOG/2LM* is partitioned into two layers — one containing information relevant for higher level processing, and another where morphological and higher level description are related. We have used a similar approach where we have a "Comprehensive Lexicon" of morphemes that store higher level information an an augmented finite state automata (AFSA) and a built-in unification component for establishing relationships. While

in LILOG/2LM they talk of "morpheme lexicon", "morpho-syntax" rules, "lexical rules" and "two-level" rules, we talk of almost similar concepts. The differences are, first in LILOG/2LM they have two-level rules while "Spelling Rules" described here are somewhat different. Secondly, morpho-syntax rules in LILOG may be recursive while we propose non-recursive rules. Finally, in LILOG, lexical rules serve to produce the feature structure of words while in our formalism, the features of morphemes are explicit in the comprehensive lexicon. Unlike the rule-based feature structure generation, in our formalism, the feature structure is obtained through straightforward unification of the features of constituent morphemes.

Although our morphological formalism has similarities with the two-level framework, there are some fundamental differences. We have made a reasonable assumption that morphological deformation during conjoining of morphemes propagate (either or both ways) from the conjoining boundary. This permits us to do away with the diacritic symbol '+'. Moreover, we can do with an augmented version of a classical finite automata unlike (for example) in LILOG/2LM, where three sets of automata — "lexical tree", two-level automata and "morpho-syntax" tree have to work in tandem. We possibly pay the price by being less general and also keep open scopes for backtracking. However, our system is inherently much simpler, takes less storage space and presumably would run faster if backtrackings are intelligently handled.

There have however been works on morphological processing of languages not using the two-level philosophy. Blank et al [19, 20] have experimented with a new kind of finite-state automaton. Nearer home, reports of morphological analysis of Sinhalese [80] are available. Kay's work on Arabic morphology [110], which is non-concatenative, is based on principles apparently beyond the scope of the two-level approach. Our technique is somewhat intermediate between Kaplan et al and Koskenniemi, where we tackle inherent parallesims in the individual networks of Kaplan et al by augmenting the capabilities of finite-state networks (see Section-2.3.2).

B. Syntax Related Works

Perhaps the earliest attempt towards introducing a mathematical structure to linguistic analysis was provided by Chomsky [37, 38]. His ideas influenced the development of a branch of computer science under the banner of Formal Language Theory. The central theme of most earlier NLP systems was the context-free grammar (CFG). As a technique, the CFG is very popular even today, partly due to its closeness to intuitive human understanding of the linguistic apparatus and partly due to certain interesting computational properties of the CFG. The theory of

parsing with CFGs was developed at length by Aho et al [1, 2] and Hopcroft et al [82]. Early [60] developed one of the first parsers for natural languages based on a CFG. He used a data structure called the "Active Chart" for the purpose. Use of the active chart in parsing, or "Chart Parsing" strategies were quite popular in the 1970s. The MIND system of Kay [106] is one example of practical chart parsing. Tutorial introduction to chart parsing is available in most text books on NLP including [6, 71, 72, 178]. Some more earlier systems based on context-free approach may be found in [175, 146, 177]. More recent efficient context free parsing strategies/ formalisms have been suggested by [7, 68, 167, 161], etc. A Japanese sentence analyzer based on CFG has been investigated in [123]. We must however point out that all the above strategies employed additional techniques of varied complexities to take care of some of the non-context-free properties of the respective target languages. Continued interest in finite state devices for syntactic parsing have been shown in [19, 28, 62]. A deterministic parser for English using three symbol "lookahead" was proposed by Marcus [122]. Horn clause based parsers were introduced by Pereira [138]. Petrick [141] has presented a NLP perspective of parsing issues. Some of the above approaches have found their place in text books like [33, 179, 178], etc.

The standard form of the CFG has some obvious inadequacies vis-a-vis natural languages. For example, the concept of "subject-verb agreement" can not be encoded in a CFG. Also, capturing certain straightforward linguistic generalities like a selectional restriction on the nature and number of complement of the verb (with respect to an English like language), require a many fold increase in the size of the required grammar. Some linguists believe that a theory that hypothesizes a similarity with human linguistic behaviour, must also take care of certain other factors. Such theories must propose identical internal representation for similar meaning utterances, irrespective of their actual surface forms. In this regard, the case of English active and passive forms of similar meaning sentences has been a popular benchmark. Clearly, classical CFG is quite incapable of abstracting the above factors. To overcome the various inadequacies of the CFG, linguistic and NLP research has diversified in the quest for a reasonable solution. A school of thought known as "transformational syntax" introduced the concept of different types of transformation as the mechanism that relate a context-free internal or deep structure and the surface or s-structure. Since the early 1980s, the transformational school led by Chomsky have reasonably stabilized. This school, characterized by the components like X-bar syntax, Extended Projection Principle, Government, Bounding theory, etc., currently propose a single "movement" operation called *move-*α. All transformations are proposed as manifestations of this single movement operation. The above principles are currently known as the "Government and Binding" (GB) theory. A description of the theory may be found in [39]. Recently, the NLP researchers at the MIT have shown interest in developing NLP

systems based on the GB-theories. Berwick et al [15] believes that such systems based on "Principle Based Parsing" would be the state-of-the-art of NLP research in the 1990s. To prove the universality of their approach, they have also provided analysis of Warlpiri based on their formalism [102].

A major procedural technique for augmenting the power of context-free based parsing strategy is the "Augmented Transition Network" (ATN). The ATN is a systematic extension on the concepts of the recursive transition network (RTN) which is the graphical representation of the CFG. The ATN mechanism, first proposed by Woods [181], is perhaps the most popular parsing strategies that still evokes interest among NLP researchers. The main contribution of ATNs is the introduction of the notion of "register", assignment and tests into network notation, much like programming languages. Registers are defined for each constituent and using them, general tests on acceptability (e.g. gender, argument) and flexible building of output can be made. These aspects make ATNs as powerful as Turing machines. A good survey of ATNs may be found in [12, 33, 178, 71, 72], etc. Further extensions on ATNs were suggested in [183].

A parallel school of thought to transformational theories did not accept transformational principles and hence came to be known as "non-transformational syntax". The main distinction between the transformational and the non-transformational school is that in the latter, different forms of even similar meaning sentences (for example the active and passive versions of the same sentence) are considered to be different entities. Identical internal representation (of similar meaning sentences) is achieved through other mechanisms. Most modern non-transformational techniques share two common properties— feature based representation of key aspects and an underlying unification component. A corollary for being feature based is conceiving of a more organized and complex lexicon and pushing down of a considerable part of recognition burden to the lexical stage. The syntactic component usually is simpler and more compact. The GPSG, the FUG, the CUG and especially the LFG are some of the more important of the "unification-based" formalisms. The difference among the above approaches lie in the relative degree of complexity between lexicon design and syntax rule design. In GPSG, the distribution is about even. The syntactic component employs a context-free formalism, strengthened with the use of complex "slashed categories" and unification principles are introduced through a type of meta-grammar constructs. In CUG, slashed categories of different types lends the essential power to the method and unification is carried out on them. In FUG, the categories are themselves complex feature structures and syntax rules have a built-in feature unification semantics encoded in them. Since the work described in this thesis is modeled on LFG, the features of LFG are described below in more detail, followed by a general review of unification and its use in NLP.

The Lexical Functional Grammar

The Lexical functional grammar is a popular syntactic formalism based on feature unification. In FUG [108], unification is an integral part of syntactic rules and in GPSG, HPSG and more recently, CUG have meta-rules to direct the unification process. In LFG on the other hand, the concept of "syntactic structure" of a sentence has been sub-classified as a c-structure and an f-structure. The c-structure is in general a CFG. The f-structure of a (major or minor) syntactic category enumerate its properties in terms of feature pairs, which may be quite complex. The c-structure rules are annotated with f-structure "schema" of various types. A schemata is used to establish logical relationships and dependencies among various constituents of a sentence. Metavariable are used in a schemata to denote f-structures of constituent(s) categories or the category dominating the constituents. Long-distance dependencies are established through a separate class of metavariables. The implementation semantics of LFG is best explained through operators Locate (a type of high-order search), Merge (a feature-unification component) and Include (for set operations). Further description of LFG relevant for the present work may be found in Chapter-4.

The best introduction to LFG and its different aspects are found in [23]. This book contains three parts dealing with lexical representation, syntactic representation and cognitive processing respectively. Interestingly, discussions on languages other than English (Icelandic, Russian, Malayalam, Romance, etc) have also been taken up. The popularity of LFG is evident from the bulk of reported work on various aspects of the formalism. In addition to Bresnan's account on lexical description [24], Zaenen et al [184] have provided an excellent overview into feasibility study of 'polytheoretical' approach to lexicon-development. Attention has been primarily on the problem of "sub-categorization" and a comparison between GB-theory and LFG in this respect, has been provided. An implementation of LFG in PROLOG has been discussed in [61]. Use of LFG in MT has been discussed in Klaus et al [114] and Kaplan et al [96]. Incidentally, the use of FUG in MT has been considered by [108]. Semantic analysis in LFG has been taken up by [76, 77, 78]. Constituent co-ordination in the LFG environment has been considered in [92, 94]. LFG usage in syntactic constraints on anaphoric binding has been the topic of interest of Dalrymple et al [41, 42, 43, 166]. The formal architecture of LFG has been taken up by Kaplan [93] and Fenstad [63]. "Functional Uncertainty" is one recent subtopic of LFG and has been taken up by Kaplan et al [89, 90, 91, 97]. Functional uncertainty concepts have led to the incorporation of regular expressions in place of single feature names in schemata.

Unification in NLP and Implementation of Unification

Unification as an operation in automatic theorem proving was introduced by Robinson [145]. It was first used in CL/ NLP by Kay in FUG. LFG is another important unification based approach. A good review on use of unification to NLP may be found in [115]. There has been a rapidly growing literature on categorical grammars with unification formalism like [22, 29, 180, 185], etc. The underlying syntactic formalism in the LILOG project is also based on categorical grammars with unification which the designers call STUF (Stuttgart Unification Formalism). The GPSG is based on DAG unification. Feature unification concepts were introduced to augment GPSG to HPSG [142]. Various workers [139, 103, 132, 127] have investigated into the underlying mathematics and logic of unification based formalisms. Barton [11] et al have analyzed the complexities of LFG and GPSG and Ritchie [143] has done the same for FUG. The formal semantics of unification formalisms have been taken up in general by [63] while [93] has discussed the formal architecture of the LFG in greater detail.

A good survey of unification implementation strategies may be found in [115]. The unification process in FUG has been described by Kay [109]. The PATR/PATR-II framework, which is the basis of GPSG/ HPSG based implementations, has been introduced in [161, 162] and extended in the D-PATR system in [101]. A graph-unification based representation is one of the central themes of STUF employed in the LILOG project. The 'language' of STUF, based on a version of categorical unification grammar [169], has been taken up in [22, 57, 58] while the issue of parsing of STUF grammars have been taken up in [150]. As have been pointed out by most researchers, the complexity of the unification problem with disjunctive constraint satisfaction is in general exponential. Methods of performing disjunctive constraint satisfaction in a more efficient manner have been considered by Kasper [104] and Maxwell et al [124]. An interesting innovation used by Maxwell at al is based on using the conjunctive normal form (CNF) instead of the more commonly used disjunctive normal form (DNF). Maxwell et al [125] have discussed the interface of the phrasal and functional constraints in an LFG environment where they have discussed about various techniques like 'non-interleaved pruning', 'factored extraction', 'factored pruning', etc. as some efficiency enhancement strategies.

C. (A Brief) Review of Semantics Related Works

Conceptual Dependency by Schank et al [148], is an elegant tool of graphically expressing sentential semantics through a set of eleven primitive actions. Semantic Grammars [79], were attempts at semantic interpretation directly from the sen-

tence. Allen [6] has concentrated more on "logical forms" — a variation of first order logic. Fillmore [64, 65] and others suggested an alternative philosophy in Case Grammars which is mostly rationalization of traditional syntactic ideas with classical theories of case. GB-theory [39] in its present form proposes a three-tier semantic description based on morphological case (morphological feature), grammatical function or gf (syntactic level feature) and thematic or theta-role (semantic level). This three-tier definition is the currently accepted notion of sentence analysis. Wilks [176] laid the foundations of "preference semantics." Allen [5] is interested in the analysis of intention in utterances. Semantic analysis of natural languages is closely related to knowledge representation studies of AI. The "semantic network", whose foundation was laid by Woods [182] and later formalized by Minsky [129], is a formalism widely used for the purpose. A review of knowledge representation vis-a-vis natural languages may be found in [173]. The *Tense* of the main verb is known to have considerable importance in semantic analysis. This was highlighted in papers [130, 133, 136, 172] appearing in *Computational Linguistics* **14**(2), which was a special issue on tense. There exists an alternate school of computational linguistics that concentrate on verb centric parsing as the central strategy. The main consideration here is on the different "types of processes" described by verbs. Types of process have been extensively studied by Halliday [75], while Kholdovich [113] considers one aspect of verb-centric parsing.

D. Review of Some Relevant Works on Indian Languages

Tradition of linguistic research in Sanskrit dates back to millennia. There had been references to grammar even in the *Vedas* and the *Upanishads* (10th century B.C to 4th century B.C.) . However, *Panini* (5th century B.C.) is considered to be the father of Sanskrit grammar. The *Sutras* or the *Ashtadhyayi* (so called because it contains eight chapters; **Ashta**=eight and **Adhyaya**=chapter) are succinct aphromisms of Sanskrit grammar which are attributed to Panini by universal consent. These are the first attempt in the history of the world to describe and analyze the components of a language on scientific lines. They are not only foremost specimens of Descriptive Grammar but also acclaimed as a notable manifestation of human intelligence. Certain inaccuracies in Panini's works were corrected by *Katyayana* (around 300 B.C.). A voluminous commentary called the *Mahabhasya*, meant to elucidate the text of Panini and other grammarians, is ascribed to *Patanjali* (150 B.C.). A good collection of works by well-known researchers on Sanskrit grammarians from 7th century A.D. (like Hiuen Tsang, the Chinese pilgrim who visited India during the time of Emperor Harshavardhana) till a couple of decades ago, may be found in [165]. A treatise [170] on Panini's *Ashtadhyayi* in English makes interesting reading. Relevance of Paninian principles in Universal Grammar

was considered by [164]. A survey of research on Panini may be found in [32]. Recently, there has been a spurt of interest in Paninian principles and Sanskrit vis-a-vis NLP and knowledge representation as is evident from [26, 86].

The ideas described in the present thesis has been greatly influenced by S. K. Chatterjee, one of the greatest linguists of modern times. His compilation on the origin and the development of Bangla [34] is a must read for anybody involved with linguistic study of Bangla. His text on the essentials of Bangla grammar [36] and for learning Bangla as a foreign language [35] have been two major sources of information for the present work. In the field of modern computational linguistics, some work have been done on different sub-paradigms. P. Dasgupta has published works on Bangla nouns [51], a class of nominal inflectional classifier [50], verb phonology [49], gerundative noun phrases [46], compound verbs[45], and Bangla relative clauses [47]. P. Dasgupta has also written a book on the importance of grammatical functions [52], which has influenced the present work to quite some extent. His works on phonology have been followed up by Dey [54] and Paul [137]. The spelling rules in Chapter-2 have mostly adopted from [137]. Recently Dasgupta M. [44] have considered certain aspects of Bangla composite verbs. A good study of negation in Bangla in general may be found in Singh [163]. A colleague from our own group has investigated linguistic aspects of verb-centric parsing of Bangla, centered around the concept of the "valency" of verbs [9].

The National Centre for Software Technology, Bombay, has been assigned with developments in the field of NLP under the "Knowledge-Based Computer Systems" project. The issue of a common core grammar for Indian languages was taken up in [151] A report by the Dept. of Electronics, Govt. of India [55] indicates government interest in the field. The IIT Kanpur has undertaken projects on MT between Indian languages. Reports on their work may be found in [16, 17, 147]. They have used a Paninian model for parsing Indian languages. An LFG implementation for an Indian language (Telugu) has been discussed in [160]. Various aspects of NLP with Indian languages were discussed in seminars [171] and [40].

To the best of our knowledge, the present work is the first concerted effort towards morphological and syntactic processing of Bangla.

Motivation for the Present Work

The importance of Natural Language Processing for Indian languages is justified by the sheer numbers of people speaking these languages. The term 'Indian language', covers the entire Indian sub-continent, i.e. India, Pakistan, Bangladesh, Nepal,

Sri Lanka, Bhutan and the Maldives with total population exceeding one billion! Moreover, the languages of many Pacific and South Asian countries like Indonesia, Kampuchea, Mauritius, etc. are very similar to the Indian languages. Even the one chosen as a case study, namely Bangla or Bengali, has a speaking population exceeding two hundred million. It is the state language of Bangladesh and the dominant language of at least two provinces of India — West Bengal and Tripura.

The work described in the present thesis is a part of a more fundamental project of proposing and implementing a computational model for comprehending modern Indian natural languages. The present work is directly concerned with the development of sub-systems for a)**Lexical** and b)**Syntactic** levels of analysis. The intention is to make the models independent of the target language. To this end, rather than proposing actual models, the approach has been to devise formalisms that lead to direct implementation of *software tools* for automatic construction of analyzers at different levels, from some basic *specifications*. Bangla has been chosen as the immediate target and the fundamental capabilities of the tools have been decided upon from examples in Bangla. To highlight the effectiveness of the tools, lexical and syntactic specification for a workable sub-set of Bangla has been carried out.

An Overview of Indian Languages and Bangla

All ethnic Indian languages (except perhaps Urdu, which has considerable Arabic influence) have in some way or the other originated from Sanskrit. However, the modern versions of almost all the languages have deviated considerably from the parent language and have developed their own idiosyncrasies. Nevertheless, being originated from a single source, certain basic philosophies of Sanskrit language have been retained. Sanskrit is an *agglutinating* language. Words tend to form conglomerates. It is highly inflectional. The grammar of Sanskrit has a strong semantic bias. The basic philosophy behind the concept of "grammaticality" of a Sanskrit sentence is:

- A sentence conveys a coherent piece of communication, i.e., every entity of a sentence satisfies mutual expectancy.
- The essence of communication is established by the verb or **Kriya**-pad.
- A rigid system of inflection establishes a near one to one correspondence between a nominal inflection and the functional role that the noun inflected plays in a sentence. The mapping from grammatical functions to thematic role (**Karaka**) is quite simple.

- Well-formedness of a sentence in almost never altered by a permutation of the entities.

Modern Indian languages have lost the agglutinating property of Sanskrit to quite an extent but have retained the strong inflectional characteristic and the basic philosophy behind grammaticality. The participating entities are in their most general form a sequence of words. The basic semantics of a participating entity is carried by a noun (or pronoun) that acts as the "head" (in the sense of X' syntax) of the entity. The inflection on the head noun may be an affix/declension, or it may occur as a separate "particle" word immediately following the head. In most Indian languages, the separate inflection plays the role of a "positional" case marker. Due to clear similarity with English prepositions but to signify its position vis-a-vis the head noun, such a particle is called a post-position. The qualifier(s) for the head noun in Sanskrit usually precede the head in order, but may be agglutinated. In the modern languages, this order criterion is maintained. A participating entity in a sentence in a modern Indian language is therefore:

- In general, a sequence of words
- Is case marked by an affix or declension or a post-position (or even both)
- Has a rigid order of occurrence of the constituent words.

A participating entity without a post-position will be called a Noun Phrase (NP) and one with a post-position will be called a post-positional phrase (PP).

Just as in Sanskrit, where word order is not important in grammaticality, in most Indian languages, phrase order is considerably free. Again, as in Sanskrit, the grammatical function (i.e., Subject, Object, etc.) of a phrase is mostly determined from the inflection on the head in an NP or the post-position in a PP.

The language of Bangla is normally used in two forms. The first form, called *Sadhu Bhasa* (**Sadhu** in the sense of "better" or "pure"), or "Formal" version, is more traditional. The *Chalit Bhasa* (**Chalit** meaning "common') is the more commonly used form. At present, the use of the formal version is restricted to sombre texts and sometimes editorials of newspapers. Actually, the line of difference between the two forms is quite fuzzy except in the inflected form of verbs. "Sadhu" forms of verbs are normally formed out of straightforward concatenation of morphemes. "Sadhu" verb declensions are generally longer in length. The "Chalit" form declensions are simpler, generally shorter in length than corresponding formal forms. Also, different types of spelling deformations, mostly due to phoneme (vowel) harmony, take place during morpheme conjoining. In the present work, we have taken

up "Chalit" forms of verbs in addition to the "Sadhu" forms. An efficient morphological analyzer is therefore necessary and has been proposed in the form of the AFSA (see Chapter-2).

Design/Programming Platform Used

Throughout the thesis, we have attempted to provide an object-oriented design of the various analytical components. This way, a "top-down" view of the entire system is immediately visible. Various "objects", abstracting different facets of the design, have been introduced. In some cases, the description of the objects have been minutely detailed to highlight their significance in the design. However, aspects of behavioral design of certain objects have been kept at a more abstract level. Inter-object interaction has been assumed to be of procedural, rather than of functional nature.

During implementation we have used C++, an object-oriented programming language with growing popularity. Most aspects of our design have corresponding C++ constructs, with the result that implementations carried out are easy to comprehend and extendible with minimal effort.

Organization of the Thesis

The contents of the thesis may be broadly divided into three major parts. They are:

Part I: Lexical Division. This part consists of two chapters, Chapter-2 and Chapter-3.

Part II: Syntax Division. This part consists of one chapter, Chapter-4.

Part III: Application Division. This part consists of three chapters, Chapter-5, Chapter-6, and Chapter-7.

Lexical Division:

This division is concerned with the development of a formalism that is useful for building an LFG specific lexical sub-system for an Indian language, particularly

Bangla. The idea is to create a software *tool* that may be used by a linguistic expert to systematically specify and guide the building up of the lexicon.

Chapter-2 . This chapter deals with the development of a formalism for the creation of a *word-level* parser for Bangla. Words here are assumed to be concatenatively constructed out of morphemic primitives. It is expected that the linguistic expert would *specify* the lexicon in four levels:

i. Specification of different morpheme classes.

ii. Specification of rules of morpho-syntax.

iii. Specification of a set of spelling rules.

iv. Specification of a list of morphemes (constituting the vocabulary).

The different specifications would be temporarily stored in different flat files and a "word-level" lexicon would be *generated* or *compiled*, from the knowledge stored in the files. The said lexicon would be represented at two different levels — a comprehensive lexicon containing all possible information for every morpheme specified, and a recognition/generation system for words in the form of an **Augmented Finite State Automata** (AFSA). The AFSA would parse an input word into the constituent morphemes and perform LFG-type *lexical projection* of the word as a *union* of the lexical projections of the constituents.

In this chapter, various aspects of the specification formalism and the process of lexical compilation has been described in depth. An object-oriented design of the different components has been provided.

Chapter-3 . In this chapter, it has been observed that there are a considerable number of lexical entities in Bangla common usage that span across multiple words. The classical LFG based techniques for handling multi-worded lexical items in English-like languages has been shown to be unsuitable for Bangla in many cases. A formalism has been suggested in this chapter for handling lexical entities spanning across any number of words (trivially one). The suggested analysis technique is called "supra-lexical" level of analysis. The basic philosophy in this chapter is similar to that of Chapter-2. Hence, a set of tools meant for the linguistic expert to provide supra-lexical specification has been described and compilation of the specifications into C++ code fragments discussed.

Syntax Division:

This division is concerned with a syntactic formalism suitable for Bangla that is primarily based on the LFG formalism. There is a single chapter (Chapter-4) in this division.

Chapter-4 The traditional LFG has a well-defined implementational semantics through the operators Locate, Merge and Include. However, there are certain built-in assumptions in LFG regarding syntactic encoding of grammatical functions for non-configurational languages. It has been shown in this chapter that the assumption "inflectional case marking is strong and non-ambiguous" is quite valid for Bangla. An excessive use of "alternations" becomes necessary as a result, causing a rapid degradation in the performance of the parser. It has been farther noted that if syntactic encoding of grammatical functions of participating noun phrases of a sentence/clause be sufficiently "delayed", the number of alternations could be reduced to a manageable level. A technique has been proposed in the chapter whereby:

i. An "under-specification" meta-variable has been introduced, that serves to "delay" encoding of grammatical function of participating noun phrases.

ii. A "Symbol Table" as an additional data structure aid for the parser has been proposed.

iii. A lexicon-driven "meta" functional structure or m-structure has been introduced.

iv. Certain alterations and additions in the semantics of the traditional operators have been suggested and a new operator called Search has been introduced. The modified operators systematically inter-relate the c-structure, the f-structure and lexical projection of words.

The basic idea of the proposed formalism is "delaying" evaluation of syntactic encoding till enough lexical information (in the case of Indian languages, the lexical projection of the verb) has been projected such that the encoding can be done with less ambiguity. The symbol table serves the purpose of delaying the evaluation. The under-specification meta-variable stores unevaluated functional forms in the symbol table while the m-structure guides belated evaluation of the stored unevaluated forms. The described formalism has been given the name *Generalized Lexical Functional Grammar* (GLFG).

In this chapter, Bangla "simple" (i.e. with one verb) sentences have primarily been studied while building up the formalism. However, to demonstrate the general nature of of the techniques, a class of complex Bangla sentences has

also been analyzed. The resulting formalism for Bangla clearly highlights a "verb-centric" approach. A systematic object-oriented design of the various operators has been taken up.

Application Division:

This division is concerned with the actual application of the formalisms developed in the first two divisions to Bangla. Three aspects have been studied in the following three chapters:

Chapter-5 . Since verb is at the kernel of a verb-centric approach, this chapter has dealt with the study of the verbal paradigm of Bangla to a reasonable depth. After providing a brief introduction to the classification and ontology of Bangla verbs, it has been pointed out that the analysis of different verb forms of Bangla is primarily providing their supra-lexical specifications. A few commonly used forms have been considered case by case accordingly.

Chapter-6 . This chapter in concerned with the analysis of Bangla Noun Phrases and Post-positional phrases. Not much state-of-the-art (i.e. computational linguistic based) research material on Bangla NP and PP is available in literature. In this chapter, certain amount of linguistic insight into various NP and PP forms of Bangla has been provided. The necessary lexical, supra-lexical and syntactic specifications have been given.

Chapter-7 . In this chapter, some commonly used sentential forms not considered in Chapter-4, has been discussed. The syntactic specifications of the forms discussed have been provided using traditional LFG tools as well as tools proposed in Chapter-4.

Part I

Lexical Division

Chapter 2

The Lexical Sub-System

The Lexical sub-system constitutes the basic building block of our proposed GLFG based syntactic system described in Chapter-4. The success of such a system depends very much upon an efficient lexical projection mechanism, including projection of m-structure (see Chapter-4) schema by verbal lexemes of the languages under study. The proposed lexical sub-system for inflectional Indian languages is primarily concerned with *parsing* surface forms (i.e. the form in which the words are spoken or written) of words into its constituent morphemes so as to obtain lexical projections of words as a *union* of the projections of the constituent morphemes. However, the proposed design is also capable of generating of surface forms of words from a sequence of lexical morphemes or *lexemes*. A major achievement of our formalism is near maximal compactness of lexical representation. We have proposed a *lexical specification* scheme which leads to a compact *lexical representation* for an Indian inflectional language. The representation scheme is an easy to understand formalism meant to be used by a linguistic expert to *specify* the morphological structure of an inflectional language that automatically leads to the building up of efficient *automata* for parsing of words. The automata in turn produces lexical projection in the form required by GLFG. Although the formalism is specially tuned to a GLFG based platform, it is general enough to be adapted to other models of syntactic analysis.

In English-like non-inflectional languages, words are considered as atomic lexical units. The lexical sub-system, or *Lexical Analyzer*, of an LFG based NLP system for such languages may be described by Fig-2.1. A word 'pumped in' by the syntactic component may be checked against the vocabulary of words by a *Finite State Automata* (FSA). A well-formed word always leads the finite control of the automata to a 'terminal' state. The terminal states of the automata are 'indexed' to the lexicon proper from where the lexical category of a word and its f-structures

may be recovered and 'projected' back to the syntactic component.

For inflectional languages, words are constituted of more elementary lexical units called *morphemes*. Usually, the overall lexical projection of a word in an inflectional language result out of union of the projections of the constituent morphemes. Just as 'syntactic' rules govern the formation of sentences from phrasal units, *morpho-syntactic* rules govern formation of words from morphemic units. In languages like Arabic, a morpheme may get *embedded* in other morphemes during word formation (Kay [110]), leading to non-concatenative morpho-syntactic rules. However, most Indian languages have concatenative rules for morpho-syntax, i.e. the constituent morphemes of a word occur side by side. Nevertheless, there are no obvious delimiters indicating 'morphemic boundaries' in words. The automata for parsing a word in an Indian language must therefore be capable of identifying morphemic boundaries. However, due to the concatenative nature of the morpho-syntactic rules, the 'power' of such an automata remains in the order of an FSA. Lexical Analysis for inflectional languages is best explained by Fig-2.2.

2.1 General Background

2.1.1 Morpho-Syntax of Indian Languages

Morphological structure of words in modern Indian languages mostly follow the tradition of classical Sanskrit grammar.The morphemes constituting a word are of two types:

Stem: The essential or root form of a word that provides the meaning.

Affix: Morphemes which in isolation do not convey any meaning but provide proper linguistic perspectives for the stems or other suffixes to which they are conjoined. Depending on position of conjoining, an affix is a *prefix*, an internal *affix* or a *declension*.

Words are produced as a result of several conjoinings:

a. Between a stem and an affix or vice versa.

b. Among two or more affixes.

c. Between two stems, thus producing *compound* stems.

d. By some combinations of the above.

A word is constituted of an optional prefix (acting as an adjectival particle for a nominal or adjectival stem), a simple or compounded stem, zero or more internal affixes and an optional terminating declension. Every constituent morpheme of a word contributes to its overall linguistic property. The rules of morpho-syntax determine which stem can be followed by which affix and/or declension. For this purpose, the morphemes are partitioned into several classes. The morpho-syntactic rules restrict the conjoinings among the various morpheme classes. In most Indian languages including Bangla, there is another class of rules of generative morphology called *spelling rules*, that are concerned with morpho-phonemic or morpho-graphemic restructuring of symbols at the boundaries of two conjoining morphemes. The spelling rules makes the job of detection of morphemic boundaries more difficult.

The major affix classes in Bangla (as well as most other Indian languages) are verbal and nominal declensions called **bibhaktis**, verbal and other internal affixes called **pratyayas**, nominal prefixes and suffixes called **anusargas** and **upasargas**, etc. The stems are also classified into several basic clusters like *verb*-stems, *noun*-stems, etc. The morpho-syntactic rewriting rules are based on these classes, as in:

a) VERB $\longrightarrow$ Verb-Stem [Caus-Affix] Verb-Decl

b) NOUN $\longrightarrow$ Noun-Stem [Def-Affix] Case-Decl

where Caus-Affix is the verbal causational affix class, Verb-Decl is the verbal declension class, Def-Affix is the nominal definiteness affix class and Case-Decl is the nominal case declension class. The items in brackets are optional. The entity on the left side of $\longrightarrow$ represents the *lexical category* of the word. Thus, in rule a) above, the word is a VERB. The basic meaning of the word is provided by the verbal stem. The causational affix, if present, makes the verb a causated verb. The declension provides information about the tense, gender, number, person, honorific value, etc., of the word. In rule b), the word is a NOUN. The definiteness affix, if present, makes the word a 'definite' noun. The case declension determines the 'case' of the noun (like DATive, POSSessive, LOCative, etc.).

Rules may be more complex rules like rule c) given below:

c) Noun-Stem $\longrightarrow$ Verb-Stem Causational-Affix

which indicates that Noun-Stem itself could be a derived morpheme, formed from a verb stem conjoined with a causational affix. However, we would not permit

such rules where derived categories may be used on the right hand side of the $\longrightarrow$. Rather, we would require that rules like c) be "multiplied out" in all other rules where the (derived) morpheme class (Noun-Stem here) occurs on the right hand side. Thus by combining rule c) with rule b) we get rule c′:

c′) NOUN $\longrightarrow$ Verb-Stem Caus-Affix [Def-Affix] Case-Decl

For example, **khel** (play) is a verb stem and **a'** is the verb causational affix. The form **khela'** (game) may be used as a nominal stem and may be conjoined with a definite- affix **t'a'** and a case-declension **ke** to get **khela't'a'ke** (game-DEF-DAT).

In many cases, a declension may be NULL, i.e. with no overt manifestation in the surface form of a word. Depending upon what the remaining morphemes are, a NULL morpheme projects its own lexical properties. For example, in Bangla, a NULL declension on a VERB, indicates second person, imperative mood with casual honorific value.

As a result of the spelling rules, the surface form of a word is not a simple joining of a morpho-syntactically consistent list of constituent morphemes. Indeed, some of the spelling rules for a particular language may be quite complex. Often, one spelling rule may affect (or be affected by) the decision of the rule at some other morpheme boundary.

2.1.2 Related Works

Finite-state approaches for morphological processing are not uncommon [73]. Kaplan and Kay [87] proposed a series implementation of finite state automata for translating the lexical form (i.e. a sequence of lexemes constituting the word) of a word to its surface form (and vice versa). Although the algorithmic feasibility of their model was discussed, it leads an uncontrolled multiple sub-path generation in the equivalent automata. Koskenniemi [116, 117, 118] has argued that the resultant automata in Kaplan and Kay's model may become prohibitively large. He proposed a *two-level* approach towards word form production and detection, where the automata will run *in parallel.* A review of the two level approach may be found in [69]. A rugged framework for lexical description using the two-level approach as the kernel, was suggested by Ritchie et al [144]. A recent description on morphological processing from LILOG scheme [149], with an approach similar to ours, has come to our notice. The above approaches have been also reviewed in Chapter-1.

In the Indian context, despite the fact that many reported NLP formalisms make

extensive references to a lexical sub-system, the real problem of tackling lexicons of reasonable size have not been seriously tackled. As a result, the work presented in this chapter appears to be a pioneering concerted effort of its kind. A description of the ideas described here may also be found in [154, 155].

2.1.3 Overview of Proposed Formalism

In our proposed formalism, the basic idea is to unite the lexicon, the lexical description and the surface description into an integrated system. The major aspects of the formalism are:

- A **stored lexical knowledge base** for morpheme classes and their inter-relationships.
- A stored knowledge base for **spelling rules.**
- A **Lexical Specification** (or Lexical Description) phase during which the lexicon writer imparts linguistic knowledge to build up the above knowledge bases.
- A **Representation** phase in which the knowledge in the knowledge bases is used to generate the following two levels of lexical representation:
  - A *Comprehensive Lexicon* for every morpheme. The comprehensive lexicon contains every relevant lexical knowledge (in the form of f- or m-structure schema) for each morpheme in the assumed vocabulary of the language.
  - A formulation of the rules of morpho-syntax and derivational morphology in the form of an Augmented Finite State Automata (AFSA).

The major problem for an automata for parsing words is the detection of morpheme boundaries. There are two interacting regular grammars governing word formation — one for morpho-syntactic rules and another for formation of morphemes from alphabetic symbols. An overall automata for parsing words must incorporate both the above grammars. In our proposal, the AFSA accomplishes that, with a built-in capability for detecting morpheme boundaries. The AFSA can not only parse words into constituent morphemes but can also generate the corresponding surface form from the given lexical form of a word. Special nodes called *active* nodes of the AFSA have *pointers* leading into the comprehensive lexicon, using which all

relevant information about the constituent morphemes of a word are recovered during parsing.

The proposed recognition system is intermediate between the approaches of Kaplan et al and Koskenniemi. Here, during lexical specification, the underlying automata are conceived as in Kaplan et al but during representation, the inherent parallelisms are taken care of by extending the power of finite state network. As a result, the size of the overall automata is kept within manageable limits, which we claim to be lower than what is required by Koskenniemi's formalism.

The formalism is presented as a software tool to be used by a linguistic expert for specification of the lexicon of the target language. These specifications are to be '*compiled*' *into the proposed representation scheme.*

2.2 Lexical Specification

We describe here how the linguistic expert should specify the lexicon for a target language. While introducing the specification process, we shall systematically specify a lexicon for a subset of nouns and verbs in Bengali with only the relevant attributes for every morpheme class. A practical system will incorporate many such attributes.

The linguistic expert would provide lexical description in four levels:

i. Specification of *different morpheme classes.*

ii. Specification of rules of morpho-syntax.

iii. Specification of a set of spelling rules.

iv. Specification of a list of morphemes (constituting the vocabulary).

2.2.1 Specification of Morpheme Classes

In this level of specification, the lexicon writer provides the names of different morpheme classes for the language. *All morphemes of a single morpheme class* project f-structure and m-structure (the m-structure is a GLFG feature described in Chapter-4) consisting of an almost identical set of attribute names, differing only in the attribute values. However, individual morphemes may project special schema which mark some special characteristics of the morpheme.

While specifying each class, a set of attribute names, along with a list of permitted attribute values, is provided. If the attribute name begins with a #, it indicates an m-structure and in the place of an attribute name, a two-place path ($gf\ p$), where gf is a *grammatical function* name and p is an agreement feature, must be mentioned. See Chapter-4 for details of the relevant terms. One of the permitted values of each attribute/ path is enclosed within brackets to indicate it to be the default value. The permitted attribute value could also be the character "$", which indicates that the actual value could be any string of characters. The default value of such an attribute is the null string. A STEM or PREFIX directive may be given to a morpheme class to indicate that a morpheme of the class can be the stem or prefix of a valid word. An END directive may be given to a class to indicate that a morpheme from this class can terminate a valid word. The morpheme class NULL has a special meaning and must not be specified.

The declaration syntax is:

```
( ( (<Morpheme Class> <Directive>) ( (<Attribute Name> <Attribute Name1>
                                       (<List of attribute values>)
                                     )
                                     (<Attribute Name> <Attribute Name1>
                                       (<List of attribute values>)
                                     )
                                     ...
                                   )
  )
  ( (<Morpheme Class> <Directive>) ( (<Attribute Name> <Attribute Name1>
                                       (<List of attribute values>)
                                     )
                                     (<Attribute Name> <Attribute Name1>
                                       (<List of attribute values>)
                                     )
                                     ...
                                   )
  )
  ...
)
```

Note:

i. `<Directive>` could be either empty, one or more from `STEM`, `PREFIX` and `END`.

ii. `<Attribute Name1>` is present only if `<Attribute Name>` is # followed by a grammatical function (gf) name like, `SUBJ`, `OBJ`, `IOBJ`, etc.

iii. <List of attribute values> enumerates all possible values of <Attribute Name> or of (<Attribute Name> <Attribute Name1>), as the case may be. One item of the list is enclosed within [and] to indicate the default value.

An Example Specification (for morpheme classes VSTEM, NSTEM, VDEC, VCAUS, NCASE and DEF):

```
(((VSTEM STEM END) ( (VALENCY  ([0] 1 2 3))
                     (PRED ($))
                     (#SUBJ  CASE ([NOM] DAT LOC POSS))
                     (#SUBJ  ANIM ([+] -))
                     (#OBJ   CASE ([NOM] DAT LOC POSS))
                     (#OBJ   ANIM (+ [-]))
                     (#IOBJ  CASE (NOM [DAT] LOC POSS))
                     (#IOBJ  ANIM ([+] -))))
  ((NSTEM STEM END) ( (CAT ([material] abstract instrument place))
                     (ANIM (+ [-]))
                     (PRED ($))))
  ((VDEC END) ( (TENSE (PAST [HABIT] PRESENT ))
                (GNPH (0 1p [2p-0h] 2p-1h 3p-1h 2/3p-2h ))
               ) )
  ((VCAUS END) ( (CAUS (0 [1] 2))))
  ((NCASE END) ( (CASE ([NOM] DAT LOC OBLQ POSS ))))
  ((DEF END) ( (DEF ([YES] NO))))
)
```

2.2.2 Specification of Rules of Morpho-Syntax

Rules of morpho-syntax, also called *Word Grammar* rules, are concerned with the formation of words from morphemes. These rules lead to restrictions on a morpheme of one class following a morpheme of some other classes in a word that must be reflected in the AFSA. (For example, a morpheme of class NCASE may not follow a morpheme of class VSTEM. A morpheme from class VCAUS may follow a morpheme of class VSTEM and in turn may be followed by a morpheme from class VDEC or class NCASE.) For this purpose, we introduce here the notion of a feasible pair of morpheme classes that will be used later during representation.

Feasible pairs: A pair of morpheme classes (M_1, M_2) is a *feasible* pair, if any morpheme of class M_2 can follow any morpheme of class M_1 in a word. A sequence of morphemes $m_1 + m_2$ is a feasible pair of morphemes if $m_1 \in M_1, m_2 \in M_2$ and M_1, M_2 is a feasible pair of morpheme classes. Thus, if $m_1 + m_2$ is a feasible pair of morphemes, m_1 and m_2 may appear in order in (the lexical form of) a word.

A major assumption in our design is that between any different morpheme classes M_i and M_j, at most one pair out of (M_1, M_2) and (M_2, M_1) is feasible. This ensures that during formation of words from word grammar rules, no cyclic reference of morpheme classes are made.

Word grammar rules are specified in the usual manner of specifying syntactic LFG rules, i.e. a lexical category is constituted of (denoted by $\longrightarrow$) morphemes of some classes. The rules may be annotated with f-structure schema. However, the *rules must not be recursive*, i.e. a lexical (i.e. word) category may not participate on the right and a morpheme class may not be on the left side of the $\longrightarrow$ of any rule. An *artificial morpheme class* called NULL may be used as a mechanism to project lexical information when certain morphemes are missing (i.e. they are null). The *design rule checks* associated with the specification of word grammar rules are:

- Out of pairs (M_1, M_2) and (M_2, M_1) of morpheme classes, at most one is *feasible*
- A NULL class must be the last (i.e. right-most) class in rules where they are used.
- The class immediately to the left of a NULL class in any rule must have an END directive specified.

The syntax for rule specification is therefore:

$$W \longrightarrow \underset{LFG\ f-structures}{M_1 \quad M_2 \quad \cdots}$$

Here, W is a lexical category like VERB, NOUN, etc., and M_i-s are morpheme classes. There may be multiple rules for the same lexical category.

The LFG f-structure schema normally refer to only those attributes belonging to the morpheme classes under which they appear (local attributes). But other attributes (global) may also be referred. These rules *project* lexical information of morphemes into a higher level GLFG based parsing system (see Chapter-4).

An Example Rule Specification:

i. $VERB \longrightarrow VSTEM \quad \underset{\substack{(\uparrow\ TENSE) = IMPER \\ (\uparrow\ GNPH) = 2p-0h}}{NULL}$

ii. $VERB \longrightarrow VSTEM \quad VDEC$

iii. $VERB \longrightarrow VSTEM \quad VCAUS \quad VDEC$

iv. $NOUN \longrightarrow NSTEM \quad NULL$
$(\uparrow CASE) = NOM$

v. $NOUN \longrightarrow NSTEM \quad NCASE$

vi. $NOUN \longrightarrow NSTEM \quad DEF \quad NCASE$

vii. $NOUN \longrightarrow VSTEM \quad VCAUS \quad NCASE$
$(\uparrow CAT) = gerund$

viii. $NOUN \longrightarrow VSTEM \quad VCAUS \quad DEF \quad NCASE$
$(\uparrow CAT) = gerund$

2.2.3 Specification of Spelling Rules

An initial assumption of our formalism was that morphological conjoining is *strictly concatenative.* In reality, this assumption happens to be too strong. In Bangla, especially in the common dialect (**Chalit Bhasa**), deformation of symbols around the boundary of morphemes occur in many cases. For example, when the two morphemes **dhu** (Verb stem "wash") and **ben** (A future tense Verb declension) conjoin, the resultant 'surface' form is **dhoben**. Note that the deformation of **u** to **o** in the example is very near the conjoining boundary. To take care of such deformities in our formalism, we assume that words may have two different levels of representation. The representation that we write, read, speak and hear is the *surface* level representation. We also conceive of a *lexical* level of representation in which morphological conjoining is strictly concatenative. Thus the lexical form of the above example would be **dhuben**. A special class of morpho-syntactic rules called *Spelling Rules* govern morpho-phonemic (in spoken form) or morpho-graphemic (in written form) restructuring of symbols at boundaries of conjoining morphemes.

Note: We shall express the lexical level representation of two conjoining morphemes L_1 and L_2 as $L_1 + L_2$.

Paul [137] has closely studied spelling deformities in Bangla, especially of the Verbal paradigm. From their observations, backed up by some of our own, we may list the following salient features of spelling deformity:

i. Any deformity may be characterized entirely by the following 'atomic' operations:

   (a) Addition of a symbol.

   (b) Deletion of a symbol.

   (c) Replacement of one symbol by one or more symbols.

(d) Replacement of more than one symbol by one symbol.

We assume a special symbol $\emptyset$ to denote the 'absence' of a symbol. Using $\emptyset$, all the above atomic operations may be expressed by (possibly multiple uses of) a single operation — replacement of *one* symbol (which can be $\emptyset$) by *one* other symbol (which can also be $\emptyset$). Replacement of $\emptyset$ by $\emptyset$ is superfluous.

ii. Although there are innumerable individual instances of deformities, they can be reasonably generalized. Some of these generalizations have "global" implications and are applicable during a conjoining between *any two morphemes*. Some of them are however "local" to conjoining between pairs of morphemes from certain particular morpheme *class* pairs.

In light of the above observations, we introduce the concept of Spelling rules and specifications thereof in the following paragraphs.

Alphabet: The set of all characters that can constitute a lexical form (resp. surface form) of a word constitutes the alphabet Σ_L (resp. Σ_R). Although most characters of Σ_L and Σ_R are identical, in general, Σ_L and Σ_R are not equal.

An l-r pair: We define $a : b$, where $a \in \Sigma_L \cup \emptyset$ and $b \in \Sigma_R \cup \emptyset$, to be an *l-r pair* or simply *pair*. A **union** $a_1|a_2|\ldots|a_k : b_1|b_2\ldots|b_k$ of pairs represents a disjunctive choice $a_i : b_i$, $1 \leq i \leq k$, from the k possible pairs. We shall call k the *length* of the union.

An R-Expression (RE) is defined as a finite string of unions of pairs. For example, $RE = (a|b) : (x|x)\,\emptyset : y\,c : \emptyset$ is an R-Expression. If there are n unions in an R-Expression RE, it represents all distinct l-r-pairs of length n obtained by opening out the disjunctive choices of the unions. If the strings represented by an R-Expression RE be listed in a list L, we call L the **string list** of RE. In the example taken above, $\{a : x\,\emptyset : y\,c : \emptyset, b : x\,\emptyset : y\,c : \emptyset\}$ is the string list. The name R-expression has been used in the present context to highlight the similarity with *regular expressions*.

LEX and SURF of an RE: Let l be a string of pairs. We define $LEX(l)$ (resp. $SURF(l)$) to be the string formed by taking only the left hand (resp. right hand) symbols from pairs $a : b$ (collapsing all $\emptyset$ symbols) of l. $LEX(RE)$, for an R-Expression RE is defined as the *ordered* list of strings formed by taking $LEX(l)$ (resp. $SURF(l)$) of all $l \in L$, where L is the string list of RE. In the example of the preceding paragraph, $LEX(RE) = [ac, bc]$ and $LEX(RE) = [xy, xy]$. There is an one to one correspondence between individual members of $LEX(RE)$ and $SURF(RE)$ for a given RE. Thus, in the example, ac and bc correspond to xy and xy respectively. Subsequently, whenever $LEX(RE)$ and $SURF(RE)$ will be

co-referred, it will mean that such a co-reference ranges over corresponding pairs from $LEX(RE)$ and $SURF(RE)$. The notions of LEX and SURF will be used in later sections. In the ensuing discussions, we would use the notation RE^L and RE^S to denote LEX(RE) and SURF(RE) respectively.

String Matching: A string t is *tail matched* by a string s if $\mid t \mid \geq \mid s \mid = n$, and the last n symbols of t spell out the string s. A string s *head matches* a string t if s is a prefix of t.

Spelling Rule: A *spelling rule* is a template of the form $RE_1 + RE_2$, where RE_1 and RE_2 are R-Expressions. The character + represents the abstract morpheme boundary. Intuitively, a rule $RE_1 + RE_2$ means the following. At the boundary between two morphemes, let the left morpheme tail match and right morpheme head match RE_1^L and RE_2^L respectively. In the surface, the matched portion of the morphemes get translated to the corresponding symbols from RE_1^R and RE_2^R. In other words, a *feasible pair* of morphemes (a notion introduced earlier) $m1 + m2$ matches a rule $RE_1 + RE_2$ and gets translated to surface $s_1t_1t_2s_2$ if and only if $m_1 = s_1r_1$, $m_2 = r_2s_2$, $r_1 \in RE_1^L$, $t_1 \in RE_1^S$, $r_2 \in RE_2^L$, $t_2 \in RE_2^S$ and r_1, t_1 and r_2, t_2 are corresponding pairs.

Often, as a shorthand, a spelling rule rule template may be written as: $\ldots S : T \ldots + \ldots$, where S and T are strings of identical size. This representation is a shorthand for $\mid S \mid$ different rules formed by taking members from S and T in order.

A rule $\emptyset : \emptyset + \emptyset : \emptyset$ called a *trivial* rule, is never explicitly mentioned.

Local and Global Rules: Rules may be defined to be applicable at the boundary between only some specified pairs of morpheme classes. In such cases, the rules are adorned by the pairs of morpheme class. Such a rule will be called a *local* rule. All unadorned rules, called *global* rules, are applicable at the boundaries of all morphemic pairs. In the event of a rule clash between a local and a global rule, the local rule prevails.

Specification of spelling rules consist of a list of local or global rules in the format:

`<Spelling Rule Template>` [at (M_1, M_2)]

where, M_1 and M_2 are morpheme classes. If the "at" clause is present, it indicates a local rule; otherwise it is a global rule. The null symbol is represented by '0'. Additionally, the symbols 'V', 'C' and $\doteq$ represent the sents of all vowels, set of all consonants and the entire alphabet set, respectively. Specifications are given in our phonetic code described in Appendix-A. However, during storage, symbols

are stored in ISCII.

In Appendix-B, we have listed a set of some spelling rules we used for our system for Bangla. The global rules have been adopted from [34] and [35]. The rules applicable at the boundary between verbal stems and verbal declensions have been taken from [45] and [137] with suitable augmentation and modification.

Storing Spelling Rules:

The Spelling rules specified by the lexicon writer is stored in a temporary text file SPELL.IN for use during generation of AFSA in Sec-2.3.5.

2.2.4 Morpheme List Specification

As the final level of lexical specification, the lexicon writer provides a list of morphemes belonging to the subset of the language being considered. A morpheme is specified on the following format:

```
<Morpheme>: ( (<Morpheme Class> (<Attribute Name/Path <Value>)
                                (<Attribute Name/Path <Value>)
                                ...
              )
              (<Morpheme Class> (<Attribute Name/Path <Value>)
                                (<Attribute Name/Path <Value>)
                                ...
              )
              ...
            )
```

This format permits a morpheme to be a member of multiple classes. The list of attributes specified for every class may include all or some of the attribute specified for the class. As mentioned in Sec-2.2.1, if an attribute is omitted in the above specification, the default value of the attribute in the specified class is assumed to be included. The list may also include other attributes/paths not included in the specified class but belong to some other class.

Example:

```
i) pa':  ( (VSTEM (VALENCY 1) (PRED 'get'))
           (NSTEM (CAT instrument) (PRED 'foot')))
```

```
 ii) pa't: ( (VSTEM (VALENCY 1) (PRED 'lay')))
iii) mar:  ( (VSTEM (PRED  'die') (CAUS 0)))
 iv) ma'r: ( (VSTEM (VALENCY 1) (PRED 'kill')))
  v) a':   ( (VCAUS))

 vi) t'a': ( (DEF))

vii) er:   ( (NCASE (CASE POSS)))

viii) ke:  ( (NCASE (CASE DAT)))

 ix) e:    ( (NCASE (CASE LOC) (CAT place))
             (NCASE (CASE OBLQ) (CAT material))
             (VDEC) )

  x) E:    ( (VDEC (TENSE CONTINF) (GNPH 0)))

 xi) te:   ( (VDEC (GNPH 2p-1h))
             (VDEC (TENSE CONDINF) (GNPH 0))
             (NCASE (CASE LOC) (CAT place))
             (NCASE (CASE OBLQ) (CAT material)) )

xii) ten:  ( (VDEC (TENSE HABIT) (GNPH 2/3p-2h)))
```

A few words on the above example specification:

- A morpheme may belong to multiple classes as in i), ix), xi).
- Even while belonging to the same class, there may be *alternations* in the lexical specification as in ix) xi).
- A morpheme may be specified only with its class as in v), vi), ix). In such cases, all default attributes are assumed.
- There is no harm in re-specifying a default attribute value as in xii).
- An attribute not included in the default attributes of the morpheme class of a word may also be specified, as in iii) where CAUS is not a default attribute specified for the VSTEM class.

2.3 Lexical Representation

As discussed in Sec-2.1.3, there are two levels of lexical representation:

- A *Comprehensive Lexicon* for every morpheme.
- An Augmented Finite State Automata (AFSA).

2.3.1 The Comprehensive Lexicon

The Comprehensive Lexicon is basically an indexed database of morphemes. The specification of an individual morpheme is first completed by copying the default specifications from the class it belongs. From this, tuples of the type:

$$< Morpheme\ Class > (< Attribute\ Name/Path >< Attribute\ Value >)(\ldots)$$

are created and stored in the database entry for the morpheme. There may be multiple tuples for the same morpheme. One implementational bottleneck is that the size of the tuples are different. Standard relational database management tools do not provide facilities for storing variable sized records. We have overcome the difficulty in the following manner. We maintain an auxiliary file (SCHEMA file) for storing schema constructs of the type $(< Attribute\ Name/Path >< Attribute\ Value >)(\ldots)$. Now, we have observed that the size of this construct for a standard lexical specification for Bangla is at most three (3). An extra flag must be incorporated for coding constraint schema (existential — code E, negative existential — code N and constraint — code C). The schema database can thus be maintained at a reasonable storage efficiency. Actually, further compaction has been achieved by encoding the morpheme class names, attribute names and attribute values in a master database file (MASTER file). Furthermore, character strings have been encoded through still another code file (STRING file). The main comprehensive lexicon file (LEXICON) has the record structure (M, s, n), where M is the code for the morpheme class of the morpheme indexed to this record. s is the record number in SCHEMA file of the first schemata in the projection of the morpheme and n is the number of schema in the projection of the morpheme. Thus, if a morpheme points to the k-th record of LEXICON file which is (M, s, n), then the projection of the morpheme consists of n schema stored in record number s through $s + n - 1$ in the SCHEMA file.

A database file called MORPH, which is a temporary access file to Comprehensive Lexicon, is also created. This file is required during automatic generation of AFSA described in Sec-2.3.5. This file consists of pairs (m, i), where m is a morpheme and i is an index into the LEXICON file for m. There may be multiple entries for the same morpheme in MORPH file. The file may be discarded after generation of the AFSA.

The actual generation of the database files is quite straightforward if utilities like

Lexical Analyzer Generator (*Lex*) and Compiler Generator (*Yacc*) are used.

Example: Consider the morpheme te specified in ix) of example morpheme specification in Sec-2.2.4. There are four entries for this morpheme in the LEXICON file (and hence four entries in the MORPH file). Of them, let us consider the one in which **te** is a finite verb declension. The complete lexical specification for the morpheme consists of two schema. The relevant entry in the LEXICON file would be $(Code(VDEC), s, 2)$ and entries of record numbers s and $s+1$ in the SCHEMA file would be $(Code(TENSE), Code(HABIT), \emptyset, \emptyset)$ and $(Code(GNPH), Code(2p-1h), \emptyset, \emptyset)$ respectively, where $\emptyset$ indicates blank entry.

2.3.2 Introduction to the AFSA

The *Augmented Finite State Automata* (AFSA) is our proposed tool for parsing words into constituent morphemes. In the normal mode of use, the AFSA accepts the surface representation of a word as input and ultimately generates index pointers into the comprehensive lexicon for the morphemes constituting the word. The lexical projection of the word can be recovered as a result of *unification* of the lexical projections of the constituent morphemes. The lexical representation of the word is trivially obtained in the process. In a less used mode, the AFSA may also be used to generate the surface representation of a word from the lexical representation given as input. The AFSA is automatically compiled out of the various specifications provided by the lexicon writer described in Sec-2.2

We have made a major assumption in designing the AFSA. It is assumed that Morphological restructuring takes place only at the boundaries of conjoining morphemes and spelling deformations propagate continuously *away* from the conjoining boundary. The application of a rule at a boundary is *context-free*, neither affecting nor being affected by an earlier or later application of a rule at some other boundary. Our observations from Bangla show that instances of rule context sensitivity are very rare. They sometimes occur when three or more morphemes conjoin and *a sandwiched morpheme*, being of very small length, is deformed, both from the right as well as from the left. It then becomes necessary to judge in which order the deformations need be carried out since the effect of one deformation may alter symbols at the other boundary as well. In Bangla, there is one major instance of the phenomenon where the verbal causational affix a' is deformed at boundaries with both the preceding verb stem and the succeeding verb declension. This situation has been taken care of by a minor manual *fine-tuning* of the AFSA.

The AFSA consists of a forest of *Directed Acyclic Graphs* (DAGs). Each DAG represents a finite state recognizer for a class of morphemes. However, there is a

single DAG for all STEM type morphemes. The DAGs consist of two types of edges — *lexical* or *l*-edge and *surface* or *s*-edge. Transition along l-edges only from the root node to a terminal node of a DAG recognizes a lexical morpheme. Transition along s-edges however, recognize *one* surface form of some lexical morpheme. The different DAGs in the system are also inter-connected by l- and s-edges. However, the inter-DAG edges are qualitatively different from intra-DAG edges. We call inter-DAG edges *active* and intra-DAG edges *passive*. The active edges encode the morpho-syntactic restrictions applicable for the language specified by the lexicon writer as described in Sec-2.2.2.

2.3.3 Formal Definition of the AFSA

The AFSA consists of a forest of DAGs, where every DAG consists of:

a. A set of *nodes* representing *states*. Nodes are labeled as *passive*, *l-active* and/or *s-active*.

b. A set of *l-passive edges* between a pair of nodes in the same DAG.

c. A set of *s-passive* edges between a pair of nodes in the same DAG.

d. A set of *l-active edges* linking a (terminal) node of one DAG to the root node of another DAG. The word *active* node will be used interchangeably with terminal node.

e. A set of *s-active* edges linking a terminal node of one DAG to a root node of another DAG. Every s-active edge has a *disjunction of one or more* index pointers into the Comprehensive lexicon. Every s-active node is associated with one or more s-active edges. Additionally, if the DAG to which an s-active node belongs represents a morpheme class with the END directive, the node has one *trivial* s-active edge. Unlike normal s-active edges, a trivial s-active edge does not link to any node in a different DAG. However, it has an index pointer into the Comprehensive Lexicon.

f. A set of *associations* connecting an l-active and one or more s-active nodes.

Every DAG has a unique *root l-node* and one or more *root s-nodes*. The root l-node is also a root s-nodes.

2.3.4 Parsing in the AFSA

Input: The surface representation of a word — s

Aim: To recover pointers into the Comprehensive Lexicon of the constituent morphemes of s. The Lexical representation is trivially obtained as a by-product of the analysis.

Data Structures: The AFSA and a *Stack* of quadruples $(dag, node, index, k)$, where *node* is an active node in DAG *dag*, *index* is an index pointer into the Comprehensive Lexicon and latest morphemic boundary is at the $k-1$-th symbol of the input.

Driving Routine of the AFSA:

Step-1 $dag \leftarrow STEM$; $node \leftarrow$ root l-node of the STEM DAG; $k \leftarrow 1$. (Here k points to the character in s currently being scanned). Clear the Stack.

Step-2 If the end of string has not been reached, proceed to Step-3. Otherwise, check if *node* is an s-active node. If not, proceed to Step-5. Otherwise, let $p \leftarrow p_t$, where p_t is the trivial index pointer for *node*. *Push* $(dag, node, p, k)$ in *stack* and *exit* with success.

Step-3 If *node* is an s-active node, non-deterministically decide whether to make an active transition. In the event such a decision is taken, let the chosen non-trivial s-active edge lead to node n in DAG d and let p be the index pointer of the chosen edge. *Push* (d, n, p, k) onto stack. Make $dag \leftarrow d$ and $node \leftarrow n$ and repeat step 3. If active transition is not taken, proceed to Step-4. An active transition is not possible from an s-active node if all s-active edges from the node are trivial edges (i.e. they lead to nowhere).

Step-4 If there is an s-passive edge in DAG *dag* from node *node* to node n on the k-th character of s, make $node \leftarrow n$; $k \leftarrow k+1$ and go to Step-2.

Step-5 If there is no s-active or s-passive transition possible in DAG *dag* from node *node* based on the k-th character of s, or if k points beyond the last character of s, *pop* $(dag, node, p, k)$ from *stack* (where p is a dummy variable) and resume in Step-2. If the *pop* operation fails, *exit* with failure, i.e. declare the input word to be ill-formed.

Output Pointers to Comprehensive Lexicon:

If the driving routine terminates successfully, the *stack* contains r, $r \geq 1$ quadruples $(d_1, n_1, p_1, k_1), (d_2, n_2, p_2, k_2), \cdots (d_r, n_r, p_r, k_r)$. The lexical projection of the word is the *union* of the r sets of schema obtained from the comprehensive lexicon by following the pointers $p_1, p_2, \cdots, p_r$.

Obtaining All Possible Word Parses:

In the form presented above, the parsing algorithm obtains *one* parse of a well-formed word. However, with slight modifications in Step-2, it is possible to get *all* parses for an input word. After a valid parse has been found, control must return to Step-2. No success *exit* is carried out but instead the top-most stack entry is *popped* out and discarded and a branch to Step-5 is taken.

Soundness and Completeness of the Parsing Algorithm:

The parsing algorithm guarantees that a well-formed word will be correctly parsed. However, some non-well-formed structures may still be accepted. To reject such inputs, the lexical structure obtained as output may be fed back to produce the corresponding surface structure. The original input structure is well-formed if and only if it is identical to the surface structure obtained out of the feedback process.

Computational Complexity of Parsing Algorithm:

The worst case time complexity of parsing in AFSA is of exponential order, primarily due to the non-determinisms at various stages. Let there be n DAGs in the system corresponding to $n-1$ non-STEM morpheme classes and one common STEM DAG. The worst case *first level* non-determinism occurs at an active node (which is also a passive node) of the STEM DAG where the next symbol has active transitions to every $n-1$ non-STEM DAG as well as a passive transition. This gives rise to an n level non-determinacy. The worst case *second level* non-determinism is of order $n-1$ and may occur at an active node of a non-STEM DAG, where there may be $n-2$ active transitions to the remaining $n-2$ non-STEM DAG (active transitions from a DAG to itself are not possible) as well as a passive transition. Similarly, k-th level non-determinism is of order $n-k+1$. Of course, while parsing a given word, there may be at most n levels of non-determinism, since otherwise there must be a cyclic active transition. As all non-determinisms are multiplicative, the worst case scenario during parsing could lead to $n!$ non-deterministic choice for every single symbol of the word, hence giving a $O(k^{n!})$ worst case complexity for a word of length k.

Now, it is generally observed that while there may be many active transitions from an active node of the STEM DAG, active transitions from an active node of a non-STEM DAG is much fewer. This is because, a non-STEM morpheme class

may be followed by only a few other DAGs in morpho-syntax rules. For example, in the rules for Bangla specified in Sec-2.2.2, only the VCAUS morpheme class has more than one (actually three here) following morpheme classes (VDEC, DEF and NCASE). Thus there may be at most three active transitions from an active node of the VCAUS DAG. We may safely conclude that higher (than one) level non-determinisms do not affect computational complexity in an appreciable manner. Pragmatic worst case complexity is therefore $O(k^n)$ — still exponential!

Focusing entirely on first level (i.e. localized in the STEM DAG) non-determinism only, it is possible to reduce it further by associating a set of *lookahead* symbols with active edges. An active transition is taken only if the next symbol belongs to the lookahead set. A non-determinism is then encountered when all the following conditions hold:

- The present node is an active node.
- Total transitions (both active as well as passive) possible from the present node, with the next symbol as "lookahead", is more than one.

While worst case complexity still remains exponential, since non-determinism is now governed by two independent events of moderate probability, the practical complexity is quite low. It is difficult to analytically compute the average case complexity since it is not easy to estimate the distribution of words being fed to the AFSA. We made the following short study for a moderate lexicon consisting of 565 stems of different classes. We identified the nodes in the STEM DAG that cause non-determinism, along with the offending characters. The results are shown below:

Total number of stems	=	565
Total number of nodes	=	764
Total number of active nodes	=	551
Total number of offending active nodes	=	124

Of the 124 offending nodes (less than 17% of all nodes), in as high as 84, the offending symbol was **i**. Invariably, these nodes recognized verbal stems and the non-determinism was between active transitions to the VCAUS (since by the effect of some spelling rules, the causational affix **i** becomes **iy**, if the following verbal declension is E realized as **e** in the surface) and VDEC (since there are many declensions like **i, ite, ila**, etc., that begin with **i**. We fine-tuned our recognizer to lookahead to the second lookahead character at active nodes, if the next symbol is *i*. This leaves us with 60 isolated offending active nodes in the AFSA. The test runs show near linear (with respect to word length) run time complexity.

2.3.5 Automatic Generation of AFSA

The specifications provided by the lexicon writer (as described in Sec-2.2), are *compiled* into an AFSA. The compilation proceeds with two passes over the list of morphemes (Sec-2.2.4), along with an intermediate pass over the list of spelling rules (Sec-2.2.2) stored temporarily in file SPELL.IN. The compilation process is pre-processed by a pass over the list of morpheme classes (Sec-2.2.1. The second pass also consults the set of spelling rules. The morphemes are assumed to be existing in the database file LEXICON (and its auxiliary files) and the temporary access file MORPH (see Sec-2.3.1).

In the ensuing discussions, the compilation process will be explained with reference to the examples cited in Sec-2.2.4. The AFSA obtained after the first pass of compilation has been shown in Fig-2.3 and the final AFSA produced is shown in Fig-2.4.

Pre-processing Stage:

Here, one skeletal DAG (i.e. DAGs with no transitions and a root node only) for every morpheme class *without* STEM directive is created along with one common DAG for all morpheme classes with STEM and PREFIX directives. The morpheme class specification of Sec-2.2.1 need only be consulted. The example specification in Sec-2.2.1 creates skeletal DAGS STEM, VDEC, VCAUS, NCASE and DEF.

First Pass:

In the first pass, morphemes are entered into the automata resulting in step-wise increase in the sizes of the respective DAGs. An attempt is made to recognize the next morpheme in the DAG specified for it, till a node is reached from where no further transitions are possible on the next symbol of the input morpheme. Thereafter, new nodes and edges are created so as to incorporate the morpheme in the specified DAG. The last node visited/created for every morpheme is made an l-active (and hence also s-active) edge. No spelling deformities are considered in this pass and as a result, l-passive and s-passive edges are identical. By the above process of creating DAGS, the path from the root to an l-active node in a DAG traces out a unique morpheme. Suppose the next morpheme incorporated is m and it traces out a path upto a l-active (hence s-active) node n in the relevant DAG. Let there be k entries $(m, i_1), (m, i_2), \cdots, (m, i_k)$ in the MORPH file for m Let the indexes $i_1, i_2, \cdots$ indicate morpheme classes $M_1, M_2, \cdots$. A trivial s-active edge e from n is created such that the index pointer of e is $i_{k_1} \vee i_{k_2} \vee \cdots$, where $M_{k_1}, M_{k_2}, \cdots$ are classes with the END directive.

Intermediate Pass:

An intermediate stage of compilation generates lexical versus surface structure association for all morphemes that can follow other morphemes by consulting the spelling rules one by one from the temporary file SPELL.IN Sec-2.2.3. As output, this stage creates a *symbol table of quadruples* ($< MorphemeClass >$, $Node, LexString, SurfString$), which means that the actual start node of the DAG (for surface representation) for $< MorphemeClass >$ is $Node$, provided the morpheme (in lexical structure) begins with $LexString$, which gets deformed to $SurfString$ by the spelling rule. This stage in general needs to split hitherto identical l- and s-edges and may also need to create new s-active nodes and s-passive edges in the following manner:

Let $RE_1 + RE_2$ be a spelling rule applicable at boundary between morpheme classes M_1 and M_2. Let RE_2^L and RE_2^S differ first at position i.

Case 1. $i = 1$ i.e. even the first characters of RE_2^L and RE_2^S are different.

Take for example, the rule $a' : a' + \emptyset : o\,\emptyset : y\; a' : a'$ at VSTEM, VCAUS. The strings $a' \in \{\emptyset : o\,\emptyset : y\; a' : a'\}^L$ and $oya' \in \{\emptyset : o\,\emptyset : y\; a' : a'\}^S$ differ at the first position.

A new sub-DAG consisting of s-passive edges only is created for all strings of the form $RE_2^S w$, where $RE_2^L w$ is a morpheme, w being the part of a morpheme unaffected by the rule. All final nodes of the sub-DAG thus created are made s-active and associated with the corresponding l-active nodes, paths to which have RE_2^L as prefixes. $(M_2, < Root\ Node\ of\ Sub\ DAG >, RE_2^L, RE_2^S)$ is entered into the symbol table.

In the example, a sub-DAG for **oya'**, with VCAUS/1 as root, is created. The terminal node of this sub-DAG is made s-active and is associated with the only node which has a' as a prefix. $(VCAUS, VCAUS/1, a', a')$ is entered into the symbol table.

Case 2. $i > 1$ A process similar to the above case is carried out, but the root of the sub-DAG is same as the root of the original DAG. Hence, the part of the sub-DAG recognizing RE_2^S upto the ith character is a part of the original DAG. As before, $(M_2, < Root\ Node\ of\ original\ DAG >, RE_2^L, RE_2^S)$ is entered into the symbol table.

In our example, for the rule $C : CV1 : R(V1)C : C + t : t$ at VSTEM,VDEC, $t \in \{t : t\}^L$ is identical to $t \in \{t : t\}^S$. So no new sub-DAGs are created for morphemes */te/* and */ten/* which has t as prefix. However, $(VDEC, VDEC, t, t)$ is entered into the symbol table.

Second Pass:

In the second pass of compilation, at first every lexical morpheme of a given class is associated to all possible following classes by constructing l-active edges. Next, the surface structure of the morpheme is woven into the same automata by associating s-active nodes with root nodes of other DAGs in the following manner:

Let the current morpheme m belong to class M_1 and let there be a spelling rule $RE_1 + RE_2$ applicable at boundary between morpheme classes M_1 and M_2. The morpheme **pa't** and the rule $C : C\,V1 : R(V1)\,C : C + t : t$ generates an example. As a result of the first pass, m (**pa't** here) traces a path upto some l-active node l (say) in DAG M_1. From this node RE_1^L (*pat* here) is backtraced to node n (say). So, there must be a bundled l-passive/l-active transition out of n (root of STEM DAG in the example) on character c, where c is the first symbol of RE_1^L (p here). Starting from node n, an attempt is made to create a branch of the DAG M_1 consisting of s-passive edges only, recognizing RE_1^S. Suppose that the process can be continued upto node n'. Note that n' could be the same as l (if $RE_1^L = RE_1^S$), or it may be a different node. In the example, two extra nodes are generated to recognize *pet*. The node n' is made an s-active node and associated with l. The symbol table is searched for an entry of the form (M_2, R, RE_2^L, RE_2^S). The intermediate pass has ensured the finding of such an entry. In the example, the triple $(VDEC, VDEC, t, t)$ is obtained from the *symbol table*. Starting from n', an s-active edge e is generated upto the node R in DAG M_2. Let $(m, i_{k_1}), (m, i_{k_2}), \cdots$ be MORPH file entries such that the morpheme class indicated by entries of the LEXICON file indexed through $i_{k_1}, i_{k_2}, \cdots$ are all M_1. The index pointer of e is made the disjunction $i_{k_1} \vee i_{k_2} \vee \cdots$. The whole process is repeated for all applicable spelling rules.

The final AFSA for the example morpheme specification from Sec-2.2.4 is given in Fig-2.4.

Let us take up some examples of words parsed by the AFSA of Fig-2.4

pa'y: This word has multiple parses — NOUN: **pa'**(NSTEM, "foot")+e(NCASE) and VERB: **pa'**(VSTEM, "get")+e(VDEC). In both cases a global spelling rule $(a|a'|o) : (a|a'|o)\ +\ e : y$ is applicable across the morpheme boundary.

peye: This word is parsed as VERB: **pa'**(VSTEM)+E(VDEC). Note that E, the colloquial form of the CONTINF verb declension, is a symbol from the lexical alphabet only. The spelling rule $V : V + E : y\ \emptyset : e$ local to the boundary between VSTEM and VDEC is applicable here. The set V is the union $a|a'|i|i'|u|u'|e|e'|o|o'$), i.e. the the set of natural vowels.

pa'yer: Parsed as NOUN: **pa'**(NSTEM)+er(NCASE). The spelling rule $V : V + \emptyset : y\ e : e$ local to boundary between NSTEM and NCASE, is applicable.

pa'oya': Parsed as NOUN: **pa'**(VSTEM)+**a'**(VCAUS). The spelling rule $(a|a'|o)$: $(a|a'|o)$ + $\emptyset : o\ \emptyset : y\ a' : a'$ local to boundary between VSTEM and VCAUS, is applicable. Note that the word is a *NOUN*.

pa'te: Will have multiple parses NOUN: **pa'**(NSTEM)+**te**(NCASE) and VERB: **pa't**(VSTEM, "lay")+**e**(VDEC). Both are by normal concatenation.

pe'te: Will have multiple parses VERB: **pa't**(VSTEM)+E(VDEC) by spelling rule $a' : e\ C : C\ + E : e$ (where C is the set of normal consonants) and VERB: **pa'**(VSTEM)+**te**(VDEC) by rule $a' : e\ + t : t$.

mara'y: Will be parsed as **mar**(VSTEM "die")+**a'**(VCAUS)+**e**(VDEC) by the AFSA. However, the word **mara'y** is ill-formed because in Bangla, there is a verb stem **ma'r** "kill" meaning the causated of **mar** and thus **mar** itself can not be causated. The AFSA as such does not detect the error. The error is detected when an attempt to unite the schemata $(\uparrow\ CAUS) = 0$ projected by **mar** with schemata $(\uparrow\ CAUS) = 1$ projected by **a'** fails.

2.3.6 Later Modifications

A modification in the structure of the AFSA in the form of lookahead symbols has been suggested earlier. The lookahead set may be generated without difficulty in the second pass of compilation. A non-trivial modification in the control program of the AFSA would be suggested in Sec-6.3 to take care of "affix-hopping", an idiosyncratic feature of Bangla.

In the next chapter (Chapter-3), we would extend the concept to lexical items spanning across more than one word.

2.4 Implementation Notes

The major object (also known as `classes` in C++ parlance) used are:

PassiveNode: This `class` represents a passive node of the AFSA. It contains lexical and surface back passive edges to the previous node. There is a *container* of pairs (c, lp), where c is a symbol from the lexical alphabet and lp is a pointer to another node. Thus the items of this container are l-passive edges. There is another similar container for s-passive edges. Member functions includes forward/ reverse lexical/ surface transition procedures.

ActiveNode: Is inherited from PassiveNode. There are two additional container for l- and s-active edges. Each contained item of these containers is an Active Edge. There are member functions for making l- and s-active transitions. Another important member data of ActiveNode is *pLexList*, which is a pointer to a *linked list* of pointers to the Lexicon `class`. All members of the list are assumed to be disjunctively pointing to different lexical entries.

ActiveEdge: Is a pair $(pLook, pNode)$ where, *pLook* is a pointer to a *set* of "lookahead" symbols and *pNode* is the root of the DAG to where the active transition is taken.

Lexicon: This `class` does not have any important member data. It serves the purpose of interacting with the lexicon through database management routines to draw lexical projections of morphemes.

Other `classes` used are FStructure, Pair, etc., along with member functions to perform Locate and Merge operations of LFG. These `classes` will be discussed in more detail in Chapter-4.

Object Returned by the Lexical Sub-system

As shown in Fig-2.2, there are two major contents in the object returned by the lexical sub-system — Word category of the input word being parsed and the f-structure. In practice, the structure of the object returned by the lexical component is not exactly so. The reason is that in our proposed system, there is an intervening supra-lexical layer (described in Chapter-3) between the syntactic component and the lexical component. The components of the object actually returned is as follows:

- The input word itself.
- Word Category. Indicating the tentative word category of the input word.
- A List of constituent morphemes.
- A List of (codes for) Morpheme Classes $M_1, M_2, \cdots$ for the constituent morphemes of the word. Thus, the details of morpho-syntactic composition of the input word is also returned.
- A List of (pointers to) f-structures $F_1, F_2, \cdots$ for the constituent morphemes.
- A (pointer to) the unified f-structure of the word.
- A List of (pointers to) m-structure schema of the word, if any.

- A (pointer to) the semantic clause of the word.

The `class` corresponding to the return object is called **LexPrimitive**. This `class` has member functions to extract the following:

- The word
- The (tentative) Word Category.
- The constituent morphemes.
- The Morpheme Class of the i-th morpheme.
- The f-structure of the i-th morpheme. As we shall show in Chapter-4, it is possible to extract out the (sub) f-structure corresponding to any attribute of an f-structure. Using similar techniques, it is possible to extract the (sub) f-structure corresponding to any attribute of the f-structure of any constituent morpheme of the word.
- The number of arguments in the semantic clause of the word.
- The i-th argument of the semantic clause. If i is zero, the semantic predicate is returned.
- The m-structure schema.

Interaction with the Lexical Sub-system is carried out through the following function:

```
int primitiveLexAnalysis( char          *word,    \\ Input word
                          LexPrimitive *lexPrim \\ Returned object
                        ); // Returns TRUE if parse successful,
                           // FALSE otherwise
```

2.5 Discussions

The formalism proposed here has been tried out for a medium sized lexicon in Bangla consisting of about three hundred verb stems, one thousand nominal stems and a few stems of some other classes. There are about forty verb declensions and about ten case declensions capable of generating 2,400 VERBs (causated and

non-causated) and 10,000 NOUNs. The results of parsing obtained were very satisfactory, with near linear recognition time complexity.

The rules that govern conjoining between one the thirteen mono-syllabic (one to three characters in length) prefixes and a noun in Bengali, are semantic in nature. The lexical projection of a prefix does not provide any advantage in syntactic processing of sentences. Moreover, since the noun stems and the prefixes reside in the same DAG (STEM), the active transition from an active node recognizing a prefix leads to the root of STEM DAG itself, causing self loops. Too many self loops lead to a rapid degeneration of efficiency. In our system, we have stored both the prefixed and unprefixed versions of nouns as distinct stems.

The lexical projection of a compound stem produced as a result of *euphony* of two stems can be derived from the conjoining stems. As a result, euphony is a more attractive subject of study than prefixes. If an AFSA is used to perform rule-based de-euphonization, it may have too many self-loops, resulting in reduction of efficiency. However, at the level of sentential syntax analysis, considerable advantages may be derived from rule-based de-euphonization. We have made some initial studies [135]. However, it is too early to report any major achievement.

The biggest advantage of our formalism is the compactness and lucidity of representation. The recognizers are finite-state networks — a well-studied formalism. The representation scheme is easy to understand and quite flexible. The underlying LFG formalism permits a broad generalization across morpheme boundaries as in the last example taken up in Sec. 2.3.5 (i.e. **mara'y**). Comparing our formalism with Koskenniemi's two level approach, we find that the latter does not incorporate morpho-syntactic restrictions in the automata itself. In the framework proposed by Ritchie et al [144], attempts have been made to include morpho-syntactic rule as extensions of Gazder's GPSG [70] formalism. However, adapting such a framework to suit a LFG based system appears difficult. Moreover, morpho-syntactic parsing in the above framework involves complex manipulations of rules unlike simple pointer tracing in the proposed formalism.

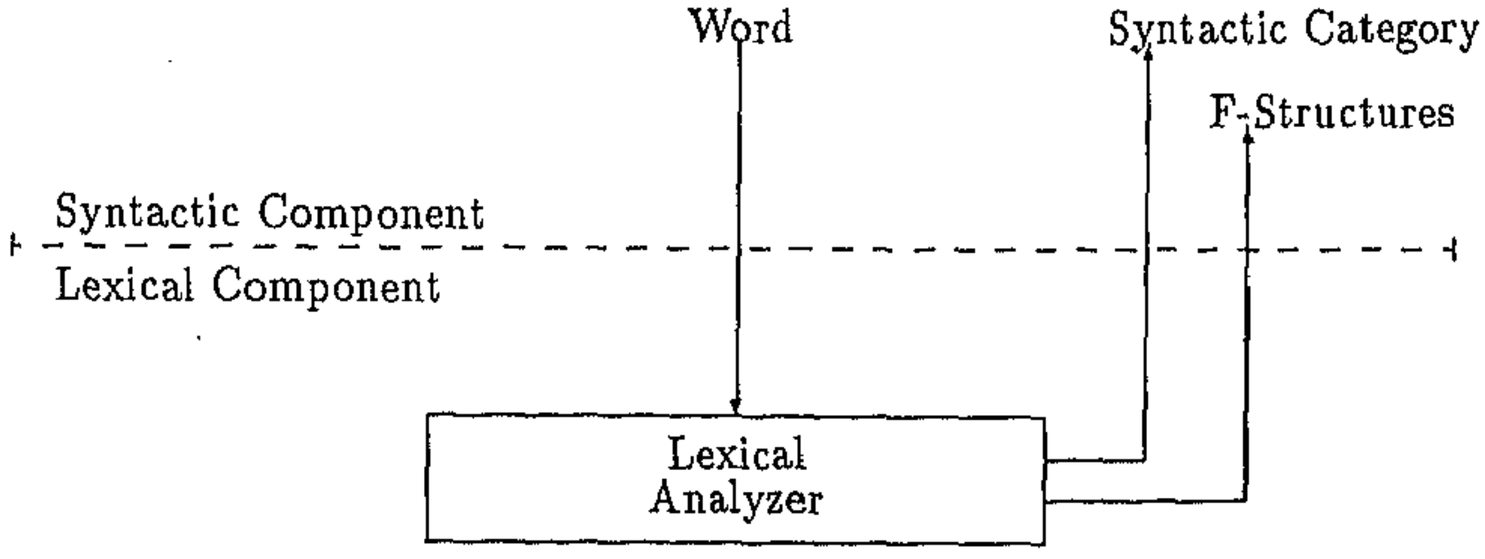


Figure 2.1: Conventional Lexical Interaction

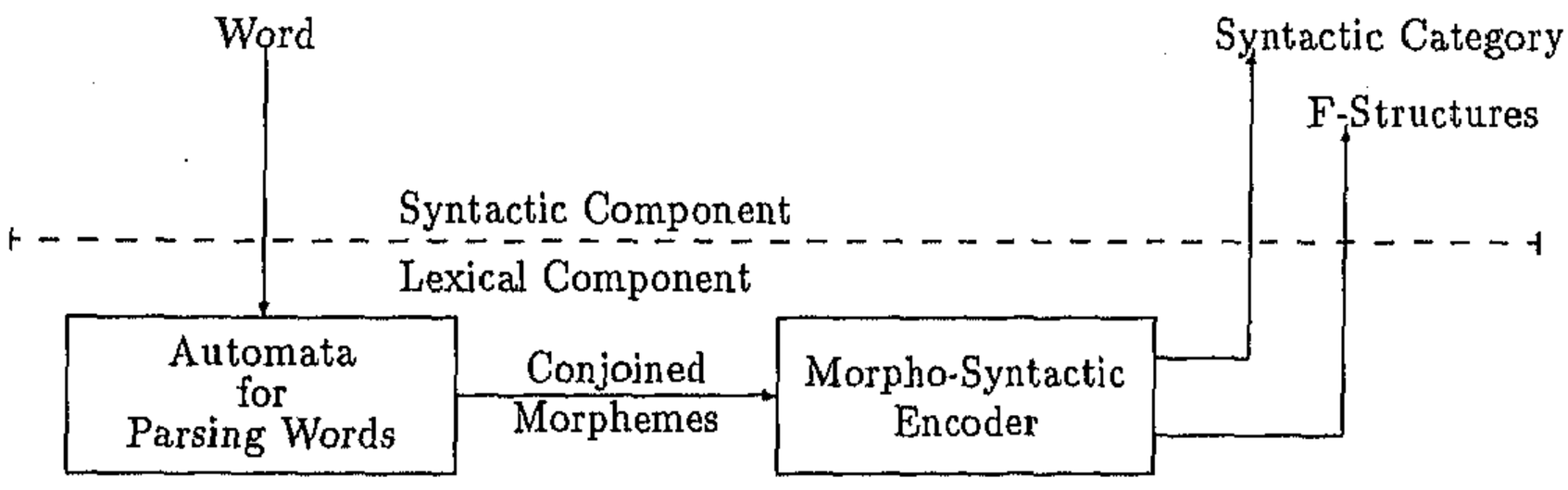
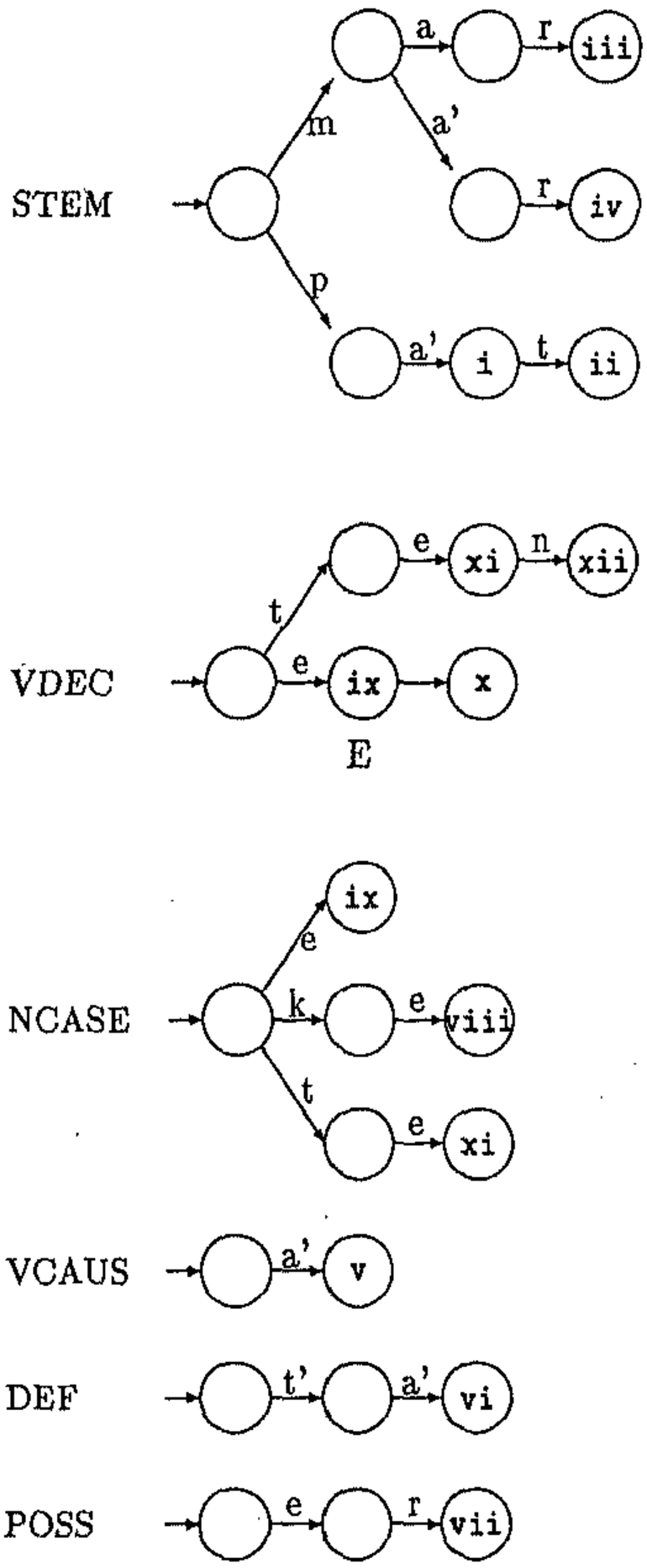
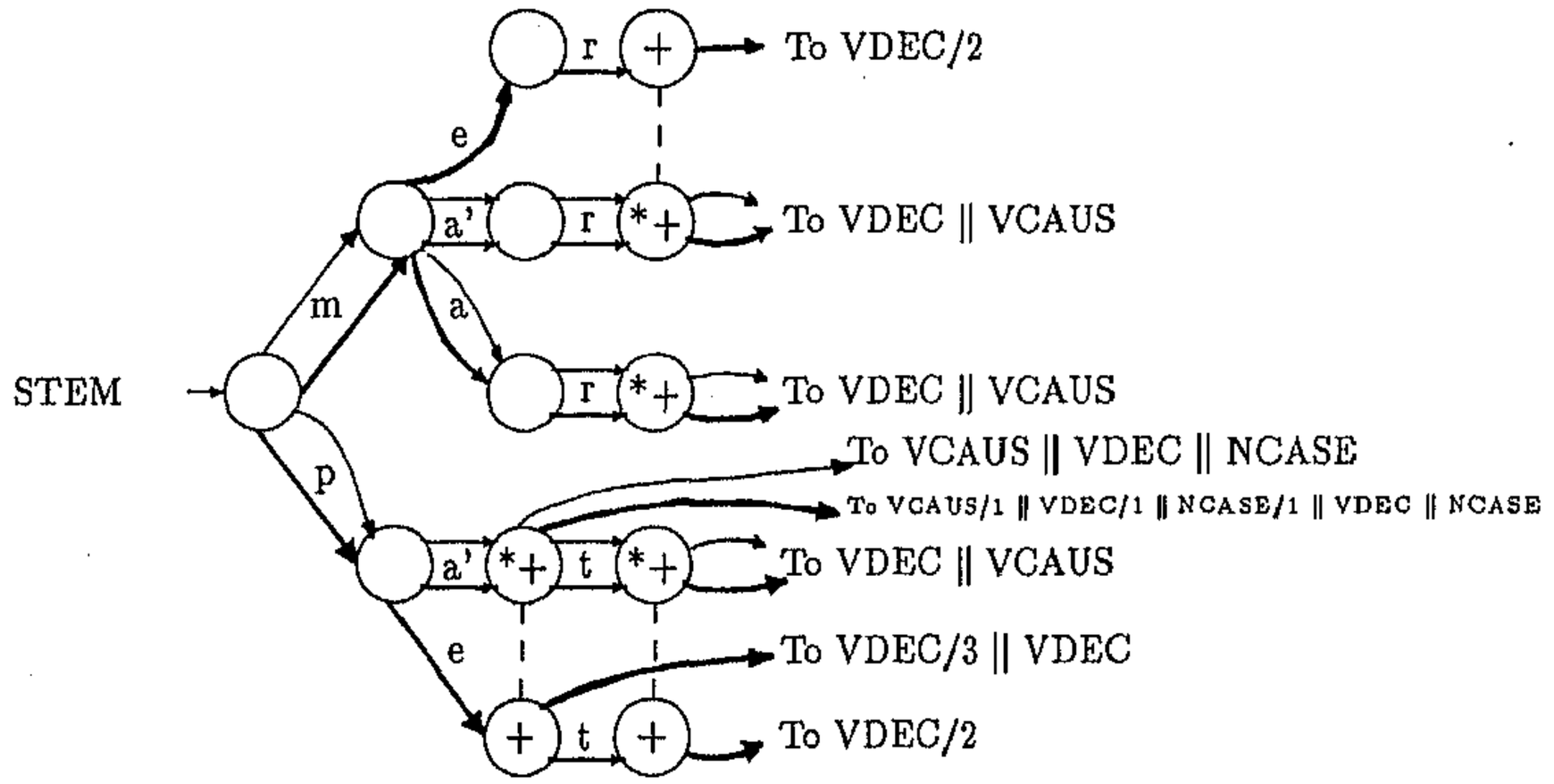


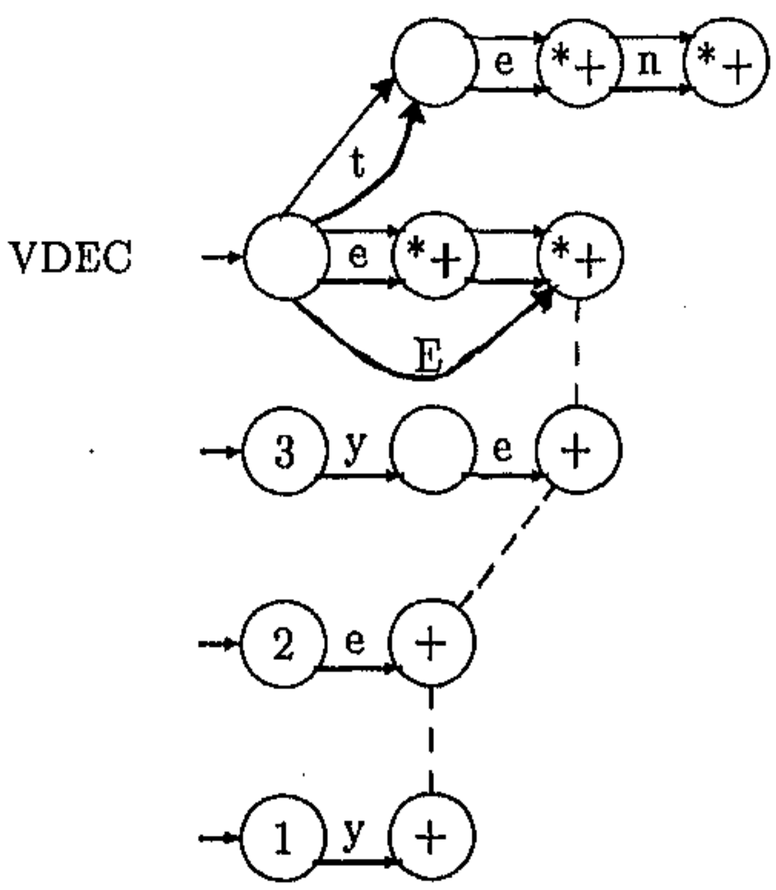
Figure 2.2: Lexical Interaction with Morpho-Syntactic Analysis



Edges are both l-passive and s-passive. Numbered circles are active nodes. Figures on terminal nodes indicate entry number of Morpheme Lexicon (Sec-2.2.4) indexed by the nodes.

Figure 2.3: The AFSA After First Pass of Compilation


r
To VDEC/2
e
a'
r
*+
To VDEC || VCAUS
m
a
STEM
r
*+
To VDEC || VCAUS
To VCAUS || VDEC || NCASE
p
To VCAUS/1 || VDEC/1 || NCASE/1 || VDEC || NCASE
a'
*+
t
*+
To VDEC || VCAUS
e
To VDEC/3 || VDEC
+
t
+
To VDEC/2


e
*+
n
*+
t
VDEC
e
*+
*+
E
3
y
e
+
2
e
+
1
y
+

...contd(Fig-2.4)

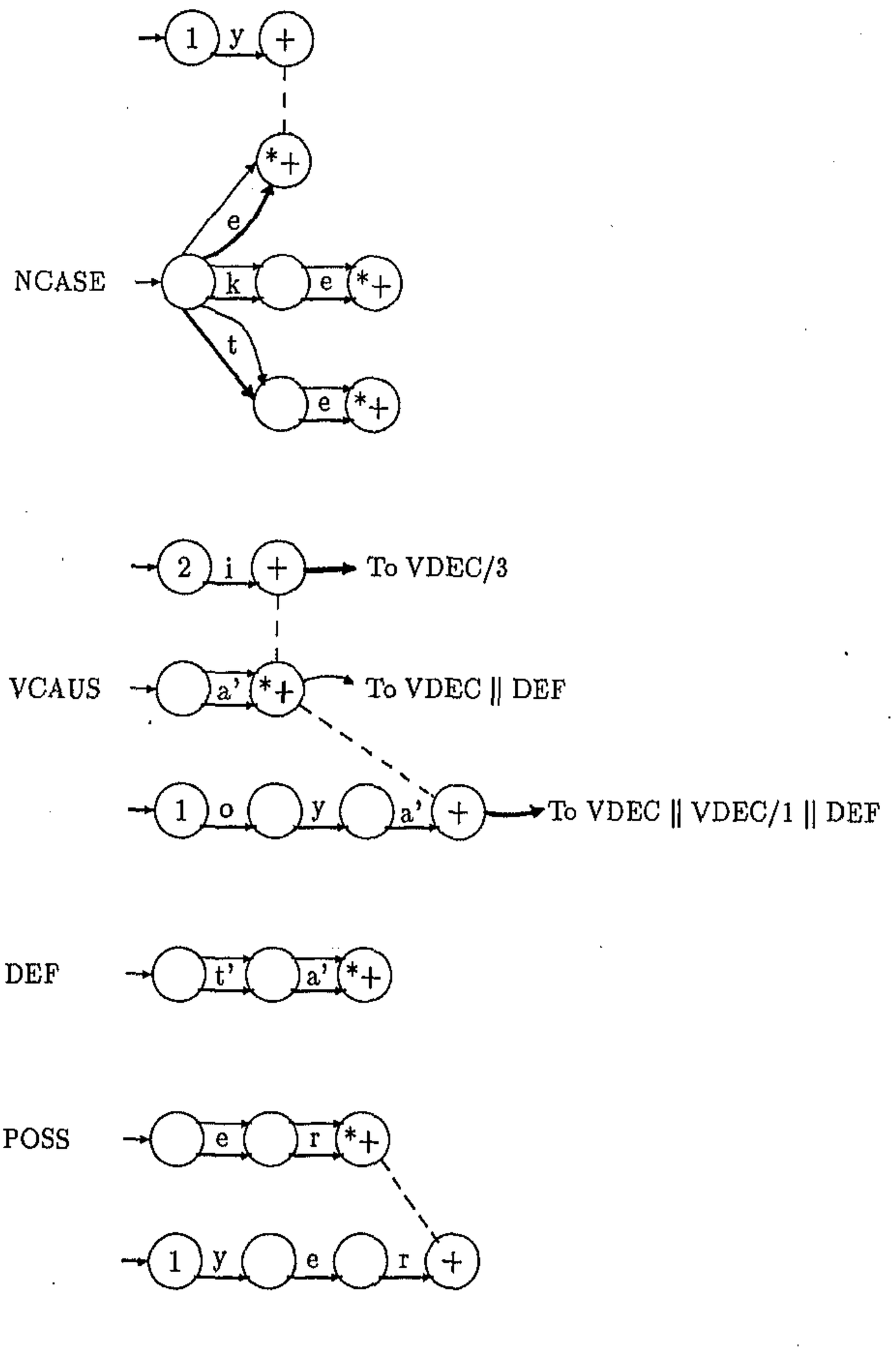


Figure 2.4: The Final Compiled AFSA

Chapter 3

The Supra-Lexical Level

Quite often a lexical item spans across more than one word in many Indian languages, especially in Bangla. Often, the overall projection of a multi-worded lexical item may not be trivially determined from the projections of the individual words, thus violating the principles of "Direct Syntactic Encoding" (a primary concept on which LFG is based). In their article on LFG [88], the authors demonstrated how similar problems in English could be tackled by the use of *constraint schema* or *functional control schema*. We shall show that these approaches are not attractive enough for Bangla, because either they lead to manifold increase in lexicon storage requirement, or because some basic assumptions of LFG are violated. We propose a *Supra-Lexical* level of analysis, which forms the actual interface between a syntactic component and the morphological (or *hitherto known* lexical) component. The primary responsibility of the supra-lexical component would be:

i. Detect co-occurring words forming a lexical item and in the process provide *feedback* to the syntactic component to "pump in" more words.

ii. Obtain normal projections of individual words using the existing lexical component as described in Chapter-2.

iii. Generate the projection of the word-group from the projections of the individual words in the group using some specified *supra-lexical* rules to do so.

The lexical interaction scheme with an intermediate supra-lexical level of analysis is explained in Fig-3.1.

In this chapter, we would establish the necessity of the supra-lexical level for Bangla by first giving a general solution outline and requirement specification in Sec-3.1.

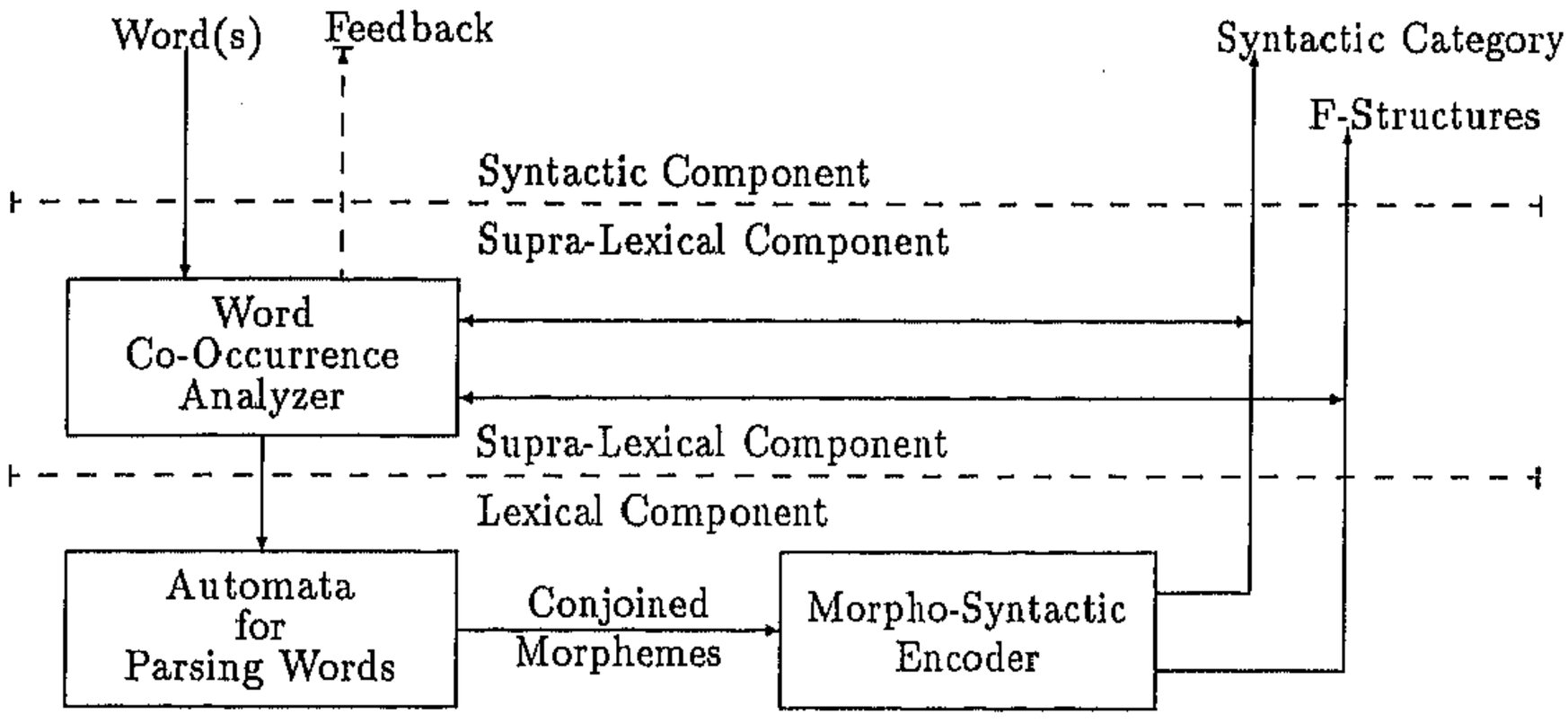


Figure 3.1: Lexical Interaction with Supra-Lexical Analysis

Next, in Sec-3.2, a complete formalism that may be used by a lexicon designer for supra-lexical specification would be introduced and its implementational aspects discussed.

3.1 The Need for a Supra-Lexical Level

We begin by citing some of the more common instances of multi-worded lexical items in Bangla. The examples chosen are mostly from the verbal paradigm as most multi-worded entities are either verbs or of categories derived from verbs. A more comprehensive discussion on Bangla verbs may be found in Chapter-5.

3.1.1 Some Examples of Multi-Worded Lexical Units in Bangla

We give in here some interesting examples of lexical entities constituted of more than one word. Although the list is not exhaustive, the basic idea is to highlight the situation that guides us to a solution technique described in this chapter. For more example from the verbal paradigm of Bangla, see Chapter-5.

A Verb in Perfect/Progressive Future Tense: Here, the future tense marking *auxiliary* **tha'k** (stay) follows any participle verb. The participle has

either the CONDitional INFinitive or the CONTinual INFinitive declension. The verb is either in future or habitual *tense* depending upon whether the declension on the participle is CONDINF or CONTINF respectively. Also, the *aspect* of the verb is either perfect or progressive, depending on the inflection on the auxiliary. This form has considerable similarity with the English "be" auxiliary usage.

Compound Verb: In this form of multi-worded Bangla verbs (also quite prevalent in other Indian languages) a verb lexeme consists of a *pole* which is a verbal stem taking a *continual infinite* (CONTINF) declension, followed by a *vector* which is a tensed verb with a stem from a well-defined set of about thirteen stems. The general meaning of the verb is generated from the pole while the inflection of the vector provides the tense and agreement features of the verb. The stem of the vector provides a finer perspective to the meaning of the verb which may have very little correlation with the actual meaning of the vector. As an illustration, in compound verb **base parla** = *sit*+**CONTINF** *fall*+**(PRESENT-PERF+3p-1h)**, the overall meaning is *sat down*. The vector meaning *fall* provides a perspective of locus to the *pole*. However, the overall action has very little relationship with the action of falling. It needs to be mentioned that any pair of consecutively occurring verbs with an CONTINF declension on the first word does *not* constitute a compound verb. This form is peculiar to many South Asian languages and has no parallel in English.

A Verb in Neuter Voice: In this multi-worded verb form, there is a *for infinitive* declension on the first word and the second word has the *copula* **Ha**. Although the overall meaning of the verb is still derived from the first stem, the syntactic requirements for the other entities in the sentence (where the verb occurs) get non-trivially modified. This form has parallel in English. However, unlike English, here the syntactic repercussions are not localized.

3.1.2 Problems With Traditional Approaches

Here we review some of the techniques used in [88] for English. Other uses of the techniques may be found in [24] where the main focus is on a consistent lexical theory for passivization.

One suggested method is the use of a set of *constraint schema* in the lexical projection of one of the words of a word-group. Constraint schema are used to enforce the other words of the group to be of a certain pattern. This method is very suitable for word groups representing a "figure of speech", like the English verb

'to keep tabs on'. One lexical entry for 'keep' has a projection meaning 'maintain watch' along with a constraint function that ensures that the following two words are 'tabs' and 'on.' The method is quite suitable if number of similar instances of word groups is reasonably low but gives rise to too many *alternations* in the lexicon if some words (for example 'keep' here) take part in hundreds of similar groups.

The other method suggested was the use of "Uniqueness Condition" together with appropriate *functional control* schema. With the English auxiliary-participle pair 'is handing', the suggested solution is to treat 'is' as a finite verb and let it take a *single* COMPlement verb phrase. The complement verb phrase accounts for functional attributes for the remaining phrases in sentences like *'The girl is handing the baby a toy'*. Also, the subject of the complement VP of 'is' is unified with the subject of 'is' ('The girl' in the above example) by means of a *functional control* schemata $(\uparrow\ SUBJ) = (\uparrow\ VCOMP\ SUBJ)$. The semantic formula of 'is' is expressed as a single argument predicate 'PROG' whose argument is made the complement of 'is' by the projected schemata $(\uparrow\ VCOMP\ SUBJ) = (\uparrow\ SUBJ)$. The details may be found in [88] (Art-4.5). Unfortunately, this method of enforcing word co-occurrence is not feasible in Bangla as is demonstrated below.

Bangla is a *non-configurational* language. The constituent phrases and the verb of a sentence may permute rather freely. Thus, there may be many well-formed surface forms for a single semantic form. The *grammatical functions* (gf-s) of participating noun phrases in a sentence are predicted from the case markers on the head nouns of the phrases. Co-occurring words however appear side by side in all the surface forms of a sentence. Let us consider a sentence (3.1). Any permutations of the underlined phrasal units in (3.1) could also be considered.

(3.1) **a'mi ra'mke galpat'a' balte tha'kba**
I-NULL Ram-DAT story-DEF-NUL tell-COND-INF stay-FUT
"I shall go on telling the story to Ram"

The verb **balte tha'kba** in (3.1) (will go on telling) is a multi-worded verb in FUTURE-PROGressive tense, consisting of a bi-transitive participle **bal** (say). The overall tense-aspect of the verb has been derived from the inflections on both the auxiliary and the participle and not directly inherited from either of them. The three noun phrases **a'mi**, **ra'mke** and **galpat'a'** can freely permute without changing the meaning of the sentence. If Bangla had a concept of a verb phrase VP, (3.2) could be the c-strucure for (3.1) and functional control schema might be used to obtain the f-structure (3.3).

(3.2) $S[_{NP}[a'mi]\ _{VP}[_{VP'}[_{NP}[ra'mke]\ _{NP}[galpat'a']\ _{V}[balte]]\ _{V}[tha'kba]]]$

(3.3)
$$\begin{bmatrix} SUBJ & \begin{bmatrix} ANIM & + \\ CASE & NULL \\ GNPH & 1p \\ PRED & 'I' \end{bmatrix} \\ COMP & \begin{bmatrix} OBJ & \begin{bmatrix} ANIM & - \\ DEF & YES \\ CASE & NULL \\ GNPH & 3p-1h \\ PRED & 'story' \end{bmatrix} \\ IOBJ & \begin{bmatrix} ANIM & + \\ CASE & DAT \\ GNPH & 3p-1h \\ PRED & 'Ram' \end{bmatrix} \\ SUBJ & \\ PRED & 'tell < (SUBJ),(OBJ),(IOBJ) >' \end{bmatrix} \\ TENSE & FUTURE \\ PRED & 'PROG < (COMP) >' \end{bmatrix}$$

However, due to the non-configurational nature of the language, there is actually no evidence of a VP node and as Mohanan has suggested [131], the c-structure rule for a sentence should rather be like (3.4). Indeed, (3.4) is the starting point of the GLFG formalism described in Chapter-4.

(3.4) $S \longrightarrow NP^* \; V \; NP^*$

Although (3.4) can correctly generate a sentence like (3.5) or any phrasal permutations, it fails to generate (3.1) or its permutations.

(3.5) **a'mi ra'mke galpat'a' balba**
"I-NULL Ram-DAT story-DEF-NUL tell-FUT"
I shall tell the story to Ram.

In fact, it is not possible to find a set of context free rules which will ensure that in the c-structure of any permutation of phrases of (3.1), the subject **a'mi** maintains a sibling relationship with the auxiliary **tha'k**. Hence, the functional encoding approach can not be used effectively to solve our problem. From similar considerations, functional control is not the proper approach for other multi-worded lexical forms described earlier.

3.1.3 Outlines of the Proposed Solution

A plausible solution to the problem at hand (for multi-worded verbs) could be:

- Use the flat structure (i.e. $S \rightarrow (NP)^*\ V\ (NP)^*$) for a sentence.
- But, treat the node V in the above structure not as a pre-lexical node but as a phrasal node (i.e. a node which may derive a form consisting of more than one word).

The above solution is unsatisfactory because :

- It unnecessarily complicates matters in case of a vast number of verbs which consist of a single lexical word only.
- The approach is not general enough to be extended to the few cases of word co-occurrence outside the verbal paradigm.
- Most importantly, the principles of *direct syntactic encoding* of LFG prevents non-trivial derivation of the projection of the word group from the projections of the individual words in the group.

A viable alternative is to strengthen the lexical sub-system. However, since lexical entities occurring as word groups is an exception rather than a rule, from efficiency considerations, non-trivial modifications in the already existing lexical sub-system is not quite attractive. As a compromise solution, we propose a layer of analysis "sandwiched" between the syntactic and morpho-syntactic layers. For the sake of continuity, we shall go on referring to the morpho-syntactic unit as the lexical sub-system and call the sandwiched layer the *supra-lexical* sub-system.

3.1.4 Requirement Specification

Some of the complex manipulations that are required to be carried out by the proposed supra-lexical component are:

- The individual *semantics* of the co-occurring words may be altered and/or may be totally eclipsed. Non-trivial procedures are therefore required for generating the semantics of the word group.

- The rules governing co-occurrence may not be simply syntactic, i.e., the co-occurring words may need to satisfy certain non-syntactic constraints. For example, the verb stems for a (prospective) compound verb need to be tested for a valid pole-vector pair.

- A lexical feature of a word-group may be a function of the *type* of co-occurrence and may not be simply derivable from the words taking part in it. For example, the TENSE feature of a Progressive-Future verb is obtained as result of the type of co-occurrence and not from the projections of either the participle or the auxiliary.

- The syntactic *requirement* of other entities in a sentence involving a word co-occurrence may get changed as a result of the grouping. Neuter verb is an example of the phenomenon. In our proposed syntactic formalism, this necessitates changing the *m-structure* of the co-occurring words. See Chapter-4 for a description of the m-structure.

3.2 The Proposed Supra-Lexical Formalism

For the proposed Supra-Lexical sub-system, it is envisaged that:

- There would be an "off-line" *specification* phase during which the lexicon designer would specify a set of *filters* and a set of *supra-lexical rules*. The filters would be specified directly in the syntax of C++ programming language. The syntax for specification of supra-lexical rules is introduced later in this section.

- As the supra-lexical rules are being specified, there would be a interpretive tool that would directly create C++ source files from them. We would call the resultant code fragment (which would also include reference to the filters) the *Supra-Lexical Analyzer*.

- After compilation of the entire system, the supra-lexical component would act as a layer between the syntactic and the morphological (lexical) components. See Fig-3.1 for clarification.

3.2.1 The Tools for Supra-Lexical Specification

Filters

A major component of the supra-lexical level is a set of functions which check the validity of occurrence of various primitives in specified relative positions. The primitives could be morphemes or morpheme classes, words or lexical entities. Such a function will be called a **filter**. A filter is identified by its name. A filter takes one or more arguments, returns values TRUE or FALSE. A particular filter is specifically designed to check a particular validity. Some of the most common (and hence built-in) filters are:

`exists(e)`: Checks if the primitive `e` exists or not.

`equals(e,f)`: Checks if primitive `e` is equal to primitive `f`. Succeeds if both e and f do not exist.

`equalsFrom(e,f)`: Checks if primitive `e` exists and is equal to any one of the primitives from a *set* `f` of primitives.

New filters may be composed from existing ones by using boolean connectives:

`&&` (and), `||` (or), `!` (not),

etc.

Also, entirely independent filters may be constructed using the syntax of the C++ programming language.

Word Co-occurrence Expression

A Word co-occurrence expression is a primary selector for co-occurring words. The syntax for a word co-occurrence expression is:

X Y Z ...

Here each item X, Y, etc., is a word category (NOUN, VERB, etc.). An item may be detailed down to constituent morpheme classes (NSTEM, VSEM, etc.) using the format $M_1 + M_2 + \cdots$, where each M_i is a morpheme class. A morphemic primitive M_i may be parenthesized to indicate its optional presence. If an item is detailed down to the morphemic level, care must be taken that the detail conform to at least one morpho-syntactic rule (as described in Chapter-2) for some primary word class.

Example: Consider the word co-occurrence for any Verb-Verb type multi-worded verb. Such verbs have been treated in more detail in Chapter-5.

```
VSTEM+(VCAUS)+VDEC    VSTEM+(VCAUS)+VDEC
```

Here, VSTEM, VCAUS and VDEC are morpheme classes corresponding to verb stems, verb causational affix and verb declensions, respectively.

In the context of any word co-occurrence rule, the items X, Y, etc., may be *logically referred to* by positional arguments `$1`, `$2`, *etc.* Individual morphemes of an item may in turn be logically referred to using the scope resolving operator . (dot). A logical reference of a primitive (whether of type `$i` or `$i.M`, `M` being a morpheme class in the detail of the `i`-th word) encompasses all its attributes — the character string for the primitive, the f-structure for the primitive and the m-structure (the primitive must be a verb). To distinguish among the three, a string reference is made by enclosing the primitive within square brackets (`[]`). The m-structure of the `i`-th word has a special notation (`$i #`). In itself, a logical reference is assumed to refer to the f-structure of the primitive. Thus, in the above example `$1.VCAUS` logically refers to the f-structure of the VCAUS morpheme of the first word, while `[$1.VCAUS]` refers to the character string for the same morpheme. Also, `($1 #)` represents the *entire* set of m-structure schema of the first word.

Mapping Operators and Mapping Function

The ultimate objective of the supra-lexical level is to obtain the proper lexical projection for a word group. The projections could result out of:

Assignment(s) from other projections or new schema. The operator `<<` will be used to denote an assignment.

Unification of an existing projection with another. The operator = will be used to denote unification.

The assignment and unification operations are associated with a word co-occurrence expression (see later) and may have the usage:

`Name << Value` *or* `Name = Value`

Here `Name` is a slot name in the f-structure of the lexical entity formed by the word group and `Value` is either a constant (scalar quantity or symbol) or a value obtained from the f-structure of the co-occurring words using `$i` indirection. The C (*or* C++) string concatenation function `strcat` may be used to compute `Value`.

Example: `TENSE << ($2 TENSE)`

Another usage of the assignment/ unification operator is of the form:

`#` = $\mathcal{M}$, where, `#` represents the m-structure of the word group and $\mathcal{M}$ is an expression of involving m-structure schema. The expression $\mathcal{M}$ may be a constant set of m-structure schema or the set of schema for the i-th word in the group expressed as `($i #)`. $\mathcal{M}$ may also be recursively defined using expressions like $\mathcal{M}' + m$ and $\mathcal{M}' - m$, where m is a single schemata. While the expression $\mathcal{M}' + m$ *unifies* the schemata m with the set $\mathcal{M}'$, the expression $\mathcal{M}' - m$ *removes* the schemata m from $\mathcal{M}'$ to give the resultant expression. In the expression $\mathcal{M}' - m$, *only the left hand side* of the schemata m must be mentioned.

Example: Consider the case where the CASE feature of the SUBJect of the clause headed by a verb in neuter voice must change to DATive. The required mapping function is:

```
# = ($1 #) - (# SUBJ CASE) + (# SUBJ CASE)=DAT
```

The first "-" operator removes the schemata `(# SUBJ CASE)=...` from the m-structure schema set of the first word of the group, while the next "+" operator adds the schemata `(# SUBJ CASE)=DAT` to the resulting set.

A mapping function may have the following forms:

i. Simple assignment/unification operation.

   Example: `TENSE << ($2 TENSE)`

ii. *Filter*?A_1 : A_2, where A_1, A_2 are lists of assignment/unification operations terminated by ; (semi-colon). If the Filter operation is successful, the A_1 operations are carried out, otherwise the A_2 operations are carried out. Compound statements in A_1 and A_2 are enclosed within braces.

Example:

```
equals(($1 TENSE), PresentSimple) ?
  {TENSE = PresentPerfect; NEGATED = YES;} : ;
```

iii. *Filter*; which simply demands a successful Filter operation, failing which the word co-occurrence being tested is unsuccessful.

Example: `compoundable($1.VSTEM, $2.VSTEM);`

Semantic Function

The overall semantics of the word group is a function of the general form of co-occurrence expression as well as the semantics of the words that are actually taking part in the co-occurrence. The LFG tradition does not permit reference to internal arguments of the semantic clauses of lexical items. However, such references are essential in the present case. The notation `~`i, $i \geq 0$ is used to denote the i-th argument of a semantic clause. `~0` denotes the predicate of a semantic clause. Thus, `$2.~3` refers to the third argument in the semantic clause of the second word in the group.

The form `~?` refers to the *number* of arguments in a semantic clause. Thus, `$1.~?` refers to the number of arguments in the semantic clause of the first word.

A *semantic function* is a specification having the form $\{n; E_0; E_1; \cdots E_n; \}$ where n is an integer denoting the number of arguments in the semantic clause of the group, E_i s are string expressions (i.e. character arrays or character arrays operated with string concatenation operator + of applied with `substring` or other character array transfer functions) which may freely refer to names and arguments of the semantics clauses of the individual words of the group. E_0 denotes the *predicate* of the semantic clause. The overall semantics of the word group is therefore is $E_0 < E_1, \cdots, E_n >$, where < and > enclose the arguments in the tradition of LFG. An E_i can be any gf name like SUBJ, OBJ, etc. An E_i is a semantic function can also be `@$`j. In this case the next `n-$j.~?` arguments are *copied* from the `$j.~?` arguments of the j th word. Normal $ indirection may be freely used in a semantic clause.

Example: As an example, consider the semantic clause for a compound verb.

```
{$1.~?; $1.~0 + vector($2.VSTEM); @$1; }
```

In the above example, the operator + concatenates two strings and `vector` is a transfer function returning the finer semantics of the verb as provided by the vector (the 2-nd word). Thus, the semantics of the compound verb has a) as many arguments as the semantics of the first word (pole), b) the predicate is derived by concatenating the predicate of the pole and the vector-meaning of the stem of the second word (vector), and c) the arguments of the constructed predicate are exactly the arguments in the semantic clause of the pole.

In case a semantic function is null (i.e. entirely omitted), the semantic clause of the first word of the group is returned as the semantic clause of the word group.

3.2.2 Supra-Lexical Rule

The tools described above are used to describe rules that govern the formation of different multi-worded lexical entities. A supra-lexical rule is a 4-tuple $< L, WG, M, S >$, where:

L is a Lexical Entity,
WG is a word co-occurrence expression,
M is a set of mapping functions and
S is a semantic function.

The proposed syntax for specifying a supra-lexical rule is:

L => WG `where` $\{m_1; m_2; \cdots;\}\{S\}$.

Here, m_i is a mapping function belonging to M and S is a semantic function.

Example (Compound Verb):

```
VERB  => VSTEM+VDEC   VSTEM+VDEC where {
  equals( ($1.VDEC TENSE), CONDINF);
  compoundable( [$1.VSTEM], [$2.VSTEM]);
  TENSE<<($2 TENSE);
  GNPH<<($2 GNPH);
  #=($1 #);
}{
  $1.~?; strcat($1.~0, vector($2.VSTEM)); @$1;
}
```

In the above example (simplified), the filter `compoundable` checks whether the two

stems may be compounded. See Chapter-5 for further discussion on compoundability.

Example (Verb in Neuter Voice):

```
VERB => VSTEM+(VCAUS)+VDEC  VSTEM+VDEC where {
equals( ($1 TENSE), FORINF);
equals( ($2.VSTEM, COPULA) );
TENSE=($2 TENSE);
GNPH=($2 GNPH);
  # = ($1 #) - (# SUBJ CASE) + (# SUBJ CASE)=DAT;
}{}
```

Note that in the above example, the *S* clause is null and hence the semantic clause of a neuter verb is identical to the semantic clause of the first verb in the group. Also, the first stem is required to be the copula through use of the `exists` filter. Only copula stems project a COPULA structure.

Any number of supra-lexical rules may be specified. More than one rule may refer to as the same lexical category for the *L* part. For example, both the rules specified above refer to VERB as the lexical category.

3.3 Generation of the Supra-Lexical Analyzer

There are two aspects of supra-lexical specification:

i. Specification of a list of `filters`.

ii. Specification of a set of `supra-lexical` rules.

Filters are specified in the C++ syntax directly into a header file FILTER.H and implementation file FILTER.CPP. Care must be taken to properly *overload* the filters for the various types — atom, string, f-structure, m-structure, etc., of the expected inputs. For example, consider the built-in filter `equals(e1,e2)`. The parameters `e1` and `e2` may be any of the above types. Hence, the filter must be overloaded for different pairs of types of `e1` and `e2` — (atom,atom), (atom,string), (string,atom), etc.

The interpreter program that generates C++ source from supra-lexical rules is not very difficult to design. As discussed earlier, the syntactic component performs lexical interaction via an intervening supra-lexical layer. The function called by the syntactic component has the following form:

```
int lexAnalysis( LexPrimitive *lex);
// Returns TRUE if a lexical entity is found,
// FALSE if no lexical entity found,
// EOF if trying to read beyond of sentence.
// If TRUE value returned, lex points to a LexPrimitive structure
// for the next single or multi-worded lexical entity.
```

This function refers directly to a list of words for the input sentence. It maintains a `static` pointer to the next word of the sentence to be read. The function may be explained by the following pseudo-code:

```
int lexAnalysis( LexPrimitive *lex)
{
  static int inputWordPosition = 1; // Initially points to first
                                    // word of sentence

  if (inputWordPosition is pointing beyond last word of sentence) {
    return EOF;
  }

  Invoke all supra-lexical rules, including the default rule, in
  parallel. If at least one succeeds, return TRUE, else return
  FALSE. The successful supra-lexical function adjusts
  inputWordPosition and f.

  // Actually the above invocation is indirectly through pointers to
  // supra-lexical functions

}
```

The `LexPrimitive` object (`class`) has been introduced in Chapter-2.

A few points must be noted:

- The function `lexAnalysis` is the primary interface between the syntactic and the lexical component.

- The function `lexAnalysis` invokes *all* supra-lexical functions in parallel.
- Hence, there is a *default* supra-lexical function.

The *Default* Supra-Lexical Function returns the f-structure and lexical category of a word as returned by the lexical component. If a word is well-formed and no other supra-lexical function runs successfully, at least the default supra-lexical function succeeds.

Supra-Lexical Functions:

All supra-lexical functions have an identical header structure. The default supra-lexical function has name `supLex0`. The other supra-lexical functions have names `supLex1`, `supLex2`, ... The i-th function in the order of specification has name `supLex`i. The names are actually not important since they are never directly used. Pointers to the functions are stored in a list that is invoked by the function `lexAnalysis`. The header structure of any supra-lexical function (for example the 3-rd one) is:

```
int supLex3(LexPrimitive *lex, int & wordPosition);
```

Note that `wordPosition` is a *reference* parameter, i.e., its value may be changed from inside a function. The pseudo-code of a supra-lexical function reflects the actions specified for the corresponding supra-lexical rule. Every function interacts intimately with the lexical component through the function `primitiveLexAnalysis` described in Chapter-2. As an example, consider the supra-lexical function for Compound Verb specified in the previous section. The supra-lexical function (say `supLex21`) corresponding to it would have the pseudo-code:

```
int supLex21(LexPrimitive *lex, int  & wordPosition)
{
  // The Supra-Lexical Function for Compound Verb.
  //  lex carries back lexical information for the lexical entity.
  //  wordPosition indicates the position of the input word pointer
  //  when this function was called, and may be altered within
  //  Return value is TRUE if this function succeeds, else FALSE

  LexPrimitive *lex1, lex2; // Pointers to lexical information for
                            // the two words of the Compound Verb
  Get memory for lex1, lex2;
```

```
int oldWordPosition = wordPosition; // Save wordPosition

// Global variable sentence, which is an array of *char, the i-th
// entry being for the i-th word, is assumed

if (!primitiveLexAnalysis(sentence[wordPosition], lex1))
  RETURN_FALSE;

// RETURN_FALSE is a macro that un-allocates memory for
// lex1 and lex2, sets wordPosition back to oldWordPosition
// and returns FALSE.

wordPosition++;

if (!primitiveLexAnalysis(sentence[wordPosition], lex2))
  RETURN_FALSE;

wordPosition++;

// Satisfying Word Co-Occurrence Expression which is
// VSTEM+(VCAUS)+VDEC   VSTEM+(VCAUS)+VDEC

int wce = FALSE;

if  ( morpheme class of first morpheme of lex1 is VSTEM  &&
       ( ( morpheme class of second morpheme of lex1 is VCAUS  &&
         morpheme class of third morpheme of lex1 is VDEC )
       || ( morpheme class of second morpheme of lex1 is VDEC ) )
     )  wce = TRUE;

if (!wce) RETURN_FALSE; // Co-Occurrence Test Failed for first word

if   ( morpheme class of first morpheme of lex2 is VSTEM  &&
      ( ( morpheme class of second morpheme of lex2 is VCAUS  &&
         morpheme class of third morpheme of lex2 is VDEC )
       || ( morpheme class of second morpheme of lex2 is VDEC ) )
    )  wce = TRUE;

if (!wce) RETURN_FALSE; // Co-Occurrence Test Failed for second word

// Filter Operation equals( ($1.VDEC TENSE), CONDINF)

Let i be the position of the VDEC morpheme in lex1;

if ( !equals( TENSE slot of i-th f-structure of lex1, CONDINF) )
```

```
  RETURN_FALSE; Filter operation failed

// Filter Operation  compoundable( [$1.VSTEM], [$2.VSTEM]);

Let i be the position of VSTEM morpheme in lex1;
Let j be the position of VSTEM morpheme in lex2;

if (!compoundable(i-th morpheme of lex1,j-th morpheme of lex2))
  RETURN_FALSE; // Filter operation failed

// Filters successful. Performing   TENSE<<($2.VDEC TENSE);

Let j be the position of VDEC morpheme in lex2;

Unify TENSE sub-structure of lex with TENSE sub-structure of lex2;


// Performing GNPH<<($2.VDEC GNPH)

Let j be the position of VDEC morpheme in lex2;

Unify GNPH sub-structure of lex with GNPH sub-structure of lex2;

// Performing #=($1 #);

Assign m-structure of lex1 to m-structure of lex

//  Performing Semantic clause evaluation.

Let n be the number of arguments in the semantic clause of lex1;

char semClause[MAX_SEM_SIZE];
int   s=0;

Set number of arguments in semantic clause of lex to n;

char temp1[MAX_SEM_SIZE];
strcpy(temp1,0-th semantic clause argument of lex1);
char temp2[MAX_SEM_SIZE];
Let j be the position of VSTEM morpheme of lex2;
temp2 = strcpy(j-th morpheme of lex2);
char temp2[MAX_SEM_SIZE];
temp2 = vector(temp2);
strcat(temp1,temp2);
```

```
semClause = temp1;
Copy semClause as s-th semantic clause argument of lex;
s++;

while (s<=n) {
  strcpy(semClause,s-th semantic clause argument of lex1);
  Copy semClause as s-th semantic clause argument of lex;
  s++;
 }
}
```

It is not difficult to generate supra-lexical functions of the above type automatically from supra-lexical rules. Indeed, it is done in our system through a *yacc* specification.

3.4 Discussions

Some of the filters that are required in an actual implementation may require considerable linguistic research of the target language. One example is the filter `Compoundable`. Some linguistic study on which pairs of verb stems may be compounded was carried out by P. Dasgupta [45]. However, subsequent studies by us revealed certain weaknesses in Dasgupta's theories, especially from computer implementation viewpoint. At present we are involved with a feature based solution for the problem. We expect to develop a solution technique, that would do away with the maintenance and look-up of a table for *all* all pairs of stems and instead, work with an $m \times m$ table with m not exceeding 10.

In this chapter, we have considered in detail only a few representative examples from Bangla that require supra-lexical specification. Many others instances, including the ones taken up here, have been taken up in in other chapters, primarily Chapter-5.

As mentioned earlier, the implementation of the different software aspects described in this chapter is quite straightforward, needing nothing more than a parser generator tool like *yacc* to construct a parser for a supra-lexical rule.

Part II

Syntax Division

Chapter 4

Syntactic Analysis

4.1 Introduction

In the preceding two chapters, the necessity of having an efficient morphological parser for lexical interaction has been established. The natural choice of a formalism for the syntactic component would be one which can be best interfaced with the morphological component and also has some inherent ability for parsing a language like Bangla. The LFG (Lexical Functional Grammar [88]) formalism satisfies almost all the requisites because:

- LFG has a built-in control structure equally meaningful for both configurational and non-configurational languages.
- The inter-relationship between grammatical function (gf), configuration (in configurational languages) and morphological case marking is clearly defined in LFG.
- Most of the grammatical formalisms of LFG have direct implementational connotation. In particular, the operators `Locate`, `Merge` and `Include` introduced in [88] have unambiguous semantics.

In spite of the above positive aspects of LFG for the problem at hand, the formalism does have certain drawbacks vis-a-vis implementation of actual parsers for Bangla. We shall show that a parser for Bangla based on pure LFG formalism may have to take care of too many non-deterministic choices. We shall also show that if certain implementation motivated additional features are added to the original formalism,

the above mentioned non-determinism comes down to a manageable level. In the present chapter, we shall establish the need for the extensions and enumerate them. The resultant extended LFG formalism will be called the **Generalized Lexical Functional Grammar** or **GLFG**.

4.2 Background of LFG

LFG is characterized by a two-level description for a sentence called the *c-structure* and the *f-structure*. The c-structure is a context-free description for the surface syntax of the sentence. The f-structure functionally inter-relates the various syntactic components of a major category with respect to the semantic component of the *head* of the category. The f-structure of a category consists of pairs of attribute-name and attribute-value where the attribute name is an atomic *name* and the attribute value could be an atom or another f-structure. The functional pairs are either instantiated through *lexical interaction*, or through *syntactic encoding*. In either case, functional *schema* are introduced into the derivation system. A schemata could be of various types, the most common being $< Left >=< Right >$. The $=$ operator has an operational semantics of *unification*. Either or both of $< Left >$ and $< Right >$ is an atom or a *function application* of the form $(f\ v)$, where f refers to (points to) an f-structure and v is an attribute name in f. The result of an application $(f\ v)$ yields the attribute value of the pair for which v is the attribute name. The notation $(f\ g\ \ldots k\ v)$ is a shorthand for $(((\ldots f\ g)\ \ldots v)$. A unification $l = r$ succeeds if either both l and r are identical atoms, or if l has an undefined value and r has a value (in which case l is made to have the same value as r), or if both l and r are undefined, in which case, both l and r are made to refer to identical structures. The semantics of the above operations are best captured by the operators `Locate` and `Merge`. `Locate` performs function application (if required) to evaluate a descriptor (an evaluation may produce an uninstantiated placeholder). `Merge` performs the unification. The operational semantics of a schemata $l = r$ is $Merge[Locate[l], Locate[r]]$. A more detailed semantics of the LFG may be found in [93].

Lexical interaction project functional descriptions into the immediate pre-lexical nodes of the c-structure of a sentence. Other nodes may be annotated with any number of schema. In the syntax rules for the c-structure, two types of *metavariables* are used to indirectly refer to the f-structure of other nodes. We are not interested here about the bounded metavariables $\Uparrow$ and $\Downarrow$. The immediate metavariables $\uparrow$ and $\downarrow$ refer to the f-structure of the parent category its own f-structure respectively. The immediate metavariables are freely used in annotating c-structure rules to capture various linguistic generalities of the target language. The annotating

schema are introduced into the system during the construction of the syntax tree for a sentence, whence the metavariables get instantiated with actual f-structures (placeholders returned by Locate and Merge). Thus, the annotated LFG rules perform *syntactic encoding* of grammatical relations. What makes LFG interesting to us is that a uniform representation scheme is permitted for both configurational and non-configurational languages in the paradigm, as has been argued and demonstrated by Bresnan [24].

4.2.1 Syntactic Encoding of Grammatical Functions

Power of LFG is derived from its ability of syntactic encoding of grammatical functions (gf-s). The LFG formalism permits efficient encoding of gf-s in both configurational and non-configurational languages. In our work, we are mostly interested about non-configurational languages. The theory of syntactic encoding of gf-s in non-configurational languages is primarily due to Bresnan. An outline of the theory is given below.

Bresnan's Theory of Non-Configurational Syntactic Encoding:

The theory of syntactic encoding of grammatical functions in the context of the LFG has been described by Bresnan in [23][Art. 5.4, pp 296–303], i.e. in [25]. The following discussion has been adopted from the same. While examples in the original were cited from the language *Warlpiri* — an Australian aboriginal language, in our description, we would consider Sanskrit because it closely conforms to the basic assumptions of the theory.

In a non-configurational languages, the c-structure has the basic form $C \longrightarrow X^*$, where C is a major non-lexical category and X is a lexical or non-lexical category. For example, in Sanskrit, the c-structure for a sentence S is of the form:

$S \longrightarrow NP^*\ V; NP^*$

where, NPs are noun phrases (non-lexical category) and V is the verb (lexical category).

According to basic principles of non-configurational encoding, there could be two types of schema annotating an arbitrary category X.

In one type, the encoding schemata has the form $\uparrow{=}\downarrow$. There can be only one category associated with this schemata due to *consistency rules* of LFG and the category is called the *head* of the major category S. In Sanskrit, the schemata $\uparrow{=}\downarrow$ annotates the verb V, making the verb the head of a Sanskrit sentence.

In the other and more common type, the encoding is in the form of a series of *alternations* of pairs of function and feature assignment equations of the form:

$$\begin{array}{rcl}(\downarrow\ F) & = & V \\ (\uparrow\ G) & = & \downarrow\end{array}$$

In the above equation pair, G is a function selected by the value V of a feature F. For example, the fact that the OBJ function of an noun phrase NP in Sanskrit is selected from the ACCusative case marker on it, may be expressed by:

$$\begin{array}{rcl}(\downarrow\ CASE) & = & ACC \\ (\uparrow\ OBJ) & = & \downarrow\end{array}$$

The two equations above may be combined into a single one if the feature names are not bound by extraneous conventions. For example, one may choose the call the ACCUsative case marker an OBJective case marker and associate the feature OBJ with it. Substituting OBJ for ACC in the above equation pair, we may get:

$$\begin{array}{rcl}(\downarrow\ CASE) & = & OBJ \\ (\uparrow\ OBJ) & = & \downarrow\end{array}$$

Now, eliminating the term OBJ from the above equations, we get a single equation:

$$(\uparrow\ (\downarrow\ CASE)) = \downarrow$$

The equation obtained is now independent of the function name G in the original pair. It expresses the notion that the grammatical function of an NP in a language like Sanskrit is *selected uniquely and non-ambiguously from the CASE feature of the phrase.* Under such an assumption, the alternation in the original form of the schema is no longer necessary since each alternating term becomes identical.

In most real natural languages (as we shall see later), the selection of grammatical functions from CASE features of the NPs of a sentence is not non-ambiguous. A particular CASE feature F_i may select more than one functions $G_{i_1}, G_{i_2}, \cdots G_{i_k}$. If there are j possible values $v_1, v_2, \cdots, v_j$ for the CASE feature in the language, the general form of the schema under NPs would be:

$$\left\{ \begin{bmatrix} (\downarrow\ CASE) & = & v_1 \\ (\uparrow \begin{Bmatrix} G_{1_1} \\ G_{1_2} \\ \vdots \end{Bmatrix}) & = & \downarrow \end{bmatrix} \begin{bmatrix} (\downarrow\ CASE) & = & v_2 \\ (\uparrow \begin{Bmatrix} G_{2_1} \\ G_{2_2} \\ \vdots \end{Bmatrix}) & = & \downarrow \end{bmatrix} \right\}$$

$$\vdots$$

Although the above schema looks formidable, it is not really so. A particular *NP* has a unique case feature (say v_i). Hence, only one choice from the outer alternation is valid, namely the one having $(\downarrow\ CASE) = v_i$ as the feature assigning (and hence testing) schemata. The inner alternation ranging over grammatical functions is language dependent. Languages like Sanskrit have lower occurrences of them and are hence more ideal. In most modern Indian languages (including Bangla) however, there are considerable alternations at this level. We shall show later in this chapter, how the projection of the verb introduces another set of schema that picks only a few (trivially one) alternate values for grammatical function from the inner alternation in the above schema.

4.2.2 Syntactic Encoding in Actual Non-Configurational Languages

Attempts have been made to use syntactic encoding of grammatical relations in non-configurational languages like Malayalam (Mohanan [131]), Icelandic ([8]) and Russian ([134]). In most cases, extensive and tedious language specific study about the phenomenon is necessary. Notational tools like alternation, situation dependent order of unification evaluation, etc. are freely used. However, the refinements used in the above works, were not mapped onto the Locate and Merge operators. In fact, implementation of the notations used could lead to non-trivial problems in schema evaluation ordering (LFG claims that schema may be evaluated absolutely in an order independent manner) as well as satisfaction of too many disjunctive constraints. As we shall discuss, even efficient disjunctive constraint satisfaction algorithms are NP-complete in the worst case.

For Indian languages, most of the non-determinisms arise due to that fact that the gf of a noun phrase or post-positional phrase is not encoded by the morphological or post-positional case marker with sufficient certainty (some of which Mohanan [131] has taken care of by complicated situation dependent control schema). Under extreme cases a case marker may encode almost every gf. It is therefore not always

possible to determine with certainty the functional role of a phrase form its case marker alone. The degree of uncertainty however reduces drastically if an a-priori knowledge of the verb is available.

4.2.3 Implementation Semantics of an LFG Parser

The implementation semantics of LFG is defined in terms of the three operators `Locate`, `Merge` and `Include`. While `Locate` is an advanced form of list search, `Include` is a set manipulator. The most interesting operator however is `Merge`, which is primarily a unification procedure of an advanced type. An overall LFG parser may be visualized as a context-free grammar combined together with a unification process.

The efficiency of an LFG parser depends mostly on the efficiency of the unification process. Performance of any unification process degrades rapidly with added non-determinism, a fact that has been underlined in [115], which is a good survey of unification algorithms of various types. Kaplan [91] and Knight [115] have shown that non-determinism due to disjunctive structure specification leads to a unification problem which is NP-complete. There have however been attempts like [91] and [124] towards developing unification (permitting non-determinacies) algorithms with better average case behaviour. These modern methods are relevant in cases where non-determinisms are otherwise unavoidable as in instances of unbounded uncertainties investigated in [89, 90, 91, 97]. For Bangla, the "uncertainties" are quite bounded, in the sense that syntactic encoding of grammatical functions of even the matrix can only be practically expressed with a large number of alternations. While such a lavish use of alternations may retain expressibility of the grammar, it leads to a very inefficient parser implementation. It therefore pays to have a formalism that leads to a lower degree of parser non-determinism, while maintaining the other advantages of the LFG.

4.3 Outline of Proposed Solution

Using Mohanan's approach, the syntax rule for a sentence in an Indian language like Bangla may be given by (4.1).

$$
\begin{array}{lllll}
S & \longrightarrow & NP^* & PP^* & S' \\
 & & (\uparrow\ (\downarrow\ CASE)) = \downarrow & (\uparrow\ (\downarrow\ PCASE)) = \downarrow & (\uparrow\ COMP) = \downarrow \\
 & & V & NP^* & PP^* \\
 & & & (\uparrow\ (\downarrow\ CASE)) = \downarrow & (\uparrow\ (\downarrow\ PCASE)) = \downarrow
\end{array}
\tag{4.1}
$$

Rule (4.1) represents a "flat" constituent structure for a sentence S and suggests that the constituent phrases (NPs and/or PPs) may freely permute among themselves. The S' category represents an "embedded clause" as in complex sentences. Embedded clauses have been discussed in Sec-4.4. Using Bresnan's approach, syntactic encoding of gf-s is carried out in (4.1) using the schemata (4.2) under the NPs.

$$(\uparrow\ (\downarrow\ CASE)) = \downarrow \tag{4.2}$$

4.1 is of course based on Bresnan's postulate of non-configurational syntactic encoding of gf-s described earlier.

Table 4.1: Case Marker Versus Possible Grammatical Functions

Marker	Name	SUBJ	OBJ	IOBJ	ADJUNCT
None	NULL	•	•	•	•
+ke, +re	DATive	•	•	•	
+e, +te	OBLique	•	•	•	•
+er	GENitive	•			

Note: The GEN case marker normally marks a genitive qualifier of a noun. However, for certain verb forms (for example, ones in pseudo-passive *voice*), it also marks the subject.

The underlying assumption in (4.2) is that a case marker so strongly predicts a function that mapping from case marker to function is nearly one-one. However, from Table-4.1, it is clear that in Bangla (and also other modern Indian languages) almost every marker has many-to-one mapping. Thus alternations must be used in (4.1). The parser would accept only those choices that are consistent with the *lexical projection of the verb.*

Use of alternations obviously adversely affects the efficiency of the unification component. It may be easily demonstrated in Bangla (and most Indian languages) that if *a-priori* lexical knowledge of the verb is available during parsing of a constituent NP, the number of disjunctive choices drastically reduces — to nearly one (the ideal situation) in a large number of cases. For example, if it is known that the verb (stem) is **di** (give), the case marker **ke** predicts uniquely the function Indirect OBJect (IOBJ). The formal treatment is as follows:

Let $G = \{g_1, g_2, \cdots\}$ be the set of relevant grammatical functions and let $C = \{c_1, c_2, \cdots\}$ be the set of case markers for NPs for the target language. Let $cToG$ be a mapping from case markers to grammatical functions such that $cToG(c)$, $c \in C$ is (are) the function(s) predictable from c. In the scenario described, $cToG(c)$ is actually a finite disjunction $g_{i_1} \vee g_{i_2} \vee \cdots$ of functions. Let f_N be the f-structure of an NP which is a child of a sentence (or clause) S with f-structure f_S. Let the case marker on the head noun of the NP be c. Under these circumstances, the implementation semantics of the schemata (4.2) that annotates the NP may be expressed as $(f_S\ cToG[c]) = f_N$, where "=" denotes unification. With $cToG[c]$ being in general a disjunction, the above form effectively "multiplies out" to $|cToG(c)|$ *non-deterministic* choices for the functional role played by the f_N in f_S. If in the ultimate analysis the NP is found to play the functional role g in f_S, the constraints set (4.3) projected by the verb must have been satisfied.

$$
\begin{array}{c}
(f_S\ g\ CASE) = c \\
(f_S\ g\ q_1) = v_1 \\
(f_S\ g\ q_2) = v_2 \\
\vdots \\
(f_S\ g\ q_k) = v_k
\end{array}
\qquad (4.3)
$$

where q_i are different normal agreement features (like NUMBer, PERSon, etc.) and/or other semantic agreement features (like ANIMacy, etc.). We shall call the schema (4.3) the *agreement* schema for the function g projected by the verb. Observations show that in most well formed sentences, the agreement schema of the verb for any function g is satisfied by *at most one* constituent NP of the sentences. However, some order of processing the agreement schema of different gf-s) must be maintained. The mapping $cToG$ is thus nearly one-to-one *in the context* of the agreement schema of the verb, which can very well serve as *test* criteria for selecting grammatical functions from internal properties of NPs. The parser must ensure evaluation of an encoding schemata of a constituent NP in the context of the agreement schema of the verb, somewhat like handling a "forward reference" (where a referred item is defined "later" than the places where it has been referred to). The trick is to "delay" the evaluation of encoding schema of constituent NPs till an appropriate moment while maintaining a persistent data

structure like a *Symbol Table* to keep track of the points of forward reference (at which actual function names get instantiated) and their local environments (the internal f-structure of the constituent NPs).

4.3.1 Solution Part-I, Initiation of Forward Reference

A forward reference discussed above is encountered during Locate-ing the left hand side of a schemata like (4.2) while processing an NP. In our "delayed" encoding proposal, the (modified) Locate operation should leave the "name" of the functional role played by the NP as "under-specified". To force the Locate operator to behave in this manner, we propose:

- The introduction of a new type of *under-specification* metavariable "?".
- The encoding schemata (4.2) be modified to (4.4). With that, the modified rule for S is as given in (4.5).

(4.4) $(\uparrow ?) = \downarrow$

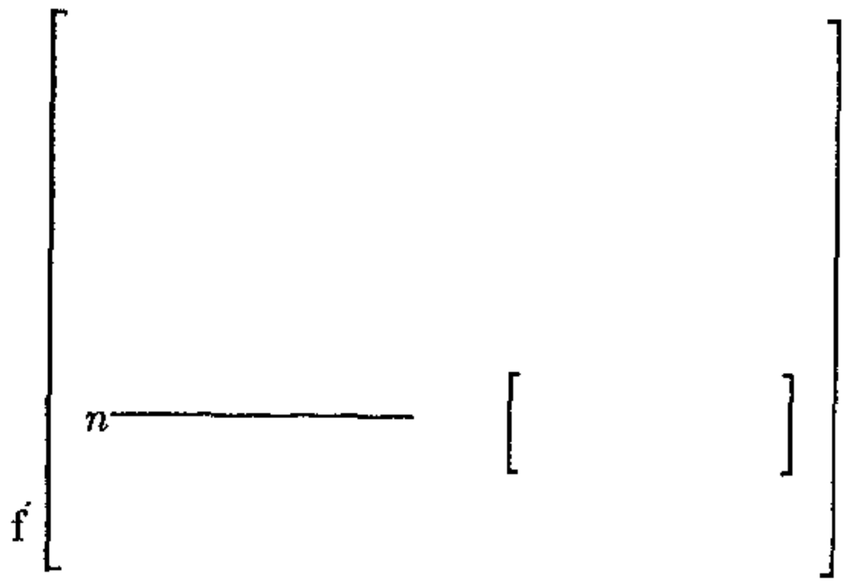
(4.5)
$$\begin{array}{lcccc} S & \longrightarrow & NP^* & PP^* & S' \\ & & (\uparrow (\downarrow CASE)) = \downarrow & (\uparrow (\downarrow PCASE)) = \downarrow & (\uparrow COMP) = \downarrow \\ & & V & NP^* & PP^* \\ & & & (\uparrow (\downarrow CASE)) = \downarrow & (\uparrow (\downarrow PCASE)) = \downarrow \end{array}$$

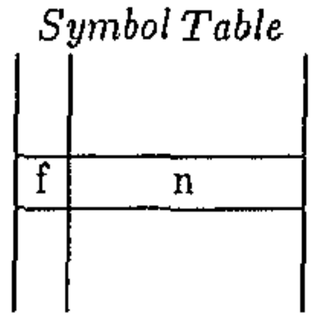
The ? metavariables generate placeholders for a hitherto anonymous grammatical functions. Such a placeholder will be called a *nameholder* and denoted by actual name variables $n_1, n_2, \cdots$. Locate-ing of a schemata like (4.4) creates such a nameholder (n say) in the *scope* of the functional placeholder (f say) for the $\uparrow$ metavariable and simultaneously stores the pair (f, n) in the Symbol Table. Locate-ing a construct like $(f\ n)$ where both f and n are already defined placeholder and nameholder respectively, returns (a pointer to) the "value" part of the pair in the f-structure (pointed at by) f, whose "name" is (pointed at by) n. With this, the semantics of Locate with respect to form (4.6), the left hand side of the schemata (4.4) is given below and pictorially represented in (4.7).

(4.6) $(\dagger ?)$

Locate[d], where d has the form (x y). Let f be the reference to a f-structure Locate[x]. If y is a ? metavariable, let n be a new nameholder for the metavariable. An anonymous slot is created in the scope of f and n is made to point to it. Simultaneously, the pair (f,n) is entered as a new entry of the Symbol Table. If however y is a nameholder n, Locate returns the value field of the pair in f whose name field is held by n.

(4.7)





4.3.2 Solution Part-II, Precipitation of Forward Reference

The next point to be considered is how to bind actual function names to nameholders. We assume that the agreement schema for a function g may select the structure which satisfies the constraints. For this, the agreement schema must be handled in a different manner than normal projection schema. We choose the notation $(\#\ g\ q) = v$ for one agreement schemata for the function g. We shall call the forms $(\#\ g\ q_i) = v_i$ a meta-structure or *m-structure*. M-structure schema are projected by the main verb of a sentence. A symbol table entry (f, n) satisfies a m-structure schemata $(\#\ g\ q_i) = v_i$ projected by the verb V of a sentence S, if f is the f-structure of S and the structure $(f\ n)$ (where n is treated as an atom) contains the pair $[q_i\ \ v_i]$. If a symbol table entry satisfies all m-structure schema for a function g, by our proposed scheme, the nameholder n that points to the entry is bound to the function name g. Also, the satisfying symbol table entry is deleted.

Testing of symbol table entries with m-structure schema and resulting binding of nameholders to actual function names are carried out by a newly introduced operator Search. The operator Search takes the entire set m-structure schema for a particular gf and carries out the process described in the previous paragraph. If more than one symbol table entry satisfy the m-structure schema for a particular function g, the one earlier in order of occurrence is chosen. The relative evaluation

(by operating with Search) order for the sets of m-structure schema for different functions is motivated by the *default* ordering of phrases in a sentence in the target language. In Bangla for example, the default ordering is SUBJ-IOBJ-OBJ-COMP. Note that the COMP function is played by an embedded clause like structure and not encoded by m-structure schema. Thus, the test for SUBJ is carried out first, followed by IOBJ and OBJ if any. The proposed solution technique can be implemented in the following sequence.

First, all f-structure schema, including ones having under-specification metavariables, annotating the children node of an S dominated c-structure tree are evaluated. This generates symbol table entries corresponding to NPs annotated with under-specification schema. Next, the m-structure schema of the main verb are operated on with the Search operator in the default phrasal order for the language. A sentence is well-formed if and only if all the m-structure schema for the verb are satisfied and all nameholders in the scope of the sentence are bound to names (i.e. at the end, symbol table is empty). Note that the evaluation process naturally satisfies the uniqueness property for sentence level grammatical functions.

Regarding the relative evaluation order of f- and m-structure schema, the general principle is "all f-structure schema are evaluated before any m-structure schemata is evaluated (i.e. fed to the Search operator). There are some exceptions to the general principle as explained in the following section.

Let us consider the Bangla simple sentence of Example-1, in which the NPs have been underlined.

Example-1:

a'pni a'ma'ke ekt'a' bai deben
You(honoured)-NULL I-DAT one-DEF book-NULL give-3p-hon-PAST
You (honoured) will give me a book

Any permutation of the underlined phrases and the verb should give identical results. The lexical entries of the head nouns and the verb are given in (4.8)–(4.11). For an explanation of the GNPH feature, see Chapter-5.

(4.8)
$$\begin{aligned} &\textbf{a'pni} \quad N \quad (\uparrow GNPH) = 2/3p - 2h, \\ &\qquad\qquad (\uparrow ANIM) = +, \\ &\qquad\qquad (\uparrow PRED) = '\,you', \\ &\qquad\qquad (\uparrow CASE) = NOM \end{aligned}$$

$$
\begin{array}{lll}
\textbf{a'ma'ke} & N & (\uparrow\ GNPH) = 1p, \\
 & & (\uparrow\ ANIM) = + \\
 & & (\uparrow\ PRED) =' I', \\
 & & (\uparrow\ CASE) = DAT
\end{array}
\tag{4.9}
$$

$$
\begin{array}{lll}
\textbf{bai} & N & (\uparrow\ GNPH) = 3p - 1h, \\
 & & (\uparrow\ ANIM) = - \\
 & & (\uparrow\ PRED) =' book', \\
 & & (\uparrow\ CASE) = NOM
\end{array}
\tag{4.10}
$$

$$
\begin{array}{lll}
\textbf{dilen} & V & (\uparrow\ TENSE) = PAST \\
 & & (\uparrow\ GNPH) = 2/3p - 2h \\
 & & (\uparrow\ PRED) = \\
 & & 'give < (SUBJ), (OBJ), (IOBJ) >' \\
 & & (\#\ SUBJ\ GNPH) = 2/3p - 2h \\
 & & (\#\ SUBJ\ ANIM) = + \\
 & & (\#\ SUBJ\ CASE) = NOM \\
 & & (\#\ IOBJ\ ANIM) = + \\
 & & (\#\ IOBJ\ CASE) = DAT \\
 & & (\#\ OBJ\ ANIM) = - \\
 & & (\#\ OBJ\ CASE) = NOM
\end{array}
\tag{4.11}
$$

The f-structure f_S of the sentence before processing the m-structure of the verb appears like (4.12) and the final solution is given by (4.13). The f-structures f_a, f_b and f_c are for the NPs in order.

$$
\begin{array}{l}
\left[\begin{array}{lll}
n_1 & & \left[\begin{array}{ll} ANIM & + \\ CASE & NOM \\ GNPH & 2/3p-2h \\ PRED & 'You' \end{array}\right] \\
 & f_a & \\
n_2 & & \left[\begin{array}{ll} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & 'I' \end{array}\right] \\
 & f_b & \\
n_3 & & \left[\begin{array}{ll} ANIM & - \\ DEF & YES \\ CASE & NOM \\ GNPH & 3p-1h \\ PRED & 'book' \end{array}\right] \\
 & f_c & \\
TENSE & & PAST \\
PRED & & 'give < \cdots >' \\
f_S & &
\end{array}\right]
\end{array}
\tag{4.12}
$$

Symbol Table

f_S	n_1
f_S	n_2
f_S	n_3

(4.13)
$$
fs\begin{bmatrix}
SUBJ & \begin{bmatrix} ANIM & + \\ CASE & NOM \\ GNPH & 2/3p-2h \\ PRED & 'You' \end{bmatrix} \\
IOBJ & \begin{bmatrix} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & 'I' \end{bmatrix} \\
OBJ & \begin{bmatrix} ANIM & - \\ DEF & YES \\ CASE & NOM \\ GNPH & 3p-1h \\ PRED & 'book' \end{bmatrix} \\
TENSE & PAST \\
PRED & 'give<(SUBJ),(OBJ)(IOBJ)>'
\end{bmatrix}
$$

If on the other hand, the sentence is as in Example-2, with the lexical entry of the verb **baklen** as in (4.14), the NP **a'ma'ke** is selected as the OBJect.

Example-2:

a'pni a'ma'ke baklen
You(honoured)-NULL I-DAT scold-3p-hon-PAST
You (honoured) scolded me

(4.14)
$$
\begin{array}{lll}
\textbf{bakben} & V & (\uparrow TENSE) = PAST \\
 & & (\#\ SUBJ\ GNPH) = 2/3p-2h, \\
 & & (\#\ SUBJ\ ANIM) = + \\
 & & (\#\ SUBJ\ CASE) = NOM \\
 & & (\#\ OBJ\ ANIM) = + \\
 & & (\#\ OBJ\ CASE) = DAT
\end{array}
$$

4.4 Embedded Clause and Complex Sentences

In most of the languages under study there is a class of complex sentences which is somewhat similar to the "to" infinitive class of sentences of English. In English, the to-infinitive dependent clause of a complex sentence is normally described as the complement of the verb, or VCOMP. But as there is no concept of VP in the languages under study, the dependent clause is described as a general COMPlement

of the matrix. Here we consider only Bangla complex sentences. Other Indian languages have similar constructs for complex sentences.

In Bangla, there is no distinctive particle (like "to" in English) to serve as a connective between the matrix and the complement. Instead, the complement clause is *embedded* within the matrix. The complement verb takes either the FORINF or any of the FUTURE tense declensions[1]. Depending upon whether the declension on the verb of the complement clause is FORINF or indicates the FUTURE tense, the SUBJect (and sometimes the OBJect) of complement clause is derived from the IOBJect and SUBject of the matrix, respectively. The case marker of the IOBJect NP of the matrix is invariably DATive. Moreover, if the verb of the complement is in FUTURE tense, the verb of the matrix must be a "say" (or *equi*) type verb, most commonly **bal** (stem). The verb (stem) bal of Bangla is quite versatile in this respect. Different possibilities of complex sentences with bal as the main verb (stem) has been discussed in Sec-7.4. The rules for the sentence, as well as the embedded clause are given in (4.15) and (4.16). However, in the subsequent discussions in this chapter, we consider only dependent clauses with FORINF verbs.

$$\begin{array}{lcccc} S & \longrightarrow & NP^* & PP^* & S' \\ & & (\uparrow ?) = \downarrow & (\uparrow\ (\downarrow\ PCASE)) = \downarrow & (\uparrow\ COMP)) = \downarrow \\ & & V & NP^* & PP^* \\ & & & (\uparrow ?) = \downarrow & (\uparrow\ (\downarrow\ PCASE)) = \downarrow \end{array} \tag{4.15}$$

$$\begin{array}{lcccc} S' & \longrightarrow & NP^* & & PP^* \\ & & (\uparrow ?) = \downarrow & & (\uparrow\ (\downarrow\ CASE)) = \downarrow \\ & & V & & \\ & & (\downarrow\ INFTYPE)) = FORINF & & \\ & & \{\ (\downarrow\ TENSE)) = FUTURE & & \} \\ & & (\downarrow\ EQUI) & & \\ & & (\uparrow\ SUBJ\ GNPH) = (\downarrow\ GNPH) & & \end{array} \tag{4.16}$$

(See Sec-7.4 for a discussion on the utility of the schemata $(\uparrow\ SUBJ\ GNPH) = (\downarrow\ GNPH)$ in (4.16)).

In complex sentences, the SUBJect of the embedded clause is generally omitted and is *derived* from some function of the matrix through a *control* schemata (introduced by Bresnan [24]) on the main verb. For example, the control schemata (4.17) derives the subject of the embedded clause from the indirect object of the matrix.

(4.17) $(\uparrow\ COMP\ SUBJ) = (\uparrow\ IOBJ)$

[1]The nomenclature has been explained in Chapter-5

From rules (4.1) and (4.16), the surface structure of a sentence with embedded clause could be NP^* V_{FORINF} NP^* V NP^*, where V_{FORINF} is the embedded verb. From (4.16), it is clear that the NPs occurring to the right of the embedded verb can never be participants of the complement clause. Although participating phrases of the complement clause do not permute with the phrases of the matrix, it is a difficult problem for the parser to determine a-priori which pre-V_{FORINF} NPs belongs to the complement and which to the matrix. Moreover, if a control schemata like (4.17) is evaluated before any m-structure, the Locate operator handling the same would tend to create slots SUBJ and IOBJ in f-structures of the embedded clause and matrix respectively, violating the principles of delayed functional encoding. Considering above difficulties, we propose the following additional strategies:

i. The m-structure of the verb of a complement clause may not project a m-structure for the SUBJect.

ii. Initially, the f-structures of the participating NPs of an S with or without a constituent S', are assumed to be defined in the scope of the matrix. Thus, initially, the f-structure of the embedded clause have no reference to any gf.

iii. All functional encoding schema are scheduled for evaluation after the m-structure of the main verb.

iv. The m-structure for the embedded verb are scheduled for evaluation after functional encoding schema.

v. After evaluating the m-structure of the main verb, all remaining symbol table entries (as well as the name-value pairs pointed at by them) are removed from the scope of the f-structure of the matrix and attached to the f-structure of the complement.

vi. The m-structure for the complement's verb is evaluated last following which emptiness of symbol table is checked to determine well-formedness.

Example-3:

a'pni a'ma'ke ekt'a' bai pad'.te ballen

You(honoured)-NULL I-DAT one-DEF book-NULL read-FOR-INF say-3p-hon-PAST

You (honoured) asked me to read a book

(4.18)

$$\begin{array}{lll} \textbf{pad'.te} & V & (\uparrow TENSE) = FORINF \\ & & (\uparrow PRED) =' read < (SUBJ),(OBJ) >' \\ & & (\#OBJ\ ANIM) = - \\ & & (\#OBJ\ CASE) = NOM \end{array}$$

(4.19)

ballen V $(\uparrow\ TENSE) = PAST$
$(\uparrow\ GNPH) = 2/3p - 2h$
$(\uparrow\ PRED) =$
$'ask < (SUBJ), (COMP), (IOBJ) >'$
$(\uparrow\ COMP\ SUBJ) = (\uparrow\ IOBJ)$
$(\#\ SUBJ\ GNPH) = 2/3p - 2h$
$(\#\ SUBJ\ ANIM) = +$
$(\#\ SUBJ\ CASE) = NOM$
$(\#\ IOBJ\ ANIM) = +$
$(\#\ IOBJ\ CASE) = DAT$

Note:
1. There is no m-structure schema for SUBJect in the lexical entry of **pad'.te** in (4.18).

2. The functional control schemata $(\uparrow\ COMP\ SUBJ) = (\uparrow\ IOBJ)$ in the lexical entry (4.19) of the "equi" verb **ballen** derives the subject of the complement from the indirect object of the matrix.

Let f_S and f_E be the f-structures for the matrix and the embedded clause, respectively. The situation just before processing the m-structure of the main verb is given in (4.20). After processing the m-structure of the main verb and removing all unprocessed entries of the symbol table from the scope of f_S to the scope of f_E, the situation is as in (4.21). Finally, after processing the m-structure of the

complement's verb, the final result is given by (4.22).

$$
\left[\begin{array}{ll}
n_1 & \left[\begin{array}{ll} ANIM & + \\ CASE & NOM \\ GNPH & 2/2p-2h \\ PRED & \text{'You'} \end{array}\right] \\
 & f_a \\
n_2 & \left[\begin{array}{ll} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & \text{'I'} \end{array}\right] \\
 & f_b \\
n_3 & \left[\begin{array}{ll} ANIM & - \\ DEF & YES \\ CASE & NOM \\ GNPH & 3p-1h \\ PRED & \text{'book'} \end{array}\right] \\
 & f_c \\
COMP & \left[\begin{array}{ll} TENSE & FORINF \\ PRED & \text{'read} < \cdots > \text{'} \end{array}\right] \\
 & f_E \\
TENSE & PAST \\
PRED & \text{'ask} < \cdots > \text{'} \\
 & f_S
\end{array}\right] \tag{4.20}
$$

Symbol Table

f_S	n_1
f_S	n_2
f_S	n_3

$$
\left[\begin{array}{ll}
SUBJ & \left[\begin{array}{ll} ANIM & + \\ CASE & NOM \\ GNPH & 2/3p-2h \\ PRED & \text{'You'} \end{array}\right] \\
IOBJ & \left[\begin{array}{ll} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & \text{'I'} \end{array}\right] \\
COMP & \left[\begin{array}{ll}
n_3 & \left[\begin{array}{ll} ANIM & - \\ DEF & YES \\ CASE & NOM \\ GNPH & 3p-1h \\ PRED & \text{'book'} \end{array}\right] \\
TENSE & FORINF \\
PRED & \text{'read} < \cdots > \text{'} \\
 & f_E
\end{array}\right] \\
TENSE & PAST \\
PRED & \text{'ask} < \cdots > \text{'} \\
 & f_S
\end{array}\right] \tag{4.21}
$$

Symbol Table

f_E	n_3

(4.22)

$$
fs\left[\begin{array}{ll}
SUBJ & \left[\begin{array}{ll} ANIM & + \\ CASE & NOM \\ GNPH & 2/3p-2h \\ PRED & 'You' \end{array}\right] \\
IOBJ & \left[\begin{array}{ll} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & 'I' \end{array}\right] \\
COMP & \left[\begin{array}{ll} OBJ & \left[\begin{array}{ll} ANIM & - \\ DEF & YES \\ CASE & NOM \\ GNPH & 3p-1h \\ PRED & 'book' \end{array}\right] \\ SUBJ & \\ TENSE & FORINF \\ PRED & 'read<(SUBJ),(OBJ)>' \end{array}\right] \\
TENSE & PAST \\
PRED & 'ask<\cdots>'
\end{array}\right]
$$

4.4.1 Operational Semantics of Locate and Search

The definitions of the Locate (*modified to take care of the "?" metavariable*) and Search operators are given below. The essential aspects of Locate have been adopted from [88][Appendix-F, pp 273]. The collection of entities and variable assignments C and the definition of the Substitute sub-operator are assumed as described in the original source. We shall use S to denote the symbol table.

Locate:

The Locate operator takes a designator d as input. A successful operation finds a new value for d and may alter the collection C and/or the symbol table S.

i. If d is an entity in C, Locate[d] is d.

ii. If d is a symbol or semantic form character string, Locate[d] is the symbol or semantic form with that representation.

iii. If d is a variable,

If d is already assigned a value in C, Locate[d] is that value.
Otherwise, a new place-holder is added to C and assigned as the

value of d. Locate[d] is that new place-holder.

iv. If d has the form $(x\ y)$, where y is ?. Let F be the entity Locate[x] and n be a new name-holder for y. A new pair with an anonymous name slot and a place-holder in the value slot is created in the scope of F and n is made to point to the name slot. Simultaneously, the pair (F, n) is entered as a new entry of S. Locate[d] is that place-holder for the value part of the newly created pair.

v. If d is of the form $(x\ n)$ where y is a nameholder. Locate[d] is the value field of the pair in Locate[x] whose name field is held by n.

vi. Otherwise, d is a function-application expression of the form $(f\ g)$. Let F and G be the entities Locate[f] and Locate[g], respectively. There is no solution if G is not a symbol or place-holder, or if F is not an f-structure or place-holder. If F is an f-structure:

> If G is a symbol or place-holder with a value defined in F, Locate[d] is that value.
>
> Otherwise, G is a place-holder or a symbol for which F has no value. F is modified to define a new place-holder as the value of G and Locate[d] is that place-holder.
>
> Otherwise, F is a place-holder. A new f-structure F' is constructed with a single pair that assigns a new place-holder value to G and Substitute[F', F] is performed. Locate[d] is the new place-holder value.

Search:

The Search operator takes as argument a collection M of m-structure schema of the form $(\#\ g\ q_i) = v_i$. Note that every m-structure of M has the same gf g as the second entity of the left hand side of $=$.

We say that a symbol table entry (f, n) *satisfies* the m-structure schemata $(\#\ g\ q_i) = v_i$ if and only if Locate[$((f\ n)\ q_i)$] yields the value v_i. A symbol table entry satisfies a collection of m-structure schema if it satisfies every schemata of the collection.

If there is no symbol table entry satisfying the argument M of Search, there is no solution. Otherwise, let (f, n) be the first entry of S satisfying M. The name-holder n is bound to the name g and the satisfying entry is deleted from S.

There is no modification in the semantics of the Merge and Include operators. However, with introduction of the delayed encoding concepts, an order has to be imposed on the evaluation of various entities in C as explained earlier in this section.

4.5 Implementation Notes

Implementation of the basic operations of LFG is easier in LISP like type-less functional languages, rather than in conventional programming languages. However, due to the type inheritance property of C++, most of the facilities of LISP are retained. C++ permits easy construction of data-structures representing *heterogeneous containers* of objects (that are inherited from a common type). We have used this fact to our advantage.

4.5.1 Object Design

At the top of our object hierarchy for GLFG parser implementation, we have an *abstract class* called `GLFGObject`. There are a few immediate sub-classes of `GLFGObject`, namely,

- `AtomObject`, for atomic entities.
- `StringObject`, for atomic strings.
- `FStructObject`, for f-structures.
- `MStructObject`, for m-structures.
- `PairObject`, for Pairs.

Each of the above objects has a unique readable Class-Id.

The `Holder` object is basically a pointer to a `GLFGObject`. The `GLFGObject` carries a *virtual overloadable* function `eval()` that is overloaded by each descendant class of `GLFGObject`. A holder object is a *variable* if it is itself non-NULL and the object pointed at by it evaluates to a non-NULL value. A NULL holder or a holder that points to an object evaluating to NULL is a *placeholder*. However, a NULL holder with only the `mayBeAtom` flag on, is a *nameholder*. Every `Holder` is associated bit-valued flags `mayBeXX`, where XX could be ATOM, STRING, FSTRUCT, MSTRUCT or PAIR. If a flag is OFF, the holder may not point to an object of the corresponding class. When any holder data receives a non-NULL value through assignment, the validity of assignment is checked against the flags. A flag may not be OFF in one and ON in the other between the holder being assigned and holder assigned from. Any `Holder` object is constructed as a NULL-pointing placeholder with all flags turned ON. The flags may be altered after construction. The `what`

member function of a holder object returns the GLFGObject pointed at by the holder. In the GLFG environment, two or more holders may be unified, whence, they not only hold identical objects but also an updation of the held object of one automatically updates the held objects of the unified holders. To achieve this, we let every Holder object contain a variable to a HolderList object.

A HolderList object is a *collection* of pointers to Holder objects. Its main purpose is to bind unified holders. At any point of time, all holders of a holder list points to identical structures.

A PairObject has two Holders name and value. The name holder has only the mayBeATOM and mayBeMSTRUCT flags turned ON while value holder has all but the mayBeMSTRUCT flag ON.

The most interesting object in our system is FStructObject. This object has two fields — pairs, which is a container of PairObjects and unionList, which is a container of Holder objects, where each contained holder in turn points to this object. The following important member functions are associated with FStructObject:

- int addPair(PairObject *p). This function searches whether there is a pair in the pairs container having the same name as p. If not, p is added to pairs and TRUE value returned. Otherwise, if the value fields of the matching pairs are identical, return TRUE, else return FALSE.

- Holder locate(AtomObject *a). Searches for the names of each pair in pairs to be a. If yes, the value part of the matching pair is returned. Otherwise a new pair with *name* pointing to a and value field a placeholder pointing to a newly created FStructObject is created and added to pairs. The placeholder value is returned.

- Holder locate(Holder *n). Here n must be a nameholder and hence must be having a NULL pointer value. Create a new pair with a nameholder as the name field and a placeholder is created in the value field as above, and returned.

- int unify(FStructObject *anotherF). This function has been described in more detail in Appendix-C. A point to remember is that f1->unify(f2) and f2->unify(f1) have identical results and may modify the contents of both f1 and f2. As a result of unification, the pairs and unionList containers of f1 and f2 must become identical.

The collection C of entities and variable assignments is a *global container of* Holder objects. During construction of C it is made to contain holder objects pointing to

all atoms and all strings defined in the system. A holder pointing to an atom has only the `mayBeATOM` flag on, while a holder pointing to a string has only the `mayBeSTRING` flag on.

The `Stack` is a global container of `StackEntry`. A `StackEntry` has three fields — two placeholders and a nameholder.

Each `GLFGObject`, as well as the `Holder` object overloads the function `locate`, the return value of which is a (pointer to a) holder. Locate of a holder object is itself. Locate of an atom or a string is a holder in `C` pointing to the atom or string. Locate of an f-structure is any of the holders in the `unionList` of the f-structure. Locate operations on a pair objects and m-structure objects are illegal.

An `ApplicationPair` object is derived from `PairObject`. However, the flags on the `name` and `value` holders are different during construction. The name field has only the `mayBeFSTRUCT` flag on. The value field may have any one of the flags — `mayBeATOM`, `mayBeFSTRUCT` or `mayBeQMARK` (a new flag) on representing GLFG applications expressions corresponding to $(f\ a)$, $(f\ g)$ and $(f\ ?)$, respectively. Here f and g are place-holders to f-structures and a is any atom. Locating an `ApplicationPair` object works as described in Step-d of `Locate` operation in Sec-4.4.1.

An `MStructObject` has two members — a `functionName` of type `AtomObject` and `metaConstraints` which is a container of `Constraint` objects (see below). The member function `search` performs precipitation of forward reference of `functionName`, consulting the global `Stack` in the process. A global container `M` of `MStructObjects` holds all the m-structure schema projected during lexical interaction.

At the next level of abstraction, there is an `Equation` object, which contains a `left` side and an optional `right` side. An equation may be `fired`. If both left and right sides of an equation exists, firing it performs the task
`left->locate->what()->unify(right->locate->what())`.
Firing of an one-sided equation, also called a `Constraint` object, involves appending the constraint to a global container `constraintList`. A constraint can be checked against validity.

C-Structure rules are specified in *yacc* syntax. Tokens represent lexical categories like VERB, NOUN, DET, etc., and other non-terminals represent phrasal categories.

4.5.2 Actual Parsing

The parser makes two "passes" over the input sentence. For the time being, let us assume that words do not have ambiguous meanings. In the first pass, the function `lexAnalysis` described in Chapter-3 is invoked repeatedly till the entire sentence has been scanned. With the above assumption, no non-determinisms are expected during lexical analysis of a well-formed sentence. Every time a new lexical entry is detected by `lexAnalysis`, a new `Holder` object pointing to the f-structure sub-object of the `LexPrimitive` object returned by `lexAnalysis` is created and appended to the global collection `C`. Also, if any m-structure schemata is projected, it is appended at the appropriate location of `M`.

During syntactic parsing, lexical interaction is carried out by associating tokens with `Holder` objects. Every non-terminal also get associated with holder objects during parsing. The metavariable $\downarrow$ for any grammatical category (phrasal or lexical) in the right hand side of any c-structure rule represents the holder associated with the category. During acceptance of a c-structure rule by the parser (i.e. during a *reduce* operation), first a *new* holder is created in `C` and associated with the non-terminal on the left side of the rule. This holder therefore represents the $\uparrow$ metavariable. The `Equation` objects associated with all the non-terminal are introduced into the system (and immediately "fired") in order of their occurrence in the rule being reduced. If the left side non-terminal is `S` (sentence or clause), the `search` operation is carried out on each `MStructObject` in `M`. If the sentence is well-formed, this operation should be performed without any error and on termination, `Stack` should be empty. Finally, when the entire input sentence (which could as well be a complex one) has been scanned, the contraints in `constraintList` are checked for validity.

4.6 Discussions

To test our formalism, a random sample of about 250 simple sentences and sentences with embedded clauses were selected from newspaper clippings. The NPs were simplified to the bare *head* nouns before feeding them to our parsers. In spite of the fact that the phrasal ordering was quite unpredictable, almost all simple sentences in active voice (constituting about 57% of the samples) were correctly parsed. Bangla has no natural passive voice. Passive-like constructs are expressed in the *intransitive passive* [2] or *neuter* voice called bhabavachya. In Bangla neuter voice sentences, the verb assumes a distinct form. The *agent* (which we assume

[2]This is a misnomer as even transitive verbs may be used in this voice.

to be the SUBject in absence of any accepted linguistic guidance regarding the matter) takes either a DATive or a GENitive case marker. This explains why in Table-1 these case markers have been shown also to mark SUBjects. As high as 30% of our samples were in neuter voice (as is expected in reports) thus justifying the use of delayed syntactic encoding. Most of the neuter voice sentences were also correctly parsed. From the 11% of the samples constituting active-voice sentences with embedded clauses, barring those which used verb forms not already stored in the lexicon (like **ud/y.og nilen** "took initiative"), the remaining were correctly parsed. However, the parser in its present form could not parse neuter voice sentences with embedded clause that constituted the remaining 2% of the samples. At present, we are investigating this form in more detail.

In Sec-4.4, we have assumed that the embedded clause plays the COMPlement functional role. In corresponding examples of English *to*-infinitive sentences, it is usual to visualize the dependent clause as VCOMP or the verb's complement. Since in Bangla there is no concept of a VP, the concept of the verb's complement is ambiguous. There are however Bangla sentences very similar in form to the ones considered in Sec-4.4 (and often confused with them) where the embedded clause is similar to English *for-infinitive clauses*. Of the two types (i.e. to-infinitive and for-infinitive) of clauses, the for-infinitive is slightly more natural. Hence, we call the verbal inflection marking such clauses the *FORINF* declension. An example of a sentence with a for-infinitive clause is given in Example-4.

Example-4:

pradha'nmantri ekha'ne baktrita' dite a'sben
Prime Minister-NULL here-OBL lecture give-FORINF come-3p-hon-FUTURE
The Prime Minister will come here to deliver a lecture

As in English, we propose to handle this by letting principal clause take dependent complements. A peculiar and widely used type of sentence in Bangla use arbitrary length chains of clauses semantically disjoint in time across the clauses. In such sentences, which we choose to call *garden-path* sentences, it is difficult to pin-point the principal clause. An example is given in below.

Example-5:

tumi ghum theke ut'he Ha't mukh dhuye dut'o bis/kut' khete khete ekt'a rik/sa'y chepe st'eSane chale y.eyo
You-NULL sleep from rise-CONTINF hand face wash-CONTINF two-DEF biscuit eat-FORINF eat-FORINF one-DEF rickshaw-OBL ride-FORINF station-OBL walk-CONTINF go-2p-IMPER
You, (after) waking up (thereafter) washing your face and hands (while) munching two biscuits, take a rickshaw to go down to the station.

Sentences like above are not very uncommon in Bangla. Note that only the final verb has a finite declension, the rest having either CONTinual-INFinitive or FORINF declensions. Note also the paired use of the same verb khete khete with FORINF declensions to indicate simultaneity of action, while the verb with CONTINF declension generates a temporal chain of events. However, in spite of a CONTINF declension on the verb chale, the finite verb of the matrix is actually the word pair chale y.eyo *meaning "go down". The finite verb here is a compound verb*, an example of a multi-worded lexical entity in Bangla. An analysis of garden-path sentences is given in Sec-7.3.

Part III

Application Division

Chapter 5

The Verbal Paradigm of Bangla

5.1 Introduction

The verb is the nucleus of our GLFG based approach for syntactic processing of Indian languages. The deep structure of any sentence in any natural language contains a verb which acts as the predicate of the logical form of the sentence. The verbal paradigm of Bangla is quite involved in that, the surface form of even a simple sentence, there may be none, one or more than one verb. Apart from providing the general semantics of the sentence, the verb(s) in Bangla provides the grammatical *glue* with which the other *participating entities* in the sentence are bound together. The *grammaticality* of Bangla sentences emanates from the *semantax* of the verb and a verb-centric parsing strategy is therefore very natural. For verb-centric parsing, an ontological study of Bangla verbs is necessary. In this chapter, the syntax and to certain extent semantics of Bangla verbs will be discussed.

As with any word, a verb or **kriya-pad** (kriya=action, pad=part-of-speech) is formed from morphemic units. The stem of a verb is called a **dhatu**. A verb consists of a **dhatu** followed by an optional *causational* affix called **sadhita pratyaya** and terminated by a verbal *declension* called **kriya bibhakti**. The study of the verbal paradigm therefore involves the study of all the three morphological units constituting a verb.

5.2 The Ontology of Bangla Verbs

5.2.1 Classification of Verbs

Bangla verbs can be classified on the basis of:

Participating Entity: By which, verbs can be marked as taking from one to three *participating entities.* Causated verbs however take one extra participating entity than the corresponding non-causated forms. The number of participating entities taken by a verb is generally known as its *valency.*

Causationality: This generally involves finding out whether there is a causational affix in the verb. A causated verb alters the semantics of the corresponding uncausated verb in a determinable manner.

Finite-ness: A finite verb may end a sentence, and every (simple) sentence must have one implicit (i.e. non-existent in the surface form as in zero-worded verbs) or explicit finite verb. For many Indian languages, especially Bangla, the ontology of infinite verbs is an interesting branch of study. Whether a verb is finite or infinite is determined from the declension of the verb.

Number of words: On this basis, a verb can be:

i. *Implied* Verb. A verb which is missing in the surface form of the sentence and has somehow to be implied by the parser.

ii. *One worded finite* verb.

iii. One worded finite verb preceded or followed by a *negative particle* which constitute the paradigm of *negative* verbs.

iv. *Two worded* verbs, where both the words have a verb stem. This is a cross-discipline with infinite verbs since in most cases the first word is an infinite verb. These verbs can be further studied under:

   (a) *Compound* Verbs

   (b) *Modal* Verbs

   on the basis of the declension on the first word of the two word sequence.

v. Two worded verbs, where the first word has a noun or adjective stem and the second word has a verb stem. Such verbs are called *composite* verbs.

vi. Complex cases of multi-worded verbs, including negative instances of such verbs.

In spite of above classifications, while speaking in the general sense, the word "verb" will be used to describe a one-worded non-negated finite verb. Concepts will be introduced and tackled initially with such verbs but the ideas will be appropriately generalized for verbs consisting of any number of words.

5.2.2 Formal and Colloquial Form

Usage of Bangla is seen in two different forms — a formal form or "Sadhu Bhasha" ("Sadhu" = Pure; "Bhasha" = Language) which is used in old-fashioned literature and a more common colloquial form or "Chalit Bhasha" ("Chalit"=Common). The difference between the two forms is mostly in the verbal inflectional system. The formal version is characterized by a well-defined and more rigid inflectional system. In the colloquial form, many cases of vowel-disharmony have been evened out. As a result, in the colloquial form, a large number of spelling rules govern the conjoining between the stem and the inflection of a verb. As an example of vowel harmonization, consider the verb pariya'=/par/+/iya'/ in the formal form becoming pere in the colloquial form. For the spelling rule governing the formation of the verb, see Appendix-B. Since current usage of the formal version is limited, our attempt has been to include the most common colloquial form as a part of the language. Hence, in many places in this chapter, examples have been considered from both the formal and colloquial forms. Whenever an example (or description) has been cited in the colloquial form, it has been parenthesized. For example, a citation could be pa'riya' (pere) is a verb that ....

5.2.3 The Verbal Inflection System

The verbal inflection system consists of the causational affix and the verbal declension.

The *causational affix* determines whether the verb is causated or not. In Bangla, the only causational affix is a', in both formal and colloquial form. However, in the colloquial form, the affix sometimes get deformed in spelling as shown by some rules in Appendix-B.

There are some verb stems (for example, da'md'.a' 'stand') that may not be morphologically causated. These stems must have *lexical* "flags" to enable the morphological processor to decide that it may not be followed by a causational affix.

The *declension* on a verb determines:

- The Tense and Aspect of the verb if it is finite. Certain declensions mark the verb to be infinite, in which case it is tenseless.
- The Gender, Number, Person and Honorific Value of the verb. These four features are henceforth to be referred to collectively as the GNPH value of the verb, although for a particular language of interest, some of the features may be absent (In Bangla, the gender and number feature are absent). The GNPH feature of the main verb of a sentence *must* agree with the GNPH feature of the grammatical subject. Thus, the GNPH is one of the strongest points for subject determination.

5.2.4 Tense and Aspect

The tense-aspect system of Indian languages is quite complex. Tense and aspect are normally determined from the declension of the verb. However, for a given language, some tensual and aspectual cases are expressed by special multi-worded verb forms.

According to Chatterjee [36], there are four basic tenses in Bangla — **Past**, **Habitual (Past)**, **Present** and **Future**. There are three instances of aspect for every tense namely, **Simple** (i.e. no aspect), **Progressive** and **Perfect**. In addition there is a distinct form for the **Imperative Mood**. Thus, in all, there are *thirteen* different tense/aspect values for the TENSE attribute for a finite verb. The thirteen tenses are shown in Table-5.1

There is no gender and number distinction with Bangla verbs. *Five* different person-honorific classes are encoded in the declension of a verb as shown in Table-5.2.

Of the thirteen possibilities of the TENSE value, the imperative mood, all the simple tenses and the different aspectual cases of the past and present tenses are completely encoded in the verb declension. The different forms are shown in Table-5.3. The last row in every entry is reserved to express the inflected form (in "Sadhu Bhasa") of the verb root **kar** meaning "do".

The remaining four forms require an auxiliary aspectual operator and hence are instances of multi-worded verbs.

There are two major defective verbs. The first one has the stem **y.a'** "go". However, when declened with a morpheme beginning with "E", the stem gets deformed to **gi**.

We take care of this defective verb by a spelling rule. Another defective verb is the aspect-verb. This verb can only be in PRESENT or PAST tense. The different forms are **chila'm, chili, chile, chila** and **chilen** in the PAST tense and **chi, chis, che, cha** and **chen** in the PRESENT tense. We have considered all the ten forms as special VSTEM morphemes that are indeclinable. These morphemes project the necessary TENSE and GNPH features. The indeclinability is assured through manual tuning of the AFSA.

5.2.5 Verb Forms with Infinite Declensions

The verb declension system also encodes three types of *infinite* forms, as shown in Table-5.4.

In our approach, the lexical projection of a verb with infinite declension will not have TENSE and GNPH features. Instead, a feature INFTYPE will be projected with possible values FORINF, CONDINF or CONTINF.

A verb with an infinite declension does not always automatically constitute an infinite verb. In fact, the infinite declensions often act as auxiliary operators for forming different types of multi-worded verbs. Also, there is one infinite verb form that is necessarily multi-worded. The usage of verb forms with infinite declension is described below. Here, V may be any verb stem or verb stem conjoined with a causational affix, and I may be any inflection. The linguistic concepts have been taken from standard text books on Bangla grammar like [36].

ite (te) **or FORINF:** This infinite form has three usages:

- As V+FORINF V_M+I. Here V_M is the stem of a verb indicating modality like *par* (can/may), *ha* (will/shall/must), etc. I determines the tense and GNPH of the two-worded verb which will be called a *modal* verb. Normally, V_M is different from V.
- As V+FORINF V+FORINF. This is the form of a *merge infinite* verb. The usage is as $S_V S'$, where S_V is a sentential form ending in the merge infinite verb and S' is the remaining part of the sentence. S' is either a sentence, or S/CP, i.e. a sentence from which a case phrase is missing (a notation taken from Gazder et al [70]). Semantically, the action described in S_V has a *temporal reference* that *merges* with the temporal reference of S'.
- As V+FORINF in isolation. This is the form for a *for-infinite* verb. Usage is similar to the merge infinite verb. Semantically, S_V is the *reason* for S'.

Another related infinite form, called the *join infinite* form, has been discussed in Sec-5.3.4.

ile (le) **or CONDINF:** The form V+CONDINF is the form of a *conditional infinite* verb. Usage is similar to merge infinite. Semantically, S_V is the precondition for (and hence temporally *meets*) S'.

iya' (E) **or CONTINF:** There are a few different usages:

- The form V+CONDINF is the form of a *continual infinite* verb. Usage is similar to merge infinite. Semantically, S_V temporally *precedes* S'.
- The form V+CONDINF V' is the form for *Compound Verbs* which has been described in Sec-5.3.3.
- There is another use of CONTINF albeit rare. The form V+CONTINF V+CONTINF often acts as an *adverbial infinite* for a following verb V'. Usually, V is a *Process* type verb-stem. Semantically, the above form describes an action V' that has been accompanied by process V.

Verbal Declension System Ambiguity: Comparison between Table-5.3 and Table-5.4 reveals ambiguity in detection of the FORINF and the CONDINF declensions since they are identical in form with the HABIT-2p-1h and PAST-2p-1h finite declensions, respectively. The lexical entries for these declensions therefore have an alternation between the finite and the infinite forms.

5.2.6 Voice

Indian languages exhibit four different forms of sentence expression — the concept having close similarities with the English voice system. The Voice system for Indian languages is described below. The description provided is a deviation from classical grammar texts. It has been adopted from a book by P. Dasgupta [52], where the author has explained the voice system in the light of the X′-syntax.

Active Voice or "Katrivachya' This is the most common and normal form. the actor is given the principal status, or according to the concepts of X'-syntax, the actor governs the verb. The actor thus becomes the grammatical function subject of the sentence.

Passive Voice or "Karmavachya" Here the verb is governed by the acted upon phrase which is given the subjective status.

Neuter, Intransitive Passive or "Bhabvachya" Usually, this form is found with intransitive verbs. No phrase (or an empty category) governs the verb. The actor takes a neutral case marker like DAT or GEN.

Quasi-passive, Middle or "Karmakatrivachya" This form is most common for ergative constructions in which inanimate entities are delegated with actor status. The verb is both governed by and governs the inanimate entity which now plays a dual functional role of subject and object.

For Bangla, active voice is the most common mode of expression and the whole tense system described above, is true for sentences in active voice. Passive voice is rare in Bangla except for *composite* verbs.

Neuter voice is denoted by a two-worded verb of the type V_1-FORINF **Ha**-DEC, where V_1 is the stem from which the basic semantics of the verb is derived, FORINF is the merge infinite verb declension, **Ha** is the stem of the *copula* and DEC is any verb declension. The tense-aspect of the verb is determined by DEC.

Neuter or intransitive-passive voice is denoted by a two-worded verb of the type V_1-CAUS **Ha**-DEC, where V_1 is the stem from which the basic semantics of the verb is derived and CAUS is the verb causational affix.

There are no special verb forms for the quasi-passive voice. The verb takes the form as in active voice. Determination of voice is from semantic considerations.

5.3 Some Common Bangla Verb Forms

The major features in the f-structure of a verb are **TENSE**, **GNPH** and **PRED** (semantics, or logical form). For causated verbs, there are two additional features — CAUS, which takes a YES value and CSUBJ, which carries information about the necessary case-marker on the participating entity in the sentence that plays the functional role of Affected SUBject. In the GLFG based verb-centric parsing system being discussed in the present thesis, syntactic control emanates from the verb, meaning thereby, the m-structure schema should be most prevalent in the lexical entries of verbs. In our approach, the lexical projection of a verb with infinite declension will not have TENSE and GNPH features. Instead, a feature INFTYPE will be projected with possible values FORINF, CONDINF or CONTINF.

Irrespective of whether a verb is finite one-worded or a more complicated form, it is a single lexical entity. In this section, we shall mainly concern ourselves with the

lexical projection of Bangla verbs including the associated m-structure schema. For multi-worded verb forms, supra-lexical specifications are obviously required. Also, since the causational affix introduces a non-trivial modification in the semantics and m-structure of a verb, supra-lexical rules are required for single worded forms too. The GLFG based analysis of Bangla verbs must therefore involve:

- Study of Supra-Lexical rules for the different verb forms.
- M-structure and F-structure associated with various verbs.

It is obviously not possible to discuss the f- and m-structures of each individual verb in the present thesis. We provide a general overview of how the different entities of the f- and m-structure of a verb (of any form) are generated, either straightway from the lexical database entries of the constituent morphemes (for single worded verbs), or through manipulations at the supra-lexical level.

5.3.1 One Worded Finite Verb

The one worded verbs discussed here may or may not be causated, and are in active voice.

Uncausated finite verbs derives the PRED feature and the m-structure directly from the lexical database of the stem. The TENSE and GNPH features are derived from the declension. For causated verbs, the PRED feature and the m-structure must be altered to include one more participating entity. However, there are many verb stems which may not be morphologically causated. To prevent such stems being followed by a causational affix, their lexical entries contain an additional f-structure schemata $(\uparrow\ CAUS) = NO$. This way, the morphological processor fails to unify the $(\uparrow\ CAUS)$ entities from the stem and a causational affix (if such an affix is present). As an example, the lexical form da'md'.a' (stem 'stand') + a' (affix) + **iben** (declension) is ill-formed, i.e., there is no single-worded verb in Bangla meaning 'cause to stand'. This mechanism permits us to have a trivial supra-lexical rule for one-worded causated or non-causated verbs.

```
VERB => VSTEM+(VCAUS)+VDEC where {
  TENSE<<($1 TENSE);
  GNPH<<($1 GNPH);
  exists($1.VCAUS) ? {
    #=$1.M + (# ASUBJ {(ANIM,YES),(CASE,($1 CSUBJ))});
```

```
  } : {
    #=$1.M;
  }
} {
  exists($1.VCAUS) ?
    $1.~?; $1.~0; @$1; :
    $1.~?+1; strcat("CAUS-", $1.~0); SUBJ; ASUBJ; @$1;
}
```

Thus, if a verb is non-causated, its m-structure is identical to the m-structure of the stem. However, if it is causated, the m-structure gets modified as follows. The affected subject of the sentence is forced to be animate and the case marker on the 'head' word of the Affected SUBJect (ASUBJ) grammatical function is constrained to be same as the CSUBJ feature of the stem. In either cases, the TENSE and GNPH of the verb remains same as returned by the morphological sub-system. If the verb is causated, the PRED feature is augmented to include the semantics of causation. In that case, an additional argument for the secondary agent is added at the second place of the semantic clause, the Affected Subject is made to occupy the new position and the predicate is expressed by adding the string prefix "CAUS-" to the predicate of the uncausated stem.

Example:
Consider the causated verb **dekha'be** (see-CAUS-FUTURE-3p-1h) "will cause to see, i.e. show". The uncausated stem **dekh** "see" projects the following features (only the necessary ones mentioned):

(↑ ASUBJ)=DAT
(# SUBJ CASE)=NOM, (# SUBJ ANIM)=YES, (# SUBJ GNPH)=3p-1h
(# OBJ CASE)=NOM
(# IOBJ CASE)=DAT, (# IOBJ ANIM)=YES

As a result of the above supra-lexical specification, the projection of **dekha'be** includes two additional m-structure schemata — (# ASUBJ CASE)=DAT, (# ASUBJ ANIM)=YES. Also, the semantic clause is — "CAUS-see<(SUBJ), (SUBJ), (OBJ), (IOBJ)>, which clearly has one additional argument than the semantic clause of the uncausated form.

5.3.2 Infinite Verbs

An Infinite verb is one which can act as the "head" of a dependent clause of a complex sentence or non-final clauses in a garden-path a sentence. Bangla is characterized by a lavish usage of infinite verbs. Complex sentences have been dealt with more detail in Sec-7.4. The infinite "head" verb of the dependent clause of a complex sentence is treated as a normal verb with additional constraints that it should be tense-less and project a (INFTYPE FORINF) feature.

The infinite verbs used in garden-path sentences (see Sec-7.3) are of three types, two of which are one-worded. Each form must project a FLOW feature indicating the temporal relation between the clause headed by the verb in question, and the next clause of the sentence. The flow feature can have three possible values. If C_1 and C_2 are two successive clauses in a garden-path sentence and if C_1 is headed by an infinite verb V (which may be multi-worded), the semantics of the three values of the FLOW feature projected by V, is explained below:

i. If FLOW has a value BEFORE, C_1 temporally *precedes* C_2.

ii. If FLOW has a value SIMULTANEOUS, C_1 and C_2 describe *simultaneous* events.

iii. If FLOW has a value IF-THEN, C_1 not only temporally precedes C_2, but also a pre-condition for event C_2 at all taking place.

iv. A FLOW value JOIN indicates C_1 temporally *meets* C_2.

For a more detailed discussion on garden-path sentences and the use of the FLOW feature, including examples, see Sec-7.3.

Infinite verbs projecting BEFORE value for FLOW is one-worded (unless negated, or the base part generated out of multi-worded forms) that takes the CONTINF declension. An IF-THEN value of the FLOW feature is also projected by a single-worded verb, the declension being CONDINF. The supra-lexical specification for infinite verbs projecting FLOW values BEFORE and IF-THEN is:

```
VERB => VSTEM+(VCAUS)+VDEC where {
  equals(($1 INFTYPE), CONTINF) ? {
    FLOW<<BEFORE ;
  } :
  equals(($1 INFTYPE), CONDINF) ? {
```

```
  FLOW<<IF-THEN ;
};
exists($1.VCAUS) ? {
  # = ... same as finite one-worded verb ...
} : {
  # = $1.M;
  }
} {
  ... same as finite one-worded verb ...
}
```

Example: The verb **dekhe** see-CONTINF "upon seeing" has a FLOW value BEFORE, while **dekhle** see-CONDINF "if seen" has a FLOW value IF-THEN.

A FLOW value SIMULTANEOUS is projected by the form V-FORINF V-FORINF, i.e. a doubly uttered CONDINF declened verb. The supra-lexical specification in this case is:

```
VERB => VSTEM+(VCAUS)+VDEC VSTEM+(VCAUS)+VDEC where {
  equals(($1 INFTYPE), FORINF) &&
    equals($1.VSTEM, $2.VSTEM) &&
    equals($1.VCAUS, $2.VCAUS) &&
    equals($1.VDEC,  $2.VDEC) {
      FLOW<<SIMULTANEOUS;
  }
  exists($1.VCAUS) ? {
    # = ... same as finite one-worded verb ...
  } : {
    #=$1.M;
  } {
  ... same as finite one-worded verb ...
  }
}
```

Example: The two-worded verb **dekhte dekhte** see-FORINF see-FORINF "while seeing" project a SIMULTANEOUS value for the FLOW attribute.

The Verb form projecting a FLOW value JOIN has been discussed in Sec-5.3.4.

5.3.3 Multi-Worded Verb Forms

There are a few types of multi-worded (finite, non-negated) verb forms in Bangla:

V+**FORINF** V_M: This is the form for modal verbs talked about in Sec-5.2.5. Verbs in progressive-future tense and NeuterPassive Voice also have this form.

V+**CONTINF** V': This is the form for compound verbs.

X V: This is the form for composite verbs where X is either a noun or an adjective.

The supra-lexical rules for the various verb forms are considered below:

Compound Verbs:

```
VERB => VSTEM+(VCAUS)+CONTINF VSTEM+VDEC where {
  compoundable($1.VSTEM,$2.VSTEM);
  TENSE<<($2 TENSE);
  GNPH<<($2 GNPH);
  exists($1.VCAUS) ? {
    # = ... same as finite one-worded verb ...
  } : {
    # = $1.M;
  }
} {
  exists($1.VCAUS) ?
    $1.~?+1; strcat("CAUS-", $1.~0, vector($2.VSTEM)); SUBJ; ASUBJ; @$1; :
    $1.~?; strcat($1.~0, vector($2.VSTEM)); @$1;
}
```

In the above rule, *compoundable* is a user supplied filter. The *compoundable* filter is a mapping from (Verbal Stem × Verbal Stem) onto {TRUE, FALSE}. The naive method of construction of this filter is to have a compiled list of all pairs that map to TRUE (i.e., all compoundable stems). However, attempts are being made to *derive* the compoundable filter using different lexical attributes of the compounding stems. A review of the status of such research may be found in the work of a colleague, Bandyopadhyaya [9].

In case the stems are compoundable, the TENSE and GNPH features of the compound verb are derived from the second verb-like word called the *vector*. The semantic predicate of the compound verb is primarily derived from first word called

the *pole*. However, it is subtly altered by the *vector meaning* of the vector. The alteration is assumed to be captured by the function `vector`.

Example:
In the compound verb **dekhe phelbe** see-CONTINF drop-FUTURE-3p-1h "will observe", the *vector meaning* of the stem phel "drop" (vector) provides a sense of completion of the main action ("see" here). Suppose that `vector(phel)` returns `COMPLETE`. Then, the semantic predicate of the above compound verb is `seeCOMPLETE`.

Some Future an Habitual Tense Forms

For future tense verbs, the supra lexical rule is:

```
VERB => VSTEM+(VCAUS)+VDEC VSTEM+VDEC where {
  equals($2.VSTEM, FUTUREMODAL);
  equals( ($1 TENSE), CONTINF) || equals( ($1 TENSE), FORINF);
  equals( ($2 TENSE), HABIT) || equals( ($2 TENSE), FUTURE);
  equals( ($2 TENSE), HABIT) ? {
    equals( ($1 TENSE), CONTINF) ? {
      TENSE << HABIT-PERF;
    } : {
      TENSE << HABIT-PROG;
    }
  } : {
    equals( ($1 TENSE), CONTINF) ? {
    TENSE << FUTURE-PERF;
    } : {
      TENSE << FUTURE-PROG;
    }
  }
  GNPH<<($2 GNPH);
  exists($1.VCAUS) ? {
    # = ... same as finite one-worded verb ...
  } : {
      # = $1.M;
    }
  } {
  }
```

In the above rule, FUTUREMODAL is a special auxiliary verb stem **tha'k** (stay). The two-worded verb is in future and habitual tense depending upon whether the

future modal auxiliary is in FUTURE or HABITual tense, respectively. The aspect of the verb is progressive or perfect depending upon whether the declension on the main verb is FORINF or CONTINF, respectively.

Example:

i. **dekhte tha'kba** see-FORINF stay-FUTURE-1p "shall go on seeing" is in FUTURE-PROGressive tense.

ii. **dekhe tha'kba** see-CONTINF stay-FUTURE-1p "should have seen" is in FUTURE-PERFect tense.

iii. **dekhte tha'kta'm** see-FORINF stay-HABIT-1p "used to go on seeing" is in HABITual-PROGressive tense.

iv. **dekhe tha'kta'm** see-CONTINF stay-HABIT-1p "used to have seen" is in HABITual-PERFect tense.

Verbs in Neuter Voice

For Neuter Voice verbs, the supra lexical rule is:

```
VERB => VSTEM+(VCAUS)+VDEC  VSTEM+VDEC where {
  equals( ($1 TENSE), FORINF);
  equals( ($2 GNPH), 3p-1h);
  equals( ($2.VSTEM, COPULA) );
  TENSE=($2 TENSE);
  GNPH=($2 GNPH);
  # = ($1 #) - (# SUBJ CASE) + (# SUBJ CASE)=DAT;
}{}
```

In the above rule, COPULA is a special verb stem **Ha** (be). Note the change in the m-structure of the verb from the m-structure projected by the stem of the main verb. Note that the GNPH of the auxiliary is constrained to be a neutral one like 3p-1h.

Example:
dekhte Hala see-FORINF be-PAST-3p-1h "had to see" is a verb in neuter voice as in the sentence **a'ma'ke ta'jmaHal dekhte Hala** *I-DAT Tajmahal-NULL see-FORINF be-PAST-3p-1h* "I had to see the Tajmahal", where the subject **a'ma'ke** takes the DATive case marker.

Modal Verbs

For all modal verbs, most of the clauses in the supra-lexical rule are common. Only the semantic clause gets modified according to the modal operator. The general form of the rule is:

```
VERB => VSTEM+(VCAUS)+VDEC  VSTEM+VDEC where {
  equals( ($1 TENSE), FORINF);
  equalsFrom( ($2.VSTEM, MODALSET) );
  TENSE=($2 TENSE);
  GNPH=($2 GNPH);
  # = ($1 #);
} {
  exists($1.VCAUS) ?
    $1.~?+1; strcat( modal($2.VSTEM), "CAUS-", $1.~0); SUBJ; ASUBJ; @$1; :
    $1.~?; strcat(modal( $2.VSTEM), $1.~0); @$1;
}
```

Here, the set MODALSET is the set of modal auxiliaries. The function modal returns the modal semantics of the modal auxiliary stems. The predicate of the modal verb is obtained by concatenating the modal semantics of the auxiliary with the predicate for the main verb. For four most common modal auxiliaries, the modal semantics are given below, along with example sentences from the prototype **a'mi ta'jmaHal dekhte** X-**la'm** *I-NULL Tajmahal-NULL see-FORINF X-PAST-Ip*, with X being the modal operator.

pa'r Can/May-). **a'mi ta'jmaHal dekhte pa'rla'm** "I could see the Tajmahal". The predicate is *Can/May-see*.

ca'h (WantTo-). **a'mi ta'jmaHal dekhte ca'ila'm** "I wanted to see the Tajmahal". The predicate is *WantTo-see*.

la'g (GoOn-). **a'mi ta'jmaHal dekhte la'gla'm** "I went on looking at the Tajmahal". The predicate is *GoOn-see*.

bas (TakeUp-). **a'mi ta'jmaHal dekhte basla'm** "I took up seeing the Tajmahal". The predicate is *TakeUp-see*.

Composite Verbs

The most difficult of verb forms in Indian languages, especially Bangla, is the composite verb. Composite verbs of Bangla represents the dynamicity of language.

Most new actions are described in Bangla by using composite verbs. As described earlier, a composite verb consists of a non-verbal (and meaningful) entity followed by a verbal auxiliary unit. The non-verbal entity can be a noun (N), an adjective (A) or sometimes a post-positional particle (P). Most verbal-noun units (morphemes formed from a verbal stem followed by a causational affix) may be used as the first entity of a compound verb. Linguistic study of Bangla composite verbs is at an early stage. Even, a universally accepted set of linguistic tests for identifying composite verbs is not available.

The form N-V of composite verbs is most common. The six most common auxiliary V stems of composite verbs are kar, ha, de, pa' kha', pa'. Of them, statistics shows that kar ("do") accounts for the auxiliary of more than half of the composite verbs used normally. Now, kar is a transitive verb stem. Quite often, the composite verb N-kar is an intransitive one. In such cases, it is tempting (and often confusing) to consider kar as the main verb with N as its object (complement). For example, consider the composite verb (stem) **cha'n kar** ("bath-do"). In the sentence **ra'm cha'n karche** ("Ram is taking a bath"), cha'n (bath) may be considered to be the object of **kar**. However, the approach is clearly not proper because first, it leads to a high degree of polysemy for kar and secondly, it leads to an awkward situation for sentences like **ra'm a'ma'ke bait'a' da'n kareche** ("Ram has gifted me the book). Here, if da'n ("gift") is considered as the object of kar, the actual object bait'a' ("the book") can not be assigned a function. Transformational approaches towards treating the N of N_V composite verb as the complement of V (even in the presence of other objects) may be found in P. Dasgupta [48]. However, criticisms thereof, and a more balanced outlook towards composite verbs of Bangla may be found in M. Dasgupta [44]. The latter approach suggests a greater bias towards treating composite verbs as separate lexical entity.

Treatment of composite verbs as separate lexical entities is not trivial in the general case. However, for the more widely used forms, this approach appears more realistic. In the present work, we employ the lexical approach by considering composite verbs at the supra-lexical level. The resulting supra-lexical specification is given below. We consider only the N-V composite verb, leaving the others for later research.

```
VERB => NSTEM VSTEM+VDEC where {
  composable( $1.NSTEM, $2.VSTEM);
  TENSE=($2 TENSE);
  GNPH=($2 GNPH);
  # = CompositeMStruct($1.NSTEM, $2.VSTEM) +
      (\# SUBJ GNPH)=($2 GNPH);
} {
```

```
    compositeArgumentNo($1.NSTEM, $2.VSTEM);
    compositePred($1.NSTEM, $2.VSTEM);
    compositeArgumentList($1.NSTEM, $2.VSTEM);
}
```

Note the functions `CompositeMStruct`, `compositeArgumentNo`, `compsitePred` and `compositeArgumentList`. They generate the m-structure, number of semantic clause arguments, semantic predicate and the semantic clause argument list, respectively. Together, these functions perform what may be termed as the *verbalization* of the nominal stem.

Example:
Let us consider the nominal stem **da'n** that is composable with verb **kar**. The results of the above functions are summarized below:

i. `CompositeMStruct`(da'n, kar) returns the list { (# SUBJ ANIM)=+, (# SUBJ CASE)=NOM, (# IOBJ ANIM)=+, (# IOBJ CASE)=DAT, (# OBJ ANIM)=-, (# OBJ CASE)=NOM }

ii. `compositeArgumentNo`(da'n, kar) returns 3.

iii. `compsitePred`(da'n, kar) returns "gift".

iv. `compositeArgumentNo`(da'n, kar) returns the list { (SUBJ), (OBJ), (IOBJ) }.

Thus the predicate of the composite verb da'n karlen gift do-PAST-2/3p-2h "gifted" has a semantic clause $'gift < (SUBJ),(OBJ),(IOBJ) >'$ and an m-structure { (# SUBJ GNPH)=2/3p-2h, (# SUBJ ANIM)=+, (# SUBJ CASE)=NOM, (# IOBJ ANIM)=+, (# IOBJ CASE)=DAT, (# OBJ ANIM)=-, (# OBJ CASE)=NOM }

Although composite verbs can be causated, we have not taken up that case in detail.

5.3.4 Negative Verbs

There are two types of finite verb negation in Bangla and one type of negation of infinite verbs. Every negation is carried out with the help of one of the two negative *particles* — **na** and **ni**. One interesting use of a negative particle along

with a type of infinite verb, that does *not* actually negate the sense of the verb, has also been discussed here.

A good study of negation in Bangla in general may be found in Singh [163], where negation of verbs have naturally enjoyed maximum predominance. It may be observed in Bangla that the negative particle in almost all negative forms maintain near proximity with the verb it negates. Also, in some of the forms, the presence of the particle alter the lexical projection of the verb and sometimes impose syntactic restrictions. It is therefore natural to use supra-lexical specifications for negative verb forms. Here we concentrate on some of the more commonly used forms:

Finite Verb followed by "na". This is the most common form of negation. The negative particle introduces a simple negation on the semantics of the preceding verb. No selectional restrictions are imposed. Other features of the verb remains unaltered. Thus the situation is somewhat as expressed by the English particle 'not' used as a singleton (i.e. with no auxiliary). We shall call this type of negation an *indefinite negation*. The necessary supra-lexical specification for this form is:

```
VERB => VSTEM+(VCAUS)+VDEC NEG where {
  equals( ($2 NEGTYPE), INDEFNEG );
  TENSE<<($1 TENSE);
  GNPH<<($1 GNPH);
  exists($1.VCAUS) ? {
    #=$1.M + (# ASUBJ {(ANIM,YES),(CASE,($1 CSUBJ))});
  } : {
    #=$1.M;
  }
} {
  exists($1.VCAUS) ?
    $1.~?; strcat("~", $1.~0); @$1; :
    $1.~?+1; strcat("~CAUS-", $1.~0); SUBJ; ASUBJ; @$1;
}
```

Here, ~ is taken to be the negation operator. The particle na' is assumed to project a NEGTYPE feature with value INDEFNEG.

Example:
The verb **dekhla'm na'** see-PAST-1p no-INDEFinite "did not see" has a semantic predicate ~see, and TENSE as PAST and GNPH as 1p.

Finite Verb followed by "ni". Here the situation is somewhat like in the English 'have not' negation operator (there is no simple equivalence of the 'had

not' operator in Bangla). Here, the preceding verb must necessarily be in PRESENT tense. Apart from negating the sense of the verb, the effect of the operator is to alter the tense of the verb to PRESENT-PERFect. We shall call this type of negation an *perfect negation*. The particle na' is assumed to project a NEGTYPE feature with value INDEFNEG. The necessary supra-lexical specification is:

```
VERB => VSTEM+(VCAUS)+VDEC NEG where {
  equals( ($2 NEGTYPE), PERFNEG );
  equals( ($1 TENSE), PRESENT);
  TENSE<<PRESENT-PERF;
  GNPH<<($1 GNPH);
  exists($1.VCAUS) ? {
    #=$1.M + (# ASUBJ {(ANIM,YES),(CASE,($1 CSUBJ))});
  } : {
    #=$1.M;
  }
} {
  exists($1.VCAUS) ?
    $1.~?; strcat("~", $1.~0); @$1; :
    $1.~?+1; strcat("~CAUS-", $1.~0); SUBJ; ASUBJ; @$1;
}
```

Example:
The verb **dekhi ni'** see-PRESENT-1p no-PERFeCT "have not seen" has a semantic predicate **~see**, and TENSE as PRESENT-PERF and GNPH as 1p.

"na'" followed by any of the infinite forms: This is the usual form for negation of infinite verbs. Treatment is same as in *indefinite* negation of finite verbs, except that the negative particle precedes the infinite verb and the verb being negated is one type of infinite verbs discussed in Sec-5.3.2. As a representative example, we give the necessary supra-lexical specification for the infinite verb that are one-worded in positive usage:

```
VERB => NEG VSTEM+(VCAUS)+VDEC where {
equals( ($1 NEGTYPE), INDEFNEG );
  equals(($2 INFTYPE), CONTINF) ? {
    FLOW<<BEFORE ;
  } :
  equals(($2 INFTYPE), CONDINF) ? {
    FLOW<<IF-THEN ;
```

```
  };
exists($1.VCAUS) ? {
    #=$1.M + (# ASUBJ {(ANIM,YES),(CASE,($1 CSUBJ))});
  } : {
    #=$1.M;
  }
} {
  exists($1.VCAUS) ?
    $1.~?; strcat("~", $1.~0); @$1; :
    $1.~?+1; strcat("~CAUS-", $1.~0); SUBJ; ASUBJ; @$1;
}
```

Example:
The verb na' dekhe no-INDEF see-CONTINF "not having seen" projects a BEFORE value for the FLOW attribute, while the verb na' dekhle no-INDEF see-CONDINF "if not seen" projects an IF-THEN value.

A "na" between two identical infinite stems: Here both the stems have FORINF declension. Recall that in the absence of the negation operator, this is the form projecting a SIMULTANEOUS value for the FLOW feature. The interesting point to be noted here is that the form being discussed *signify no negation at all.* In fact, this is one way of expressing temporal *join* of two events and a JOIN value is projected for the FLOW feature. The supra-lexical specification for this form is:

```
VERB => VSTEM+(VCAUS)+VDEC NEG VSTEM+(VCAUS)+VDEC where {
  equals( ($2 NEGTYPE), INDEFNEG );
  equals(($1 INFTYPE), FORINF) &&
    equals($1.VSTEM, $3.VSTEM) &&
    equals($1.VCAUS, $3.VCAUS) &&
    equals($1.VDEC,  $3.VDEC) {
      FLOW<<JOIN;
  }
  exists($1.VCAUS) ? {
    #=$1.M - SUBJ + (# SUBJ {(ANIM,YES),(CASE,($1 CSUBJ))});
  } : {
    #=$1.M;
  }
} {
  ... Same as positive forms, i.e. no ~ operator used ...
}
```

Example: dekhte na' dekhte see-FORINF no-INDEF see-FORINF "

There are some other forms where the negation operators locally group with verbs. However, they have not been considered here. Treatment of negated forms of multi-worded verbs, may be carried out in a line similar to the negated join infinitive verb discussed above.

5.4 Discussions

In the present chapter we looked at the verbal paradigm of Bangla from the angle of implementation of a computer based parser. As our implementation approach is verb-centric, the more properties of Bangla verbs we can encompass in our formalism, the more general will be the resultant parser. We started with the observation that the common Bangla verb is one-worded and may be quite efficiently handled by our proposed morphological formalism. However, we also noted that there are some verb forms that are used quite frequently, but are not one-worded. Also, negated verb forms are always multi-worded. Contemporary linguistic research on Bangla verbs is involved with in-depth study of syntactic and semantic properties of the various multi-worded forms. For example, other workers of our group have studied the "compoundability" condition for Bangla verbs. The present chapter is however more concerned in bringing all the forms under a single umbrella of supra-lexical specifications. We have been successful in incorporating the more common forms under our formalism as may be observed from the preceding discussions. The task is however far from over. There are many more details to be worked out and many questions to answer. Some verb forms not discussed in this chapter but study of whose properties have been earmarked for future study are:

i. Infinite Compound Verbs and Composite Verbs.

ii. Verbal Adjectives.

iii. Negated Compound Verbs and Negated Composite Verbs.

iv. Negated Infinite Compound/Composite Verbs.

v. Foreign language Verbs and their assimilation into Bangla.

vi. Different type of ambiguities in the verbal paradigm.

Also earmarked for deeper study is handling all types of composite verbs. At present only some common forms have been taken up.

Table 5.1: The Thirteen different Tenses-Aspects of Bangla

	Different values of the TENSE attribute	Abbreviation used in Text
1	Past Simple	PAST
2	Habitual Simple	HABIT
3	Present Simple	PRESENT
4	Future Simple	FUTURE
5	Past Progressive	PAST-PROG
6	Habitual Progressive	HABIT-PROG
7	Present Progressive	PRESENT-PROG
8	Future Progressive	FUTURE-PROG
9	Past Perfect	PAST-PERF
10	Habitual Perfect	HABIT-PERF
11	Present Perfect	PRESENT-PERF
12	Future Perfect	FUTURE-PERF
13	Imperative Mood	IMPER

Table 5.2: The Five GNPH attributes of Bangla

	Different values of the GNPH attribute	Abbreviation used in Text
1	First Person	1p
2	Second Person Intimate	2p-0h
3	Second Person Casual	2p-1h
4	Third Person Casual	3p-1h
5	Second/Third Person Honoured	2/3p-2h

Table 5.3: The Verb-Declension System of Bangla

GNPH ⟶ / TENSE ↓	1p	2p-0h	2p-1h	3p-1h	2/3p-2h
PAST	ila'm (la'm) karila'm	ili (li) karili	ile (le) karile	ila (la) karila	ilen (len) karilen
HABIT	ita'm (ta'm) karita'm	iti (ti) kariti	ite (te) karite	ita (ta) karita	iten (ten) kariten
PRESENT	i (i) kari	is (is) karis	a (a) kara	e (e) kare	en (en) karen
FUTURE	iba (ba) kariba	ibi (bi) karibi	ibe (be) karibe	ibe (be) karibe	iben (ben) kariben
PAST-PROG	itechila'm (chila'm) karitechila'm	itechili (chili) karitechili	itechile (chile) karitechile	itechila (chila) karitechila	itechilen (chilen) karitechilen
PRESENT-PROG	itechi (chi) karitechi	itechis (chis) karitechis	iteche (che) kariteche	itecha (cha) karitecha	itechen (chen) karitechen
PAST-PERF	iya'chila'm (echila'm) kariya'chila'm	iya'chili (echili) kariya'chili	iya'chile (echile) kariya'chile	iya'chila (echila) kariya'chila	iya'chilen (echilen) kariya'chilen
PRESENT-PERF	iya'chi (echi) kariya'chi	iya'chis (echis) kariya'chis	iya'cha (eche) kariya'che	iya'che (echa) kariya'cha	iya'chen (echen) kariya'chen
IMPER	-	NULL NULL kar	aha (a) kara	uk (uk) karuk	un (un) karun

Table 5.4: The Infinite Verb-Declensions

Infinite Form	Abbreviation Used	Declension "Sadhu"	Declension "Chalit"	kar Inflected
Merge/Reason Infinitive	FORINF	ite	(te)	karite
Conditional Infinitive	CONDINF	ile	(le)	karile
Continual Infinitive	CONTINF	iya'	(E)	kariya'

Note: The "Chalit" form of the CONTINF inflection is actually e at the surface level. However, to distinguish it from the *PRESENT-3p-1h inflection* (the distinction is especially required since the two inflections are subject to different types of spelling rules), the *lexical* form E has been used. The symbol E belongs to the lexical alphabet only. Spelling rules are used to obtain surface forms of this inflection.

Chapter 6

The Bangla Case Phrase

It has already been discussed that sentences in Indian languages mostly have a "flat" structure, with the verb as the key entity. Depending upon the verb, there may be other *participating* phrasal entities in the sentence which normally permute freely among themselves. In Bangla, the participating entities are generally of two types — the noun phrase (NP) and the post-positional phrase (PP). The main distinction between the two is morphological. Noun phrases are *case marked* by inflectional case markers. Post-positional phrases on the other hand are case marked by certain types of particles called post-positions. A case phrase therefore have two essential associated features — the semantics or PREDicate feature and a CASE-marking feature. The semantics is derived from the "head" noun or pronoun of an NP (if CP is an NP) or the head of the "object of post-position" (POBJ) NP (if CP is a PP).

6.1 The Noun Phrase

A *complete* linguistic study of the Bangla noun phrase is beyond the capability of the present work. The study of noun phrases is essentially a study of the various modifiers/qualifiers of the head noun and various selectional restrictions imposed by these modifiers. Frey [67] has pointed out that the amount of available literature on noun phrases is surprisingly small. For a language like Bangla, the situation is even worse. The present study is a modest attempt to fill the gap, at least from the viewpoint of computer implementation. The discussions in this chapter are greatly influenced by Jackendoff [84] and Frey [67]. The original analysis offered by Jackendoff for simple NPs in English, which was predominantly based on a transformational approach, is given in (6.1), (6.2. Frey has on the other hand,

suggested a LFG based analysis in [67]. He has pointed that Jackendoff's analysis have certain inadequacies vis-a-vis imposing some common selectional restrictions. Frey introduced some new phrasal types to take care of the selectional restrictions. In the present work, we have adopted Frey's approach in principle. However, Bangla NPs are quite different from English NPs and require different techniques to tackle the intricate details.

$$(6.1)\quad N'''\ (=NP) \rightarrow \left\{ \begin{array}{c} DEM \\ POSS \\ Q1 \end{array} \right\}\ N''$$

$$(6.2)\quad N'' \rightarrow \left\{ \begin{array}{c} DEM \\ POSS \\ Q1 \end{array} \right\}\ AP^*\ N'$$

where AP is the adjective phrase and the other categories are as follows:

DEM: Demonstrative pronouns, interrogative pronouns, 'the', 'a', 'some' (singular), etc.

POSS: Possessives.

*Q*1: 'Each', 'all', 'no', 'every', etc.

*Q*2: 'Many', 'few', 'several', 'much', 'little', etc.

NUM: Numerals, 'a few', 'a little', 'a dozen', etc.

The paradigm of Bangla NPs is a vast one. Here we investigate some of the more common forms. The head of a Bangla NP is a noun or pronoun (N). An N consists of a nominal stem (NSTEM), an optional definite-marking affix (DEF) and a nominal (NDEC) or possessive declension (POSS). The declension is mandatory in the deep-structure of an N. However, in the surface, it may be manifested as a NULL declension. Constituent morphemes of a noun are extracted during parsing by the lexical sub-system. The lexical sub-system also takes care of some of the selectional restrictions vis-a-vis inflections and the stem. We have incorporated one such restrictions in our system — "personal nouns and pronouns may not take the DEF affix". With this, words like a'mit'a', ra'mt'a', etc, are declared ill-formed at the morphological parsing level.

6.1.1 The Nominal Inflections

There are three classes of nominal inflections in Bangla — the DEF affix, the POSS declension and NDEC declension.

The DEF affix acts like a determiner like "the" in English. The affixes t'a', t'i specify singular number for the noun. Affixes kha'na', kha'ni are also used in place of t'a', t'i. The affixes gulo, guli are used to specify plural numbers. Thus bait'a', baiikha'ni, etc., (bai=book) means "the book", whereas baigulo means "the books". The DEF affix is optional. In its absence, the noun is said to be *indefinite*.

The POSS declension in Bangla is a single lexical form er. However, due to spelling deformation resulting out of vowel harmony, it may also become a'r, r in the surface. The usage of the POSS declension is roughly similar to the apostrophe-s of English. Thus pa'khit'a'r (bird-DEF-POSS) means "the bird's". A noun with a POSS declension will be said to have a POSS case marker.

The nominal declensions normally act as case markers for nouns and pronouns. Compared to Sanskrit, the number of nominal declensions (NDEC) is quite small in Bangla. They may be broadly classified in four groups depending upon the type of case marking they perform:

i. The NULL declension, which has no surface form. Since this declension is normally used on the subject of the sentence, it will be called the NOMinative case marker.

ii. The declensions ke and re (which may deform to ere) form the class of DATive case marker.

iii. The declensions e,te (which may deform to y,ete respectively) are the LOCative case markers.

iv. The declensions e, te are also called OBLiQue case markers. To avoid confusion, these declensions will be called OBLQ if conjoined with noun stems with no PLACE feature and LOC with noun stems with PLACE feature. For example, ha't (hand), being a body part, has a PLACE-YES feature. Thus Ha'te means "on the hand" but gorute (goru=cow) means "the cow". The OBLQ case normally points to a SUBJ grammatical function except in some poetic forms.

6.1.2 The Different Modifiers of the Head Noun

The Modifiers of the head of an NP could be of the following types:

i. Inflectional, as a DEF affix or a declension on the head N. The nominal inflections have been considered above.

ii. Numeric, involving *number* with associated inflectional paradigm.

iii. Adjectival — qualitative or gerundative modifiers.

iv. Demonstrative and Interrogative modifiers.

v. Quantifiers

vi. Genitive or Possessive modifiers.

There are order restrictions on the different types of modifiers. An NP may take none or some of the above types of modifiers. However, a demonstrative and a quantifier may not occur together and neither can a quantifier and a numeral. There may at most one of numeric, demonstrative and quantifier modifiers. Each modifier (except inflections) are complex linguistic objects in the sense that they have their own associated inflectional and modifier paradigms and should be treated as phrases.

The numeric and adjective modifiers have the highest syntactic proximity with the head. The adjectival modifier is a chain of simple adjectives of quality (like **la'l** "red", **chot'a** "younger" or "smaller"). Numbers may occur both to the left (normal attachment in a "head last" language like Bangla) and right of the head N. A pre-numerically-modified NP is indefinite while a post-numerically-modified NP is definite. Thus, **tint'e chele** (three-def boy-NULL) "three boys" is different from **chele tint'e** (boy-NULL three-def) "*the* three boys". For a post-numerically-modified NP, all non-NULL inflections that normally occur with the head N, *hops* over to the numeric modifier. Thus, "to the three boys" would be **chele tint'eke** and not ***cheleke tint'e**. Finally, all adjectives must be between the number and N in a pre-numerically modified NP. With this, we obtain the following analysis for N′, a numerically modifies N as given in (6.3), (6.4) and (6.5). The schemata $(\downarrow\ COUNT) = +$ ensures that the head N is countable in (6.3) and (6.4).

(6.3) $N' \longrightarrow \underset{}{NUM} \quad \underset{\downarrow \in (\uparrow\ ADJ)}{A^*} \quad \underset{\substack{\uparrow=\downarrow \\ (\downarrow\ COUNT) = +}}{N}$

(6.4) $N' \longrightarrow$ A^* ($\downarrow \in (\uparrow ADJ)$) N ($\uparrow = \downarrow$) NUM ($(\downarrow DEF)$ $(\downarrow COUNT) = +$)

(6.5) $N' \longrightarrow$ A^* ($\downarrow \in (\uparrow ADJ)$) N ($\uparrow = \downarrow$)

The lexical entries of every number project a NUMBER feature. However, the lexeme **kayek** "some" (plural) projects NUMBER=?, i.e. an unknown number. Moreover, the belief modifiers **besh** "quite", **sa'ma'nya** "only" may co-occur to the left of **kayek**. It is economical to take care of this co-occurrence by a supra-lexical rule.

The rule for N'', i.e. a N' with a demonstrative (DEMO) modifier or a quantifying (QUANT) modifier, is given in (6.6), (6.7).

(6.6) $N'' \longrightarrow$ $DEMO$ N' ($\uparrow = \downarrow$)

(6.7) $N'' \longrightarrow$ $QUANT$ N' ($\uparrow = \downarrow$) ($\sim (\downarrow DEF)$)

Note that in (6.7), the quantifier imposes a selectional restriction that the N' should not be definite. Thus, while **pratit'i cheleke** "each-DEF boy-DAT" 'to each boy' is correct, ***prati chelet'ike** "each-NULL boy-DEF-DAT" is not. This restriction automatically prevents quantification of numerically modified phrases like ***prati duit'i chele** "each-NULL two-DEF boy-NULL".

The internal structures of the modifiers DEMO and QP have been analyzed in Sec-6.2.1.

The noun phrase NP is an N'' with an optional *genitive phrase modifier* GENIP. Hence we have rule (6.8) for the NP.

(6.8) $NP \longrightarrow$ $GENIP$ ($(\uparrow POSSESSOR) = \downarrow$) N'' ($\uparrow = \downarrow$)

The GENIP has been analyzed in Sec-6.2.2.

6.2 Internal Structure of NP Modifiers

6.2.1 Demonstratives and Qualitatives

The three commonly used *demonstratives* are **ei** this/these, **oi** that/those and **sei** that/those. They are indeclinables and project the appropriate DEM features. Mention must also be made here of the relative demonstrative **y.e** which marks a relative NP construction. *Relative NPs have not been considered in the present work.* The difference between **oi** and **sei** is that the latter is more often used as an anaphoric binder for a relative clause. The following are the lexical entries for the common demonstratives (all demonstratives are indeclinables):

ei: (DEMO) ($\uparrow$ DEM) = $THIS$
oi: (DEMO) ($\uparrow$ DEM) = $THAT$
sei: (DEMO) ($\uparrow$ DEM) = $THAT$

The following *quantifiers* (phrases) are most common:

prati, prat/y.ek "each", "each and every". These quantifiers may either take the singular DEF affixes **t'a, t'i** or no affix but a following count particle (CNT) from {**kat'a, kat'i**}. Thus we may have **pratit'a, prat/y.ek kat'i**, etc. The COUNT=+ feature is also projected to ensure that the head N is countable.

anek, sab "many", "all". They may take either the plural DEFINITE affixes **gulo, guli** or no affix but a following count particle from {**kat'a, kat'i, kichu**}. Thus we may have **anekgulo, sab kat'a, sab kichu**, etc. The COUNT=+ feature is projected.

bes', sa'ma'n/y.a, kichu "quite a few", "some", "few". These *belief* quantifiers must co-occur with count particle **kichu** to give **bes' kichu** "quite a few", **sa'ma'nya kichu** "only some" **kichu kichu** "just a few".

Note that **kichu** is both a quantifier and a count particle.

For proper analysis of the QP, we create a lexical category QUANT. QUANT words may be formed from stems belonging to QUANTROOT with an optional DEFINITE affix. The lexical entries of each quantifier stem project a QNT and a NUMB feature. The NUMB feature (SINGular, PLural or NO) serves in restricting conjoining of a wrong DEFINITE affix. The following lexical entries for the more common quantifiers are:

prati, prat/y.ek: QUANTROOT ((QNT UNIV), (NUMB SING) (COUNT +)
)
anek: QUANTROOT ((QNT LARGE), (NUMB PL) (COUNT +))
sab: QUANTROOT ((QNT UNIV), (NUMB PL) (COUNT +))
bes': QUANTROOT ((QNT HIGHBELIEF), (NUMB NO) (COUNT -))
sa'ma'n/y.a: QUANTROOT ((QNT LOWBELIEF), (NUMB NO) (COUNT -)
)
kichu: QUANTROOT ((QNT MEDIUMBELIEF))

The rule for the quantifier is given by supra-lexical specification:

```
QUANT => QUANTSTEM+DEF (CNT) where {
  exists($2) || ( ($1 QNT) != HIGHBELIEF && ($1 QNT) != LOWBELIEF )
  QNT<<($1 QNT);
  COUNT<<($1 COUNT);
}
```

The filter in the above specification ensures that quantifiers bes' and sa'ma'n/y.a are followed by a count particle. The COUNT feature ensures that the modifies head noun has compatible countability.

6.2.2 The Genitive

A genitive modifier for an NP is itself a noun phrase. However, the case marker on the head noun or pronoun must be for the POSSessive case. The rule for GENIP is as given in (6.9).

(6.9) $$GENIP \longrightarrow \begin{array}{c} NP \\ (\downarrow CASE) = POSS \end{array}$$

Thus a GENIP and an NP are syntactically equivalent units except that a GENIP requires that the head noun take a POSSessive case marker. Since the GENIP is a full fledged NP, it may have its own genitive modifier, which in turn may have its own modifier and so forth. Thus, an NP may actually have an attached chain of any number of genitive modifiers. However, every such modifier qualifies the most closely nested NP. In the form $g_1\ g_2\ \cdots g_n\ n$, where every g_i is an NP with POSSessive case markers on their "heads" and n is an NP with or without a possessive marker on the head, the phrase g_i, $i < n$ qualifies g_{i+1} and g_n qualifies

n. Such phrases are quite commonly used in Bangla. For example, consider the NP in Ex-1:

Ex-1 **a'ma'r bandhur bad'.a bha'iyer la'l kalamt'a'**
I-POSS friend-POSS elder brother-POSS red pen-DEF
"The red pen belonging to the elder brother of my friend"

Ex-1 should have an f-structure as given in (6.10). The schemata $(\downarrow\ CASE) = POSS$ under NP in (6.9) correctly carries out the analysis.

$$(6.10)\quad \begin{bmatrix} POSSESSOR & \begin{bmatrix} POSSESSOR & \begin{bmatrix} POSSESSOR & \begin{bmatrix} PRED & 'I' \end{bmatrix} \\ PRED & 'friend' \end{bmatrix} \\ ADJ & \{"bad'.a"\} \\ PRED & 'brother' \end{bmatrix} \\ DEF & YES \\ ADJ & \{"la'l\} \\ PRED & 'pen' \end{bmatrix}$$

6.3 Empty "Heads" and Affix "Hopping"

Sometimes, in discourse usage, the "head" noun or pronoun of an NP is omitted entirely (i.e. it manifests as empty or ϵ "heads"), especially if modifiers are present. Anaphoric resolution is made through matching of modifiers. As an interesting side effect, the affixes on the empty head remain explicit in the surface and *gets conjoined to the immediately preceding modifier.* The "hopping" affixes do not obey the rules of morpho-syntax during conjoining in that they get conjoined even *after* the terminating declension of the modifier. Let us consider Ex-2:

Ex-2 **ama'rt'a ama'y da'o**
"I-POSS-DEF I-DAT give-IMPER"
Give my (X) to me.

In Ex-2, the NP **a'ma'rt'a** has a deep structure **a'ma'r ϵ-t'a'** "I-POSS ϵ-DEF". Due to affix hopping, the DEF affix on the empty head has hopped to get conjoined with **a'ma'r**. Note that the morpho-syntactic rule that requires the DEF affix to precede the POSS declension, has been violated. This way **a'ma'r ϵ-t'a'r** "I-POSS ϵ-DEF-POSS" becomes **a'ma'rt'a'r** (the X belonging to my Y) and **a'ma'r ϵ-t'a'ke** "I-POSS ϵ-DEF-DAT" becomes **a'ma'rt'a'ke** (to the X belonging to me). Affix hopping is also observed with other modifiers like in **eit'a'ke** = **ei ϵ-t'a'ke** "this-DEF-DAT = this ϵ-DEF-DAT" to this X. The affix hopping phenomenon may give rise to strange instances when indeclinable particles may get declened. For example, **bad'.a** "big" is an indeclinable adjective. To mean 'to *the bigger/biggest of some X-s*, the NP **bad'.at'a'ke** "big-DEF-DAT" may be used. Note that the

indeclinable has conjoined with two affixes in bad'.at'a'ke.

Affix hopping is a hard implementational problem to solve as it involves gross violations of rules of morpho-syntax. In addition, during parsing of a word with "hopped" affixes, the empty "head" must be projected into the syntactic component by the lexical sub-system.

For Bangla, we offer a solution to the above problem after making the following observations:

i. All empty heads must have the DEF affix t'a, t'i on itself in the deep-structure. Thus, the "hopped" affix(es) may either be only the DEF affix or may be the DEF affix followed by other affixes. For example, we may have eit'a' "this-DEF = this ϵ-DEF", or we may have eit'a'ke "this-DEF-DAT = this ϵ-DEF-DAT". However, we may not have *eike "this-DAT" to mean " this ϵ-DAT" since in this case the empty head ϵ does not take a DEF affix.

ii. If hopped affixes conjoin with the head noun/pronoun of a genitive modifier, it does so only after the POSS declension on the noun/pronoun. Since almost all other (non-numeric) modifiers are headed by indeclinable words (indeclinable words are those that may not be inflected by any affix or declension), the hopped affix *conjoins directly* with the word itself. Thus, in chelert'a' "boy-POSS-DEF = the ϵ-DEF of the boy", the hopped t'a' has conjoined after the POSS declension on the genitive's stem chele. But in la'lt'a "red-DEF = the red ϵ-DEF", the hopped t'a' has conjoined directly after lal, the stem of the adjective modifier. The case with numeric modifiers has been taken up below.

iii. A pre-nominal NUM modifier itself may take a DEF affix. With such modifiers, affix hopping is never observed. For example, one may never have an NP * ekt'a't'a'ke "one-DEF-DEF-DAT". However, if a pre-nominal NUM modifier does *not* take a DEF affix as in ekt'a'ke "one-DEF-DAT", all affixes from but not including the DEF (i.e. only ke in this case) should be considered as hopped. The question of affixes hopping to a post-nominal NUM modifier does not arise as affixes may only hop on to the previous word.

iv. The DEF affix does not take part in any spelling rule from the right, i.e., the morpheme boundary between the DEF affix an the preceding morpheme can be easily determined.

We offer a solution in two parts. The first part necessitates certain enhancements in the morphological analysis component. The enhancements, that had been briefly mentioned in Sec-2.3.6 will be detailed below.

The first part of the proposed solution for tackling the affix hopping problem deals with non-NUMeral modifiers. During parsing, if the finite control of AFSA comes to a state s which may properly recognize any NP modifier word (whether really a modifier has been recognized may be checked from the comprehensive lexicon for the hitherto recognized morphemes). Let there be more unscanned symbols in the input word. The AFSA tries to recognize the remaining symbols from the root node of the DEF affix. In case the latter recognition process is successful, the lexical sub-system informs the syntactic component that two, rather than one words have been recognized and parsed. The first word consists of the stem and affixes recognized upto s. The second word has PRED as DREF where DREF is a lexical entity for Discourse REFerence. It may be used by a higher level discourse analyzer for actual binding. There is one restriction in the above process. State s can not be in the DAG DEF.

As an example, consider the word **a'ma'rt'a'r**. After scanning upto **a'ma'r**, the finite control is in a state s say. It can easily be checked from state s that the lexical form for **a'ma'r** is /ami/+/er/ "I+POSS". Now, POSS is a terminating class for nouns and the derived f-structure of a'ma'r' indicates a noun with POSS declension and hence a possible NP modifier. Attempt is next made to recognize the remaining symbols t'a'r from the root node of the DEF DAG. This yields a parse /t'a'/+/er/ "DEF+POSS". Two words are reported to be recognized — the first is "I-POSS" and the second "DREF-DEF-POSS".

The second part of the solution deals with NUMerical modifiers. The situation is different with NUMerals since they may take the DEF affix as their own right, which in turn may be followed by a nominal declension. It will thus be difficult to detect the hopping affixes at the lexical level. But noting that an NP may not have both pre- and post-numeric modifiers, we may consider a new rule (6.11) for N' as consisting of a NUMeral only. Note that in (6.11), the empty head DREF is projected syntactically through the schemata associated with NUM.

$$(6.11)\quad N' \longrightarrow \begin{array}{c} NUM \\ (\uparrow PRED) = DREF \end{array}$$

An associated implementational problem of empty heads is in detection of the boundary of an NP where the trace head had no inflection to hop. For example, consider **sab kichu** ϵ ,all some "everything". Here, both **sab** "all" and **kichu** "some" both qualify an empty head. In a sentence like **a'ma'r sab kichu toma'r theke peyechi** I-POSS all some you-POSS from get-1p-PRES-PERF, "I got whatever I have from you", the modifiers **sab** and **kichu** do not qualify **toma'r**. While rules (6.3)–(6.5) require definite presence of a 'qualified' noun with nominal modifiers, none can be detected here, not even from traces left in hopped affixes. To

solve this problem, we propose duplication of rules (6.3)–(6.5) in (6.12)–(6.14), to take care of a non-existing head in N' where the DREF entity is projected through functional control. Note that these rules are invoked only when the trace head has a NULL case marker. This is because if there had been a non-NULL affix on the trace, it would have hopped to a modifier and lexical parse of the modifier would have *introduced* the head as a DREF entity.

(6.12) $N' \longrightarrow \underset{\downarrow \in (\uparrow ADJ)}{\underset{(\uparrow PRED) = DREF}{NUM}} \quad A^*$

(6.13) $N' \longrightarrow \underset{\downarrow \in (\uparrow ADJ)}{A^*} \quad \underset{(\downarrow DEF)}{\underset{(\uparrow PRED) = DREF}{NUM}}$

(6.14) $N' \longrightarrow \underset{\downarrow \in (\uparrow ADJ)}{\underset{(\uparrow PRED) = DREF}{A^*}}$

6.4 Post-positional Phrase

Simply speaking, a post-positional phrase is an NP modified post-position. The post-position acts as the "head" of the phrase and behaves like the case marker for the entire phrase. The rule for a PP is given by (6.15).

(6.15) $PP \longrightarrow \underset{(\uparrow POBJ) = \downarrow}{NP} \quad P$

Most post-positions used in Bangla are positional. PPs headed by such post-positions do not play any functional role but act as adjuncts. All these post-positions require a GENIP as the NP. Examples of adjunctive post-position are **upare** 'above', **bhitare** 'inside', **madh/y.e** 'inside', **ba'ire** 'outside', etc. There are three classes of non-adjunctive post-positions. They are analyzed below.

The Agentive/Instrumental Post-Position

The post-positions **diye** and **d/Ba'ra'** ("Sadhu") 'by', signify agentive or instrumental role for the governed phrase. Depending upon the verb, a **diye** governed

PP may play either the INSTrumental or the SUBJect or even the affected subject (ASUBJ) functional role. d/Ba'ra' requires that the governed NP actually be a GENIP. For **diye** the case feature of the governed NP is NULL if the head of the NP has a CAT=instrument feature; otherwise it is DAT. In our analysis, we let PPs governed by **diye, d/Ba'ra'** project an INST value for the PCASE feature. Thus we have:

$$\text{d/Ba'ra'} \quad P \quad \{ \begin{array}{l} (\uparrow\ POBJ\ CAT) = instrument; (\uparrow\ POBJ\ CASE) = NOM \\ (\uparrow\ POBJ\ CASE) = POSS \\ (\uparrow\ PCASE) = INST \end{array} \}$$

$$\text{diye} \quad P \quad \{ \begin{array}{l} (\uparrow\ POBJ\ CAT) = instrument; (\uparrow\ POBJ\ CASE) = NOM \\ (\uparrow\ POBJ\ CASE) = DAT \\ (\uparrow\ PCASE) = INST \end{array} \}$$

Examples:

ra'mke diye Ram-NULL by "by Ram".

ra'mer d/Ba'ra' Ram-POSS by "by Ram".

*ra'm d/ba'ra' *Ram-NULL by. Because "Ram" does not have an INTRUMENT=+ feature.

la't'hi diye stick-NULL by "by (a) stick".

la't'hir d/Ba'ra' stick-POSS by "by (a) stick".

The Source Locative Post-Position

The most used source locative post-position is **theke** 'from'. The case feature of the governed NP is NULL if the head of the NP has a PLACE=+ feature; otherwise it is POSS (i.e. NP is a GENIP). We let **theke** project the LOC value for PCASE feature. Thus we have:

$$\text{theke} \quad P \quad \{ \begin{array}{l} (\uparrow\ POBJ\ CAT) = place; (\uparrow\ POBJ\ CASE) = NOM \\ (\uparrow\ POBJ\ CASE) = POSS \\ (\uparrow\ PCASE) = LOC \end{array} \}$$

Examples:

ra'mer theke Ram-POSS from "from Ram".

dilli' theke Delhi-NOM from "from Delhi".

The Reason Post-Position

The post-position jan/y.a 'for' (also seldom used tare) requires a governed GENIP. We let this post-position project PCASE=FOR.

jan/y.a P $(\uparrow\ POBJ\ CASE) = POSS$
$(\uparrow\ PCASE) = LOC$

Examples:

ra'mer jan/y.a Ram-POSS for "for Ram".

t'a'ka'r jan/y.a money-POSS for "for money".

6.5 Discussions

In the present chapter, we have attempted a GLFG (actually LFG, since no special GLFG feature has been used) based analysis for Bangla Noun Phrases and Post-Positional Phrases. We were limited by non-availability of pertinent literature. Our analysis is mostly modeled after the one provided by Frey [67]. A major portion of the chapter dealt with the Noun Phrase, more particularly, the different modifiers of the head noun/pronoun of a noun phrase and their selectional restrictions. In our estimate, the analysis covers a very large portion of day to day Bangla NP usage and may be implemented in a straightforward manner.

The aspect of "affix hopping" that has also been considered in this chapter however has some serious implementational connotations. We have shown that this phenomenon might lead to chaotic behaviour of the lexical component as certain basic assumptions are violated. However, from certain observations that we have made, we were able to provide a technique for tackling the problem with no unreasonable increase in complexity. The problem of detecting a noun phrase where the head is missing but modifier(s) are present, has also been taken up. A solution has been proposed that takes care of most examples. Nevertheless, the offered solutions are mostly ad-hoc and the phenomenon deserves further and deeper investigation.

An NP in Bangla may also be a gerundative phrase. A gerundative phrase is a clause like structure in which the verb is a participle and there are no auxiliaries to provide the tense. As in English, two closely resembling but semantically different gerundative forms are observed. In either case, the verb consists of a (simple or compound stem) terminated by affix a' (not causated) or a'no (causated). In the passive form, the verb is preceded by an agentive GENIP, an optional patient

NP playing the IOBJ function if the verb is causated (whose case requirement is determined by the m-structure of the verb) and followed by an OBJ NP. An example is given below:

ama'r kha'oya' a'pelt'a ...
"I-POSS eat-PARTICIPLE apple-DEF ..."
The apple eaten by me ...

In the other form, the OBJ NP is optional and if present must precede the verb as in:

ama'r a'pelt'a kha'oya' ...
"I-POSS apple-DEF eat-PARTICIPLE ..."
Eating of the apple by me ...

We shall refer to the first form as a *verbally modified* and the second one as a *verbal* noun phrase. Some linguistic work on Bangla gerundative phrases has been carried out by P. Dasgupta [46]. Presently, we are also investigating into this form to bring it under the purview of the GLFG. For this we have taken help of supra-lexical specifications. Some breakthrough is expected in the near future.

Chapter 7

The Sentential Paradigm of Bangla

7.1 Types of Sentences

Sentences are classified according to the number of verbs (not the number of words constituting the verb) contained in them as well as the relative importance of the roles of the verbs. A sentence can be:

Simple Sentence. Here there is only one verb which is the main verb.

Garden-Path Sentence. Here there are more than one verb which are involved in providing the sense. However, one verb, called the main verb, is given a superior status and occurs last in a chain of verbs. The verb-chain describes a sequence of actions partially ordered in time or causationally. The superiority of the main verb is reflected in agreement between its GNPH and the GNPH of the subject of the overall sentence. Also, the main verb is the only finite verb in the sentences. All other verbs are infinite ones.

Compound Sentence. Here also there are multiple verbs. However, in this case there are more than one main verb, i.e. that there are multiple subjects. Like English sentences, Bangla compound sentences involve sequences of simple sentences connected by conjunctive operators. Compound sentences have not been considered in the present work.

Complex Sentences: The complete class of Bangla complex sentences include those in which the dependent clause is embedded (See Sec-4.4 and also Sec-7.4), as well as the class of sentences having multiple relative clauses. In

the latter type, successive clauses are linked by anaphoric operators (like the anaphoric "that" operator linking English relative clauses. A major work on Bangla relative clauses may be found in P. Dasgupta [47]. We have not considered relative clauses in the present work.

These other class of complex sentences is similar to the "to" or "for" infinitive sentences of English except that in Bangla, the infinitive clause is embedded in the outer sentence. Theoretically, dependent clauses may be embedded to arbitrary depths, implying thereby that Bangla is an NP-complete language in this respect. However, normal speakers rarely use embedding beyond first (or at most second) level. This is one of the most difficult features of Bangla from NLP point of view.

Sentential Adverbial/Adjectival Clauses: These clauses are somewhat like the adverbial and adjectival clauses of English. Strictly speaking, they are not sentences since such a clause in isolation does not constitute a well-formed sentence. We have not covered this type of clauses as most of aspects pertaining to their analysis have not yet been sufficiently studied from a computational linguistic point of view.

Interrogative Sentences: There are two types of interrogative sentences in Bangla ---- "Yes-No" queries and "k-phrase". The former type has a relatively simple structure. The latter type is so named because of similarities with English "Wh-phrases". We are presently studying the aspects of interrogative sentences with a view of providing a GLFG based analysis for them. However, they have not been considered in the present work.

7.2 Simple Sentences

Simple sentences have been discussed in some detail in Chapter-4. In this section we consider some special aspects of simple sentences not considered there.

7.2.1 Sentences Without a Finite Verb

In Bangla (not necessarily in all Indian languages; for example, not in Hindi), sentences with forms like "X *is* Y", the verb, which is the *copula*, is omitted. For example:

i. **tumi bha'la chele** *You good boy* "You (are a) good boy". This the simplest type with form $< Object >$ *is* $< Object >$, where $< Object >$ is a noun

phrase NP (whose head may be adorned by adjectives).

ii. **a'mi toma'r bha'i** *I you-POSS brother* "I (am) your brother". Here the form is $< Object > \ is \ < Function > \ of \ < Object' >$. Here *Function* is an *attributive* noun like "brother", "capital", "president", "name", etc. The syntactic form of the $< Function > \ of \ < Object' >$ part of the sentence is an NP with $< Function >$ as the *head* noun and $< Object' >$ being a GENIP (genitive phrase).

iii. **balt'a' la'l** *Ball-DEF red* "The ball is red". Here the form is $< Object > \ is \ < Quality >$, where *is* should be interpreted as *has property*, and $< Quality >$ is an adjective phrase AP. An AP is normally a simple adjective, but may be followed by $< QualityType > -POSS$ phrase (where $< QualityType >$ is a noun for a quality name like "colour", "size", etc.) thus highlighting the *type* of quality qualified by $< Quality >$. For example, the sentence considered above could as well be written as **balt'a' la'l raNer** *Ball-DEF red colour-POSS* "The ball is of red colour". We have not analyzed the AP to sufficient depth in the present thesis.

iv. **chelet'a'ke dekhte bha'la** *Boy-DEF-DAT see-FORINF good* "The boy (is) good to look at". Here the form is NP-DAT VERB-FORINF ADVERB. Alternative forms may lack the DAT case marker on the NP and/or the VERB and the ADVERB may interchange position as in **chelet'a' bha'la dekhte** *Boy-DEF good see-FORINF.*

Since the verb is missing from the class of sentences considered above, the question of m-structure projection does not arise. Hence, there is no difference between GLFG and LFG rules for the above class of sentence. Rules for the different types of sentences considered above have been enumerated below in the same order.

i. In these sentences, it is not clear which is the subject and which is the object. We arbitrarily choose the first phrase as the subject and the second as the object. The semantics of the sentence imply a *unification of the concepts* described by the subject and the object. With these considerations, the rule is:

$$S \longrightarrow \underset{(\uparrow SUBJ)=\downarrow}{NP} \quad \underset{\substack{(\uparrow OBJ)=\downarrow \\ (\uparrow PRED)=unify<(\uparrow SUBJ),\downarrow>}}{NP}$$

Example: The semantics of the sentence **tumi bha'la chele** is $unify < YOU, GOOD-BOY >$, where YOU and $GOOD-BOY$ are the f-structures for the case phrases **tumi** "you" and **bha'la chele** "good boy" respectively.

ii. In this case, the semantics of the sentence implies a *relationship* $< Function >$ between $< Object >$ and $< Object' >$. The rule is:

$$S \longrightarrow \underset{(\uparrow\ SUBJ) = \downarrow}{NP} \qquad \underset{\substack{(\uparrow\ PRED) = \\ (\downarrow\ PRED) < (\uparrow\ SUBJ), (\downarrow\ POSSESSOR) >}}{NP}$$

Example: The semantics generated for the sentence a'mi toma'r bha'i I-NULL you-POSS brother-NULL "I am your brother" is $BROTHER < I, YOU >$, where $BROTHER$, I and YOU are the semantics of the NPs bha'i, a'mi and tom'ar, respectively.

iii. This type of sentences do not have any semantic representation as such. Here the syntactic form of $< Quality >$ is that of an adjective phrase. A sentence simply *asserts* a $< Quality >$ qualifier for $< Object >$ for future discourse reference. Thus we have:

$$S \longrightarrow \underset{(\uparrow\ SUBJ) = \downarrow}{NP} \qquad \underset{(\uparrow\ PRED) = asserta < (\uparrow\ SUBJ), \downarrow >}{AP}$$

Example: The semantics of the sentence balt'a' la'l (raNer) *Ball-DEF red (colour-POSS)* "The ball is (of colour) red"is an assertion clause $asserta << BALL, RED >$, where $BALL$ is the semantics of the subject phrase balt'a and RED is the semantics of the adjective of quality la'l "red".

iv. In this type of sentences, the verb must pertain to an action in which something concrete or abstract is taken into body or mind. The sentence lacks an explicit subject as in ergative constructions. However, unlike ergative constructions, the subject includes the universal set of objects capable of performing the action described by the verb. The semantics is more involved than can be expressed by the notation currently being used. Informally, the semantics of the sentence chelet'a'ke dekhte bha'la should generate the semantic clause as in the English sentences with an anonymous subjects like "It is good to look at the boy" or "The boy is good to look at". We are presently carrying out further investigation into the matter.

7.3 Garden-Path Sentences

A garden-path sentence is a chain of clauses headed infinite verbs, with a clause headed by finite verb completing the overall construct. The SUBJects of all the clauses are identical and may be physically absent in all but one of the clauses. Similarly, the TENSE of the overall sentence is identical to the TENSE of the final clause. The clauses with infinite verbs naturally do not have any TENSE passed on from the verb. The TENSEs of the non-final clauses are unified with the TENSE of the sentence. In the overall sentence, the part constituting the clauses with

infinite verbs shall be denoted as the SCOMP attribute of the sentence. The point of interest is *the flow of temporal context in the chain of the clauses constituting SCOMP and between SCOMP and the final clause.* A Garden-path sentence will be given a semantics "$garden-path < FLOW, SCOMP, LAST >$". Here, FLOW is a temporal attribute which can be either "*BEFORE*" or "*SIMULTANEOUS to*". SCOMP refers to the chain of non-final clauses represented by the SCOMP attribute. LAST is a reference to the last clause in the sentence. The semantics may be interpreted as "SCOMP took place FLOW LAST". With this, the GLFG rule for garden-path sentences is:

$$
\begin{array}{lcll}
S & \longrightarrow & INF & S \\
 & & (\uparrow SCOMP) = \downarrow & (\uparrow SUBJ) = (\downarrow SUBJ) \\
 & & (\uparrow SCOMP\ SUBJ) = (\uparrow SUBJ) & (\uparrow TENSE) = (\downarrow TENSE) \\
 & & (\uparrow TENSE) = (\downarrow TENSE) & \\
 & & (\uparrow FLOW) = (\downarrow FLOW) & (\uparrow PRED) =' garden-path < (\uparrow FLOW), \\
 & & & (\uparrow SCOMP), (\downarrow PRED) >'
\end{array}
$$

In the above rule, a FLOW attribute is being propagated from INF, which is used in the PRED clause of the sentence. For every clause with infinite verb, the FLOW attribute gets projected lexically by the verb (see Chapter-5). The resursion in the rule permits garden-path sentences of arbitrary length to be generated. The schemata for semantic clause denotes passage of temporal context from left to right.

INF is a clause (with a possible absence of subject) headed by an infinite verb, as described by the following rule:

$$
\begin{array}{lccc}
INF & \longrightarrow & NP^* & V \\
 & & (\uparrow;?) =? & (\downarrow FLOW)
\end{array}
$$

In the above rules, the NP* V structure is the usual for verb-terminated clauses. The schemata $(\downarrow FLOW)$ under V ensures that the verb is infinite and projects a FLOW attribute.

Example: Consider the garden path sentence:

a'mi ba'd'.i giye ca' khete khete bait'a' pad'.ba
I-NULL home-NULL go-CONTINF tea-NULL eat-FORINF eat-FORINF book-DEF read-FUTURE-1p
"I, after reaching home, while drinking tea, shall read the book".

Let the semantic clauses for the verbs be:

giye go-CONTINF. $go < (SUBJ), to < (OBJ) >>$

khete khete eat-FORINF eat FORINF. $ingest < (SUBJ), (OBJ) >$

pad'.ba read-FUTURE-1p. $read < (SUBJ), (OBJ) >$.

Let the symbolic f-structures of the NPs a'mi, ba'd'.i, ca' and bai be "I", "HOME", "TEA" and "BOOK", respectively. The semantic clause for the entire sentence is then:

$garden - path \quad < BEFORE, go < I, to < HOME >>,$
$garden - path < SIMULTANEOUS, eat < I, TEA >, read < I, BOOK >>>$

7.4 Complex Sentences

We have already talked about the class of complex sentences in which the dependent clause is embedded in the matrix, in Sec-4.4. There we considered a sub-class in which the verb of the complement was infinite with a FORINF declension. In the present section, we would talk about embedded-clause complex sentences where the main verb is bal "say", that is, the main verb is an *equi* verb. We have not covered relative clauses in the present work.

It has already been pointed out in Sec-4.4 that the Bangla equi verb bal is quite versatile. With bal as the main verb, the following cases could occur (considering a single embedded clause), with the complement verb having different valencies:

i. The complement verb is mono-valent (i.e. intransitive). In this case, the subject of the complement clause is not explicit, with the result that the complement clause takes the simple form of a single verb (with the appropriate declension) as in:

ra'm a'ma'ke baste/basbe baleche
Ram-NULL me-DAT sit-FORINF/sit-FUTURE-3p-1h say-PAST-3p-1h
"Ram has told me that I should/ he would sit"

Thus, depending upon whether the declension on the complement verb is FORINF or of FUTURE TENSE, the SUBJect of the complement is derived from IOBJect or the SUBJect of the matrix respectively. The SUBJect and the IOBject phrase can permute freely anywhere to the left of the complement. Thus the following example is a alternate well-formed version of the above example:

a'ma'ke ra'm baste/basbe baleche
Me-Dat Ram-NULL sit-FORINF/sit-FUTURE-3p-1h say-PAST-3p-1h
"Ram has told me that I should/ he would sit"

This sort of permutation is seen in all the other cases discussed below.

ii. The complement is bivalent (i.e. mono-transitive). Here, there can be a few different cases:

(a) Where both the SUBJect and the OBJect of the complement are implicit and are derived from the matrix. The surface form remains identical to the intransitive complement verb case. Only the ibe declension is allowed on the verb of the complement. The SUBJect and the OBJect of the complement is derived from the SUBJect and the IOBJect of the matrix respectively, as in:

ra'm a'ma'ke ma'rbe baleche
Ram-NULL me-DAT kill-FUTURE-3p-1h say-PAST-3p-1h
Ram has said (to me) that he will kill me".

(b) Where the complement verb's object is inanimate. Relevant for both declension forms on the complement's verb. Here, the SUBJect of the complement is derived from the SUBJect/IOBJect of the matrix depending upon whether the verb of complement is in FUTURE tense or whether it is a FORINF type infinite verb. For example:

ra'm a'ma'ke bait'a' pad'.be baleche
Ram-NULL me-DAT book-DEF read-FUTURE-3p-1h say-PAST-3p-1h
"Ram has said (to me) that he will read the book".

ra'm a'ma'ke bait'a' pad'.te baleche
Ram-NULL me-DAT book-DEF read-FORINF say-PAST-3p-1h
"Ram has said (to me) that I should read the book".

(c) If the object of the complement verb is animate, there is problem. Normally, for such verbs, the object takes the DAT case marker. The equi verb of the matrix necessarily has an IOBJect with a DAT case marker. It is quite easy to confuse between the IOBJect of the matrix and the OBJect of the complement. We have conducted small experiments with native speakers who have generally agreed upon this confusion. However, whereas some feel that they would rather not try to comprehend a sentence like:

? ra'm a'ma'ke toma'ke ma'rte baleche
? Ram-NULL me-DAT you-DAT kill-FORINF said "? Ram has told me (you) that I (you) should kill you (me)"

some are of the opinion that the rule for binding the object of the complements to the nearest CP with DAT marker (as in the previous case) should be the approach. Indeed, we have incorporated the latter point of view in our system.

iii. The complement verb is trivalent (i.e. bi-transitive). This case leads to a more complicated form of the situation encountered with transitive complement verb with animate OBJects. We have kept it outside our purview for the time being.

From the above discussions, it is clear that the Bangla verb (stem) bal is used to mean request, command, persuade, etc., on one hand and acquiesce, promise, agree, etc., on the other. For the former sense to be valid, the embedded clause must have a FORINF verb, while for the latter, the embedded clause must have a verb in FUTURE tense. bal therefore has the following alternate projections:

$$
\begin{array}{lll}
\text{bal} & V & (\uparrow\ PRED) =' ask < (SUBJ),(COMP),(IOBJ) >' \\
 & & (\uparrow\ COMP\ SUBJ) = (\uparrow\ IOBJ) \\
 & & (\uparrow\ COMP\ INFTYPE) =_c FORINF \\
 & & (\#\ SUBJ\ ANIM) = + \\
 & & (\#\ SUBJ\ CASE) = NULL \\
 & & (\#\ IOBJ\ ANIM) = + \\
 & & (\#\ IOBJ\ CASE) = DAT
\end{array}
$$

$$
\begin{array}{lll}
\text{bal} & V & (\uparrow\ PRED) =' promise < (SUBJ),(COMP),(IOBJ) >' \\
 & & (\uparrow\ COMP\ SUBJ) = (\uparrow\ SUBJ) \\
 & & (\uparrow\ COMP\ TENSE) =_c FUTURE \\
 & & (\#\ SUBJ\ ANIM) = + \\
 & & (\#\ SUBJ\ CASE) = NULL \\
 & & (\#\ IOBJ\ ANIM) = + \\
 & & (\#\ IOBJ\ CASE) = DAT
\end{array}
$$

Examples:

Let us consider the following example sentences:

i. a'pni a'ma'ke bait'a' pad'.te ballen
You-NULL I-DAT book-DEF-NULL read-FORINF tell-3p-hon-PAST
You (honoured) asked me to read the book

ii. a'pni a'ma'ke bait'a' pad'.ben ballen
You-NULL I-DAT book-DEF-NULL read-FUTURE tell-3p-hon-PAST
You (honoured) promised me to read the book

Let the lexical projections for the complement verbs be:

$$
\begin{array}{lll}
\text{pad'.te} & V & (\uparrow\ PRED) =' read < (SUBJ),(OBJ) >' \\
 & & (\uparrow\ INFTYPE) = FORINF \\
 & & (\#\ OBJ\ ANIM) = - \\
 & & (\#\ OBJ\ CASE) = NULL
\end{array}
$$

pad',ben V $(\uparrow\ PRED) =' read < (SUBJ),(OBJ) >'$
$(\uparrow\ TENSE) = FUTURE$
$(\#\ OBJ\ ANIM) = -$
$(\#\ OBJ\ CASE) = NULL$

We have already considered the rules for embedded clause complex sentences in Sec-4.4, equations (4.15) and (4.16). Some evaluation order restrictions were also discussed there. Using them, the f-structures for the above two examples would be:

$$
\left[\begin{array}{ll}
SUBJ & \left[\begin{array}{ll} ANIM & + \\ CASE & NULL \\ GNPH & 2/3p-2h \\ PRED & 'You' \end{array}\right] \\
IOBJ & \left[\begin{array}{ll} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & 'I' \end{array}\right] \\
COMP & \left[\begin{array}{ll} OBJ & \left[\begin{array}{ll} ANIM & - \\ DEF & YES \\ CASE & NULL \\ GNPH & 3p-1h \\ PRED & 'book' \end{array}\right] \\ SUBJ & \\ TENSE & FORINF \\ PRED & 'read < (SUBJ),(OBJ) >' \end{array}\right] \\
TENSE & PAST \\
PRED & 'ask < (SUBJ),(COMP),(IOBJ) >'
\end{array}\right]
$$

$$
\begin{bmatrix}
SUBJ & \begin{bmatrix} ANIM & + \\ CASE & NULL \\ GNPH & 2/3p-2h \\ PRED & 'You' \end{bmatrix} \\
IOBJ & \begin{bmatrix} ANIM & + \\ CASE & DAT \\ GNPH & 1p \\ PRED & 'I' \end{bmatrix} \\
COMP & \begin{bmatrix} OBJ & \begin{bmatrix} ANIM & - \\ DEF & YES \\ CASE & NULL \\ GNPH & 3p-1h \\ PRED & 'book' \end{bmatrix} \\ SUBJ & \\ TENSE & FUTURE \\ PRED & 'read < (SUBJ),(OBJ) >' \end{bmatrix} \\
TENSE & PAST \\
PRED & 'promise < (SUBJ),(COMP),(IOBJ) >'
\end{bmatrix}
$$

The utility of the schemata $(\uparrow\ SUBJ\ GNPH) = (\downarrow\ GNPH)$ in (4.16) may be discussed now. It restricts the GNPH attribute of the verb of a FUTURE type embedded clause to be the same as that of the main verb. With this restriction, the following sentence is ill-formed:

*** a'pni a'ma'ke bait'a' pad'.bi ballen**
You-NULL I-DAT book-DEF-NULL read-FUTURE-2p-0h tell-3p-hon-PAST

7.5 Discussions

In this chapter, certain types of Bangla sentences have been analyzed in the LFG/GLFG framework. Most of the sentential forms discussed are typical for Bangla. For example, sentences without a single verb is not so prevalent in other Indian languages. The garden-path sentences discussed here is another typical Bangla sentence type. Whereas other Indian languages do not prohibit usage of garden-path sentences of the type discussed here, they are not so abundantly used as in Bangla. This aspect of Bangla interested Chatterjee [34], who has pointed out that even very long garden-path sentences in Bangla are accepted as stylistically and idiomatically adequate. In this chapter, we have shown that the above two forms may be explained in the light of the GLFG. We have also carried out a more involved discussion on the use of the *equi* verb bal in forming a class of complex sentences. The pertaining analyses are a continuation from Sec-4.4. However, here

we have not only pointed out the versatility of bal but have also drawn parallels with similar complex forms of English. Both, in Sec-4.4 and here (Sec-7.4), we have demonstrated how judicious use of control schema may be used to impose various selectional restrictions in class of complex sentences discussed.

Many aspects of Bangla sentences have not been covered in the present work. While it is difficult to provide analyses for some of the aspects like gerundal infinitives, sentential adjectives, etc., we have already carried out some study into analyzing interrogative sentences. Of the interrogative sentences, the "k-phrase" are an interesting study. Like English, in Bangla, there are query "keywords" for different thematic roles of the sentences. Furthermore, like English, all the query keywords have k as the prefix, as in ke "who", ki "what", kon "which", kabe, kakhan "when(day)/when(time)", kotha'y "where", etc., and hence the name k-phrase. Like English "Wh-phrase" queries, the k-phrase is substituted for the entity in the corresponding position of the non-interrogative sentence. Just as with normal sentences, entities in k-phrase queries may freely permute. For example, all the interrogative sentences below, have identical meaning.

ke bait'a' pad'.be
Who book-DEF read-FUTURE-3p-1h
"Who will read the book ?".
bait'a' ke pad'.be
Book-DEF who read-FUTURE-3p-1h
"Who will read the book ?".
bait'a' pad'.be ke
Book-DEF read-FUTURE-3p-1h who
"Who will read the book ?".

Bangla "Yes-No" interrogative sentences differ from English in that first, in Bangla, there is a special verbal yes-no operator ki used for constructing a query, and secondly, there is no subject-verb position interchange. The ki operator may occur anywhere, except at the sentence initial position, as in:

ra'm bait'a' pad'.be ki
Ram-NULL book-DEF read-FUTURE-3p-1h yes-no
"Will Ram read the book ?".
ra'm ki bait'a' pad'.be
Ram-NULL yes-no book-DEF read-FUTURE-3p-1h
"Will Ram read the book ?".
ra'm bait'a' ki pad'.be
Ram-NULL book-DEF yes-no read-FUTURE-3p-1h
"Will Ram read the book ?".

We intend to provide a comprehensive GLFG analysis of Bangla interrogative sentences in the near future.

Chapter 8

Conclusion

The motivation behind the present thesis was to provide a realistic computational model for a computer-based lexicon-driven syntactic processing of modern Indian languages, with Bangla as the target language. The model was intended to have a well-defined and extendible implementational connotation.

The Lexical Functional Grammar was chosen as the primary formalism. The goal was set at:

- Formulation of a pre-syntactic processing stage meant for strong lexical interaction. This includes techniques for parsing isolated words into constituent morphemes (called lexical level analysis), as well as detection of multi-worded lexical entities and lexical projection thereof (called supra-lexical level analysis). The goal includes development of software tools to enable a linguistic expert to systematically specify morpho-syntactic and supra-lexical properties of the language and automatic generation of various analytical components from them.

- Formulation of a syntactic processing stage as extensions on the basic LFG approach. It has been shown that Bresnan's concept of non-configurational encoding leads to a less-efficient parser implementation for Bangla and similar languages. The suggested extensions were intended at circumventing the problem.

- Providing an object-oriented design for implementation of the formalisms suggested above and directly implementing some of the key aspects in a C++ based programming platform.

- Applying the developed formalism to different sub-paradigms of Bangla natural language processing, namely, the verb, the case phrase and certain classes of sentences and clauses.

Goals Achieved

- A *finite-state tool for parsing words, called the Augmented Finite State Automata* (AFSA) has been proposed. The AFSA is capable of parsing words into constituent morphemes, even in the presence of spelling rules that render detection of morpheme boundaries difficult.

- A *software tool for use by the lexical "expert" was proposed.* This tool permits specification of the lexicon of the target language in four parts — major morpheme classes with abstract lexical properties for each class, rules of morpho-syntax with knowledge of constructing the lexical properties of a word class from the properties of the constituent morphemes, spelling rules and the morphemes constituting the vocabulary along with specific lexical properties of the morphemes. A part of the tool creates a Comprehensive Lexicon — an indexed database of morphemes. Another part compiles the AFSA in a systematic manner. The compiled AFSA contains pointers leading into the comprehensive lexicon through which lexical projections of constituent morphemes can be easily recovered. The projection of the word is derived as a unification of the projections of the constituents. The implementation of the software tool has been achieved almost to the complete specifications. However, prefixes and compound stems (due to euphony) have not been incorporated. Some amount of manual tuning of the AFSA may be required to take care of some special cases.

- A software tool for supra-lexical specification by the lexical expert has also been proposed. Through this tool, the lexical expert informs the system how to recognize multi-worded lexical entities and how to generate the projection of these entities. The aspects that are specified include different word groups and how they co-occur, filters for validating different properties of co-occurring words, how the different attributes of the combined entity are instantiated and how the semantics function of the entity is created. The major achievement in this respect is proper identification of the different aspects of supra-lexical specification and proposing a non-ambiguous tool for the same. Implementation of the component for automatic construction of the supra-lexical analyzer from the given specifications can be carried out without much difficulty using tools like *lex* and *yacc*.

- The analysis described above can be carried out in a manner transparent to the syntactic component. Thus, syntactic specification for phrases and sentences, need not be detailed down to the morphemes.

- A viable (for parser implementation) syntactic formalism that takes care of most of the problems arising from Bresnan's postulates, has been proposed in the form of the Generalized Lexical Functional Grammar (GLFG). The main theme of the GLFG is delayed syntactic encoding of participating noun phrases (NPs) of a sentence. The key features of the GLFG are — an under-specification metavariable "?" that assigns unnamed functions to NPs, a meta-structure projected lexically by verbs for late binding of names of functions through tests and a symbol table for co-ordinating the entire process. We have illustrated that the above features succeeds in decreasing potential non-determinism arising out of Bresnan's postulates, to a reasonable extent. We have mapped our extensions into a new operator `Search` and have extended the semantics of the traditional `Locate` operator of LFG. Some of the key components like the operators `Locate` (extended) `Merge` (unification component) and `Search` have been implemented. However, since only few data on internal grammars of NPs, PPs, etc., for bangla were available, we have implemented our ideas only for a restricted types of sentences (one worded NPs).

- Our syntactic formalism is clearly centered around the verb. Bangla verbs constitute a considerably complicated paradigm. We have provided a systematic study of the different aspects of the Bangla verb, including classification, morphological properties, nature of infinite verbs and multi-worded verbs. For some of the more complicated cases like certain difficult examples of negated verbs, verbs consisting of three or more words, composite verbs, etc., could not be covered in much depth due to lack of sufficient linguistic data based on detailed analyses of usage of such forms.

- A description of Bangla noun and post-positional phrases has also been provided. However, here again, we are hampered by near absence of linguistic data. We have carried out a preliminary study based on our approaches. We expect that our analysis would initiate more involved investigations in the field by linguists.

- We have reserved a chapter for discussing some typical sentential forms of Bangla. The situation here is also similar to the one with NPs, i.e., there is virtually no computational linguistic works covering the different sentence types discussed (or just mentioned) by us. Again, we expect to initiate in-depth linguistic studies in the field.

Application Potentials

The obvious application domain is in building natural language understanding systems for Bangla and similar languages. This requires further development in the semantic aspects that we have not covered in this thesis. Such a system may be applied almost every domain of man-machine interaction and also in machine abstraction of texts. While we have not investigated machine translation aspects in this thesis, we have a strong feeling that the techniques would be quite suitable in translations environment also. In the restricted domain, the techniques may be used to build up query languages for databases for various domain. However, for this, a deeper study of question type sentences, especially the "ka-phrases" (so called because of similarities with English Wh-phrases), is necessary. The morpho-syntactic components may be applied in isolation in spelling checker and corrector. Moreover, it may be used in a speech recognition system to provide high level word hypothesis verification.

Scope for Future Work

The problem of a realistic computer implemented NLP system for Bangla (or even any Indian language) is a vast domain, virtually unexamined. There is scope for further work in practically every aspect of NLP. However, we have identified certain fields as of immediate interest:

- A more rugged morphological parser incorporating prefixes as well as euphony. As we have pointed out, with the current version of the AFSA, the above enhancement may lead to potential violation of some basic premises. At present, we are investigating the nature of the violation and trying to hypothesize about their nature.

- It is clear from the present thesis that the efficiency of the syntactic component is most dependent on lexical design, especially the m-structure of verbs. A more discriminating m-structure of verbs leads to faster and better rejection of wrong parses. It is however a daunting task to manually construct the m-structure of all verbs of any language. A better alternative would be to have a software tool for the same. We have also been investigating this point for some time now. Our approach is based on the fact that the entire set of verb may be conceptually clustered such that the m-structure of the verbs in a cluster are similar to one another. The proposed software tool would be

used by a linguist for incremental betterment of the clusters. This way, the ideal situation would be attained asymptotically.

- We would like to carry out detailed study for the incorporation of:
  - Negation in general and of all verb forms in particular. For this, a clearer understanding of the semantics of the negation operators and negative forms is necessary.
  - Rule-based detection of compound and composite verbs. A breakthrough in these domains would not only result in tremendous reduction in storage requirement but also provide considerable linguistic insight into the general phenomenon of "compounding".
  - Gerundative phrases.
  - Relative Clauses. Analysis of relative clauses is important from the angle of natural language database query languages. At the primary level, we would like to identify the anaphoric operators, which normally occur in pairs and syntactically mark relative clauses.
  - Questions — "Yes-No" clauses as well as "ka-phrases". We have already done some preliminary study in this field. Indications are that simple interrogative sentences may be incorporated into our system without much trouble.
  - Co-ordination and Compound Sentences. As with English, study of compound sentences and co-ordinated phrases is a difficult branch of study. Ambiguity and its efficient handling must be seriously studied before taking up this field.

Bibliography

[1] Aho A. V.; Ullman J.D. 1972 *The Theory of Parsing, Translation, and Compiling*. Prentice- Hall: Englewood Cliffs, NJ.

[2] Aho A. V.; Ullman J. D. 1977 *Principles of Compiler Design* Addison Wesley: Reading, MA.

[3] Aho A. V.; Hopcroft J. E.; Ullman J. D. 1982. *Data Structures and Algorithms*. Addison Wesley: Reading,MA.

[4] AlamY. S. 1983. A Two-Level Morphological Analysis of Japanese. *Texas Linguistic Forum* **22**. 229–252.

[5] Allen J. F.; Perrault C. R. 1981 Analyzing Intention in Utterances *Artificial Intelligence* **16**: 441–458.

[6] Allen J. 1987 *Natural Language Understanding*, Benajamin Cummings : Menlo Park, CA, 1987.

[7] Alshawi H.; Boguraev B. K.; Briscoe E. J. 1985 Towards a Dictionary Support Environment for Real Time Parsing. *Proc. ACL Second European Conference*, 171-178.

[8] Andrews A. D. 1982 The Representation of Case in Modern Icelandic In: Joan Bresnan (ed). *The Mental Representation of Grammatical Relations*. MIT Press : Cambridge, MA : 427–503.

[9] Bandyopadhyay D. (in Press, 1992) The Valency of Bangla Verb and Some related Issues. *Monograph of Institute of Linguistic Sciences*, India. Calcutta.

[10] Barnett J.; Knight K.; Mani I.; Rich E. 1990 Knowledge and Natural Language Processing. *Comm. of the ACM* **33**(8) 50–71.

[11] Barton E. G. Jr.; Berwick R. C.; Sven Ristad E. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge MA.

[12] Bates M. 1978 The Theory and Practice of ATN Grammars. In: L. Bolc ed., *Natural Language Communication with Computers* 191–259.

[13] Bear J. 1986 A Morphological Recognizer with Syntactic and Phonological Ruless. In: *COLING-86*, 272–276.

[14] Bear J. 1988 Morphology with Two-level Rules and Negative Rule Features. In: *COLING-88*, 28–31.

[15] Berwick R. C. and Fong S. 1991 Principle-Based Parsing: Natural Language Processing for the 1990s. In: *Artificial Intelligence at MIT, Vol-I* Winston and Shellard eds. 287–325.

[16] Bharti A.; Chaitanya V.; Sangal R. 1989. A Karaka Based Approach to Parsing of Indian Language Processing. *TRCS-89.* Kanpur, India.

[17] Bharti A.; Chaitanya V.; Sangal R. 1990. A Computational Grammar for Indian Language Processing. In: *Seminar on Development of Core Grammar of Indian Languages for Computers*, Central Institute of Indian Languages, Hyderabad, India.

[18] Blaberg O. 1985 A two-level description of Swedish. In: Karlsson F. Ed. *Computational Morphosyntax: Report on Research 1981-1984*, University of Helsinki, Helsinki 43–62.

[19] Blank G. D. 1985 A New Kind of Finite-state Automation: Register Vector Grammar. In: *IJCAI-85* **2** 749–755.

[20] Blank A.; Ritchie G.; Pulman S. G.; Russell G. 1987 Formalismas for Morphographemic Description. In: *ACL Proceedings, Third European Conference* 11-18.

[21] Bougarev B. K.; Carter D.; Briscoe E. J. 1987 A Multipurpose Interface to an On-line Dictionary. In: *ACL Proceedings, Third European Conference.* 63–69.

[22] Bouma G.; Konig E.; Uszkoreit 1988. A Flexible Graph-unification Formalism and its Application to Natural Language Processing. *IBM Journal of Research and Development* **32**(2), 170–184.

[23] Bresnan J. (ed) 1982 *The Mental Representation of Grammatical Relations.* MIT Press : Cambridge, MA.

[24] Bresnan J. 1982 Passive in Lexical Theory In: Joan Bresnan (ed). *The Mental Representation of Grammatical Relations.* MIT Press : Cambridge, MA : 3–86.

[25] Bresnan J. 1982 Control and Complementation In: Joan Bresnan (ed). *The Mental Representation of Grammatical Relations.* MIT Press : Cambridge, MA : 282–390.

[26] Briggs R. 1985. Knowledge Representation in Sanskrit and Artificial Intelligence. *AI Magazine* **Spring 85**. 32–39.

[27] Briscoe E. J.; Boguraev B. K. 1988 *Computational Lexicography for Natural Language Procssing.* Longman/ Wiley: London/ New York.

[28] Brodda B. 1986 BetaText: an Event Driven Text Processing and Text Analyzing System. In: *COLING-86.* 421–422.

[29] Calder J.; Klein E.; Zeevat H. 1988. Unification Categorical Grammar: a Concise Extendable Grammar for Natural Language Processing. *COLING-88.* 83–86.

[30] Calzolari N.; Picchi E. 1988. Acquisition of Semantic Information from an On-line Dictionary. In: *COLING-88.* 87–92.

[31] Carden G. 1983. The Non-finite-state-ness of the Word Formation Component. *Linguistic Inquiry* **14** 537-541.

[32] Cardona G. 1976. *Panini: A Survey of Research.* Motilal Banarasidas. Delhi, varanasi, Patna, Madras.

[33] Charniak E.; Mc Dermott D. V. 1985. *Introduction to Artificial Intelligence.* Addison- wesley: Reading, MA.

[34] Chatterjee S. K. 1986 (Reprint) *The Origin and the Development of the Bengali Language — Vols I and II* (in English). Rupa and Co., Calcutta.

[35] Chatterjee S. K. 1986 (Reprint) *Bengali Self Taught* Rupa and Co. Calcutta.

[36] Chatterjee S. K. 1988 *Bhasa Prakash Bangala Byakaran* (a Book on Bengali Grammar in Bengali). Rupa and Co., Calcutta.

[37] Chomsky N. 1957 *Syntactic Structures* Mouton, s'Gravenhage.

[38] Chomsky N. 1965 *Aspects of the Theory of Syntax* MIT Press, Cambridge, MA.

[39] Chomsky N. 1981 *Lectures on Government and Binding: the Pisa Lectures* Foris Publications: Dordrecht, Holland.

[40] Sinha R. M. K. (ed.) 1991 *Proc. CPAL-2* (Second Regional Conference on Computer Processing of Asian Languages, Kanpur, India), Tata McGraw Hill, New Delhi.

[41] Dalrymple M.; Zaenen A. 1989 Modeling Syntactic Constraints on Anaphoric Binding *Technical Paper No. P89-00142*, Systems Sciences Laboratory, XEROX PARC.

[42] Dalrymple M.; Zaenen A. 1991 Modeling Anaphoric Superiority *Technical Paper No. P91-00161*, Systems Sciences Laboratory, XEROX PARC.

[43] Dalrymple M. 1991 An LFG Account of Anaphoric Binding Constraints *Technical Paper No. P91-00052*, Systems Sciences Laboratory, XEROX PARC.

[44] Dasgupta M. 1992 Composite Verbs in Bangla *Indian Journal of Linguistics* **17** (1): 1–31.

[45] Dasgupta P. 1977 The Internal Grammar of Compound Verbs in Bangla *Indian Linguistics* **38**. 68–85.

[46] Dasgupta P. 1979 The Bangla -WA/-No Forms Participle and Gerund *Indian Linguistics* **40** (3) 185–197.

[47] Dasgupta P. 1980 *Questions and Relative and Complement Clauses in a Bangla Grammar* PhD Thesis, New York University.

[48] Dasgupta P.; Dhongde R. V.; Rajendran S. 1981 Complement Verb Reparsing in Bangla, Marathi and Tamil. *Indian Linguistics*, **42** (39–47).

[49] Dasgupta P. 1982 Phonology and the Bangla Verb. *Indian Linguistics*, **43**(1–2).

[50] Dasgupta P. 1983 On the Bangla Classifier Ta, its Penumbra and Definiteness. *Indian Linguistics* **44** (1–4). 11–26.

[51] Dasgupta P. 1985 On Bangla Nouns *Indian Linguistics* **46** (1–2). 37–66.

[52] Dasgupta P. 1990 *Kathar Kriya Karma (Bengali)* [Grammatical Functions of Verbs]. Dey's Publishers, Calcutta, India.

[53] De Smedt K. 1984 Using Object-Oriented Knowledge-representation Techniques in Morphology and Syntax Programming. In: *Proc. of the European Conference on Artificial Intelligence (ECAI-84)*. 181–184.

[54] Dey P. 1979. On Rule Ordering in Bangla Phonology *Indian Linguistics* **40** (1). 24–34.

[55] 1991. Computers and the Indian languages — Unpublished Report of the Department of Electronics, Govt. of India, New Delhi.

[56] Domenig M.; Shann P. 1986. Towards a Dedicated Database Management System for Dictionaries. In: *COLING-86*. 91–96.

[57] Dorre J.; Seiffert R. 1991 A Formalism for Natural Language – STUF. In: Herzog O.; Rollinger C. -R. (Eds) *Text Understanding in LILOG* (LNAI-546), Springer-Verlag, Berlin Heidelburg. 33–38.

[58] Dorre J. 1991 The Language of STUF. In: Herzog O.; Rollinger C. -R. (Eds) *Text Understanding in LILOG* (LNAI-546), Springer-Verlag, Berlin Heidelburg. 39–50.

[59] Dowty D. R.; Karttunen L.; Zwicky A. M. eds. 1985. *Natural Language Parsing.* Cambridge University Press: Cambridge.

[60] Early J. 1970. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* **14**. 453–460.

[61] Eisele A.; Dorre J. 1986. A Lexical Functional Grammar System in PROLOG. *COLING-86* 551-553.

[62] Ejerhed E.; Church K. W. 1983 Finite State Parsing. In: Karlsson (ed.) *Papers from the Seventh Scandinavian Conference of Linguistics.* Helsinki. 410–432.

[63] Fenstad J. E.; 1987 Natural Language Systems. In: *Advanced Topics in Artificial Intelligence, 2nd Advanced Course, ACAI, '87, Oslo, Norway* (Lecture Notes in AI Series No. 345) Springer-Verlag: 189–233.

[64] Fillmore C. J. 1968 The Case for Case. In: Bach E.; Harms R. (eds.) *Universals in Linguistic Theory*, Holt, Rinehart & Winston, New York. 1–88.

[65] Fillmore C. J. 1977 The Case for Case Reopened. In: Cole P.; Sadock J. M. (eds.) *Syntax and Semantics Vol-8 (Grammatical Relations)*, Academic Press, New York, San Francisco, London. 59–81.

[66] Flickinger D. P.; Pollard C. J.; Wasow T. 1985 Structure Sharing in Lexical representation. In: *ACL Proceedingsd, 23rd Annual Meeting.* 262–267.

[67] Frey W. 1985 Noun Phrases in Lexical Functional Grammar. In: Dahl V.; Saint-Dizier P. (eds) Natural Language Understanding and Logic Programming, Elsevier Science Publishers B. V. (North Holland), 121–137.

[68] Fukumoto F.; Sano H.; Saitoh Y. and Fukumoto J. 1991 A Framework for Restricted Dependency Grammar. In: *Proc. Third International Workshop on Natural Language Understanding and Logic Programming, Stockholm, Sweden* Brown C. G. & Koch G. eds. 61–74.

[69] Gazder G.; 1985 Review Article: Finite State Morphology *Linguistics* **23**, 597–607.

[70] Gazder G.; Pullum G. K.; Klein E. and Sag I. 1985 *Generalized Phrase Structured Grammar.* Oxford: Blackwell.

[71] Gazder G.; Mellish C. 1989. *Natural Language Processing in PROLOG.* Addison-Wesley Publishing Co. Reading, MA.

[72] Gazder G.; Mellish C. 1989. *Natural Language Processing in LISP.* Addison-Wesley Publishing Co. Reading, MA.

[73] Gorz G.; Paulus D. 1988 A Finite-state Approach to German Verb Morphology. In: *Proceedings of COLING-88 (Conference on Computational Linguistics)*: 212–215.

[74] Grosz B. J.; Jones K. S.; Webber B. L. eds. 1986 *Readings in Natural Language Processing.* Morgan Kaufmann : Los Altos, CA.

[75] Halliday M. A. K. 1976 Types of process. In: Kress G. (ed.) *Halliday: System and Function of Language.* 159–173.

[76] Halvorsen P-K. 1983 Semantics for Lexical Functional Grammars. *Linguistic Enquiry* **14**(3) 567–613.

[77] Halvorsen P-K. 1988 Situation Semantics and Semantic Interpretation in Constraint-based Grammars In: Proc. of the International Conf. on Fifth Generation Computer Systems, Tokyo Japan 471–478.

[78] Halvorsen P-K.; Kaplan R. M. 1988 Projections and Semantic Description in Lexical Functional Grammar. In: Proc. of the Int. Conf. on Fifth Generation Computer Systems (FGCS'88), Tokyo. Vol. 3, 1116–1122.

[79] Hendrix G. G.; Sacerdoti E.; Sagalowics D.; Slocum J. 1978 Developing a Natural Language Interface to Complex Data *ACM Transactions on Database Systems* **3** (2): 105–147.

[80] Herath I.; Yokoyama S.; Isahara H.; Ishizaki S. 1989 Sinhalese Morphological Analysis: A Step Towards Machine Processing of Sinhalese. *Proc. IEEE Workshop on Tools for AI – TAI-89* 100–107.

[81] Herzog O.; Rollinger C. -R. (Eds) 1991. *Text Understanding in LILOG* (LNAI-546), Springer-Verlag, Berlin Heidelburg.

[82] Hopcroft J.; Ullman J. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading MA.

[83] Iosda M.; Aiso H.; Kamibayashi N.; Matsunaga Y. 1986. Model for Lexical Knowledge Base. *COLING-86*, 451-453.

[84] Jackendoff R. 1977 X-Bar Syntax: A Study of Phrase Structure. *Linguistic Inquiry Monograph Two.* MIT Press, Cambridge MA.

[85] Joshi A. K. 1985 Tree Adjoining Grammars: How Much Context-sensitivity is Required to Provide Reasonable Structural Descriptions? In: Dowty D., Kartunnen L.; Zwicky A., eds. *Natural Language Processing: Psycholinguistics, Computational and Theoretical Properties* Cambridge University Press: Cambridge: 206–249.

[86] Kak S. C. 1987 The Paninian Approach to Natural Language Processing. *International Journal of Approximate Reasoning* **1**. 117–130.

[87] Kaplan R. M.; Kay M. 1981 Phonological Rules and Finite State Transducers *ACL/LSA (Assoc. of Computational Linguistics/ Linguistic Society of America) paper.* New York.

[88] Kaplan R. M.; Bresnan J. 1982 Lexical Fuctional Grammar : A Formal System for Grammatical Representation. In: Joan Bresnan (ed). *The Mental Representation of Grammatical Relations.* MIT Press : Cambridge, MA : 173–281

[89] Kaplan R. M.; Zaenen A. 1987 Long-distance Dependencies, Constituent Structure, and Functional Uncertainty. (to appear in) Baltin M.; Kroch A. (eds) Alternative Conceptions of Phrase Structure, Chicago Univ. Press.

[90] Kaplan R. M.; Maxwell J. T.; Zaenen A. 1987 Functional Uncertainty. In: The CSLI Monthly, Center for the Study of Language and Information, Stanford University.

[91] Kaplan R. M.; Maxwell J. 1988 An Algorithm for Functional Uncertainty. *Technical Paper No. P88-00084*, Systems Sciences Laboratory, XEROX PARC. (Also in Proc. COLING 88, Budapest 297–302)

[92] Kaplan R. M.; Maxwell J. T. 1988 Constituent Coordination in Lexical Functional Grammar. In: Proc. of COLING 88, Budapest Aug 88, 303–305. Kaplan R. M.; Zaenen A. Functional Uncertainty and Functional Precedence in Continental West Germanic. *Technical Paper No. P88-00082*, Systems Sciences Laboratory, XEROX PARC.

[93] Kaplan R. M. 1989 The Formal Architecture of Lexical Functional Grammar . *Journal of Information Sciences and Engineering* **5**. 305–322.

[94] Kaplan R. M.; Zaenen A. 1989 Functional Information and Constituent Structure in West G—ermanic Infinitival Constructions. *Technical Paper No. P89-00067*, Systems Sciences Laboratory, XEROX PARC.

[95] Kaplan R. M. 1989 The Formal Architecture of Lexical-Functional Grammar. Journal of Information Science and Engineering 5, 305–322.

[96] Kaplan R. M.; Netter K.; Wedekind J.; Zaenen A. 1989 Translation by Structural Correspondences In: Proc. of the Fourth Conference of the European chapter of the Association for Computational Linguistics. 272–281.

[97] Kaplan R. M.; Zaenen A.; 1990 Long-distance Dependencies, Constraint Structure and Functional Uncertainty In: M. Baltin, A. Kroch (eds) *Alternate Conceptions of Phrase Structure*. Chicago University Press.

[98] Karttunen L. 1983. KIMMO: a General Morphological Processor. *Texas Linguistic Forum* **22** 165-186.

[99] Kartunnen L.; Wittenburg K. 1983 A Two-Level Morphological Analysis of English. In: *Proceedings of ACL, 23rd Annual Meeting*: 217–228.

[100] Kartunnen L.; Kay M. 1985 Parsing in a Free Word Order Language. In: Dowty D. R., Kartunnen L. and Zwicky A. M. (eds). *Natural Language Parsing*, Cambridge University Press, London: 279–306.

[101] Karttunen L. 1986. D-PATR: A Development Environment for Unification-based Grammars. *COLING-86* 74-80.

[102] Kashket M. 1986 Parsing a Free Word Order Language: Warlpiri. In: *Proc. ACL* 60–66.

[103] Kasper R.; Rounds W. 1986. A Logical Semantics for Feature Structures. *ACL Proceeding, 24th Annual Meeting* 257-266.

[104] Kasper R. 1987 A Unification Method for Disjunctive Feature Descriptions. *ACL Proceedings, 25th Annual Meeting* 235-242.

[105] Katajia L.; Koskenniemi K. 1988. Finite-state Description of Semitic Morphology: A Case Study of Ancient Accadian. *COLING-88* 313-315.

[106] Kay M. 1973. The MIND System, In: Rustin R. (ed.) *Natural Language Processing*. Algorithmics Press, New York.

[107] Kay M. 1979. Functional grammar. In: Chiarello C. et al., eds. *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society* 142-158.

[108] Kay M. 1984. Functional Unification Grammar: A Formalism for Machine Translation. *COLING-84* 75–78.

[109] Kay M. 1985 Parsing in Functional Unification Grammar. In: Dowty D. R., Kartunnen L. and Zwicky A. M. (eds). *Natural Language Parsing*, Cambridge University Press, London: 251–278.

[110] Kay M. 1987 Nonconcatenative Finite-state Morphology. In: *Proceedings of ACL, Third European Conference*: 2–10.

[111] Khan R. 1983. A Two-level Morphological Analysis of Rumanian. *Texas Linguistic Forum* **22** 253-270.

[112] Khan R.; Liu J. S.; Ito T.; Shuldberg K. 1983. KIMMO User's Manual. *Texas Linguistic Forum* **22** 203-215.

[113] Kholdovich A. A. (ed) 1974 *Typology of Passive Construction: Diathesis and Voices*(in Russian). Nauka Leningrad.

[114] Klaus N.; Wedekind J.; 1986. An LFG-based Approach to Machine Translation. In: *Proc. IAI-MT 86, Saarbrucken.*

[115] Knight K.; 1989 Unification: A Multidisciplinary Survey. *ACM Computing Surveys* **21** (1): 93–124.

[116] Koskenniemi K. 1983 Two Level Model for Morphological Analysis. In: *Proceedings of IJCAI-83* (International Joint Conference on Artificial Intelligence). Karlsruhe, West Germany: 683–685.

[117] Koskenniemi K. 1984 A General Computational Model for Word-form Recognition and Production. In: *Proceedings of COLING-84* : 178–181

[118] Koskenniemi K.; Church K. W. 1988 Complexity, Two-level Morphology and Finnish. In: *Proceedings of COLING-88*: 335-340.

[119] Lindstedt J. 1987. A two-level description of Old Church Slavonic Morphology. *Scandu-Slavica* **30** 165-189.

[120] Lun S. 1983. A Two-level Morphological analysis of French. *Texas Linguistic Forum* **22** 271-278.

[121] Marcus M. P. 1978 A Computational Account of Some Constraints on Language In: Waltz D. ed. *Theoretical Issues in Natural Language Processing-2*, ACL: Urbana-Champaign.

[122] Marcus M. P. 1980. *A theory of Syntactic Recognition for Natural Language* MIT Press: Cambridge, MA.

[123] Maruyama N.; Morohashi M.; Umeda S.; Sumita E. 1988 A Japanese Sentence Analyzer *IBM Journal of Research and Development* **12** (2): 238–250.

[124] Maxwell J. T.; Kaplan R. M. 1991 A Method for Disjunctive Constraint Satisfaction. In: Tomita M. (ed) *Current Issues in Parsing Technology* Kluwer Academic Publishers, 173–190.

[125] Maxwell J. T.; Kaplan R. M. 1992 The Interface between Phrasal and Functional Constraints (submitted for consideration to) Computational Linguistics.

[126] McCord M. C. 1982 Using Slots and Modifiers in Logic Grammar for Natural Languages. *Artificial Intelligence* **18**: 327–367.

[127] Mellish C. S. 1988. Implementing Systemic Classification by Unification. *Computational Linguistics* **14** 40-51.

[128] Meya M. 1987. Morphological Analysis of Spanish for Retrieval. *Literary and Linguistic Computing* **2** 166-170.

[129] Minsky M. 1981 A Framework for Representing Knowledge. In: Haugeland J. ed. *Mind Design* MIT Press: Cambridge, 95-128.

[130] Moens M.; Steedman M. 1988 Temporal Ontology and Temporal Reference. *Computational Linguistics* **14** (2): 15–28.

[131] Mohanan K. P. 1982 Grammatical Relations and Clause Structure in Malayalam. In: Joan Bresnan (ed). *The Mental Representation of Grammatical Relations.* MIT Press : Cambridge, MA : 504–589.

[132] Moshier M. D.; Rounds W. C. A Logic for Partially Specified Data Structures. *Conference of the Fourteenth ACM Symposium on Principles of Programming Languages*, Munich, W. Germany. 156–167.

[133] Nakhimovsky A. 1988 Aspect, Aspectual Class And the Temporal Structure of Narrative. *Computational Linguistics* **14** (2): 29–43.

[134] Neidle C. 1982 Case Agreement in Russian In: Joan Bresnan (ed). *The Mental Representation of Grammatical Relations.* MIT Press : Cambridge, MA : 391–426.

[135] Panda H. R. 1992 Rule Based "Sandhi Bicched" (de-euphonization) of Bengali. *M. Tech Dissertation*, Indian Statistical Institute.

[136] Passonneau R. J. 1988 Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics* **14** (2): 44–60.

[137] Paul J. 1986 Bangla Verb Morphology: A Reconsideration. *Indian Linguistics* Vol 47(1–4): 73–79.

[138] Pereira F. C. N.; Warren D. H. D. 1980 Definite Clause Grammars for Language Analysis — a Survey of the Formalism and a Comparison with ATNs *Artificial Intelligence* **13**: 231–278.

[139] Pereira F. C. N.; Shieber S. M. 1984. The Semantics of Grammar formalism seen as Computer Languages. In: *COLING-84*. 123–129.

[140] Perrault C. R. 1984 On the Mathematical Properties of Linguistic Theories. *Computational Linguistics* Vol 10: 165–176.

[141] Petrick S. R. 1987. Parsing. In: Shapiro S. C. (ed.) *Encyclopaedia of Artificial Intelligence*. Wiley, New York. 687–696.

[142] Pollard C.; Sag I. A. 1988 *An Information-Based Approach to Syntax and Semantics: Vol 1 Fundamentals*. CSLI Lecture Notes No. 13, Chicago University Press, Chicago.

[143] Ritchie G. D. 1986. The Computational Complexity of Sentence Derivation in Functional Grammar. In: *COLING-86*. 584–586.

[144] Ritchie G. D.; Pulman S. G.; Black A. W.; Russell G. J. 1987 A Computational Framework for Lexical Description. *Computational Linguistics* Vol 13(3–4): 290–307.

[145] Robinson J. A. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM* **12**. 23–41.

[146] Sager N. 1981 *Natural Language Information Processing: A Computer Grammar of English and its Applications* Addison Wesley: Reading, MA.

[147] Sangal R.; Chaitanya V. 1987. An Intermediate Language for MAchine Translation: An Approach Based on Sanskrit Using Conceptual Graph Notation, Computer Sc. & Informatics. *Journal of the Computer Society of India* **17** (1).

[148] Schank R. C. 1980 Language and Memory *Cognitive Science* 4 (3): 243–284.

[149] Schiller A.; Steffens P. 1991 Morphological Processing in the Two-Level Paradigm. In: Herzog O.; Rollinger C. -R. (Eds) *Text Understanding in LILOG* (LNAI-546), Springer-Verlag, Berlin Heidelburg. 112–126.

[150] Seiffert R. 1991 Chart Parsing of STUF Grammars. In: Herzog O.; Rollinger C. -R. (Eds) *Text Understanding in LILOG* (LNAI-546), Springer-Verlag, Berlin Heidelburg. 51–54.

[151] 1990. *Seminar on Development of Core Grammar of Indian Languages for Computers*, Central Institute of Indian Languages, Hyderabad, India.

[152] Sengupta P.; Dutta Majumder D. 1987 Natural Language Processing and Artificial Intelligence Techniques In: *Proc. Congress of Cybernetics and Systems* New Delhi 1987.

[153] Sengupta P.; Chaudhuri B. B. 1989 A Morphological Verb and Case Analyzer for NLP of a Major Indian Language. In: *Proc. National Symposium for Natural Language Processing.* Vishakapatnam, India.

[154] Sengupta P.; Chaudhuri B. B. 1992 A Lexical Representation System for Efficient Word Form Analysis and Lexical Projection in Inflectional Languages (with Bengali as a Case Study) In: Sinha R. M. K. (ed) *Proc. CPAL* (Second Regional Conference on Computer Processing of Asian Languages, Kanpur, India), Tata McGraw Hill, New Delhi: 199–208.

[155] Sengupta P.; Chaudhuri B. B. 1993 A Morpho-Syntactic Analysis Based Lexical Sub-System *International Journal of Pattern Recognition and Artificial Intelligence* (forthcoming).

[156] Sengupta P.; Chaudhuri B. B. 1993 Natural Language Processing in anIndian Language (Bengali)–I: Verb Phrase Analysis. *IETE Technical Review* (The Institution of Electronics and Telecommunication Engineers, India) 10(1). 27–41.

[157] Sengupta P.; Chaudhuri B. B. 1993 Delayed Syntactic Encoding in LFG Parser for an Indian language — Bangla. (submitted for consideration for publication to) *Computational Linguistics.*

[158] Sengupta P.; Chaudhuri B. B. 1993 Projection of Multi-Worded Lexical Entities in an Inflectional Language. (submitted for consideration for publication to) *International Journal of Pattern Recognition and Artificial Intelligence.*

[159] Sengupta P.; Chaudhuri B. B. 1993. On Parsing a Class of "Free Word-Ordered" languages. *Proc. Third Regional Workshop on Theoretical Computer Science.* Kharagpur, India. 202–210.

[160] Sharma P. C. S.; Srinivas B. 1989 Lexical Functional Grammar for Telugu. In: *Proc. National Symposium for Natural Language Processing.* Vishakapatnam, India : 38–57.

[161] Shieber S. M. 1985 Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In: *ACL Proceedings, 23rd Annual Meeting.* 145–152.

[162] Shieber S. M. 1986 *An Introduction to Unification-Based Approaches to Grammar.* Chicago University Press, Chicago.

[163] Singh U. N. 1976 Negation in Bengali and the Order of Constituents. *Indian Linguistics* Vol 37(3): 295–303.

[164] Stall J. F. 1967 *Word Order in Sanskrit and Universal Grammar.* Dordrecht..

[165] Stall J. F. (Ed.) 1972 *A Reader on the Sanskrit Grammarians.* MIT Press.

[166] Susumu Kuno. 1987 *Functional Syntsx: Anaphora, Discourse and Empathy.* Chicago University Press, Chicago.

[167] Tomita M. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer, Boston.

[168] Trost H.; Buchberger E. and Heinz W. 1989 On the Interaction of Syntax and Semantics in a Syntactically Guided Caseframe Parser. In: *Proc. COLING BUDAPEST* Vargha D. ed. 677–682.

[169] Uszkoreit H. 1986 Categorical Unification Grammar. *COLING-86*, 187–194.

[170] Vasu S. C. 1891 *The Ashtadhyayi of Panini* (2 Vols.). Allahabad Univ. Press. Allahabad, India.

[171] 1989. *Proc. National Symposium for Natural Language Processing.* Andhra University, Vishakapatnam, India.

[172] Webber B. L. 1988 Tense as a Discourse Anaphor. *Computational Linguistics* **14** (2): 61–73.

[173] Weischedl Y. 1986 Knowledge Representation and Natural Language Processing *Proc. IEEE* **Vol 7G** (7): 905–920.

[174] Weizenbaum J. 1966 ELIZA — A Computer Program for the Study of Natural Language Communication Between Man and Machine, *Communicationas of the ACM* 9 (1): 36–45.

[175] Wilks Y. 1975 An Intelligent Analyzer and Understander of English *Communications of the ACM* **18** (5): 264–274.

[176] Wilks Y. 1978 Making Preferences More Active *Artificial Intelligence* **11**: 197–223.

[177] Winograd T. 1972 *Understanding Natural Language* Academic Press, New York.

[178] Winograd T. 1983 *Language as a Cognitive Process : Vol-I Syntax*, Addison-Wesley : Reading, MA.

[179] Winston P. H. 1984. *Artificial Intelligence.* Addison-Wesley Publishing Co., Reading, MA. (2nd-edition).

[180] Wittenburg K. 1986. A Parser for Portable NL Interfaces Using Graph-Unification-Based Grammarss. In: *Proc. AAAI-86* **2**. 1053–1058.

[181] Woods W. A. 1970 Transition Network Grammars for Natural Language Analysis. em Comm. of the ACM **13** 591–606.

[182] Woods W. A. 1975 What's in a Link: Foundations for Semantic Networks In: Bobrow D. G.; Collins A. eds. *Representation and Understanding: Studies in Cognitive Science* Academic Press, NY.

[183] Woods W. A. 1980 Cascaded ATNs. *American Journal of Computational Linguistics*

[184] Zaenen A.; Engdahl E. (To appear) Descriptive and Theoretical Syntax in the Lexicon. (to appear in) Computational Approaches to the Lexicon: Automating the Lexicon II, Atkins B. T. S.; Zampolli A. (eds) Oxford University Press.

[185] Zeevat H. 1988. Combining Categorical Grammar and Unification. In: Reyle U.; Rohrer C. (eds.) *Natural Language Parsing and Linguistic Theories*, D. Reidel, Dordrecht. 202–229.

[186] Zernik U.; Dyer M. G. 1987 The Self-Extending Phrasal Lexicon. *Computational Linguistics* **13** (3-4): 308–327.

Publication List of Probal Sengupta

i. **Sengupta P.**; Dutta Majumder D. 1987 Natural Language Processing and Artificial Intelligence Techniques In: *Proc. Congress of Cybernetics and Systems* New Delhi 1987.

ii. **Sengupta P.**; Chaudhuri B. B. 1989 A Morphological Verb and Case Analyzer for NLP of a Major Indian Language. In: *Proc. National Symposium for Natural Language Processing.* Vishakapatnam, India.

iii. **Sengupta P.**; Chaudhuri B. B. 1992 A Lexical Representation System for Efficient Word Form Analysis and Lexical Projection in Inflectional Languages (with Bengali as a Case Study) In: Sinha R. M. K. (ed) *Proc. CPAL* (Second Regional Conference on Computer Processing of Asian Languages, Kanpur, India), Tata McGraw Hill, New Delhi: 199–208.

iv. **Sengupta P.**; Chaudhuri B. B. 1993 A Morpho-Syntactic Analysis Based Lexical Sub-System *International Journal of Pattern Recognition and Artificial Intelligence* (forthcoming).

v. **Sengupta P.**; Chaudhuri B. B. 1993 Natural Language Processing in anIndian Language (Bengali)–I: Verb Phrase Analysis. *IETE Technical Review* (The Institution of Electronics and Telecommunication Engineers, India) **10**(1). 27–41.

vi. **Sengupta P.**; Chaudhuri B. B. 1993 Delayed Syntactic Encoding in LFG Parser for an Indian language — Bangla. (submitted for consideration for publication to) *Computational Linguistics.*

vii. **Sengupta P.**; Chaudhuri B. B. 1993 Projection of Multi-Worded Lexical Entities in an Inflectional Language. (submitted for consideration for publication to) *International Journal of Pattern Recognition and Artificial Intelligence.*

viii. **Sengupta P.**; Chaudhuri B. B. 1993. On Parsing a Class of "Free Word-Ordered" languages. *Proc. Third Regional Workshop on Theoretical Computer Science.* Kharagpur, India. 202–210.

Appendix A

The Bangla Character Set

Like most Indian language alphabets, the Bangla alphabet is predominantly phonetic. There are two separate sections of alphabetic symbols — the vowels and the consonants. The consonants are grouped together into well-defined phonetic classes, alongwith a sundry list of consonants that can not be grouped as easily. When vowel symbols appear with a consonant, in graphemic representation, it (the vowel) is manifested as a identifiable marker to the left, right, left and above, right and above or below the consonant. Two or more consonants may form a *consonant cluster.* A consonant cluster indicates back to back consonant utterance without intervening vowel bursts. Such a cluster behaves as a normal consonant. Every consonant cluster has a unique graphemic symbol, usually, but not always, formed by superimposing scaled down graphemes of the consonants forming the cluster.

The standard for all Indian script symbols is known as the ISCII — Indian Standard Code for Information Interchange. Specially designed PC add-on cards are available for displaying and printing ISCII texts. Such cards are known as GIST cards. We have used such a card manufactured by Mssrs. AEM Ltd., for our printing purpose. However, understanding Bangla text in Bangla print requires knowledge of the alphabet as well as knowledge of the finer points of GIST card usage, we have proposed a phonetically motivated code for the purpose. The code proposes one, two or rarely three English letter codes for Bangla vowels and consonants. Consonant cluster characters are generated by using a special code (in our case "/") for composition. The code is explained below.

The Vowels

a	অ	i	ই	u	উ	r'	ঋ	e	এ	o	ও
a'	আ	i'	ঈ	u'	ঊ			e'	ঐ	o'	ঔ

The Consonants

The Main Consonant Groups

k	ক	c	চ	t'	ট	t	ত	p	প
kh	খ	ch	ছ	t'h	ঠ	th	থ	ph	ফ
g	গ	j	জ	d'	ড	d	দ	b	ব
gh	ঘ	jh	ঝ	d'h	ঢ	dh	ধ	bh	ভ
N	ঙ	N'	ঞ	n'	ণ	n	ন	m	ম

Other Consonants

y,	য	y	য়	r	র	l	ল
d,'	ড়	d'h,	ঢ়	(Note the dots after d' and d'h)			
s'	শ	S	ষ	s	স	H	হ
M	ং	h'	ঃ	m'	ঁ		

The Consonant "k" with different Vowels

ka	ক	ki	কি	ku	কু	kr'	কৃ	ke	কে	ko	কো
ka'	কা	ki'	কী	ku'	কূ			ke'	কৈ	ko'	কৌ

A Few Consonant Clusters

k/k	ক্ক	k/m	ক্ম	k/y,	ক্য	k/r	ক্র
r/k	র্ক	k/l	ক্ল	k/B	ক্ব	k/S	ক্ষ
k/S/m	ক্ষ্ম	N'/j	ঞ্জ	j/N'	জ্ঞ	N/k	ঙ্ক
N'/c	ঞ্চ	N'/ch	ঞ্ছ	H/m	হ্ম	p/t	প্ত

To provide a clear idea of the code, we include below a coded and printed text form of the Indian National Anthem.

Janaganamana adhina'yaka jaya He	জনগনমন অধিনায়ক জয় হে
bha'rata bha'g/y.a bidha'ta'	ভারত ভাগ্য বিধাতা
pa'N'/ja'ba sin/dhu gujara't'ha ma'ra't'ha'	পাঞ্জাব সিন্ধু গুজরাঠ মারাঠা
d/ra'bid'.a ut/kala baN/ga	দ্রাবিড় উৎকল বঙ্গ
bin/dh/y.a Hima'cala y.amuna' gaN/ga'	বিন্ধ্য হিমাচল যমুনা গঙ্গা
uc/chala jaladhitaraN/ga	উচ্ছল জলধিতরঙ্গ
taba s'ubha na'me ja'ge	তব শুভ নামে জাগে
taba s'ubha a's'i'S ma'ge	তব শুভ আশীষ মাগে
ga'He taba jayaga'tha'	গাহে তব জয়গাথা
janaganamaN/galada'yaka jaya He	জনগনমঙ্গলদায়ক জয় হে
bha'rata bha'g/y.a bidha'ta'	ভারত ভাগ্য বিধাতা
jaya He jaya He jaya He	জয় হে জয় হে জয় হে

Appendix B

Some Spelling Rules and Morphologically Parsed Words

B.1 Spelling Rules

Given below are the spelling rules used in our system for Bengali.

Vowel Harmony Rules (Global):

i. `V:V + a:o`

ii. `aa':aa' + 0:o 0:y a:a`

iii. `eiuo:eiuo + 0:y a:a`

iv. `aa':aa'+ e:y`

v. `eiuo:eiuo + 0:y e:e`

Spelling Rules at VSTEM-VDEC Boundary

i. `V:V + 0:y aa'eiou:aeeiou at VSTEM, VDEC`

ii. `V:V + 0:c 0:/ ch:ch at VSTEM, VDEC`

iii. `V:V + ieu:000 nsk:nsk at VSTEM, VDEC`

iv. C:C a:e C:C + E:e at VSTEM, VDEC

v. a:e C:C + E:e at VSTEM, VDEC

vi. C:C iu:eo C:C + ae:oe at VSTEM, VDEC

vii. aa'iuo:00eo0 + e:y at VSTEM, VDEC

viii. aa'iuo:0e000 + 0:y E:e at VSTEM, VDEC

ix. a:a h:i + bc'tl:bc'tl at VSTEM, VDEC

x. a:a h:0 + oi:oi at VSTEM, VDEC

xi. a:a h:0 + e:y at VSTEM, VDEC

xii. a:a h:0 + e:e =:= at VSTEM, VDEC

xiii. a:a h:0 + ei:00 ns:ns at VSTEM, VDEC

xiv. a:e h:y + E:e at VSTEM, VDEC

xv. y.:g a':i + E:e at VSTEM, VDEC

The last rule is meant for the defective verb stem y.a' "go".

Spelling Rules at VCAUS-VDEC Boundary

i. a':i + i:e y:0 a':0 at VCAUS, VDEC

ii. a:a + a':o 0:y 0:a' at VCAUS, VDEC

iii. u:0 + a':o 0:y 0:a' at VCAUS, VDEC

iv. a':a' + e:0 n:n at VCAUS, VDEC

v. a':a' + e:y at VCAUS, VDEC

vi. a':a' + 0:c 0:/ ch:ch at VCAUS, VDEC

B.2 Some Examples of Morphological Parsing of Verbs

Input Word: **kari**
Lexical Form: **kar + i**
Category: VERB
F-Structure:

```
[PRED      do    . ]
[TENSE     PRESENT]
[GNPH      1p      ]
```

Input Word: **kara'i**
Lexical Form: **kar + a' + i**
Category: VERB
F-Structure:

```
[PRED      do      ]
[CAUS      1       ]
[TENSE     PRESENT]
[GNPH      1p      ]
```

Input Word: **pa'n**
Lexical Form: **pa' + en**
Category: VERB
F-structure:

```
[PRED      get     ]
[TENSE     PRESENT]
[GNPH      2/3p-2h]
```

Input Word: **pa'c/chilen**
Lexical Form: **pa' + chilen**
Category: VERB
F-structure:

```
[PRED      get       ]
[TENSE     PAST-PROG]
[GNPH      2/3p-2h   ]
```

Input Word: **pa'oa'c/chilen**
Lexical Form: **pa' + a' + chilen**
Category: VERB
F-structure:

```
[PRED      get        ]
[CAUS      1          ]
[TENSE     PAST-PROG]
[GNPH      2/3p-2h    ]
```

Input Word: **s'uc/chilen**
Lexical Form: **s'u + chilen**
Category: VERB
F-structure:

```
[PRED      lay        ]
[TENSE     PAST-PROG]
[GNPH      2/3p-2h    ]
```

Input Word: **s'oa'c/chilen**
Lexical Form: **s'u + a' + chilen**
Category: VERB
F-structure:

```
[PRED      lay        ]
[CAUS      1          ]
[TENSE     PAST-PROG]
[GNPH      2/3p-2h    ]
```

Input Word: **hayechili**
Lexical Form: **ha + Echili**
Category: VERB
F-Structure:

```
[PRED      be         ]
[TENSE     PAST-PERF]
[GNPH      2p-0h      ]
```

```
Input Word: {\bf kheyechila'm} \\
```

```
Lexical Form: {\bf kha'} + {\bf Echila'm} \\
Category: VERB\\
F-structure:\begin{verbatim}
[PRED      eat       ]
[TENSE     PAST-PERF]
[GNPH      1p        ]
```

Input Word: **kha'iyechila'm**
Lexical Form: **kha' + a' + Echila'm**
Category: VERB
F-structure:

```
[PRED      eat       ]
[CAUS      1         ]
[TENSE     PAST-PERF]
[GNPH      1p        ]
```

Input Word: **bhegechen**
Lexical Form: **bha'g + Echen**
Category: VERB
F-structure:

```
[PRED      flee      ]
[TENSE     PERF-PRES]
[GNPH      2/3p-2h   ]
```

Input Word: **pod',e**
Lexical Form: **pud'. + e**
Category: VERB
F-structure:

```
[PRED      burn      ]
[TENSE     PRESENT   ]
[GNPH      3p-1h     ]
```

Input Word: **pod',a'y**
Lexical Form: **pud'. + a' + e**
Category: VERB
F-structure:

```
[PRED      burn    ]
[CAUS      1       ]
[TENSE     PRESENT ]
[GNPH      3p-1h   ]
```

Input Word: **ca'iten**
Lexical Form: **ca'h + ten**
Category: VERB
F-structure:

```
[PRED      want    ]
[TENSE     HABiT   ]
[GNPH      2/3p-2h]
```

Input Word: **ga'o**
Lexical Form: **ga'h + a**
Category: VERB
F-structure:

```
[PRED      sing    ]
[TENSE     PRESENT]
[GNPH      2p-1h   ]
```

Input Word: **diye**
Lexical Form: **di + E**
F-structure:

```
[PRED      give    ]
[TENSE     CONTINF]
```

Input Word: **diiye**
Lexical Form: **di + a' + E**
F-structure:

```
[PRED      give    ]
[CAUS      1       ]
[TENSE     CONTINF]
```

Input Word: **y.a'c/chilen**
Lexical Form: **y.a'** + **chilen**
Category: VERB
F-structure:

```
[PRED      go          ]
[TENSE     PAST-PROG]
[GNPH      2/3p-2h     ]
```

Input Word: **giyechilen**
Lexical Form: **y.a'** + **Echilen**
Category: VERB
F-structure:

```
[PRED      go          ]
[TENSE     PAST-PERF]
[GNPH      2-3pHon     ]
```

Appendix C

The Unification Function

First we talk about unification of two `GLFGObjects`. Clearly, two unifiable objects must be of identical type. Thus, for example, an `AtomObject` can not be unified with an `FStructObject`. We propose a overloadable member function `unify` for `GLFGObject`. The return value of the function is a pointer to `GLFGObject`. Thus we have:

```
virtual GLFGObject *unify(GLFGObject)
```

as a top level member function of `GLFGObject`.

Every sub-class of `GLFGObject` overloads the above function to reflect the unified objects. Non-unifiable objects return NULL. Note that the unified object is *entirely new* `GLFGObject`, which is neither same (semantically) as the `this` object or the parameter `GLFGObject`. It is thus the duty of the caller function to retrieve the unified object and destroy the objects unified.

Top level access to the unification mechanism is through a global function described in the following header:

```
int unify(Holder *h1, Holder *h2);
```

We now provide a pseudo-code for the above function.

```
int unify(Holder *h1, Holder *h2)
{
  if (p1 == p2) return TRUE;
  GLFGObject *o1 = object pointed by h1,
             *o2 = object pointed by h2;
  if (o1==o2) return TRUE;
```

```
// If pointed objects of two holders are same, they are already unified.

if (h1 is free, h2 free or holds something) {
  destroy the HolderList member of h1;
  make the holderList member of h1 same as that of h2;
  add h1 in HolderList member of h2;
  make h1 point to what ever h2 points to;
  return TRUE;
}
else if (h2 is free, h1 is holds something) {
  destroy the HolderList member of h2;
  make the holderList member of h2 same as that of h1;
  add h2 in HolderList member of h1;
  make h2 point to what ever h1 points to;
  return TRUE;
}
else { // Both h1 and h2 holds something
  GLFGObject *g = (held object of h1)->(held object of h2);
  if (!g) { // the held objects are not unifiable
    return FALSE;
  }
  // g points to unified object
  destroy held object of h1;
  make held object of h1 to be g;

    destroy held objects of all holders contained in
      HolderList collection of h2;
    make held objects of all holders contained in
      HolderList collection of h2 to be g;

    merge to HolderList objects of h1 and h2;

    return TRUE;
  }
}
```