

A Computational Linguistic Analysis of Bangla

by

Zeeshan Rahman Khan

Submitted to the Department of Electrical Engineering and
Computer Science

in partial fulfillment of the requirements for the degrees of

Master of Engineering

and

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

© Zeeshan Rahman Khan. MCMXCV. All rights reserved.

The author hereby grants to MIT permission to reproduce and
distribute publicly paper and electronic copies of this thesis
document in whole or in part, and to grant others the right to do so.

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

AUG 10 1995

Barker Eng

Author

LIBRARIES

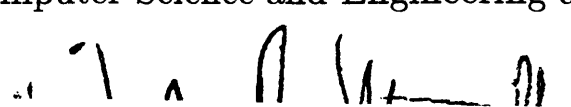
Department of Electrical Engineering and Computer Science

May 26, 1995


Certified by

Robert C. Berwick

Professor of Computer Science and Engineering and Computational
Linguistics

 Thesis Supervisor

Accepted by

 Frederic R. Morgenthaler

Chairman, Departmental Committee on Graduate Theses

A Computational Linguistic Analysis of Bangla

by

Zeeshan Rahman Khan

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 1995. in partial fulfillment of the
requirements for the degrees of
Master of Engineering
and
Bachelor of Science in Computer Science and Engineering

Abstract

The 'Government and Binding'(GB) framework developed by Noam Chomsky and others describes the grammars of different human languages in terms of universal, atomic principles and language specific parameters. In this thesis, I describe the design and implementation of a parser that analyzes a range of Bangla sentences according to these universal principles. The system was built on the Pappi principles-and-parameters interface system, and it successfully handles leftward and rightward scrambling, anaphor binding, quantifier raising, and clausal extraction. I discuss the issues and difficulties faced in implementing such a parser, comparing it against other possible implementations. I also discuss the potential for using it as the front-end for a Bangla-English translator.

Thesis Supervisor: Robert C. Berwick

Title: Professor of Computer Science and Engineering and Computational Linguistics

Acknowledgments

I would like to express my gratitude to my supervisor Professor Robert Berwick for giving me the opportunity to explore the computational linguistic phenomena of my native language Bangla. I would also like to thank Sandiway Fong for helping me with the implementation on his Pappi development system. I would like to specially thank my wife Fahria for putting up with my late hours at the AI Lab. And last but not least I would like to thank my father, mother and brother, without whose encouragement I would never had made it through five years at MIT.

Contents

1	Introduction	8
1.1	The Principles and Parameters framework	9
1.2	‘Government and Binding Theory’	11
1.2.1	X-bar theory	11
1.2.2	Case Theory	13
1.2.3	Movement	13
1.2.4	Binding Theory	14
1.2.5	Other Theories	15
2	Implementation	17
2.1	P & P parsers	17
2.1.1	Pappi	18
2.1.2	Other parsing approaches	18
2.2	Bangla in the p & p framework	20
2.2.1	Parameters	20
2.2.2	Lexicon	26
3	Bangla	27
3.1	Bangla linguistic phenomena	27
3.1.1	Scrambling	27
3.1.2	Quantifier Raising	36
3.1.3	Clausal Extraposition	37
3.1.4	Semantic Filtering	39

3.1.5	Questions	40
4	Conclusion	42
4.1	Remaining Problems and Limitations	42
4.2	Future Work	43
A	Tables	45
B	Figures	47
C	Code	49
C.1	parametersBangla.pl	49
C.2	lexiconBangla.pl	51
C.3	peripheryBangla.pl	62

List of Figures

1-1	Scrambled sentences	8
1-2	Phrase Structure Representation	12
2-1	The pappi interface	19
2-2	Non-configurational parses	21
2-3	Case assignment for scrambled objects	23
2-4	Pro drop example	24
2-5	Wh-question example	24
2-6	Preposition stranding example	25
3-1	VP internal scrambling	28
3-2	Medium distance scrambling	29
3-3	“Bob John-er maa-ke taake dekhalo”	31
3-4	“John-er maa taake bhalobashe”	32
3-5	Rightward Scrambling	34
3-6	Anaphor Binding	35
3-7	No weak-crossover effect in scrambling	36
3-8	Implementation of Quantifier Raising in Bangla	38
3-9	Clausal Extraposition with overt complementizer	40
3-10	Clausal Extraposition with empty complementizer	41
3-11	Wh-questions and Binding	41
B-1	The Bangla Font implementation	48

List of Tables

2.1	GB Parameter Settings for Bangla, Japanese, English, French and Dutch	21
A.1	Run-time for parses:simple scrambling examples	45
A.2	Run-time for parses:general examples	46
A.3	Run-time for parses:detailed scrambling examples	46

Chapter 1

Introduction

The word order of sentences in Bangla or Hindi is more flexible than corresponding sentences in English, Japanese or Korean. For example, in English, there is not much flexibility in the word-order of the sentence 'She loves Karim'. But in Bangla, all six permutations of the three words of the corresponding sentence 'shey karim-ke bhalobashe' are acceptable. This thesis describes a parser implementation that can correctly and efficiently identify and parse this kind of free word order or scrambling in Bangla sentences. Figure 1-1 is a preview of our Bangla system, and it shows parses of three of those six derivations of the basic Bangla sentence.

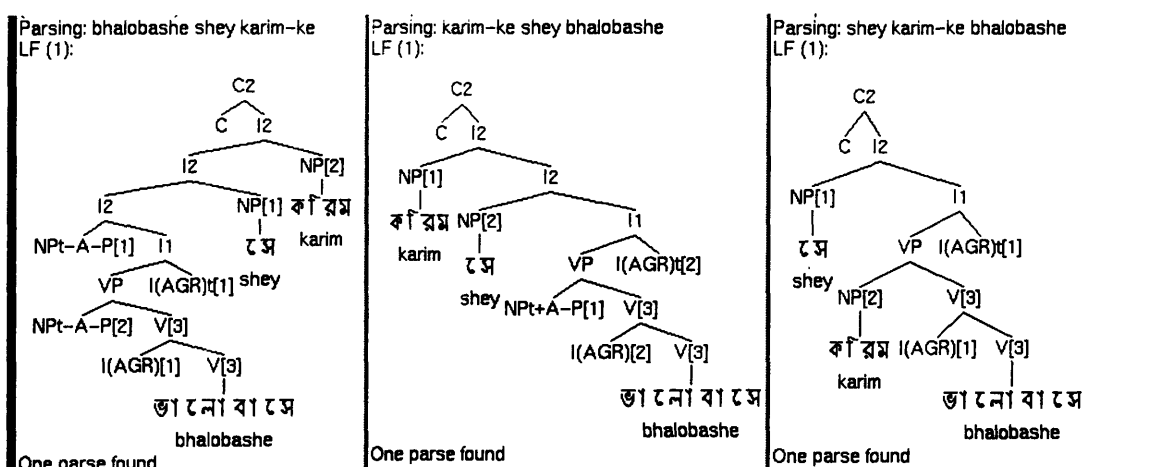


Figure 1-1: Scrambled sentences

The parser described in this thesis incorporates Bangla into the recent linguistics

trend where different human language grammars are described in terms of just a few universal principles and language-specific parameters. It accomplishes this by outlining a parametrized description of Bangla and then using that description to parse Bangla sentences. A detailed specification of grammar rules is not necessary, but with only a few parameter settings, simple Bangla sentences can be parsed. A few extra rules are needed to handle free word order. This is because we have to ensure the correct parsing of sentences that involve the interaction of scrambling with other components of grammar. To understand these interactions and more specifically how the Bangla parser implementation successfully handles interactions with different syntactic binding conditions, I shall start out by describing in this chapter the ‘Government and Binding’ theory. This is the theory that specifies the different grammar components in the principles-and-parameters (p & p) framework.

In Chapter two of this thesis, I will go over the p & p approach from a computer implementation point of view and show how it is implemented in Sandiway Fong’s ‘Pappi’[Fon94] system. In this chapter, I will also give an overview of the parameter specifications for Bangla and compare these parameters with the settings for other languages such as French and Japanese. In Chapter Three I will describe specific Bangla syntactic phenomena, and explain how my system handles sentences with these kinds of phenomena. Chapter Four concludes by outlining the prospects of implementing a translator based on the Bangla parser.

1.1 The Principles and Parameters framework

When we examine sentences from different languages, at first glance it might seem that there is a vast difference between each of these languages. But, for simple sentences from different languages, other than using a different set of words, the main differences seems to be the word order:

- (1) (i) **Bangla:** (Subject-Indirect Obj-Direct Obj-Verb)

Hasina Karim-ke ek-ti boi dilo
Hasina Karim-dat one book gave

“Hasina gave Karim a book.”

(ii) **Hindi:** (Subject-Obj-Verb)

raam-ne kelaa khayaa
raam banana ate

Ram ate a banana.

(iii) **Korean:** (Subject-Indirect Obj-Direct Obj-Verb)

Sunhee-ka Youlee-eykey [chayk hankwen]-ul senmwulhayssta
Sunhee-nom Youlee-dat [book one-volume]-acc gave-a-present

“Sunhee gave Youlee a book as a present.”

(iv) **Japanese:** (Subject-Indirect Obj-Direct Obj-Verb)

Mearii-ga taroo-ni sono hon-o watashita
Mary-nom Taroo-DAT that book-ACC handed

“Mary handed that book to Taroo.”

(v) **Dutch:** (Verb second in matrix clause, but verb final in embedded clauses)

- Hilde verslaat Adje
Hilde defeats Adje
Hilde defeats Adje.

- Ik weet dat Janneke de auto probeert te naderen
I know that Janneke the car tries approach
I know that Joanne tries to approach the car.

(vi) **French** (Subject-object-verb)

Il lit le livre
He reads the book

He reads the book

English and French sentences have the subject-verb-object order, while languages like Bangla and Japanese exhibit subject-object-verb order. We can see that if we know what each word means in a new language and if we know the word-order in that language, we should be able to understand and compose the simplest sentences in the language. In other words, we can describe the grammars for simple sentences in

different languages just by describing a general principle: “sentences contain subjects, objects and verbs”. Of course, we also need to specify the parameter ‘word-order’ for a particular language - whether the order of the sentence elements specified in the principle is ‘subject-object-verb’ or ‘subject-verb-object’, etc.

This is the main idea behind the principle-and-parameters framework. The logical next step is to ask whether this approach can be extended to cover more and more complicated sentences of different languages. The ‘Government and Binding’(GB) theory, developed by Chomsky and others, answers the question by allowing the analysis of a wide range of sentences from different languages using relatively few principles and parameters.

1.2 ‘Government and Binding Theory’

Government and Binding (GB) theory[Cho81a, Cho81b] describes the knowledge of language grammars as an interlocking set of subtheories, consisting of a universal component and a language-specific component. The universal component contains principles that are shared among all languages in the world. The language-specific component consists of a lexicon and a set of parameter settings; so the difference between language grammars stem from parametric settings of universal principles within highly constrained limits.

Here I will not go into many details of the GB theory. Rather, I shall briefly survey the subtheories on different universal principles, which are relevant to understanding the analysis of Bangla.

1.2.1 X-bar theory

X-bar theory describes how the syntactic structure of a sentence is hierarchically formed by successively smaller units called phrases. In natural languages, every phrase contains a head word. The head of a noun phrase(NP) is a noun(N), and the head of a verb phrase(VP) is a verb(V). The head is a single word that determines the main characteristics of the phrase and how the phrase as a whole can be use. For

example, in the NP ‘the big tower by the Thames’, the head word is ‘tower’. X-bar theory outlines the universal constraint that all phrases are headed. The associated parameters are whether a language is head-final or head-initial. For example, since Japanese verb phrases end with a verb, it is a head-final language, but English is a head-initial language. The X-bar theory states that phrases must adhere to the following schema in their structure:

1. $XP \rightarrow YP Xbar$ where YP is called the **specifier** of X.
2. $Xbar \rightarrow Xbar YP$ where YP is called an **adjunct** of X
3. $Xbar \rightarrow X YP_1 \dots YP_k$ where YP_i 's are called the **complements** of X

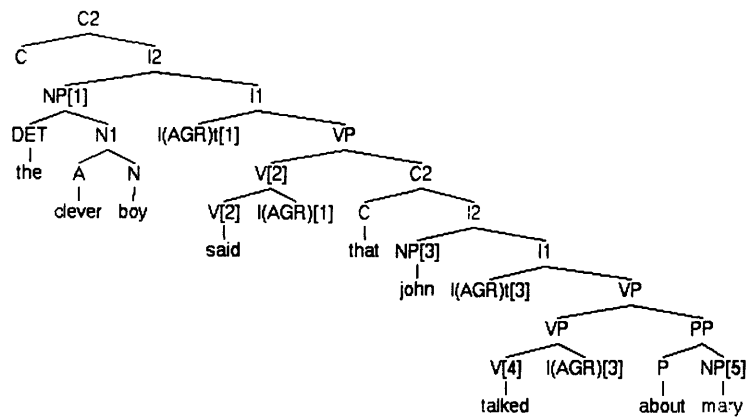


Figure 1-2: Phrase Structure Representation

For example, in the phrase-structure representation for the sentence ‘The clever boy said that John talked about Mary’ in Fig 1-2, the *determiner* ‘the’ and the *adjective* ‘clever’ are specifiers in the first noun-phase. The *prepositional phrase* ‘about Mary’ is an adjunct to the inner verb phrase which has the head ‘talked’. The clause marked by ‘C2’ – ‘that John talked about Mary’ is the complement of the verb ‘said’. A clause consists of a *complementizer*, such as, ‘that’ followed by a sentence ‘I2’. The head of I2 is the node marked as ‘I’ in the parse-tree. ‘I’ represents an inflectional element which indicates the tense or verbal inflection aspects of a sentence. There are two types of the inflectional phrases(IP) marked as ‘I2’ here. A finite IP is headed

by an 'I' with tense, while an infinite IP is 'tenseless'. An example of an infinite IP is the infinitive 'to go' in "I don't want to go". In GB theory, the sentence subject is considered to be the specifier of 'I', but for infinite IPs it is postulated that an empty category 'PRO' occupies the subject position. GB theory differentiates between the specifier, complements and other positions in a parse-tree. The specifier of IP and the verb-complements are known as **A-positions**(argument positions) while all other positions are **A'-positions**(non-argument positions).

1.2.2 Case Theory

A case is an attribute of a noun or pronoun that indicates the type of position occupied by it. In English, the four cases are *Nominative* for subjects, *Accusative* for objects, *Genitive* for nouns expressing possession, and *Oblique* for complements of prepositions. There might be other cases in other languages, for example, in Japanese, the *Dative* case is used to express indirect objects. Case can be morphologically manifested, e.g. in English, from the form of the noun 'John's', it is obvious that it has genitive case. But other cases are assigned structurally by case assigners, such as active verbs, and prepositions. The Case Theory requires that every overt NP be assigned an abstract case. The Case Filter rules out sentences containing an NP with no case.

1.2.3 Movement

1. I read the book.
2. The book was read by me.

For sentences as above where the meaning is essentially the same, but the surface structure is different, GB theory exposes the similarity through an underlying structure called the **D-structure**. In the underlying D-structure¹ for the second sentence above, 'the book' is in the complement position of the verb 'was read'. It moves to the subject position from its base position so that it can be assigned 'nominative' case.

¹[*IP* e [*I*' was [*VP* read [*NP* the book] by me]]]

This is because the passive verb ‘was read’ cannot assign case to its complement. In general, in GB theory the rule **Move- α** specifies that any element can be moved anywhere. Whether a particular movement is allowed depends on other constraints in the grammar. One of these constraints is the Structure Preserving Principle: ‘The result of a movement must satisfy the X-bar schema’. When a head moves out of its base position, it leaves an empty element, a *trace*, behind. GB theory specifies a special syntactic relationship that must hold between the moved head(*the antecedent*) and the trace it left behind: the trace must be ‘governed’ by the antecedent.

There are two common types of movements: WH-movements and NP-movements[LD94].

1. Who₁ does Kim like t₁?²
2. Kim₁ was defeated t₁.

In the first sentence, a WH-question, the initial trace is case-marked, while in the passive construction of the second sentence, the initial trace is not case-marked. ‘Kim’ had to move to the subject position to get case. The first sentence is an instance of WH-movement while the second one is an instance of NP-movement. In WH-movement, a wh-element moves to an A’-position, which is an instance of A’-movement, while NP-movement is an instance of A-movement.

1.2.4 Binding Theory

In the example sentences above there are coreference relationships between traces and NPs. Binding theory is concerned with this kind of coreference relationship of Noun Phrases. Binding is a special structural relationship in a parse tree for a sentence. And the principles of binding apply to a specific part in a parse-tree - the ‘binding domain’ or ‘the governing category’. The three principles outlining binding conditions for NPs are as follows:

1. **Condition A:** An anaphor³ must be bound in its governing category.

²Here t stands for trace and the subscript shows the co-reference relationship. In this case ‘who’ and the person Kim likes, stands for the same person.

³The term anaphor covers reflexives, such as ‘himself’, ‘herself’ and reciprocals, such as ‘each other’

2. **Condition B:** A pronoun must be free in its governing category.
3. **Condition C:** An R-expression⁴ must be free everywhere.

1.2.5 Other Theories

Other than these three theories that are directly pertinent to understanding the Bangla implementation, I will just briefly mention some of the other subtheories of GB theory[vdA93].

1. The *Projection Principle* requires that all the levels of syntax should observe the specifications for each lexical item given in its entry in the lexicon.
2. *Bounding Theory* prevents the relationship of movement from extending too far in the sentence.
3. *Control Theory* deals with the subject of infinitival clauses, i.e, the properties of 'PRO's.
4. *θ -theory* deals with the assignment of semantic-roles (θ -roles) to elements in the sentence
5. *The Phonetic Form (PF) Component* interprets the surface-structure to represent is as sounds
6. *The Logical Form (LF) Component* represents the sentence as syntactic meaning, one aspect of semantic representation

In general, every principle included in the theories above makes a statement about the (un) grammaticality of a sentence. The interaction of all these principles gives an overall judgement of the grammaticality of a sentence. If these constraints succeed in assigning a legal structure to the sentence, the sentence is considered grammatical; otherwise, it is considered somehow deviant. A sentence that can be assigned more

⁴Referential(R) expressions are all the other NPs except for anaphors and pronominals that select a referent from the universe of discourse[Hae91],e.g, Kim, the boy.

than one structural representation fulfilling all the requirements set by the principles, is considered syntactically ambiguous.

Chapter 2

Implementation

Now that we know how GB theory and the principle-and-parameters framework works, the next question is how this framework can be used to build parsers for natural languages. We also need to know how Bangla fits in the p & p framework so that the Bangla language can be implemented in a principles-and-parameters based parser. This chapter addresses these two questions.

2.1 P & P parsers

Currently there are few parser implementations available that have been built around the principles-and-parameters framework. Among the the most notable are Dekang Lin's *Principar*, Bonnie Dorr's *Unitran*, and Sandiway Fong's *Pappi*. *Unitran* is a p & p based translation system implemented in LISP, while *Principar* is implemented in C++ and based on a message-passing algorithm. *Pappi*, implemented in Prolog, is the most easily extensible of these three systems, with over eight languages already implemented on the system¹. Its prolog implementation of the GB theory principles is the closest among the three parsers to a plain English description of the GB principles. Also it comes with a user-friendly customizable interface. Although a potential future goal of my work is machine translation of Bangla, because of the modular nature of

¹As of now, Japanese, Korean, Dutch, US English, Hindi, Bangla, Spanish, French, and German have been implemented on Pappi

the principles in Pappi, and the accumulated experience in implementing different languages on the system, I opted for Pappi as the platform to implement the Bangla parser.

2.1.1 Pappi

Pappi can be viewed as a kind of direct translation of GB Theory into a principle and parameter based parser. The parser is *true* to the principles and parameters approach of the underlying theory[vdA93]. This is evident from figure 2-1 which shows the interface to the Pappi system. The input is typed in the space on top, and a tree representation is printed out in the output panel. On the left panel, we can see a number of principles, which are applied to derive the output tree.

Pappi uses the generate-and-test approach in parsing. The surface-structure candidates for input sentences are generated in accordance with the X-bar theory (*generate*). Then all the linguistic constraints as described in the GB principles filter out invalid parses (*test*). Pappi is implemented in the PROLOG language, and to ensure that the parser parses grammatical sentences, and filters out ungrammatical ones, the GB theory has to be implemented in an accurate and unambiguous way.

2.1.2 Other parsing approaches

We could ask the question, is there a better way of implementing a parser other than the p& p approach? Among the other approaches to parsing human languages, the main one is a rule-based approach. In this approach usually a ‘context-free grammar’ is written to handle different types of sentences[C.B94]. This grammar would include a different set of rules for different types of sentences, such as, passive constructions, scrambling and wh-phrases. For simple scrambling between subject and verbal complement, the following kinds of rules will be needed:

1. Sentence \longrightarrow Subject NP + Verb + Object NP
2. Sentence \longrightarrow Subject NP + Object NP + VP

Principles and Parameters Parser

Examples ...

nije-ke karim bhalobashe
 Bob john-ke oi bhari chithi-ta dilo
 bhalobashe

Run Language Theory Parsers History Options

Parsing: bhalobashe
 LF (1):

One parse found
 Parsing: Bob john-ke oi bhari chithi-ta dilo
 LF (1):

One parse found

New Tree Layout option settings are now in effect

Info ...
 Demo ...

Filters

2	Theta Criterion
2	D-structure Theta Condition
4	Subjacency
m	Wh-movement in Syntax
i	S-bar Deletion
i	Case Filter
8	Case Condition on ECs
2	Coindex Subject
2	Condition A
4	Condition B
4	Condition C
3	ECP
2	Control
2	License Clitics
4	License Object pro
2	ECP at LF
2	Fi: License operator/variables
2	Fi: Quantifier Scoping
2	Fi: Reanalyze Bound Proforms
2	License Clausal Arguments
i	License Syntactic Adjuncts
2	Wh Comp Requirement
2	Semantic Restrictions

Generators

1	Parse PF
3	Parse S-Structure
11	Assign Theta-Roles
104	Inherent Case Assignment
104	Assign Structural Case
104	Trace Theory
104	Functional Determination
4	Free Indexation
4	Expletive Linking
8	LF Movement

Figure 2-1: The pappi interface

3. Sentence \rightarrow Object NP + Subject NP + VP

So we can see a huge amount of very specific language-particular systems of rules are needed to parse different kinds of sentences. Also, no knowledge of linguistics is reflected in the parsing systems. On the other hand, the p & p approach takes advantage of the universality of human language structures. Rather than viewing the grammar as a large set of ad hoc language specific rules, it views the grammar as a modular system of principles. The principles in the p & p approach are compiled into language-specific rules at a lower level. So, this approach abstracts away from unnecessary details when we are specifying a new language. We need to be aware of the generated rules only for efficiency concerns. For example, in a p & p parser like Pappi, a large number of illicit structures have to be generated to be filtered out by the filter principles. When we design the permutation mechanism for scrambling in Bangla we will have to make sure that we do not have the problem of *over-generation*[BF92].

2.2 Bangla in the p & p framework

Before I explain my implementation of Bangla *scrambling*, I will describe the overall Bangla implementation. To incorporate Bangla into the p & p framework of Pappi, other than specifying relevant parameter settings, I needed to build a lexicon with Bangla lexical entries, and make enhancements to the universal principles in Pappi to handle phenomena specific to Bangla.

2.2.1 Parameters

Table 2.1 shows parameter settings for Bangla and compares the settings to those of Japanese, U.S. English, French, and Dutch.

The reasoning and explanation of these parameter settings is outlined below.

Head position

Bangla is a SOV language, i.e, the default word order in Bangla sentences is subject-object-verb. Sengupta[Sen90] establishes this by first proving the parses of Bangla

Table 2.1: GB Parameter Settings for Bangla, Japanese, English, French and Dutch

Parameters	Bangla	Japanese	US English	French	Dutch
Head-Initial	Only c_i	No	Yes	Yes	Except v, i, neg
Head-Final	Except c_i	Yes	No	No	Only v, i, neg
Spec-Initial	Yes	Yes	Yes	Yes	Yes
Agreement	weak	strong	weak	strong	strong
Bounding Nodes	i2,np	i2,np	i2,np	i2,np	i2,np
Case Adjacency	no	no	yes	yes	no
Pro Drop	yes	yes	no	yes	no
Anaphor Drop	no	no	no	yes	no
Null Case Markers	yes	yes	no	no	no
Allow Stranding	no	no	yes	no	no
Wh in Syntax	no	no	yes	yes	yes
Clitics	no	no	no	yes	yes

sentences to be not flat as in Figure 2-2 but configurational or tree-structured as in Figure 1-1. He shows that among all the different scrambled permutations of Bangla sentences only one is base-generated and the other parse-trees are created from the *deep structure* representation by applying the move- α rule²[Sai85].

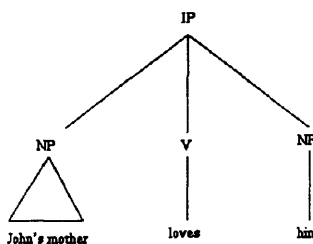


Figure 2-2: Non-configurational parses

We pick out the SOV order as the base-order and the other ones as moved/scrambled structures because this way we can capture the most common types of sentences as the principle structure, and the others as derived ones. This assignment fits with the universal descriptions of different languages. Asian languages with similar syntactical

²ref: described in chapter 1.2.3

properties as Bangla, such as, Japanese, Korean, and Hindi are all Head-Final SOV languages. More importantly, making head-final the base-case and other examples derived-cases, facilitates the parsing of Bangla sentences despite the constraints imposed by other GB principles. We have to remember that at the base-position in the D-structure, different binding conditions and GB principles must hold. For this reason, we cannot take the object-subject-verb ordering as the default because in this configuration the verb cannot govern its complement. the object. The subject in the middle blocks the governing relationship.

We could have probably adopted a description of Bangla grammar that closely reflected that of English. In this representation, sentences like ‘shey bhalobashe karim-ke’ with subject-verb-object ordering would be considered base-generated and the parameter Head-Initial would be set to true, so that verbs heading verb-phrases would begin VPs. But if we do this, there would be no uniformity in the way we treat different sentence elements. In Bangla prepositional phrases, such as, ‘tomar jonno’³ the prepositions follow their complements. So PPs are compulsorily Head-Final. With all these reasons in mind, I set the Head-Final parameter to be true for Bangla. But as we shall see in the next chapter we can not ensure total uniformity. In the case of complementizers (‘c’) in clausal constructions, ‘c’ has to be head-initial.

Specifier Position

Most languages are spec-initial as shown in the table above. Although Bangla subjects(specifier of IP) can appear in any position in scrambled sentences, the concern of universality led me to specify the position of Bangla specifiers as ‘initial’. This was done also to preserve uniformity. In Bangla noun phrases the determiners precede the noun. For example, in the noun-phrase ‘oi boi’ (that book)⁴ the determiner(a specifier) precedes the noun(the head, in this case).

³‘tomar jonno’ = ‘for you’. Preposition: ‘jonno’ = for; Complement: ‘tomar’=your

⁴oi = that, boi = book

Agreement

Unlike Japanese, French and Dutch, but similar to English, Bangla has weak agreement between the verb and the subject NP. This stems from the fact that only two parameters affect subject-verb agreement in Bangla: person and status[Kla81]. Bangla has auxiliary verbs, just as in English, and the movement of lexical verbs is not allowed in Bangla, as in Dutch[Hae91, page 602].

Case Adjacency

An NP does not have to be adjacent to a case assigner in a Bangla sentence. This is in general true for scrambling languages. Figure 2-3 shows an example scenario when object NPs are scrambled out of the VP. In the sentence ‘dilo bob john-ke boi-ta’ meaning ‘Bob gave John the book’, the subject Bob is in between the case assigner verb ‘dilo’ and the two objects. But the object NPs are still assigned case by the verb.

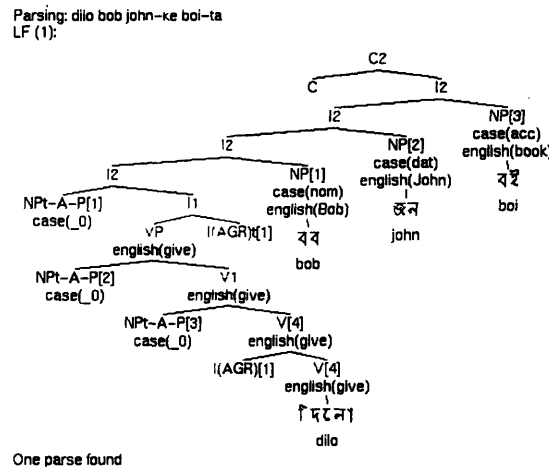


Figure 2-3: Case assignment for scrambled objects

Pro-Drop

In Bangla, a pronominal subject or object might be left unexpressed depending on the context of utterance. For example, in response to the question, ‘Has he brought

it?', one can answer only 'eneche' which means 'brought'. English is not a pro-drop language and so just 'brought' is not a valid sentence, but in pro-drop languages like Japanese, Korean and Bangla. the empty category pro⁵ occupies the positions of the unexpressed NPs[Hae91. page 455]. This is why as shown in Figure 2-4, 'eneche' is an acceptable sentence in Bangla.

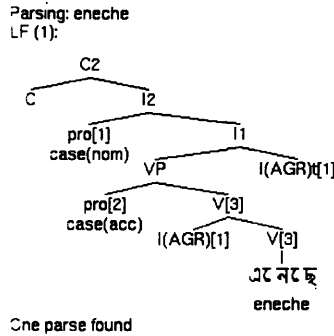


Figure 2-4: Pro drop example

Wh in Syntax

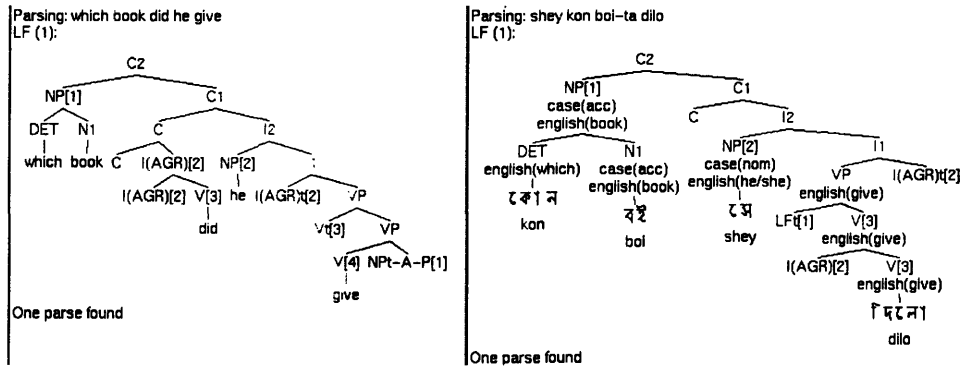


Figure 2-5: Wh-question example

Figure 2-5 compares English wh-questions with Bangla wh-questions. In English, at the surface structure level, the wh-question word has to move to the beginning of the sentence. But in Bangla and Japanese, in the surface structure, the wh-question

⁵as explained in section 1.2.1

word can be situated at the base-position. Only at the logical form level, discussed in section 1.2.5. does the question word have to move to the beginning of the sentence. This is because Bangla is a wh-in-situ language [Hae91, pages 501-503].

Stranding

In English wh-questions, a wh-phrase can move out of a prepositional phrase leaving the head of the PP behind. In this kind of *preposition-stranding*, case assignment has to look at the other end of the antecedent-trace chain to assign (oblique) case. This is shown in Figure 2-6. But preposition-stranding is not allowed in Bangla (sentence (2ii) is unacceptable in Bangla) and so the parameter is set accordingly.

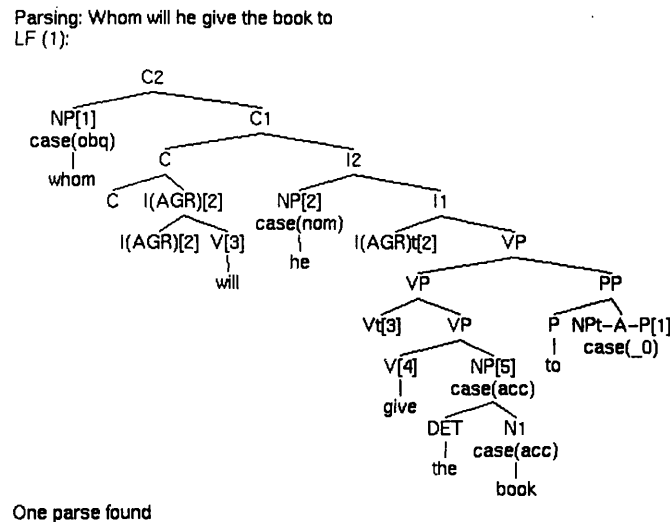


Figure 2-6: Preposition stranding example

- (2) (i) noyon kon des theke eymatro fireche?
noyon which country from just-now returned
Which country has Noyon just returned from?
- (ii) *[kon des]; noyon t; theke eymatro fireche?
which country Noyon from just-now returned

Other parameters

Unlike French, anaphors cannot be dropped in Bangla. For example, in the following sentence the anaphor ‘nije-ke’ or ‘poroshpor-ke’ cannot be dropped.

- (3) (i) shey nije-ke bhalobashe
he/she himself/herself loves
“He/She loves himself/herself”.
- (ii) taaraa poroshpor-ke bhalobashe
they each-other love
“They love each other”.

Also in Bangla, there are no clitics as in French, and as in most other languages the bounding nodes for movement are i_2 and np [Hae91, page 402].

2.2.2 Lexicon

Other than specifying the relevant parameters, I built a lexicon of about a hundred entries. This included proper names, common nouns, adjectives, prepositions, adverbs, markers and verbs. In the specification of verbs, the respective argument structures had to be specified, so that the parser would know what arguments each verb would require. Also, information about what kind of semantic properties should be associated with the agent performing a verb was included to facilitate semantic filtering⁶. Different case-markers were specified for the different cases, such as: ‘-ke’ for the dative and accusative case, and ‘-re’ or ‘-er’ for the genitive case. In Bangla the two determiners ‘ta’ and ‘ti’, which are similar to the articles ‘a’, ‘an’ and ‘the’ in English, append to the end of NPs. These were specified as markers also, but only as markers that do not assign any case.

⁶described in the next chapter

Chapter 3

Bangla

In this chapter we will take a closer look at Bangla and present an overview of the syntactic phenomena associated with this language. I will give an overview of the analyses proposed in the literature to account for these phenomena. As we shall see, for some of the phenomena, the analyses are not sufficient and as a result we came up with alternative analyses. These alternative analyses will help me explain the last step of the Bangla implementation – enhancements to the universal principles in Pappi.

3.1 Bangla linguistic phenomena

In general, the grammar rules for Bangla are similar to that of Japanese and Hindi. All of these languages exhibit scrambling of noun-phrases consisting of nouns, adjectives, and determiners, etc. But unlike Japanese, Bangla sentences exhibit clausal extraposition, rightward scrambling, PP and adverbial scrambling, etc., which are distinct linguistic properties. Bangla language sentences also exhibit standard behavior such as, quantifier raising, and wh-questions. We begin the discussion of these phenomena with an overview of scrambling.

3.1.1 Scrambling

In Bangla we can distinguish between different features of the scrambling process.

1. Short-distance (VP-internal) scrambling:

- (4) (i) John Karim-ke boi-ta dilo
 John Karim book gave
 John gave Karim the book.
- (ii) John boi-ta Karim-ke dilo
 John book Karim gave
 John gave Karim the book.

In the second sentence above the direct object 'boi' scrambles inside the VP towards its left as shown in figure 3-1. This kind of short-distance scrambling occurs when the complements in a verb-phrase are scrambled.

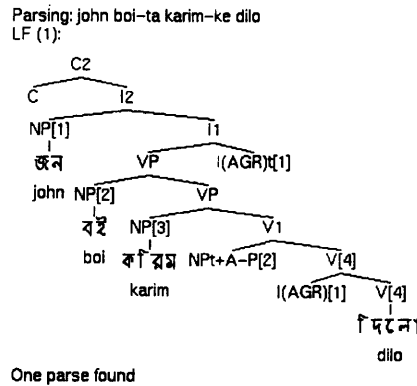


Figure 3-1: VP internal scrambling

2. Short (or medium) distance scrambling to IP:

- (5) (i) John Karim-ke boi-ta dilo
 John Karim book gave
 John gave Karim the book.
- (ii) boi-ta John Karim-ke dilo
 book John Karim gave
- (iii) Karim-ke John boi-ta dilo
 Karim John book gave
- (iv) Karim-ke boi-ta John dilo
 Karim book John dilo

- (v) boi-ta Karim-ke John dilo
 book Karim John gave

As we can see in the last four examples above, the VP-internal arguments can scramble leftward and adjoin IP at any possible order. The last two sentences are examples of multiple scrambling: more than one noun phrase belonging to the same verb's argument structure is moved. This movement is shown explicitly in the parse-tree output in Figure 3-2. In this figure we can see that both 'boi' and 'karim' has scrambled to the left of 'john', the specifier of IP.

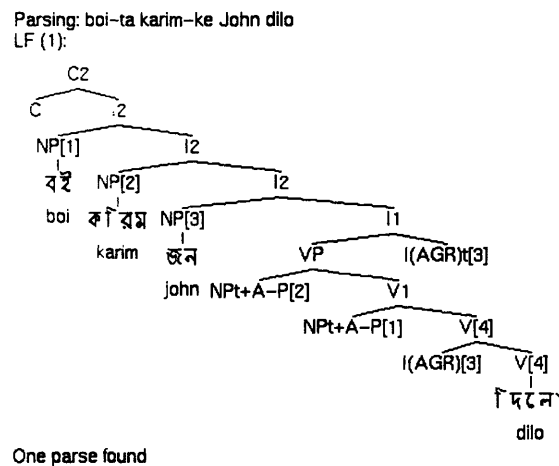


Figure 3-2: Medium distance scrambling

3. Long-distance scrambling:

- (6) (i) bob bollo je john karim-ke boi-ta dilo
 Bob said that John Karim book gave
 Bob said that John gave Karim the book.
- (ii) *boi-ta bob bollo je john karim-ke dilo
 book Bob said that John Karim gave
- (iii) *karim-ke boi-ta bob bollo je john dilo
 Karim book Bob said that John gave

The last two sentences above show that scrambling cannot cross clausal boundaries. If either or both of the VP-internal arguments of the internal clause¹

¹'boi-ta' or 'karim-ke'

scramble out to the front of the sentence, it produces unacceptable sentences. This is unique to Bangla, in Japanese or Korean, there can be long-distance scrambling.

These scrambling processes exhibit interesting results when considered in conjunction with other GB principles. We shall consider the interactions with different Binding Conditions. the weak-crossover effect and anaphor binding.

Interaction with Binding conditions

- (7) (i) bob taake john-er maa-ke dekhalo
 Bob him/her John's mother showed
 Bob showed him John's mother.
- (ii) taake bob john-er maa-ke dekhalo
 him/her Bob John's mother showed
- (iii) bob john-er maa-ke taake dekhalo
 Bob John's mother him/her showed
- (iv) john-er maa-ke bob taake dekhalo
 John's mother Bob him/her showed

All the sentences above have two different senses. Since the marker assigning dative and accusative case² in Bangla is the same ('-ke'), the sentences have two senses: 'Bob showed to him John's mother' and 'Bob showed him to John's mother'. For this reason, we get two parses for the first sentence above. But for the third sentence as shown in figure 3-3 there are a total of four parses. This is because in parse 1 and 3 in the figure, 'John' and 'him' refer to the same person. But in parse 2, Bob shows John's mother to some person other than John, and in parse 2, Bob shows some other person to John's mother. This we know by the numbers associated with the NPs in the parse-tree. If they are equal, then they refer to the same person.

The same kind of situation happens for sentence (7iv) above. The explanation of the different number of parses lies in the interaction of scrambling with Binding Condition C described in section 1.2.4. In sentence (7i) and (7ii) the pronominal

²for indirect and direct objects, respectively

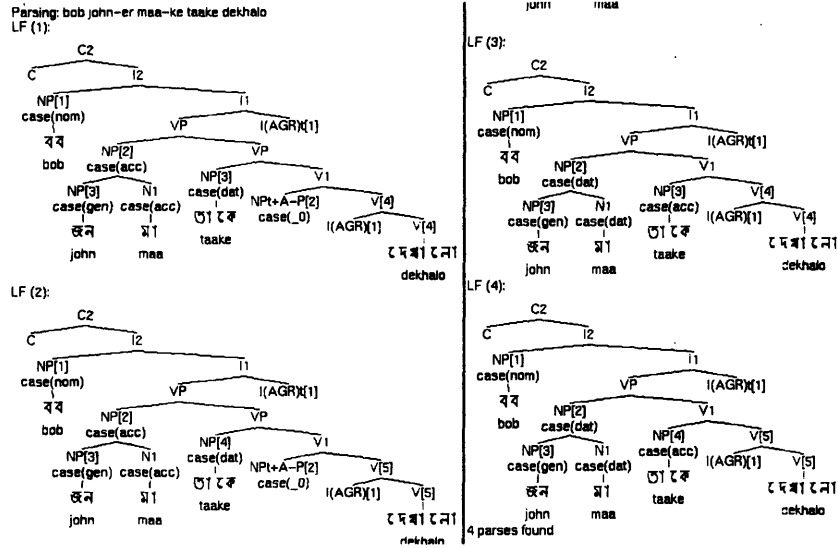


Figure 3-3: “Bob John-er maa-ke taake dekhalo”

‘taake’ precedes the R-expression ‘John’. If they refer to the same person (when they are co-indexed by the same number), then ‘taake’ binds ‘John’, and this directly violates condition C. But in sentence (7iii) and (7iv) scrambling ‘saves’ the parses where ‘taake’ and ‘John’ are co-indexed, because in these cases ‘taake’ cannot bind ‘John’.

The reverse situation happens in the following sentences:

- (8) (i) john-er maa taake bhalobashe
 John’s mother him/her loves
 John’s mother loves him.
- (ii) taake john-er maa bhalobashe
 him/her John’s mother loves

In this case, the first sentence has two parses as shown in figure 3-4. In the first parse, ‘John’s mother’ loves ‘John’, and in the second parse, ‘John’s mother’ loves someone else. But because of filtering by Condition C, the second sentence has only one parse meaning ‘John’s mother’ loves someone else. Note that these are the actual meanings that these two sentences convey to native Bangla speakers. GB theory and the parser implementation associated with it, correctly retrieves all of the senses of these sentences.

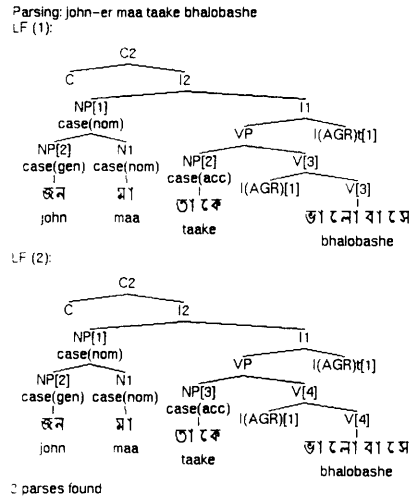


Figure 3-4: “John-er maa taake bhalobashe”

A scenario similar to the above sentences occurs for the following two sentences:

- (9) (i) shey john-er bhai-ke bokbe
 he/she John’s brother will scold
 He will scold John’s brother.
- (ii) john-er bhai-ke shey bokbe
 John’s brother he/she will scold

Here also in the first sentence, Condition C filters out the parses where ‘shey’ and ‘john’ are co-indexed. But this does not happen in the second sentence where ‘shey’ cannot bind ‘john’. Note that this is different from Korean, where Condition C would be applied to sentences similar to the second one above. This is explained by Lee[Lee94] through a generalization of subject binding. But ‘Subject Binding Generalization’ is not necessary in Bangla.

Rightward Scrambling

In Bangla, as well as in Hindi, subject and object NPs can be displaced towards the right of the verb. Anup Mahahan and Gautam Sengupta mention these kind of sentences[Mah90, Sen90] in their treatments of these two languages. But an adequate explanation of this phenomena is not included in their exposition on scrambling. In

general. when NPs move to the right of a verb in a verb-final language like Hindi or Bangla. it is thought of as a specialized form of topicalization³. Since ‘pro-drop’ is allowed in Bangla⁴, sentence (10ii) below is thought of as analogous to sentence (10i) in English.

- (10) (i) He loves it, the apple.
- (ii) bhalobashe aapel
 loves apple
 ‘pro’ loves ‘pro’, apple.
- (iii) john bob-ke boi dilo
 John Bob book gave
 John gave Bob books.

But in Bangla, a much wider variety of sentences are available. If we take sentence (10iii) all 24 permutations of the 4 words is acceptable in Bangla. Of these 24 only 6 have the verb at the end of the sentence. In all other configurations there are NPs after the verb. So this cannot be just a phenomenon of repeated stress of one or two words. All the NPs of the matrix clause can move after the verb, in any order. As shown in figure 3-5, I decided to implement this as rightward scrambling by adjunction to the IP. I could have made a special case for VP-internal NPs and right-adjoin them to VPs. But this would not allow free-order between the VP-internal NPs and the spec-IP(subject) moved to the right of the verb.

Scrambling and Anaphor Binding

- (11) (i) shey nije-ke bhalobashe
 he/she self loves
 She loves herself.
- (ii) nije-ke shey bhalobashe
- (iii) shey bhalobashe nije-ke

³From personal communications with Douglas Jones[Jon93]

⁴explained in chapter 2

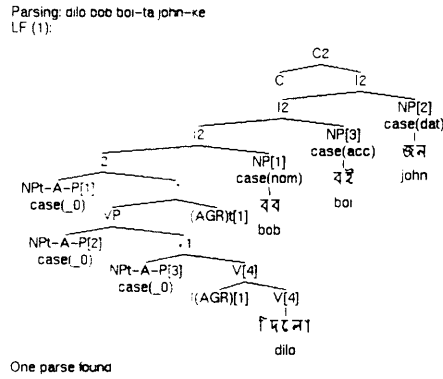


Figure 3-5: Rightward Scrambling

- (iv) nije-ke bhalobashe shey
- (v) bhalobashe nije-ke shey
- (vi) bhalobashe shey nije-ke
- (vii) taaraa poroshpor-ke bhalobashe
they each-other love
They love each other.

Condition A of GB theory⁵ stipulates that an anaphor must be bound in its binding domain. This means that in the parses for all of the above sentences, the anaphor has to be coreferenced with the binder element. But Condition C poses problems for sentence (11ii). This is because the anaphor ‘nije-ke’ is before its binder ‘shey’ and therefore cannot be bound by it. GB theory resolves this problem by moving back the anaphor to its previous position, and applying Condition A to the reconstructed structure. This is known as ‘reconstruction’ in GB theory. Note that the other scrambling examples do not require machinery for ‘reconstruction’ in order to be parsed correctly. Two examples are shown in figure 3-6. This is because the scrambling mechanism posits the subject before the object in the base position.

⁵discussed in section 1.2.4

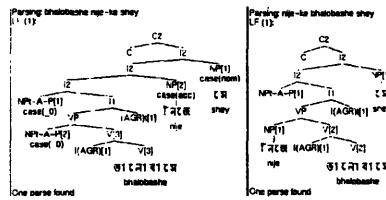


Figure 3-6: Anaphor Binding

Linguistic explanation and implementation

There is some disagreement among linguists regarding whether scrambling is an instance of A'-movement or A-movement. Sengupta in his treatise claims that scrambling is A'-movement. Mahajan argues that some instances of scrambling are NP-movement(A-movement) and other ones(A'-movement). But I adopt the hypothesis of Fong[Fon94]:

1. Scrambling is movement by adjunction in Syntax; adjoining to either VP (short-distance) or IP (medium).
2. The landing site is an A-position.

One of the main reasons for this is the 'weak-crossover effect'(WCO). This is the effect where a noun phrase "crosses over" a co-indexed pronoun, resulting in ungrammaticality. An example is the sentence, "Who_i [does [his_i mother] [love e_i]]?". This effect is caused by syntactic relationships between A'-positions. Since scrambling in Bangla does not show this effect, we consider scrambling to be A movement. This is shown in the examples in Figure 3-7. In the first parse all the schema satisfying the WCO conditions are in effect, but still the sentence is grammatical in Bangla, and so it is parsed by the Bangla parser implementation.

It may be noted here that a lot of GB principles, especially the binding conditions are affected by the A/A' distinction. As shown above I have succeeded in parsing quite a few sentence constructions with the A-movement hypothesis.

Once I decided on the theory behind scrambling, the implementation of scrambling in Pappi, was not very difficult. This is because of the framework already built by

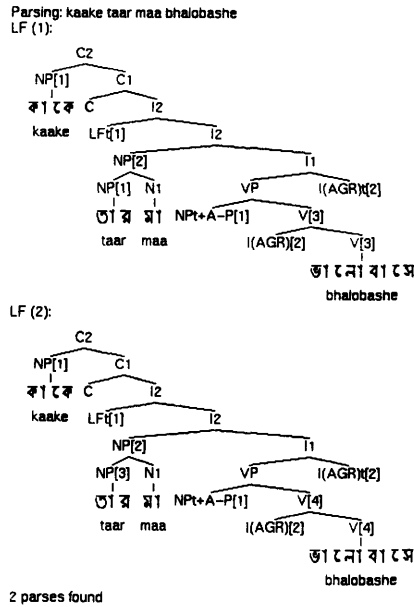


Figure 3-7: No weak-crossover effect in scrambling

Sandiway Fong to handle Japanese scrambling. Most of my work involved modification of the scrambling mechanism to handle rightward scrambling, and to decide the appropriate landing sites for the moved elements. I also implemented **prepositional phrase and adverb scrambling**⁶. A printout of the ‘periphery’ file containing my Prolog code is included in the appendix. I also had to design Bangla fonts in order to display them on the sun and X-windows architecture. A printout of all these characters is also appended at the end of this report.

3.1.2 Quantifier Raising

In English the following sentence has two meanings,

(12) Everyone saw someone.

- (i) For every x there is some y such that it is the case that x saw y.
- (ii) There is some y, such that for every x, it is the case that x saw y.

⁶Different permutations of the sentence ‘shey shohoje boi-ta dilo’= ‘he easily gave the book’ is possible

In the first interpretation, the persons that observe or see. the number of x 's, depend on the quantifier⁷ *everyone*. The persons seen, the number of y 's depend on the quantifier *someone*. In the second example, the quantifiers *everyone* and *someone* determine the scope of the variables x and y in the reverse manner. GB theory explains this kind of ambiguity through the process of Quantifier Raising. In this process, quantifiers are assumed to be moved to the leftmost position in a sentence, where they can be interpreted as binding variables or determining the scope of variables in the sentence[R.L93]. This process is very similar to the way quantifiers are represented in predicate calculus in the field of Logic. For this reason, GB theory assumes that there is a separate level of representation for sentences, where this kind of logico-semantic properties are encoded; The level is called **Logical Form** or **LF** and it is this level that the Pappi interface displays in its output. For example, sentence (13i) and (13ii) are the two logical forms for sentence (12), corresponding to its two interpretations.

- (13) (i) everyone_i someone_j e_i saw e_j
(ii) someone_i everyone_j e_i saw e_j

The same can be said for Bangla, and it is shown for the corresponding Bangla sentence (14) in figure 3-8. The figure explains the pappi implementation. In this case, the the parse is exactly the same as the corresponding English example.

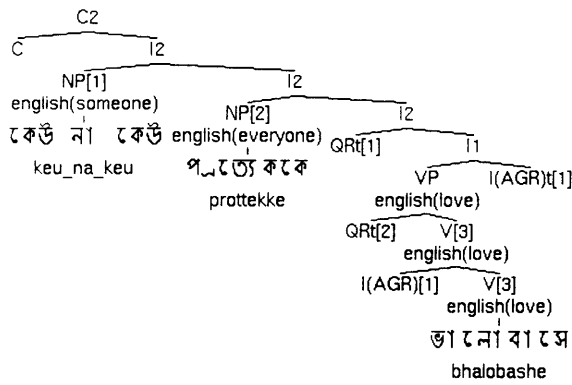
- (14) keu-na-keu prottekke bhalobashe
someone everyone-acc loves
“Someone loves everyone.”

3.1.3 Clausal Extraposition

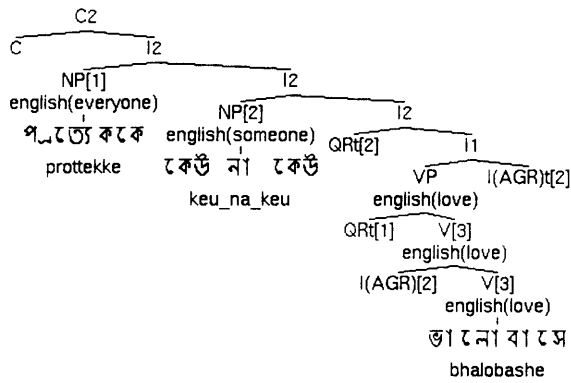
Since Bangla is a verb-final language, in complex sentences we would expect that the verb would follow the embedded clause which is a complement of the verb. For

⁷[R.L93]A quantifier is a determiner whose meaning expresses some notion of quantity: *many*, *lots of*, *few*, *some*, *every*, *no*, etc.

Parsing: keu_na_keu prottekke bhalobashe
 LF (1):



LF (2):



2 parses found

Figure 3-8: Implementation of Quantifier Raising in Bangla

example, in the Bangla translation of ‘Karim said that he loves himself’ we would expect that the clause ‘that he loves himself’ would precede the verb ‘said’. But this is not the case. On the contrary, the clause follows the verb. Also the head of the clause, the complementizer is not at the final position in the clause, it begins the clause. This is shown in sentence (15i). The complementizer might be empty, in which case we will end up with sentences like (15ii).

- (15) (i) john bollo je shey nije-ke bhalobashe
 John said that he self loves
 John said that he loves himself.
- (ii) john bollo shey nije-ke bhalobashe
 John said he self loves
 John said he loves himself.
- (iii) john nije-ke je bhalobashe shey taa bollo
 John self that loves he that said
 That John loves himself. he said that.

We account for this anomaly with a special construction called ‘clausal extraposition’. We assume that the clause was originally situated in the complement position of the verb, but since the verb assigns case, and a clause cannot take case, the clause has to move out of the VP. This phenomenon is also exhibited by dutch sentences. But in dutch, to avoid case assignment the verb can move also, which is not possible in Bangla. I considered a more complicated form of embedded clauses in Bangla(not implemented yet) exhibited in sentence (15iii) to be closer to Dutch ‘verb-raising’[vdA93]. But this also might be considered as just an irregular form of topicalization. The clausal extraposition implementation for sentences (15i) and (15ii) is shown in figures 3-9 and 3-10

3.1.4 Semantic Filtering

For the Bangla sentence ‘John Bob-ke boi-ta dilo’, initially, I would get two parses: one meaning ‘John gave the book to Bob’ and the other meaning ‘The book gave John to Bob’. The nonsensical second parse was the result of the purely syntactical

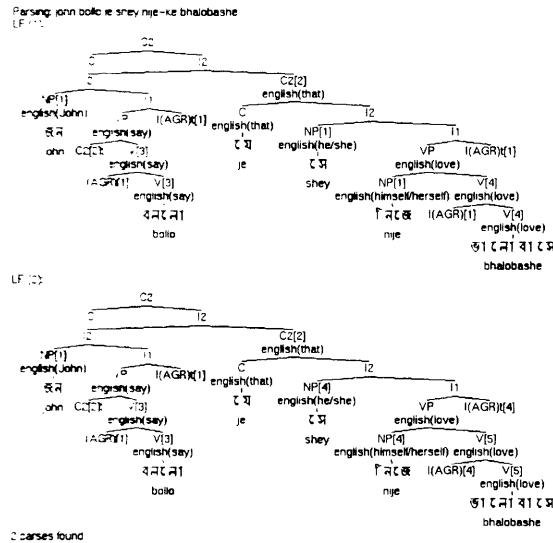


Figure 3-9: Clausal Extraposition with overt complementizer

nature of my parser implementation. To filter out these kind of non-sensical parses, I incorporated a semantic filter⁸ into the Bangla implementation. By specifying that the agent who performs the action ‘giving’ should be animate and possibly a human, I filtered out extraneous parse where ‘book’ was the agent.

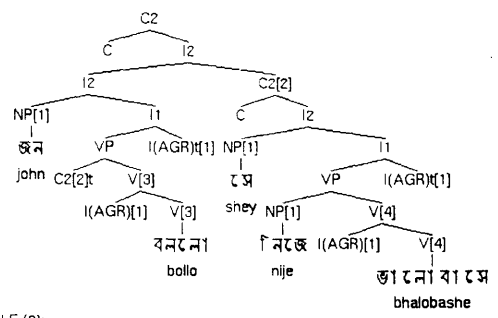
3.1.5 Questions

My Bangla implementation also handles different types of questions. As outlined by Sengupta [Sen90, page 95], the question words move to C at LF. Also, the Binding Conditions interact with the wh-movement to filter out unnecessary parses. For example, sentence (16i) results in the parse in figure 3-11. Here the parse shown in sentence (16ii) is filtered out. Parses for other questions parsed by my system are attached in Appendix B.

- (16) (i) taar bhai kaake dekheche?
his brother who saw
Who did his brother see?
- (ii) taar; bhai kaake; dekheche?

⁸written by Karen Kohl

Parsing: john bollo shey nije-ke bhalobashe
 LF (1):



LF (2):

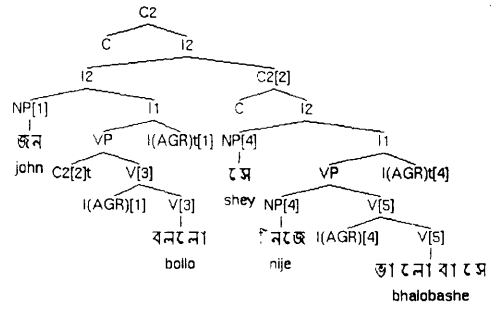
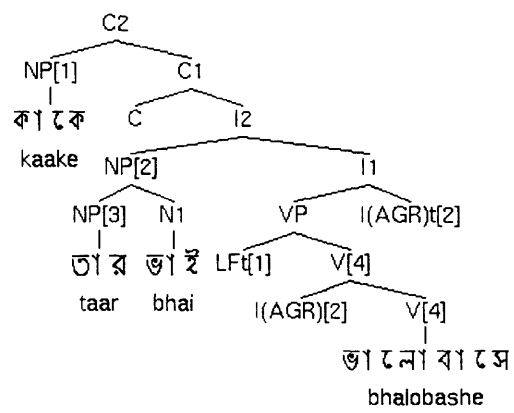


Figure 3-10: Clausal Extraposition with empty complementizer

Parsing: taar bhai kaake bhalobashe
 LF (1):



One parse found

Figure 3-11: Wh-questions and Binding

Chapter 4

Conclusion

The parser implementation outlined in this thesis handles a wide range of Bangla sentences. But because of the vast nature of the natural language domain, there is room for much improvement over my implementation. I shall conclude by outlining some of the problems and limitations that I faced in implementing Bangla, and some of the future enhancements that I would like to make to my system.

4.1 Remaining Problems and Limitations

The main problem I encountered when I started my work on Bangla was the shortage of linguistic explorations of Bangla syntactic phenomena. I started out by comparing Bangla to other languages, and thus coming up with my own explanations for many Bangla properties. I found the work of Sengupta and Klaiman[Kla81, Sen90] at a stage where I had already formulated my main hypotheses about the Bangla parser implementation. Now, I would like to explore the linguistic theory in more detail.

Other than going in deeper into the linguistic theory, my parser implementation needs work on cleaning up the interactions between the different components. For example, I implemented clausal extraposition and Binding Condition filters on scrambling. But I have not been able to completely test out and implement the different combinations of these two phenomena.

I also need to explore means to make the Bangla scrambling implementation more

efficient. Scrambling, in general, adds additional computational complexity into parsing. This is why on complex sentences, the performance of our parser implementation isn't very fast. But it was a tradeoff designed into the Pappi system - the tradeoff between flexibility in implementing languages and efficiency concerns. In the appendix, I have included a table describing the timing and total number of parses considered while parsing different example sentences. Considering the wide variety of sentence structures that the system parses, the performance is still comparable to other efficient parsers[LD94].

4.2 Future Work

Future extensions to the Bangla parser implementations will be in two main directions:

1. Towards a more powerful system by providing the capability of parsing more and more complicated Bangla sentences and
2. Towards an integrated Translator implementation which would translate from English to Bangla and vice versa.

For making the parsing system more powerful, the first step would be to scale up the Bangla dictionary. Also a mechanism to handle Bangla verbal morphology should be added. A mechanism to handle the volitional and non-volitional aspect of passive construction in Bangla[Kla81] would be a useful addition.

For translation, I would like to follow a structure like Bonnie Dorr's unitran[Dor87]. This uses an inter-lingua approach - the text in the source language is first translated into an intermediate language, and then a generation module builds sentences in the target language, based on the GB parameters of that language. Right now we have the English transcriptions of each Bangla word listed as a feature in the lexicon. This can be adapted to be used with an inter-lingua. The translator would map the tree-structure generated in Bangla, to the tree-structure in English, for example. For Bangla-English translation, this would mean that the subj-obj-verb word order would be mapped to subj-verb-obj order. But there are lots of other problems we have to

deal with before making such a syntax-directed translator powerful enough, for instance, to translate some of the noted Bengali poet Tagore's work. I will just mention one category of problems: the problem of translating idioms and complex phrases. By making special entries for idioms in the lexicon and adopting other heuristic approaches, we can attempt to overcome some of these problems. That is precisely the next step in the computational linguistic analysis of Bangla – to build a powerful translation system based on our Bangla parser implementation.

Appendix A

Tables

The following run-time data is based on testing done on a networked Sun IPX sparcstation running OpenWindows. In the tables, the second column denotes the maximum number of parses that were generated in the process of finding the right parse. The third column describes the sum of the time taken to generate these candidate parses, and the time required to filter out the incorrect parses. The correct parse-trees for all the sentences mentioned here are included in appendix B.

Table A.1: Run-time for parses:simple scrambling examples

Sentences	maximum parses generated	runtime(in seconds)
mita o-ke bhalobashe	3	1.20
o-ke mita bhalobashe	6	1.75
bhalobashe mita o-ke	47	3.82
bhalobashe o-ke mita	47	3.83
mita bhalobashe o-ke	20	2.38
o-ke bhalobashe mita	14	1.96

Table A.2: Run-time for parses:general examples

Sentences	maximum parses generated	runtime
keu-na-keu prottekke bhalobashe	5	2.39
karim bollo je shey nijeke bhalobashe	18	9.06
karim bollo shey nijeke bhalobashe	2583	225.14
eneche	3	1.23
taar bhai kaake bhalobashe	4	2.74
kaake taar maa bhalobashe	16	5.28

Table A.3: Run-time for parses:detailed scrambling examples

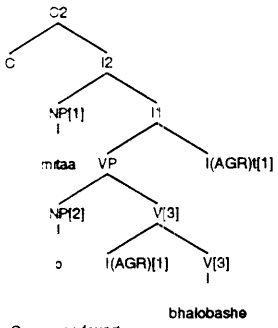
Sentences	maximum parses generated	runtime
bob karim-ke boi-ta dilo	104	8.94
bob boi-ta karim-ke dilo	49	6.27
karim-ke bob boi-ta dilo	116	9.43
karim-ke boi-ta bob dilo	116	9.73
boi-ta karim-ke bob dilo	104	8.86
boi-ta bob karim-ke dilo	49	6.06
bob karim-ke dilo boi-ta	141	11.41
bob boi-ta dilo karim-ke	370	23.08
karim-ke bob dilo boi-ta	192	16.46
karim-ke boi-ta dilo bob	192	14.95
boi-ta karim-ke dilo bob	141	11.88
boi-ta bob dilo karim-ke	370	24.72
boi dilo bob karim-ke	274	17.39
boi dilo karim-ke bob	274	17.03
karim-ke dilo bob boi-ta	198	14.32
karim-ke dilo boi-ta bob	198	14.60
bob dilo karim-ke boi-ta	274	17.53
bob dilo boi-ta karim-ke	274	17.69
dilo boi bob karim-ke	660	38.09
dilo boi karim-ke bob	660	38.08
dilo karim-ke bob boi-ta	660	38.58
dilo karim-ke boi-ta bob	660	37.82
dilo bob karim-ke boi-ta	660	38.27
dilo bob boi-ta karim-ke	660	39.16

Appendix B

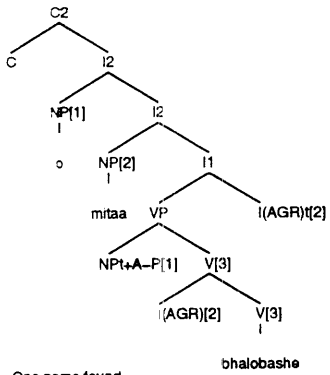
Figures

This appendix contains the parse-tree output of the sentences mentioned in Appendix A and throughout the report. A listing of the Bangla font implementation is also included here.

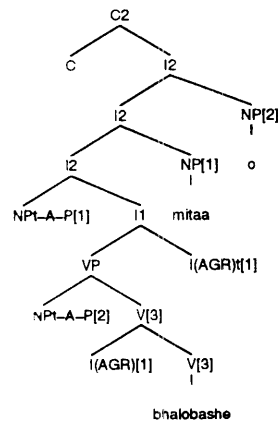
Parsing: mitaa o-ke bhalobashe
 LF (1):



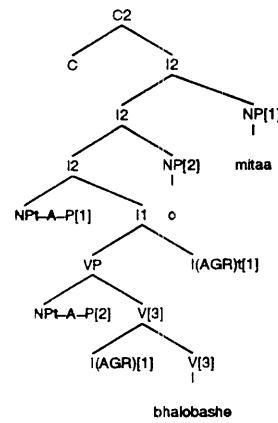
One parse found
 Parsing: o-ke mitaa bhalobashe
 LF (1):



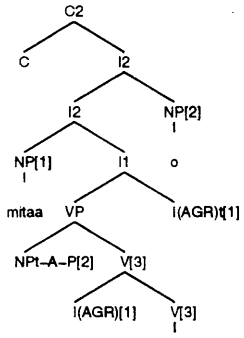
One parse found
 Parsing: bhalobashe mitaa o-ke
 LF (1):



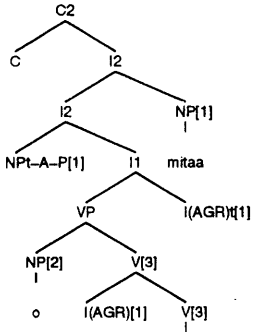
One parse found
 Parsing: bhalobashe o-ke mitaa
 LF (1):



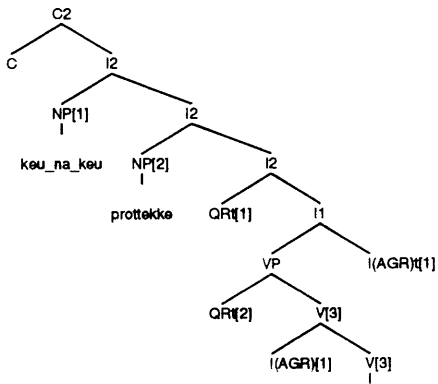
One parse found
 Parsing: mitaa bhalobashe o-ke
 LF (1):



bhalobashe
 One parse found
 Parsing: o-ke bhalobashe mitaa
 LF (1):

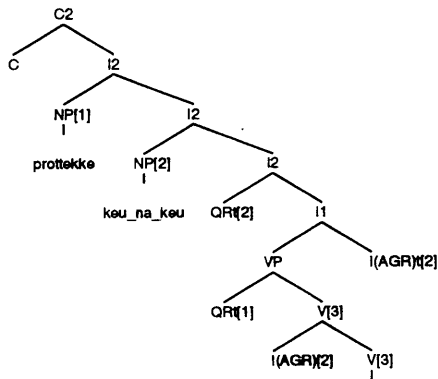


bhalobashe
 One parse found
 Parsing: keu_na_keu prottekke bhalobashe
 LF (1):



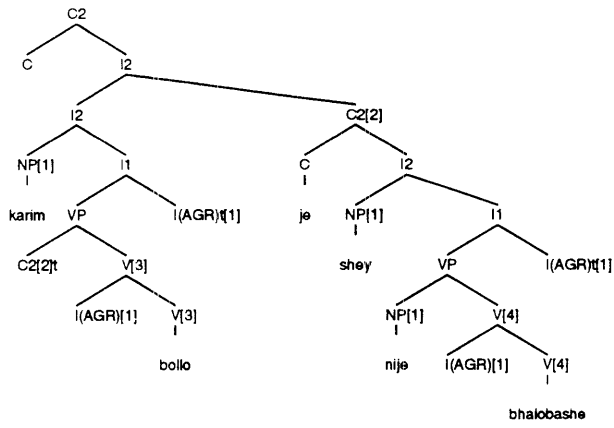
bhalobashe

LF (2):

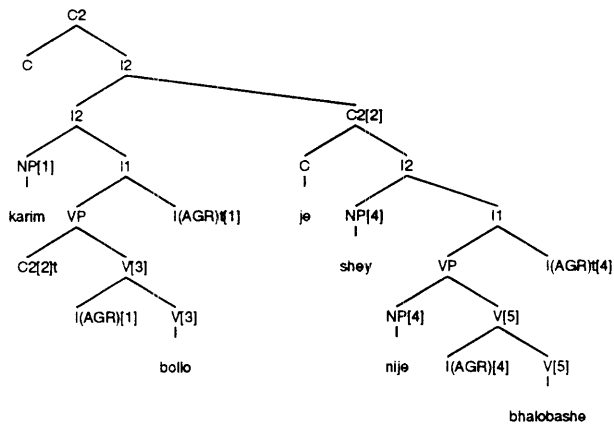


bhalobashe

2 parses found
 Parsing: karim bollo je shey nije-ke bhalobashe
 LF (1):



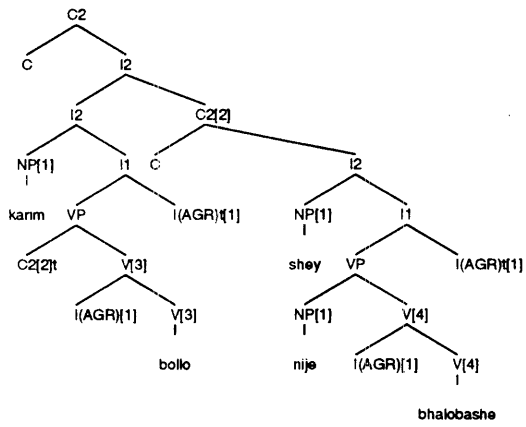
LF (2):



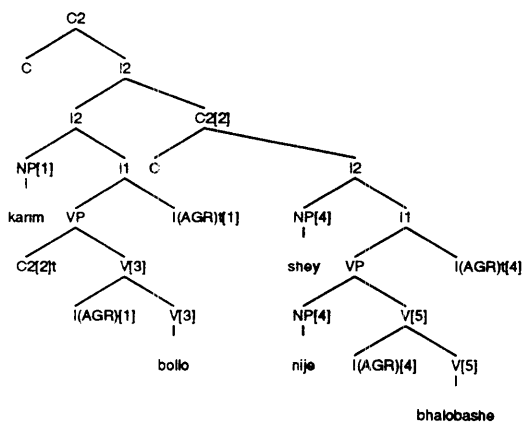
2 parses found

Parsing: karim bollo shey nije-ke bhalobashe

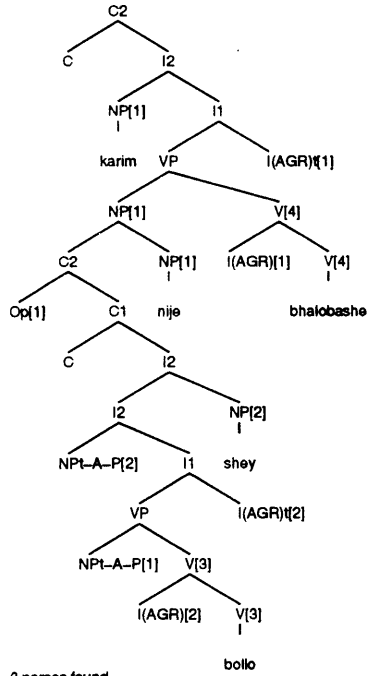
LF (1):



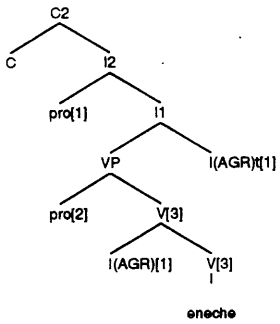
LF (2):



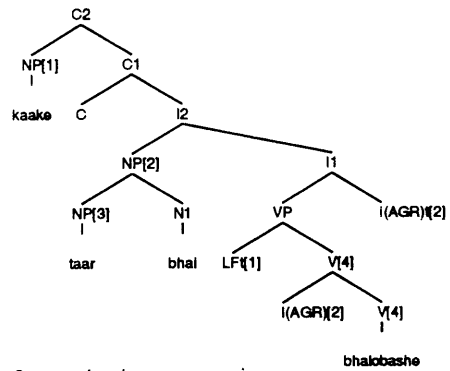
LF (3):



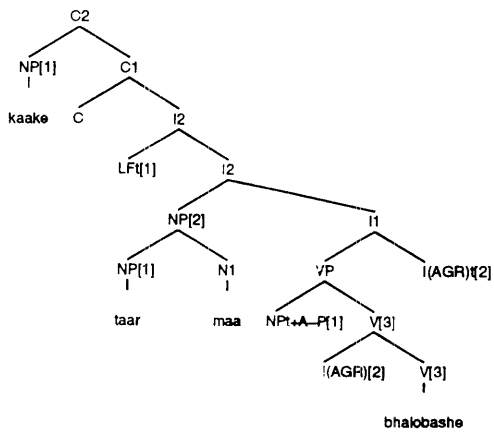
3 parses found
 Parsing: eneche
 LF (1):



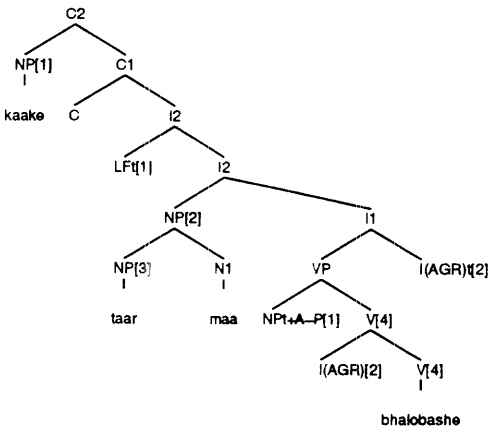
One parse found
 Parsing: taar bhai kaake bhalobashe
 LF (1):



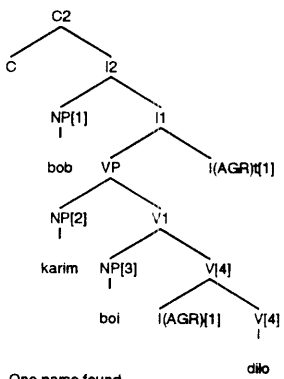
One parse found
 Parsing: kaake taar maa bhalobashe
 LF (1):



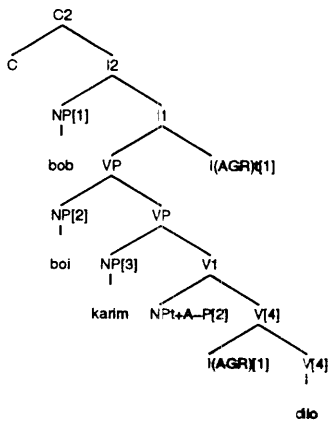
LF (2):



2 parses found
 Parsing: bob karim-ke boi-ta dilo
 LF (1):

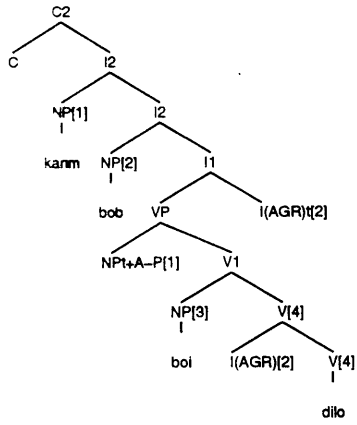


One parse found
 Parsing: bob boi-ta karim-ke dilo
 LF (1):

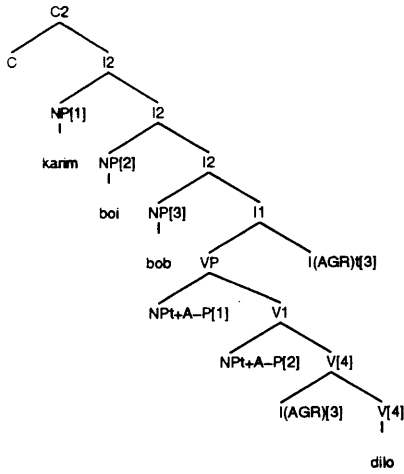


One parse found

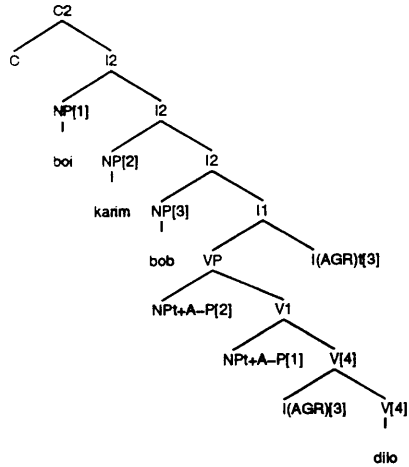
Parsing: karim-ke bob boi-ta dilo
 LF (1):



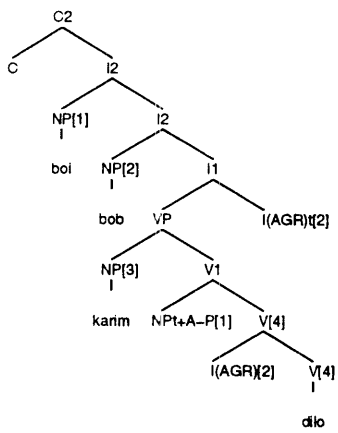
One parse found
 Parsing: karim-ke bob-ta bob dilo
 LF (1):



One parse found
 Parsing: boi-ta karim-ke bob dilo
 LF (1):

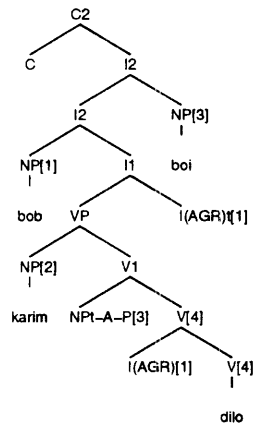


One parse found
 Parsing: boi-ta bob karim-ke dilo
 LF (1):



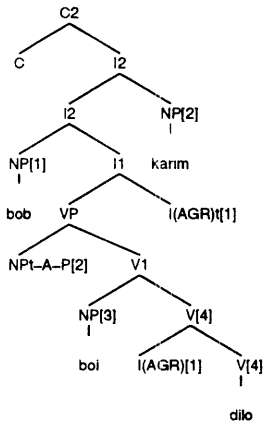
dilo

One parse found
 Parsing: bob karim-ke dilo boi-ta
 LF (1):



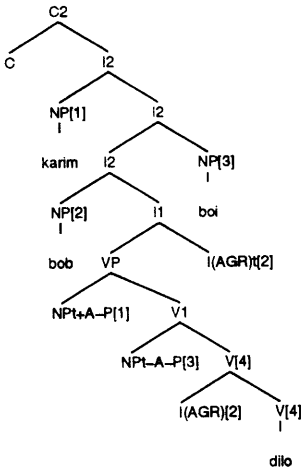
dilo

One parse found
 Parsing: bob boi-ta dilo karim-ke
 LF (1):

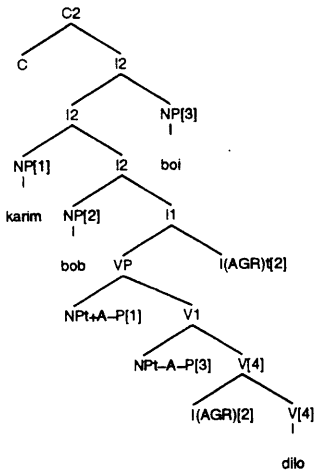


dilo

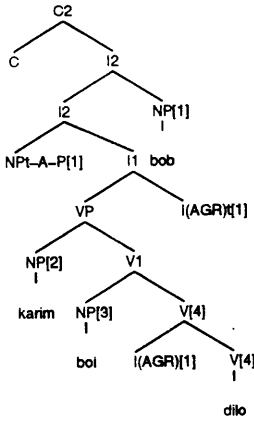
One parse found
 Parsing: karim-ke bob dilo boi-ta
 LF (1):



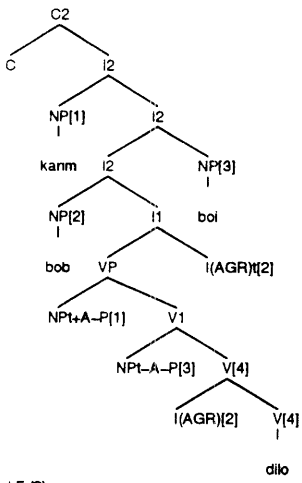
LF (2):



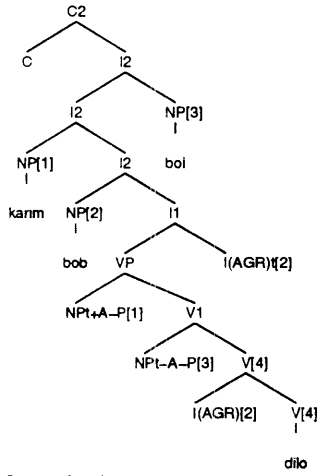
2 parses found
 Parsing: karim-ke boi-ta dilo bob
 LF (1):



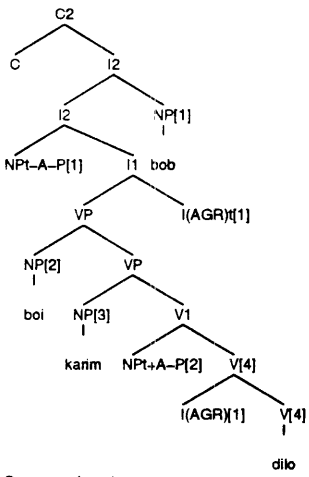
One parse found
 Parsing: boi-ta karim-ke dilo bob & & \hline
 Error: "&" not present in lexicon!
 Error: "&" not present in lexicon!
 Error: "\hline" not present in lexicon!
 Parse blocked by Parse PF
 No parses found
 Parsing: karim-ke bob dilo boi-ta
 LF (1):



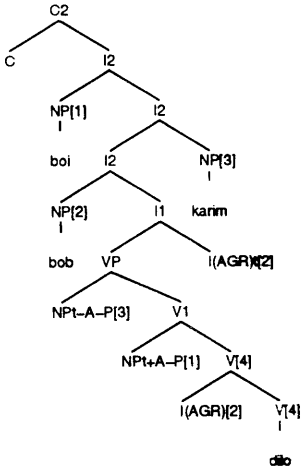
LF (2):



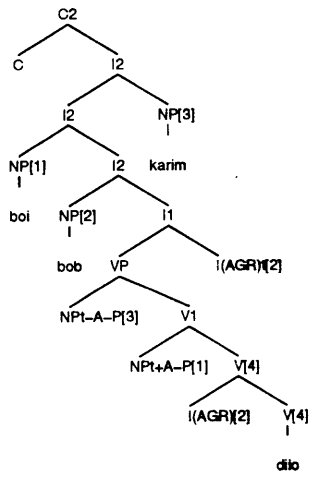
2 parses found
 Parsing: bob-ta kanm-ke dilo bob
 LF (1):



One parse found
 Parsing: boi-ta bob dilo karim-ke
 LF (1):



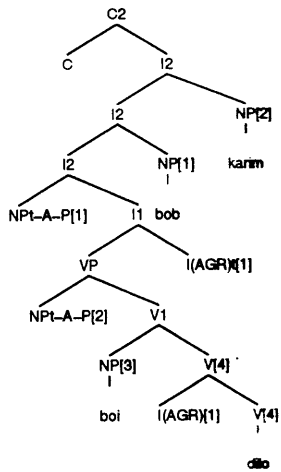
LF (2):



2 parses found

Parsing: boi dilo bob karim-ke

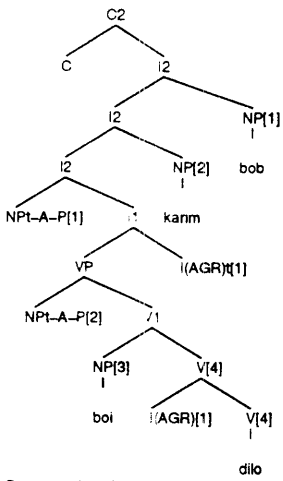
LF (1):



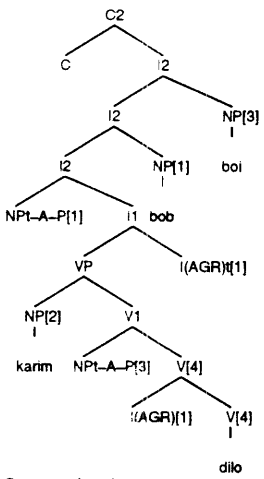
One parse found

Parsing: boi dilo karim-ke bob

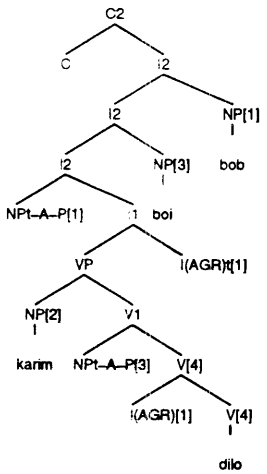
LF (1):



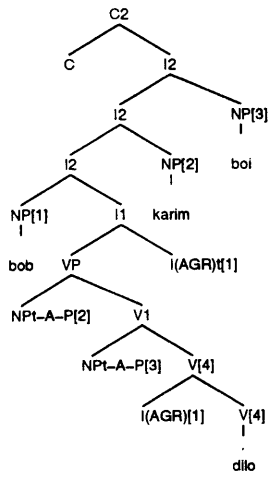
One parse found
 Parsing: kanm-ke dilo bob boi-ta
 LF (1):



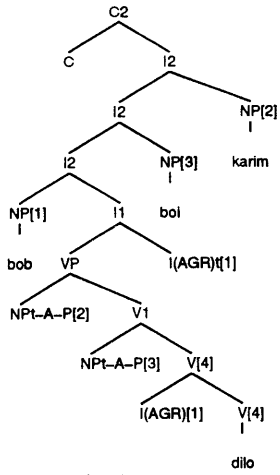
One parse found
 Parsing: kanm-ke dilo bob boi-ta
 LF (1):



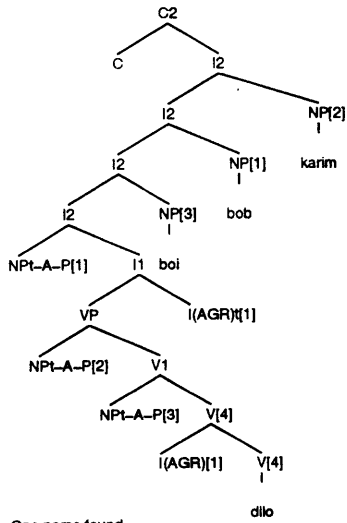
One parse found
 Parsing: bob dilo kanm-ke boi-ta
 LF (1):



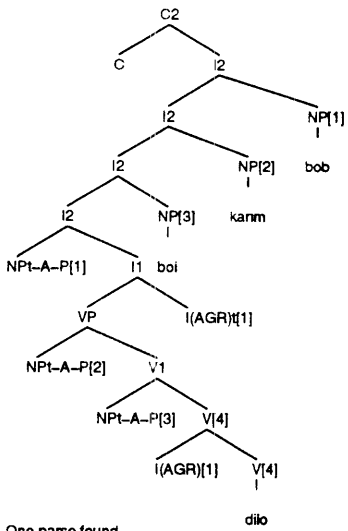
One parse found
 Parsing: bob dilo boi-ta karim-ke
 LF (1):



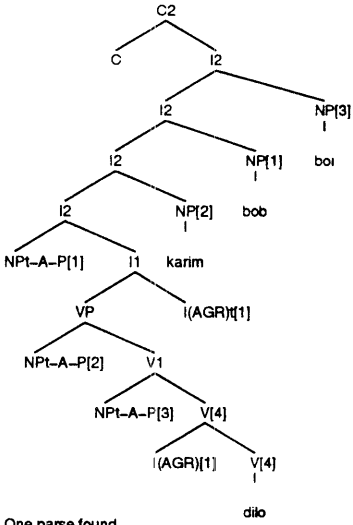
One parse found
 Parsing: dilo boi bob karim-ke
 LF (1):



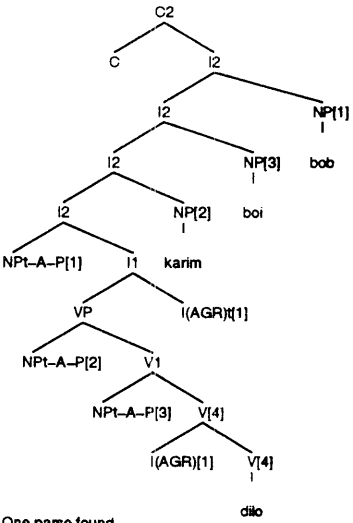
One parse found
 Parsing: dilo boi karim-ke bob
 LF (1):



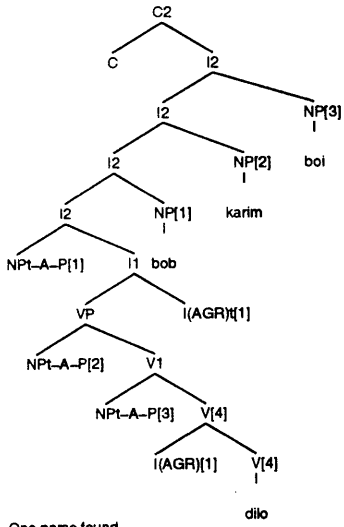
One parse found
 Parsing: dilo karim-ke bob boi-ta
 LF (1):



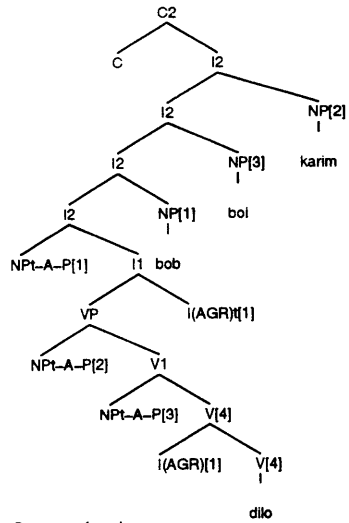
One parse found
 Parsing: dilo karim-ke boi-ta bob
 LF (1):



One parse found
 Parsing: dilo bob karim-ke boi-ta
 LF (1):



One parse found
 Parsing: dilo bob boi-ta karim-ke
 LF (1):



One parse found

Appendix C

Code

The file `lexiconBangla.pl` outlines the Bangla dictionary and the file `peripheryBangla.pl` outlines the extra code necessary to modify the principles. The parameter specifications for Bangla is included in the file `parametersBangla.pl`.

C.1 `parametersBangla.pl`

```
%%% -*- Mode: PROLOG; Package: PROLOG-USER -*-
```

```
%%% BANGLA PARAMETER SETTINGS
```

```
%%% (c) 1991, 1992, 1993 Sandiway Fong, NEC Research Institute, Inc.
```

```
%%% 1995 Zeeshan R.Khan
```

```
%%% REFERENCES
```

```
%%% no P utilities
```

```
%% X-Bar Parameters
```

```
specInitial.
```

```
specFinal :- \+ specInitial.
```

```
headInitial(c).
```

```
headFinal(X) :- \+ headInitial(X).
```

%% agreement

agr(weak).

%% V2 Parameters

% C is not available as adjunction site

% Empty C is null C

%% Subjacency Bounding Nodes

boundingNode(i2).

boundingNode(np).

%% Case Adjacency Parameter

:- initialization(no caseAdjacency).

%% Wh In Syntax Parameter

:- initialization(no whInSyntax).

%% Pro-Drop

proDrop.

%% Negation

negationMoves.

%% No Stranding

:- initialization(no allowStranding).

%% Allow null Case markers for empty Chains

:- initialization (no nullCasemarkers). %changed later

%% null Anaphor

:- initialization(no anaphorDrop).


```

%% Clitics
:- initialization(no clitic(_)).

%% License object pro parameter
:- initialization(no licenseObjectPro).

```

C.2 lexiconBangla.pl

```

%%% -*- Mode: PROLOG; Package: PROLOG-USER -*-

%%% SAMPLE BANGLA LEXICON
%%%
%%% (c) 1991, 1992, 1993 Sandiway Fong, NEC Research Institute, Inc.
%%%      1995 Zeeshan R.Khan

%%% EXPORT
%%%      term(C)                terminals
%%%      lexicon(Word,C,Fs)     Word has category label C
%%%                               and feature list Fs
%%%      probeLexicon(Word)     Word is in lexicon
%%%      vMorphToFs(Base,Form,Features) TNS/AGR features
%%%      inf(Verb,Type)
%%%      relevant(C)           constraints imposed by markers
%%%                               apply to C

%%% REFERENCES
%%%      optWhComplement(X)     xbar
%%%      (list processing)      utilities

term(n). term(v). term(a). term(p). term(c).
term(adv). term(det). term($). term(nq).
term(neg). term(mrkr).

```

```

%%% Most lexical entries are stored directly as
%%%
%%% lex(Word,Category,Features)
%%%
%%% Non-base forms require inference:
%%%   1. plural nouns           all features except agr(_) inherited
%%%                               from the sg. form
%%%   2. nominalized verbs     inherits verb features except morph(,_)
%%%   3. non-base verb forms   all features except morph(,_) inherited
%%%                               from the base form

:- dynamicedrJap:jlookup/4.

lexicon(Word,C,Fs) :-
    nonvar(Word)
    -> (probeLexicon(Word)
        -> builtin(Word,C,Fs)
        ; edrJap:jlookup(Word,_,C,Fs))    %if not in lexicon
    ; builtin(Word,C,Fs).                %if not a variable

probeLexicon(Word) :- lex(Word,_,_) ; lex(Word,_,_,_).

builtin(Word,C,Fs) :- lex(Word,C,Fs). % directly available
builtin(Form,v,Fs) :- % non-base verb forms
    lex(Form,v,Base,F1),
    verbFeatures(Base,F2),
    append1(F1,F2,Fs).

%%% NEGATION
lex(ni, neg, [polarity(-)]).

```

%%% POSTPOSITIONS

%%% features: grid & predicate(pretty much the same)

%%% transparent/adjR/selR,confj

%%% k().

lex(shomporke,p,[grid([], [theme]),predicate(theme),english(about),
k('0073004d0050003c004b00c4'),adjR([goal(roleNotPresent(X,theme),X))]]).
lex(jonne,p,[grid([], [beneficiary]),predicate(beneficiary),english(for),
k('007a006e003e'), adjR([goal(roleNotPresent(X,beneficiary),X))]]).
lex(hote,p,[grid([], [source]),transparent,predicate(source),k(a4aba4e9), % from
adjR([goal(vpAllowExt(source,X),X))]]).

%%% ADVERBS

%%% features: adjoin(left/right), predicate, wh?, k()

lex(kibhabe,adv,[adjoin(left),predicate(manner),wh,k(a4c1a4a6)]). % how
lex(kokhon,adv,[adjoin(left),predicate(time),wh]). % when
lex(gotokal,adv,[adjoin(left),predicate(time),k('00670071004b0068006c'),
english(yesterday)]).
lex(aaj,adv,[adjoin(left),predicate(time),k('00610068007a'),english(today)]).
lex(shohoje,adv,[adjoin(left),predicate(manner),english(easily),
k('00730041003c007a')]]).

%%% DEGREEE ADVERBS(modify adjectives) %might need work to constrain overflow
% permit the resulting AP to take an optional clausal adjunct

lex(oti,adv,[degree,adjR([addFeatures([allowExt(reason),
adjR([goal(nonfiniteClause(X),X))]]))]]).

%%% DETERMINERS

%%% features: don't required person, number, vowel,definitiveness features

%%% as in English

```

lex(ei,det,[k('00450065'),english(this)]).
lex(oi,det,[k('0059'),english(that)]).
lex(shei,det,[k(a4a2a4ce),english(that)]).
lex(kon,det,[wh,k('003c004b0068006e'),english(which)]).
lex(prottek,det,[k('00500040003c0071003e004b'),english(each),op(+)]).

```

%%% ADJECTIVES

```

lex(lomba,a,[grid([theme],[ ]),k(a4caa4aca4a4)]). % long (kanji c4b9a4a4)
lex(bhari,a,[grid([theme],[ ]),k('0076006800720049'),english(heavy)]).
lex(chupchap,a,[grid([theme],[ ]),k(c0c5a4ab)]). % quiet (na)
lex(bhalo,a,[grid([theme],[ ]),k(c0c5a4ab)]). % good
lex(chalakh,a,[grid([theme],[ ]),npprops([animal]),k(c5b7bacd)]). % clever
lex(buddhiman,a,[grid([theme],[ ]),npprops([living])]. %intelligent

```

%% NOUNS

% Wh-nouns

```

lex(ki,n,[a(-),p(-),agr([3,[ ],n]),wh,grid([ ],[ ]),
    morphC(acc),k('0069004b'),english(what)]). % what, nani
lex(ke,n,[a(-),p(-),agr([3,sg,[m,f]]),wh,grid([ ],[ ]),
    morphC(nom),k('003c004b'),english(who)]). % who,dare
lex(kaake,n,[a(-),p(-),agr([3,sg,[m,f]]),wh,grid([ ],[ ]),
    morphC(acc),k('004b0068003c004b'),english(whom)]).
lex(kaake,n,[a(-),p(-),agr([3,sg,[m,f]]),wh,grid([ ],[ ]),
    morphC(dat),k('004b0068003c004b'),english(whom)]).
lex(kaaraa,n,[a(-),p(-),agr([3,pl,[m,f]]),wh,grid([ ],[ ]),
    morphC(nom),k('004b006800720068'),english(who)]). % who,dare
lex(kontaa,n,[a(-),p(-),agr([3,[ ],n]),wh,
    k('003c004b0068006e'),english(which)]). %which
%%lex(kothae,english(where),n,[a(-),p(-),agr([ ],[ ],n)],
%% wh,grid([ ],[ ]),k('003c004b006800700068004f')]).

```

% Proper Nouns

```
lex(bob,n,[a(-),p(-),agr([3,sg,m]),grid([],[]),english('Bob'),
      class(person),props([solid,living,animal,human,male]),
      k('00620062'))].

lex(john,n,[a(-),p(-),agr([3,sg,m]),grid([],[]),english('John'),
      class(person),props([solid,living,animal,human,male]),
      k('007a006e'))].

lex(karim,n,[a(-),p(-),agr([3,sg,m]),grid([],[]),english('Karim'),
      class(person),props([solid,living,animal,human,male]),
      k('004b00690072006d'))].

lex(shumon,n,[a(-),p(-),agr([3,sg,m]),grid([],[]),english('Shumon'),
      class(person),props([solid,living,animal,human,male]),
      k('00730075006d006e'))].

lex(orun,n,[a(-),p(-),agr([3,sg,m]),grid([],[]),english('Orun'),
      class(person),props([solid,living,animal,human,male]),
      k('006100720075006e'))].

lex(fahria,n,[a(-),p(-),agr([3,sg,f]),grid([],[]),english('Fahria'),
      class(person),props([solid,living,animal,human,female]),
      k('00660068004100690072004f0068'))].

lex(hasina,n,[a(-),p(-),agr([3,sg,f]),grid([],[]),english('Hasina'),
      class(person),props([solid,living,animal,human,female]),
      k('0041006800690073006e0068'))].

lex(mitaa,n,[a(-),p(-),agr([3,sg,f]),grid([],[]),english('Mita'),
      class(person),props([solid,living,animal,human,female]),
      k('0069006d00710068'))].

lex(noyon,n,[a(-),p(-),agr([3,sg,f]),grid([],[]),english('Nayan'),
      class(person),props([solid,living,animal,human,female]),
      k('006e004f006e'))].
```

% Quantifier nouns

```
lex(keu,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]),
```

```

        k('003c004b0079'),english(someone),morphC(nom)]).
lex(keu_na_keu,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]) ,
        k('003c004b0079008c006e0068008c003c004b0079'),
        english(someone),morphC(nom)]).
lex(protteke,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]) ,
        k('00500040003c0071003e003c004b'),english(everyone),morphC(nom)]).
lex(kauke,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]) ,
        k('004b00680079003c004b'),english(someone),morphC(acc)]).
lex(prottekke,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]) ,
        k('00500040003c0071003e004b003c004b'),
        english(everyone),morphC(acc)]).
lex(kauke,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]) ,
        k('004b00680079003c004b'),english(someone),morphC(dat)]).
lex(prottekke,n,[a(-),p(-),agr([3,sg,[m,f]]),op(+),grid([],[]) ,
        k('00500040003c0071003e004b003c004b'),
        english(everyone),morphC(dat)]).
lex(shobai,n,[a(-),p(-),agr([3,pl,[m,f]]),op(+),grid([],[]) ,
        k('0073006200680065'),english('everyone-all')]).

%%      op(+) elements that are moved by QR and
%%      form operator-variable structures

% Pronouns and Anaphors
lex(aami,n,[morphC(nom),a(-),p(+),agr([1,sg,[m,f]]),
        grid([],[]),k('006100680069006d'),english('I'),
        props([solid,living,animal,human])]).
lex(aamraa,n,[grid([],[]),morphC(nom),a(-),p(+),agr([1,pl,[m,f]]),
        english(we),k('00610068006d00720068'),
        props([solid,living,animal])]).
lex(tumi,n,[morphC(nom),a(-),p(+),agr([2,sg,[m,f]]),grid([],[]) ,
        english('you/sg'),props([solid,living,animal,human]),
        k('007100750069006d')]).

```

```

lex(tomraa,n,[grid([],[]),morphC(nom),a(-),p(+),agr([2,pl,[m,f]]),
    english('you/pl'),props([solid,living,animal,human]),
    k('003c00710068006d00720068')]).

lex(shey,n,[agr([3,sg,[m,f]]),grid([],[]),k('003c0073'),
    english('he/she'),morphC(nom),
    a(-),p(+),props([solid,living,animal,human])]).

lex(o,n,[agr([3,sg,[m,f]]),grid([],[]),k('006f'),
    english('he/she'),a(-),p(+),props([solid,living,animal,human])]).

lex(taaraa,n,[morphC(nom),a(-),p(+),agr([3,pl,[m,f]]),grid([],[]),
    props([solid,living,animal]),k('0071006800720068'),english(they)]).

lex(taake,n,[grid([],[]),morphC(acc),agr([3,sg,[m,f]]),
    english('him/her'),a(-),p(+),props([solid,living,animal]),
    k('00710068003c004b')]).

lex(taake,n,[grid([],[]),morphC(dat),agr([3,sg,[m,f]]),
    english('him/her'),a(-),p(+),props([solid,living,animal]),
    k('00710068003c004b')]).

%%%%%% note: shey taake bhalobashe -can't be co-indexed

lex(nije,n,[grid([],[]),a(+),p(-),agr([3,sg,[m,f]]),english('himself/herself'),
    k('0069006e003c007a')]).

lex(nijera,n,[grid([],[]),a(+),p(-),agr([3,pl,[m,f]]),english('themselves'),
    k('0069006e003c007a003c006b00720068')]).

lex(nijederke,n,[grid([],[]),a(+),p(-),agr([3,pl,[m,f]]),
    english('to themselves'),morphC(acc),
    k('0069006e003c007a003c00640072003c006b')]).

lex(taar,n,[grid([],[]),morphC(gen),agr([3,sg,m]),
    goal(a(A),inv_plus_minus(A,P)),p(P),english(his),k('007100680072')]).

lex(poroshpor,n,[grid([],[]),a(+),p(-),agr([3,pl,[m,f]]),k('003c'),
    english(each_other)]).

% nijeke,eke-oporke

% Common nouns

lex(eita,n,[a(-),p(-),agr([],[],n),grid([],[]),

```

```

    english(this),k('0045006500740068'))]. % this one
lex(oita,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english(that),k('005900740068'))]. % that one (1)
lex(sheta,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english(that),k('003c007300740068'))]. % that one (2)

lex(kotha,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),english(word),
    k('006b00700068'))].
lex(bhat,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english(rice),k('007600680071'))].
lex(boi,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),english(book),
    props([solid,thought,written]),class(volume),k('00620065'))].
lex(khabar,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english(meal),k('00630068006200680072'),
    props([solid]))].
lex(chul,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english(hair),k('00430075006c'))].
lex(gola,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english('neck/throat'),k('0067006c0068'))].
lex(shohor,n,[a(-),p(-),agr([[ ], [ ],n)],grid([ ], [ ]),
    english(town),k('005300410072'))].
lex(maa,n,[a(-),p(-),agr([3,sg,f]),grid([possessor], [ ]),
    english(mother),class(person),k('006d0068'),
    props([solid,living,animal,human]))].
lex(mayera,n,[agr([3,pl,f]),
    k('006d0068003c004f00720068')|F)] :- nounFeatures(ma,F).
lex(chele,n,[agr([3,sg,m]),grid([possessor], [ ]),
    english(boy),class(person),k('003c004a003c006c'),
    props([solid,living,animal,human]))].
lex(baabaa,n,[a(-),p(-),agr([3,sg,m]),grid([possessor], [ ]),english(father),
    class(person),k('0062006800620068'))]. % father
lex(babara,n,[agr([3,pl,m]),k('006200680062006800720068')|F)] :-

```



```

nounFeatures(baba,F).
lex(bhai,n,[a(-),p(-),agr([3,sg,m]),grid([possessor],[ ]),english(brother),
class(person),k('007600680065'))).
lex(chatro,n,[a(-),p(-),agr([ ],[ ],[m,f]),grid([ ],[ ]),
english(student),class(person),k(b3d8c0b8))].
lex(manush,n,[a(-),p(-),agr([ ],[ ],[m,f]),grid([ ],[ ]),english(man),
class(person),k('006d0068006e0075005a'))).
lex(shishu,n,[a(-),p(-),agr([ ],[ ],[m,f]),grid([ ],[ ]),english(child/children),
class(person),k('0069005300530075')). % child/children
lex(biral,n,[a(-),p(-),agr([ ],[ ],n),grid([ ],[ ]),k(c7ad),
props([solid,living,animal]),english(cat))].
lex(taka,n,[a(-),p(-),agr([ ],[ ],n),grid([ ],[ ]),english(money),k(a4aab6e2)].
lex(idur,n,[a(-),p(-),agr([ ],[ ],n),grid([ ],[ ]),english(mouse),k(c1cd)].
lex(table,n,[a(-),p(-),agr([ ],[ ],n),grid([ ],[ ]),
k(a5c6a1bca5d6a5eb),props([solid]),english(table)]).
lex(chithi,n,[a(-),p(-),agr([ ],[ ],n),grid([ ],[ ]),k('0069004300690054'),
english(letter),props([solid,thought,written]))).

```

%%% Verbs

```
% base-forms
```

```
lex(dewa,v,[morph(dewa,[ ]),grid([agent],[[theme],[goal]]),idoCase(dat),
english(give),agent:[solid,living,animal,human])).
```

```
% since both theme and goal are optional
```

```
lex(bhalobasha,v,[morph(bhalobasha,[ ]),grid([agent],[goal]),english(love),
agent:[solid,living,animal]])).
```

```
lex(bokaa,v,[morph(boka,[ ]),grid([agent],[goal]),english(scold),
agent:[solid,living,animal,human]])).
```

```
lex(dekha,v,[morph(see,[ ]),grid([agent],[goal]),english(see),
agent:[solid,living,animal]])).
```

```
lex(dekhano,v,[morph(dekhano,[ ]),grid([agent],[[proposition],[recipient]]),
idoCase(dat),agent:[solid,living,animal,human],
```

```

    english(show)]).
lex(bolaa,v,[morph(bolaa,[]),grid([agent],[[proposition],[recipient]]),
    idoCase(dat),agent:[solid,living,animal,human],
    english(say)]).
lex(aanaa,v,[morph(aanaa,[]),grid([agent],[patient]),allowExt(destination),
    agent:[solid,living,animal],patient:[solid],
    adv:destination:[near],english(bring)]).

% non-base forms
lex(bhalobashe,v,bhalobasha,[morph(bhalobasha,past(-)),
    k('00760068003c006c006800620068003c0073')]).
lex(bokbe,v,bokaa,[morph(bokaa,past(-)),k('0062006b003c0062')]).
lex(bollo,v,bolaa,[morph(bolaa,past(+)),k('0062006c003c006c0068')]).
lex(dekhalo,v,dekhamo,[morph(dekhamo,past(+)),
    k('003c006400630068003c006c0068')]).
lex(dekhechi,v,dekha,[morph(dekha,past(+)),k('003c0064003c00630069004a')]).
lex(dilo,v,dewa,[morph(dewa,past(+)),k('00690064003c004c0068')]).
lex(dewa,v,dewa,[morph(dewa,past(+)),k('00690064003c004c0068')]).
lex(eneche,v,aanaa,[morph(aanaa,past(+)),k('0045003c006e003c004a')]).

% complementizers
lex(je,c,[selR([not(feature(inf(_)))]),english(that),k('003c006a')]). % that

% markers
lex(ke,mrkr,[left(n,[],morphC(acc)),k(a4f2)]).
lex(ke,mrkr,[left(n,[],morphC(dat)),k(a4f2)]).
lex(r,mrkr,[left(n,[],morphC(gen)),k(a4ce)]).
lex(er,mrkr,[left(n,[],morphC(gen)),k('0068')]).
lex(ta,mrkr,[left(n,[],det),k('0068')]).
lex(ti,mrkr,[left(n,[],det),k('0068')]).

lexFeature(morphC(_),n).

```

```

% relevant for marker constraints
relevant(n). relevant(v). relevant(p).

:- initialization(no contraction(_,_,_)).

% MISCELLANEOUS

verbalize([Y|s]) :- lex(Y,n,_).
nounFeatures(Base,F) :- lex(Base,n,F1), pick1(agr(_),F1,F2),
    pick1(k(_),F2,F).
verbFeatures(Base,F) :-
    lex(Base,v,F1),
    pick1(morph(_,_),F1,F2),
    pickNF1(k(_),F2,F).

%% MAPS MORPHOLOGY INTO SYNTACTIC FEATURES
%%     Verb morphology and Agreement
%%
%%     Form      Tense          AGR
%%     base      infinitival
%%     past(+)   past(+)         agr(_)
%%     past(-)   past(-)         agr(_)

vMorphToFs(_,Form,Features) :-
    formToFeatures(Form,Features).

formToFeatures([], []).
formToFeatures(neg, []).
formToFeatures(te, []).
formToFeatures(past(X), [past(X), agr(_)]).
formToFeatures(npast(X), [past(X), agr(_)]).

```

```
formToFeatures(prog,[prog,past(-),agr(_)]).
```

```
% from Japanese
```

```
% verb morphology
```

```
% iru1 (exists) -> ite
```

```
% iru2 (prog) -> ite
```

```
% iu (say) -> itte
```

```
% iru3 (need) -> itte
```

```
:- dynamic inf/2.
```

```
% Head movement and negation
```

```
negFromV(Neg,V) :-
```

```
    V has_feature neg,
```

```
    mkFs([index(_),polarity(-)],Fs),
```

```
    mkEC(neg,Fs,Neg).
```

```
% From English
```

```
agrConstraint(X) :- intersectAGR([_ , _ , [n]],X) if cat(X,np).
```

C.3 peripheryBangla.pl

```
%%% -- package: PROLOG-USER; Mode: PROLOG --
```

```
%%% PERIPHERY FOR BANGLA
```

```
%%%
```

```
%%% (c) 1991, 1992, 1993 Sandiway Fong, NEC Research Institute, Inc.
```

```
%%% (c) 1995 Zeeshan R.Khan
```

```
%%%
```

```
%%% Language-particular operations + kludgy stuff
```

```
%%% 1. case agreement
```

```
%%% 2. constrained scrambling
```

```
%%% 3. clausal extraposition
```

%%% S-STRUCTURE GRAMMAR ADDITIONS

% Experimental feature pushing

pushFeature(morphC(_)).

:- multifile (rule)/1.

rule ecNP -> [np(NP)] st ec(NP).

rule opC2\$c2 -> [ecNP,c1].

rule head_adjoined _ adjoins_to_the left. % in head movement

% Pushed features: Will be automatically generated...

rule dObjectNP -> [np(NP)] st \+ C==nom if NP has_feature morphC(C).

rule ioObjectNP -> [np(NP)] st C==dat if NP has_feature morphC(C).

rule overtONP -> [overtNP(NP)] st (C==acc;C==dat) if NP has_feature morphC(C).

rule objectNP -> [np(NP)] st (C==acc;C==dat) if NP has_feature morphC(C).

rule subjectNP -> [np(NP)] st \+ (C==acc ; C==dat) if NP has_feature morphC(C).

rule rovertNP -> [overtNP(NP)] st \+ (C==obq ; C==gen) if NP has_feature morphC(C).

rule npSubjectNP -> [np(NP)] st C==gen if NP has_feature morphC(C).

rule npObjectNP -> [np(NP)] st C==gen if NP has_feature morphC(C).

:- multifile (adjunction)/1.

% adjunction rule i2 -> [vp,rovertsubjNP].

% Tuesday, to break inf.loop

adjunction rule i2 -> [i2,rovertNP].

adjunction rule vp -> [overtONP,vp]. % object scrambling (VP-int)

adjunction rule i2 -> [overtONP,i2]. % no intermediate traces

```

adjunction rule i2 -> [pp,i2].
adjunction rule i2 -> [i2,pp].
adjunction rule i2 -> [i2,adv].
adjunction rule i2 -> [adv,i2].

% Base adjunction
adjunction rule np -> [overtNP,nq]. % freely adjoin NQ to NP
adjunction rule np -> [nq,np].
adjunction rule np -> [pp,np] st lexicalProperty(pp,conj).

:- multifile add_goals/2.

rhs [overtONP(NP),vp] add_goals [aPos(NP)]. % scramble object to A-pos
rhs [overtONP(NP),i2] add_goals [aPos(NP)]. % A-pos (tentatively)

rhs [vp(VP),v] add_goals [\+ adjoined(VP)]. % eliminate unnecessary
% non-determinism
% NQ NP Agreement
rhs [overtNP(NP),nq(NQ)] add_goals [agreeNPNQ(NP,NQ)]. % eliminate non-det.
rhs [nq(NQ),np(NP)] add_goals [agreeNPNQ(NP,NQ)].

% Scrambling
lhs overtONP add_goals [pushReq(es(i),es(o))].
lhs dObjectNP & rhs [np(X)] add_goals [cReq(X,es(i),es(o))].
lhs ioObjectNP & rhs [np] add_goals [cReq(es(i),es(o))].
lhs subjectNP & rhs [np(X)] add_goals [ldReq(X,es(i))].
lhs leftvgridcsr1stnp add_goals [oneReq(es(i))].
lhs v0 add_goals [zeroReq(es(i))].

rhs [det,n1] add_inherit plus(2,[1,[wh,op(_)]]).

```

```

% rhs [pp,vp] replace_rhs [coPP,vp].
rhs [c2,relC1NP] replace_rhs [opC2,relC1NP].

% Experimental feature pushing, again...
rhs [np,vgrid] replace_rhs [dObjectNP,vgrid]. % opt. direct object
rhs [np,v1] replace_rhs [ioObjectNP,v1]. % opt. indirect object
rhs [np,i1] replace_rhs [subjectNP,i1]. % opt. subject
rhs [np,pgrid] replace_rhs [overtNP,pgrid]. % disallow post-pos stranding
rhs [np,n1] replace_rhs [npSubjectNP,n1]. % genitive Case
rhs [np,n1] replace_rhs [npObjectNP,n1].

:- multifile (left_bracket)/1.
left_bracket c2 substitute openReq for open.

%%% S-STRUCTURE GRAMMAR DELETIONS

%% kind of redundant, will not be needed in next version

block rule adv -> [adv(Adv)] st maybeSubcategorized(adv,Adv).

%%% OTHER LANGUAGE-SPECIFIC AREAS

%%% EMPTY COMP

%%% similar as in English
% null C is only permitted in matrix clauses and for A-bar clauses
% emptyCompFeatures(Fs) :-
% nullFeatures(Fs),
% addF(goal(apos,fail),Fs) if \+ isMatrixLevel.

% To satisfy the WhInSyntax filter,
% matrix [C] is freely [+/-Wh], e.g. who left, john left

```

```

emptyCompFeatures(Fs) :-
    isMatrixLevel
    -> mkFs([wh(_)],Fs)
    ; nullFeatures(Fs).

%%% Move-Alpha (D-structure to S-structure)

moves(CF,np) :- cat(CF,np).

% compatibleCase(AssignedCase,MorphologicallyRealizedCase)

compatibleCase(X,X).
% compatibleCase(nom,gen).
% compatibleCase(dat,acc).                % for they taake boi dilo
                                           % to consider dative as acc
compatibleCase(_,topic). % deal with topicalization later

% Case Transmission: Need it for scrambling (complement to adjunct)
% NB. need to do [NP NQ NP-t], despite extraction, NQ-NP is overt

caseTransmission(Hd,NP,Case) :-
    baseTrace(NP),
    headOfChain(Head,NP),
    Head has_feature adjunct, % scrambling
    NP has_feature compl,
    assignSCase(Hd,Case,Head),
    NP has_feature case(Case) if \+ ec(NP). % [NP NQ NP-t]

realizedAsMarker(gen).
realizedAsMarker(dat).

```



```

caseRealizationMode(_NP,morphC).

% NQ NP agreement

agreeNPNQ(NP,NQ) :-
    NQ has_feature classifier(Class),
    ((\+ ec(NP) ; NP has_feature class(_))
    -> agreeClassifier(NP,Class)
    ; NP has_feature ec(trace), % force trace
    transmitViaChain([], [goal(agreeClassifier1(X,Class),X)],NP))..

agreeClassifier1(NP,Class) :- agreeClassifier(NP,Class).

agreeClassifier(NP,Class1) :-
    NP has_feature class(Class)
    -> Class = Class1
    ; Class1 = default.

%% Chain Formation conditions

chainLinkConditions(Head,Trace,_,UpPath,DownPath) :-
    \+ vacuousScrambling(UpPath,DownPath)
    if Trace has_feature_set [apos,adjunct], % scrambling
    longDistABarPos(Head,UpPath)
    if Head has_feature_set [apos,adjunct].

vacuousScrambling([],_). % no topmost segment crossed
vacuousScrambling(_,Down) :- \+ Down == [].

% Long Distance scrambling is A-bar

```

```

longDistABarPos(Head,UpPath) :-
    addFeature(goal(apos,fail),Head) if in(c2,UpPath). % inter-clausal

%% SCRAMBLING
%%
%% Must prevent vacuous scrambling

shiftRequest(n,es).

% request carrier r(State)
% State = Var or 1

%% pushReq(ES,ES') start new req state 0
%% shiftReq(ES) all requests state 0 -> 1
%% cReq(ES,ES') ticks off a state 1 req
%% cReq(X,ES,ES') X must be ec if req found
%% openReq(ES,ES') put in place of open, barf if state 0 req found

% initiates a request
pushReq(ES,[r(_)|ES]) :- kReq(ES).

% handles r([X])
kReq([X|ES]) :-
    open(X)
    -> true
    ; (functor(X,r,_)
    -> kReq1(ES)
    ; kReq(ES)).

% fails if we get to r(_) before an open
kReq1([X|ES]) :- open(X) -> true ; \+ functor(X,r,_), kReq1(ES).

```

```

% change state of all open requests
% handles r([X])

shiftReq([X|ES]) :-
    open(X)
    -> true
    ; ((X = r(1) ; X = r([1])) % state <- 1
        -> shiftReq(ES)
        ; shiftReq(ES)).

ldReq(Item,ES) :- ec(Item) -> ldReq(ES) ; true.

ldReq([X|ES]) :-
    open(X)
    -> true
    ; (X = r(V)
        -> (var(V) -> V = [_] ; true),
        shiftReq(ES)
        ; shiftReq(ES)).

% consume one shifted request

cReq(ES,ESp) :-
    ES = [X|ES1],
    (open(X)
    -> ESp = ES
    ; (X = r(S)
        -> (S == 1 % consume
            -> ESp = ES1
            ; ESp = [X|ESp1],
            cReq(ES1,ESp1))
        ; ESp = [X|ESp1],

```

```

        cReq(ES1,ESp1))).

% obligatory consume shifted request

cReq(Item,ES,ESp) :-
    ES = [X|ES1],
    (open(X)
    -> ESp = ES
    ; (X = r(S)
    -> (S == 1    % shifted
    -> withEmpty(Item),
        ESp = ES1
        ; ESp = [X|ESp1],
        cReq(Item,ES1,ESp1))
    ; ESp = [X|ESp1],
    cReq(Item,ES1,ESp1))).

withEmpty(X) :- ec(X) -> true ; adjoined(X,_,X1), withEmpty(X1).

% non-local request propagation
% ES = [...Rs...]
% ES' = [...Rs...,open,...]
% Translates r([1]) -> r(1)

openReq(ES,ESp) :-
    nlReq1(ES,ES1,Rs),
    append1(Rs,ES2,ESp),
    open(ES1,ES2).

% separates local requests Rs leaving ES'
nlReq1([],[],[]).

```

```

nlReq1([X|ES],ESp,Rs) :-
    open(X)
    -> ESp = [X|ES],
        Rs = []
    ; (X = r(S)
        -> (S == 1 % already shifted
            -> Rs = [X|Rsp],
                nlReq1(ES,ESp,Rsp)
            ; S == [1], % shift, xform r([1])->r(1)
                Rs = [r(1)|Rsp],
                    nlReq1(ES,ESp,Rsp))
        ; ESp = [X|ESp1],
            nlReq1(ES,ESp1,Rs)).

% <= 1 state 1 req, no state 0 req
oneReq([X|ES]) :-
    open(X)
    -> true
    ; (X = r(S)
        -> S == 1,
            zeroReq(ES)
        ; oneReq(ES)).

% no reqs of any state allowed

zeroReq([X|ES]) :- open(X) -> true ; \+ functor(X,r,_), zeroReq(ES).

%%% LEXICON SUPPORT

externalRolesForNi(X,Y) :-
    vpAllowExtL([goal,source],X)
    -> Y = goal

```

```

; unsaturatedExtRole(X,agent),
  Y = agent.

%%% CLAUSAL EXTRAPOSITION

:- multifile (rule)/1.
% Availability of empty Comp
rule empty c with Fs st isMatrixLevel, mkFs([wh(_)],Fs).

% Allow the option of local rightwards CP/IP Extraposition
rule xpCP$c2 with Fs -> [] st rwmTrace(c2,Fs,input(i),es(i),es(o)).
% rule xpCP -> [c2] st true.

rule xIP -> [i2] st cpExtpd(es(i)). % for efficiency only

% rule xIP -> [i2].

rule xpdCP -> [c2(CP)] st rwmChain(CP,es(i),es(o)), licenseXpdCP(CP).

:- multifile (adjunction)/1.
adjunction rule i2 -> [xIP,xpdCP].

rhs [c2,vgrid] replace_rhs [xpCP,vgrid]. % option of CP extraposition

:- multifile (app_goals)/2.

rhs [c2(CP),vgrid] app_goals [antiCaseCP(CP)]. % Dutch no Case for CP/IP

%%% EMPTY COMP

% Empty C is Q only
% emptyCompFeatures(Fs) :- mkFs([wh],Fs).

```

```

%% Chain Formation conditions

% chainLinkConditions(_New,_Head,_L,_UpPath,_DownPath).

%% Dutch differs from German (and archaic Dutch) in that the base
%% order is not possible. Traced to Case reasons.

antiCaseCP(CP) :- CP has_feature case(block) if \+ ec(CP).

% trace of rightwards movement
% push rw(i2/c2,ChainItem) onto environment stack
%
% Lemma: Extraposition not required in matrix clauses.

rwmTrace(C,Fs,ES,ESp) :-
    mkFs([ec(_)],Fs),
    mkEC(C,Fs,CP),
    phraseToChainItem(CP,Item),
    push([rw(C,Item)],ES,ESp).

rwmTrace(C,Fs,Input,ES,ESp) :-
    unboundedLookaheadTest(Input),
    mkFs([ec(_)],Fs),
    mkEC(C,Fs,CP),
    phraseToChainItem(CP,Item),
    push([rw(C,Item)],ES,ESp).

cpExtprd(ES) :-
    \+ \+ pop(rw(_,_),ES,_).

% extraposed clause

```

```
% form chain with trace
```

```
rwmChain(CP,ES,ESp) :-  
    cat(CP,C),  
    pop(rw(C,Trace),ES,ESp),  
    phraseToChainItem(CP,Head),  
    chainLink(Head,Trace,_,_),  
    coindex(Head,Trace),  
    instantiateChain([Head,Trace]).
```

```
rwmChainOpen(CP,ES,ESp) :-  
    cat(CP,C),  
    popUpOne(rw(C,Trace),ES,ESp),  
    phraseToChainItem(CP,Head),  
    chainLink(Head,Trace,_,_),  
    coindex(Head,Trace),  
    instantiateChain([Head,Trace]).
```

```
licenseXpdCP(CP) :-  
    notEmptyOperator(NP) if NP specifier_of CP.
```

```
%%% D-structure Binding
```

```
%% A beta-marks B in S
```

```
binds(A,B,CF) :-                % redefinition of "binds"  
    (beta_mark(A,B,CF) ; c_commands(A,B,CF)),  
    coindexed(A,B).
```

```
beta_mark(A,B,SS) :-  
    c_commands_in_DS(A,B,SS),
```



```

A has_feature subject, % A is a subject
make A have_feature beta_marked.

%% A c-commands B in D-struct

c_commands_in_DS(A,B,S) :-
    recoverDsRecur(D,S),
    D has_constituents Cs,
    in(A1,C,Cs),
    transparent(A1,A),
    dominates_mine(C,B).

c_commands_in_DS(A,B,S) :-
    S has_constituent C,
    c_commands_in_DS(A,B,C).

recoverDsRecur([DS,DLeft,DRight], [SS,SLeft,SRight]) :-
    recoverDsElement([DS],[SS]),
    [DS] has_feature _ % DS is not empty
-> recoverDsRecur(DLeft, SLeft),
    recoverDsRecur(DRight, SRight)
; true.

recoverDsRecur(DS, [SS,Word]) :-
    recoverDsElement(DS, [SS,Word]).

recoverDsRecur(DS, [SS]) :-
    recoverDsElement(DS, [SS]).

dominates_mine([C$_$_[Fs1|_]--_], [C$_$_[Fs2|_]--_]) :-
    sameCategory(Fs1,Fs2),
    !.

dominates_mine([C$_$_[Fs1|_]--_,Word], [C$_$_[Fs2|_]--_,Word]) :-

```

```
    sameCategory(Fs1,Fs2),  
    !.  
dominates_mine(A,B) :-  
    A has_feature _,  
    A has_constituent C,  
    dominates_mine(C,B).
```

Bibliography

- [BF92] Robert C. Berwick and Sandiway Fong. Madama butterfly redux: Parsing english and japanese with a principles-and-parameters approach. In R.Mazuka, editor, *Japanese Sentence Processing*. Lawrence Erlbaum, 1992.
- [C.B94] Robert C.Berwick. Class notes, natural language processing and knowledge representation. Course 6.863, Massachusetts Institute of Technology, Cambridge, MA, 1994.
- [Cho81a] Noam A. Chomsky. *Lectures on Government and Binding*. Foris Publications, Dordrecht, 1981.
- [Cho81b] Noam A. Chomsky. Principles and parameters in syntactic theory. In N.Hornstein and D.Lightfoot, editors. *Explanation in Linguistics*. Longman, London and New York, 1981.
- [Dor87] Bonnie Jean Dorr. Unitran: A principle-based approach to machine translation. Technical Report 1000, MIT Artificial Intelligence Laboratory, 1987.
- [Fon94] Sandiway Fong. Towards a proper linguistic and computational treatment of scrambling: an analysis of japanese. In *Proc. COLING-94*, Kyoto, Japan, August 1994.
- [Hae91] Liliane Haegeman. *Introduction to Government and Binding Theory*. Blackwell Publishers, second edition, 1991.
- [Jon93] Douglas Arnold Jones. *Binding as an interface condition: an investigation of Hindi scrambling*. PhD thesis, MIT. 1993.

- [Kla81] M.H. Klaiman. *Volitionality and Subject in Bengali: A Study of Semantic Parameters in Grammatical Processes*. Indiana University Linguistics Club, Bloomington, 1981.
- [LD94] Dekang Lin and Bonnie Dorr. Government-binding theory and principle-based parsing. October 1994.
- [Lee94] Young-Suk Lee. *Scrambling as Case-Driven Obligatory Movement*. PhD thesis, University of Pennsylvania, Philadelphia, PA 19104, 1994.
- [Mah90] Anoop Kumar Mahajan. *The A/A-Bar distinction and movement theory*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, 1990.
- [R.L93] R.L.Trask. *A dictionary of Grammatical Terms in Linguistics*. Routledge, London and New York, 1993.
- [Sai85] M. Saito. *Some Asymmetries in Japanese and Their Theoretical Implications*. PhD thesis, MIT, 1985.
- [Sen90] Gautam Sengupta. *Binding and Scrambling in Bangla*. PhD thesis, University of Massachusetts, September 1990.
- [vdA93] Tim van der Avoird. Accomodating gb theory for dutch by using pappi. Pappi document, March 1993.