ANALYSING HINDI TEXT

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & TECHNOLOGY

822

D. K. LOBIYAL

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067
JANUARY 1991

CERTIFICATE

This is to certify that the dissertation entitled "Analyzing Hindi Text ", being submitted by me to Jawaharlal Nehru University in the partial fulfilment of the requirements for the award of the degree of Master of Technology, is a record of original work done by me under the supervision of Dr. G. V. Singh , Associate Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University during the year 1990, Monsoon Semester.

The results reported in this dissertation have not been submitted in part or full to any other University or Institute for the award of any degree or diploma, etc.

Daya Krishan Lobiyal
D. K. Lobiyal

[illegible]

Prof. N. P. Mukherjee
Dean,
School of Computer and
Systems Sciences,
J.N.U.,
New Delhi.

[illegible]

Dr. G. V. Singh
Associate Professor,
School of Computer and
Systems Sciences,
J.N.U.,
New Delhi.

ACKNOWLEDGEMENT

I express my humble gratitude towards Dr. G. V. Singh , Associate Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi who was always there to guide me to the safe exit in case of any crisis and was all the way with me to boost my morale and to provide his valuable guidance.

I am also highly thankful to Prof. N. P. Mukherjee, Dean, School of Computer and Systems Sciences, who took great pains to provide me with the environment and all the facilities required for the successful completion of my project.

I also take this opportunity to thank all faculty and staff members as well as my friends who have been directly or indirectly helpful in eliminating a variety of problems encountered by me in the course of completing this dissertation.

D. K. Lobiyal

CONTENTS

CHAPTER 1

INTRODUCTION

Natural language is any language that a human learns from the environment and uses to communicate with each other. Natural language processing refers to the operations analysis, synthesis, transformation and inference performed on specific kinds of natural language elements. These operations can be defined as follows : Analysis is breaking of natural language units, like sentence into constituent components (part of speech), e.g., breaking a sentence into its actor form (subject) and action form (predicate); Synthesis is generating language units from smaller components, e.g. combining a nominative form and indefinite verb form. Transformation is changing of the units into another units and subunits that belong to a different class of the original unit, e.g. converting a declarative (positive) sentence into its interrogative form. Inference is drawing of conclusions or bringing out implicit knowledge from the knowledge already explicitly stored [1].

The basis of all these operations lies in a set of rules. Each type of these operations are governed by distinct set of rules and each set governs a particular type of operation. The set of rules governing an operation is referred to as a grammar for that particular language operation. For example, analysis fo a sentence is governed by a set of rules together defining a syntax grammar. Similarly, there is grammar defining the semantics (meaning) of the language.

The significant aspects of a natural language, however can be considered at two levels :

1. Syntax level
2. Semantics level

In almost all the languages there is a close relationship between the syntax (the form) and the semantics (the meaning), that is to say that the form upto a large extent determines the meaning behind it. For example, the subject and the verb of a sentence must be agreed over number, gender and person. But for the convenience of implementation, semantics analysis is many times separated from syntax analysis.

1.1 SYNTAX

Syntax concerns how words can be put together to form sentences that look correct in the language and how one word is related to another [2]. Thus syntax analysis performs two functions - Determining the structure of the natural language input and Regularizing the syntactic structure [3]. Various grammars are used to represent the acceptable structural relationship among different components of the language unit. A grammar (syntax) formalism G is four-tuple and is denoted as G = (N,T,P,S), where T is the terminal vocabulary, the words (or symbols) of the language being defined; N is the non-terminal vocabulary, the symbols (distinct from the terminal vocabulary T) which are used in specifying the word, phrase categories etc. We defined the vocabulary V to be the union of the terminal and non-terminal vocabularies. P is a finite set of productions (rewrite rules), each production is of the form a ::= b where a is sequence of one or more symbols from V and b is sequence of zero or more symbols from V. S is the start symbol, a member of N [6]. A grammar type is determined by the type of rules in P. Chomsky has defined four types of grammars referred to as Chomsky hierarchy of grammars, which consist of:

1. Unrestricted grammar (type 0)
2. Context-sensitive grammar (type 1)
3. Context free grammar (type 2)
4. Regular grammar (type 3)

1.1.1 Unrestricted Grammar

The largest family of grammars in the Chomsky hierarchy permits productions of the form a ::== b, where a and b are arbitrary strings of grammar symbols, with a <> empty. These grammars are known as unrestricted grammars.

1.1.2 Context-Sensitive Grammar

Suppose we place the restriction on productions a ::= b of a grammar that b be at least as long as a. Then we call the resulting grammar Context-sensitive grammar. The term "context-sensitive" comes from a normal form for these grammar, where each production is of form a1 A a2 ::= a1 a2, with <> empty. This production permits the replacement of variable A by string only in the "context" a1 - a2 [5].

1.1.3 Context-Free Grammar

In a context-free grammar, every rule is of the form

A ::= b

Where A is a non-terminal symbol and b is a sequence of zero or more terminal and non-terminal symbols [5]. There must be only one non-terminal symbol in the left side of the rule.

1.1.4 Regular Grammar

If all productions of the CFG are of the form A ::== wB or A ::== w, where A and B are variables and w is a (possibly empty) string of terminals, then we say the grammar is right-linear. If all productions are of the form A ::== Bw or A ::== w, we call it left-linear grammar. A right- or left-linear grammar is called a regular grammar.

A grammar for natural language in general turns out to be context-sensitive in nature. A context-sensitive grammar however, is computationally intractable. Therefore to reduce the computational complexity, the context-free grammar is used to define the syntax. This context-free grammar is then supported by another grammar called morphological grammar, representing morphological phase. Context-free grammar framework, however, fails to capture many aspects of a natural language. Another framework whicch is more close to context-sensitive grammar framework is called definite clause grammar framework. This framework is directly supported by a programming language Turbo Prolog. The definite clause grammar is a grammatical form close to index grammar. In our approach, for analysis of a Hindi sentence we have adopted DCG formalism.

1.2 SEMANTICS

Semantics concerns with what words mean and how these meanings combine in sentences to form sentence meaning [2]. The meaning of a natural language word and sentence is the entity and action it denotes. Semantic interpretation is process of determining the meaning. There are various formalisms to represent semantic interpretation. Some of the main formlisms are:

1. Case grammar
2. Conceptual dependency
3. Augmented Transition Network

1.2.1 Case Grammar

The notion of "case" has been used to refer to several related concepts. Traditionaly, it has meant the classification of noun according to their syntactic role in a sentence, signalled by various inflected forms. In English, only pronouns have these case inflections. For instance, first person singular pronoun is "I" (nominative case), "Me" (accusative/objective case) or "My" (genitive /possessive case) according to its use as subject, object or possessive article. The idea of a direct relationship between inflections and cases is one kind of case, also called "surface" or "syntactic level" case. Fillmore argued that the concept of case only dealing with surface structure was not significant in any meaningful way. The deep structure reveals the cases of importance to the meaning of the sentence. The following examples illustrate the point. In the sentence

He told his son a bedtime story.

He is the subject; told is the verb; His son is the indirect object; a bedtime story is the direct object. In the sentence

His son was told a bedtime story by him.

the subject is now "his son", there is no indirect object, "him" is objective form of "he". Note that in these two examples the surface cases are different even though the meaning of both the sentences is same. If the case definitions are related to the meaning then this kind of anomalous situations would not arise.

Fillmore postulated that a sentence in its basic structure consists of a verb and one or more noun phrases, each associated with the verb in a particular case relationship. There can be a compound instance of a single case but each case relationship occurs only once in a simple sentence.

According to Fillmore's case grammar, an action (i.e. a verb) can be related to at most five following cases :

Agent	-	the instigator of the action, an animate being.
Instrumental	-	the thing used to perform the action, an inanimate object.

Locative	-	the locaton of the action and other cases.
Dative	-	the recipient of the action.
Neutral	-	the thing being acted upon, combining the objective and the factitive.

The case grammar formalism is a propositional framework where the verb representing the action is taken to be representing the proposition and the cases representing arguments of the proposition. This framework does incorporate the information about, tense, aspect, mood, etc. representing the modality of the sentence. Therefore, to capture the complete meaning of the sentence, the proposition P arrived at considering the case framework must be supported by modality M. The sentence meaning is then represented by :

sentence ::== modality + proposition

or

S ::== M + P

The proposition (P), as already stated, is a tenseless structure consisting of verb and cases:

P ::== V + C1 + C2 + C3 + ... + Cn

Where each Ci is a case name that generates either an NP or an embedded S. In a sentence, at least one case must be present but no case may appear twice. Case markers are produced by the language-specific Kasus element (K):

Ci ::== K + NP

The Kasus element K generates a preposition, postposition or a case affix. Kasus may be null [1].

The natural language Hindi presents itself to the case frame analysis. Hindi cases, however, are slightly different from Fillmore's cases. A Hindi sentence can have at most six of the following cases :

1. Nominative
2. Accusative
3. Instrumental
4. Dative
5. Ablative
6. Locative

In Hindi, genitive and vocative are not considered as cases but genitive shows relationships between noun phrases and vocative shows exclamation.

1.2.2 Conceptual Dependency

Another approach to represent the sentence meaning is conceptual dependency (CD). Conceptual dependency theory is similar to case grammar in that the representation of the meaning of a sentence revolves around the action of the sentence. But there are significant differences between the case grammar and conceptual dependency(CD) theory. The most important difference is that the action of the sentence is not represented by the the verb, but rather by the interrelationship of a set of primitive acts.

A major axiom of conceptual dependency is that any two sentences with the same meaning will have the same internal representation, irrespective of the words and their order used in the sentence. Therefore, the active and passive forms of a sentence are considered to have the same meaning. So, in conceptual dependency theory they would have same internal representation. For example, the sentence

John threw the ball

and

The ball was thrown by John.

have the same following CD representation :

```
ACTOR  : John
ACTION : PTRANS
OBJECT : ball
DIRECTION : FROM : John
            TO   : somewhere.
```

Thus, in CD framework the information in a sentence is represented by an internal structure known as conceptualization. A conceptualization can be active or stative. An active conceptualization consists of the following slots (roles to be played by objects in the action): actor; action; object; and direction, (source (from) destination (to)). The reader is requested to see the above example again. A conceptualization can have another conceptualization as an Instrument. A stative conceptualization consists of the following slots: object, state and state value.

The concept which can play various roles are referred to as PPs (Picture Producer). PPs can have attributes referred to as PAs (Picture Adders). For example, a PP DOG has attributes PAs : animate and colour. A PP serving various roles like that of ACTOR, OBJRECT or DIRECTION must have appropriate PAs.

Shank has defined primitive actions or ACTs :

1. MOVE is any action which involves movement of a part of a person's body.
2. PROPEL is an action performed by a person which causes some object to change location.
3. INGEST involves the action of taking something into person's body and is primarily used to describe eating and drinking.
4. EXPEL is the opposite of INGEST.
5. GRASP is used to refer to the action of a person taking hold of an object.
6. ATRANS is the abstract transfer of some object.
7. PTRANS is the physical transfer of some object.
8. SPEAK means to produce a sound, e.g. singing.
9. ATTEND is the opposite of SPEAK (receiving information from external source).
10. MTRANS involves a transfer of information to or from the brain.

11. MBUILD involves combining mental information and used for concepts such as thinking or considering.

A conceptualization can have tenses. These tenses are :

1. Preesnt
2. Past
3. Future
4. Negation
5. Start of a transition
6. Finish of a transition
7. Conditional
8. Continuous
9. Question

1.2.3 Augmented Transition Network

An augmented transition network (ATN) grammar is a recursive transition network (RTN) that is augmented with a set of registers (associated wuth each category node) and with arbitrary conditions and actions associated with the arcs of grammar [16]. To understand briefly the concept of ATNs consider a small sample grammar :

S ::== NP V

S ::== NP V NP (PP)*

S ::== NP AUX V

S ::== NP AUX V NP (PP)*

S ::== AUX NP V

```
S ::== AUX NP V NP (PP)*

NP ::== DET (ADJ)* N (PP)*

NP ::== NPR

PP ::== PREP NP
```

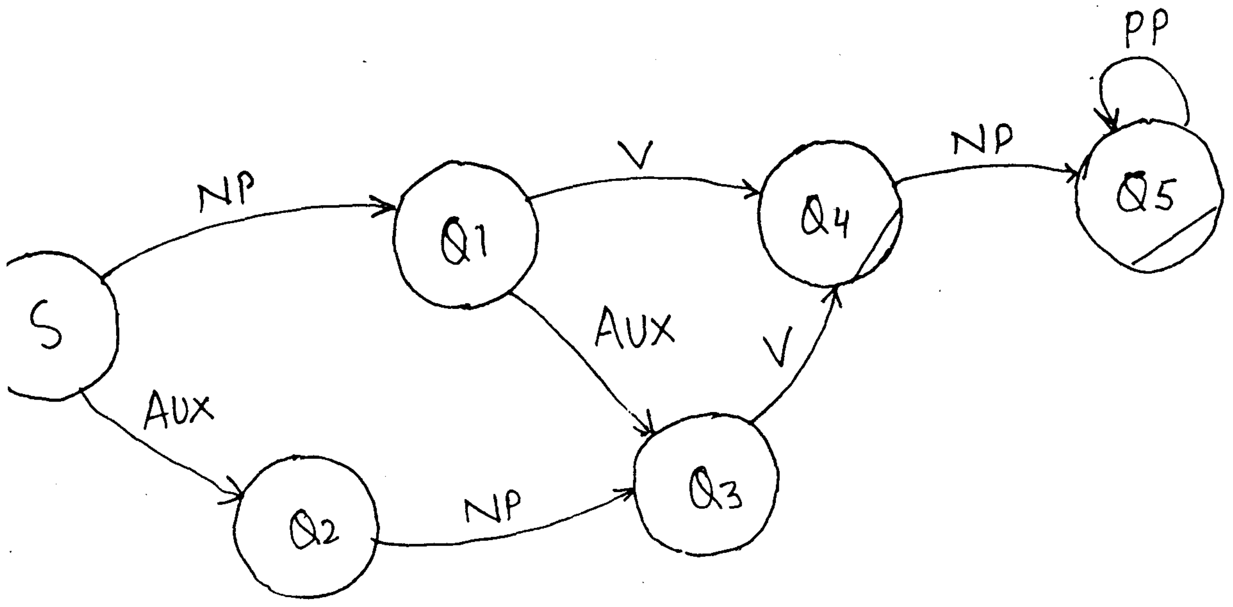
The associated RTN is given in figure 1a, 1b and 1c.

The networks are referred to as RTNs as a labelled arrow (e.g. NP) instead of causing a transition to a node results into traversing a complete network followed by a return to the calling position. When each node (e.g. NP) is associated with a set of registers, the RTN is referred to ATNs. These registers contain the modality information (e.g. tense, aspect etc) of the sentence being analyzed.

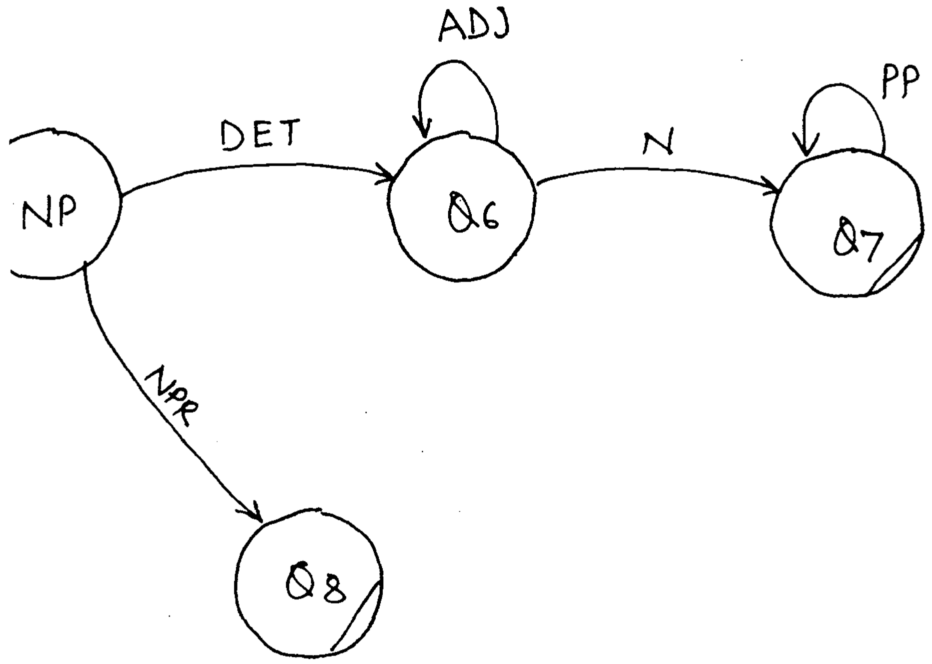
ATNs are powerful grammatical formalisms used both for syntax analysis as well as semantic analysis. ATNs, in fact, are equivalent to Transformational grammar. A transformational grammar consists of a base component and a transformational component, The base component is a context-free grammar and the transformational component is a set of rewrite rules.

1.3 THE SCOPE OF THE PRESENT WORK

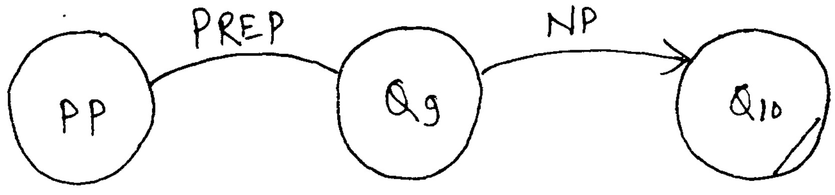
The area of natural language processing (NLP) in computer science is considered to fall in the domain of Artificial Intelligence (AI). AI problems are computationally hard for two reasons: the first, no appropriate data structures (grammars) are available that suit all (even majority) of the problems even in a single given domain. The second , it is seldom possible to comprehend all the knowledge related



1. a.



1. b.



1. C

A RECURSIVE TRANSITION NETWORK FOR SIMPLE ENGLISH SENTENCE to the problem at hand and thus clearcut algorithmic solutions can not be proposed for the problem. In area of NLP no syntax or semantic grammar frameworks are available that can capture all the aspects of a given natural language. Even in limited semantic domains (like law, medicine and natural language interface to DBMS etc) success rate is of 80% is considered to be very good [3].

Designing a general purpose NLP system still remains a dream. Even to achieve this efficiency, hundreds of man hour efforts are needed. Considering the time framework available, we aim to analyze only simple sentences of Hindi. We have not considered compound and complex sentences for our analysis. Complex noun_phrases consisting of verbal adjective have also been kept out of the scope of this work. Inspite of these limitations this work could be a significant step towards processing Hindi text. Removing the limitations just mentioned above would involve extending the grammar used and writing appropriate procedures.

As mentioned earlier we have written Hindi grammar using Definite Clause Grammar. Since Definite Clause Grammar is directly supported by Turbo Prolog, the language chosen for the implementation is Turbo Prolog.

The next chapter i.e. chapter II deals with syntax and morphology of the Hindi. Chapter III deals with the design of syntax parser, morphological analyzers for different constituents and semantic parser. The implementation of different analyzers and parsers is described in chapter IV. The final chapter, i.e., chapter V gives a brief summary and limitations of our present work.

CHAPTER 2

HINDI GRAMMAR

As explained in chapter-1, a grammar is a formal specification of the structure allowable in the language. Like other languages, Hindi sentence can also be represented using a formal grammatical formalism. A sentence is a basic element of a language, that expresses a complete thought and is made up of words. Different kinds of ideas or concepts like physical objects, actions, relationships among ideas require different representations. These different representations are collectively called Parts of Speech. The parts of speech in Hindi are called "Shabd Vichar". Hindi grammar has following Parts of Speech (Shabda Vichar):

1. Noun (Sangya)
2. Pronoun (Sarvnaam)
3. Adjective (Visheshan)
4. Verb (Kriya)
5. Adverb (Kriya Visheshan)
6. Postposition (Parsarg)
7. Conjunction
8. Interjunction

Hindi grammar recognizes two genders - masculine and feminine (There is no concept of neutral gender in Hindi. A word denoting an inanimate object is either masculine or feminine), two numbers - singular and plural, three persons - first, second and third, and six cases (refer to chapter 1).

There are some rules in a valid Hindi sentence to specify the order of constituents (i.e. words and phrases) and their relationship with each other. The order is referred to as construction and the relationship of constituents with each other is referred to as agreement.

2.1 CONSTRUCTION RULES

The order or sequence of constituents is of importance so that the meaning of a sentence may be clear. A formal Hindi sentence follows the following construction rules :

2.1.1 Position of nominative

The general rule is that in a sentence the nominative (subject) comes first and the verb comes at the end. Other parts of speech, if any, come in between the nominative and the verb. For example, in the sentence

मैं अच्छा हूँ.

मैं is a nominative and हूँ is the verb.

2.1.2 Position of object

If the verb is transitive, the object comes in between the nominative and the verb. For example, in the sentence

वह मिठाई खाता है.

मिठाई is the object.

2.1.3 Position of adjective

Normally, adjective comes before the noun or pronoun which it qualifies, however, it may also come after the noun (or pronoun) it qualifies. For example, in the sentences

1. वह सुन्दर लड़का है. 2. वह लड़का सुन्दर है.

सुन्दर is the adjective. In both of these sentences adjective qualifies the noun लड़का. Note that in the first sentence the adjective is being used in attributive and in the second sentence in predicative manner.

If a pronominal or quantitative and an attributive adjective is used together, the former precedes the latter. For example, in the sentences

1. कोई सुन्दर लड़का आ रहा है.
2. कई सुन्दर लड़के आ रहे हैं.
3. आठ सुन्दर लड़के आ रहे हैं.

the pronominal adjectives कोई and कई and the quantitative adjective आठ precedes the attributive adjective सुन्दर.

2.1.4 Extension of the subject and predicate

A sentence is the basic unit of a language, and the essential elements of a sentence are the subject and the predicate. The subject is what the sentence is about and the predicate says something about the subject. The subject of a Hindi sentence may be:

1. A noun or a pronoun, e.g. in the sentences

1. तुलसी दास आया है.   2. वह आया है.

तुलसी दास is the noun and वह is the pronoun.

2. Two or more nouns or pronouns, e.g. in the sentence

तपस्वी और गौतमी दूसरी ओर गये.

तपस्वी and गौतमी are the nouns.

3. An adjective or numeral used substantively, e.g. in the sentence.

दो वहाँ हैं.

दो is a numeral.

4. Any phrase or sentence, e.g. in the sentence

राम का लड़का मोहन अच्छा है.

राम का लड़का मोहन is the phrase.

The predicate of a Hindi sentence may be:

1. A verb only, e.g. in the sentence

वह जायेगा

verb जायेगी is the predicate.

2. Noun or pronoun followed by a verb, e.g. in the sentence

वह राजा का है.

राजा का is the noun.

3. An adjective followed by a verb, e.g. in the sentence

वह अच्छा है.

अच्छा is the adjetive.

4. Any word or phrase used as noun followed by a verb, e.g. in the sentence

मैं राजा भीष्मक का पढ़ाया हूँ.

राजा भीष्मक का पढ़ाया हूँ is a phrase.

The extension of subject comes before the subject and the extension of predicate comes before the predicate. A subject can be extended by modifiers such as adjectives or noun followed by possessive postpositions का, के, की, called relative adjectives. A predicate can be extended by any noun, pronoun or any word or phrase used substantively [9].

2.1.5 Position of words showing Locative, Ablative, Dative and Instrumental case

These cases ordinarily come between the nominative and the verb in the order, Locative, Ablative, Dative and Instrumental. For example, in the sentence

जंगल मे जाकर पेड़ से पूजा के लिये कुल्हाड़ी से कुछ लकड़ी काट लाओ.

जंगल में is the Locative case; पेड़ से is the Ablative case;
पूजा के लिये is the Dative; कुल्हाड़ी से is the Instrumental case;
कुछ लकड़ी is the objective and काट लाओ the verb.

Often locative of time comes at the beginning, even before the nominative. Where there are two locatives one of the time, and the other of place, generally the locative of time comes before the locative of place. For example, in the sentence

जाड़े में पहाड़ पर सर्दी पड़ती है.

जाड़े में is the locative of time and comes before the the locative of place, पहाड़ पर.

The general rule is that the ablative, the dative and the instrumental come after the nominative, but this is often violated for the sake of emphasis. For example, in the sentences

1. पेड़ से पत्ते झड़ते हैं.
2. तुम्हारे लिये मैंने यह टोपी खरीदी है.
3. इस हाथ से मैं नहीं लिख सकता.

पेड़ से is the ablative case, तुम्हारे लिये is the dative case इस हाथ से is the instrumental case.

TH-3468

2.2 AGREEMENT

Various constituents in a sentence are related to each other in accordance with certain rules. Generally, their relationship takes certain specified forms in a given language. Before we discuss the general relation of words and phrases in Hindi sentences, we must discuss various morphological forms of nominals and verbals in Hindi.

DISS
681·3·06:491·43
L735
an

2.2.1 Morphology of Hindi Words

The morphological analysis is performed together with syntax analysis. The smallest unit of meaning in a language is called morpheme. A morpheme may be a separate word or made up of more than one morpheme. The study of combining morphemes to form words and deriving the morphemes from words is called morphology.

When a morpheme is a separate word which can function alone, such as the word लड़का, it is called a free morpheme. Other morphemes which are combination of morphemes, such as लड़कों the plural of लड़का, made up of the free morpheme लड़का and plural ending -ओं, are called bound morphemes.

Natural language systems may not always require morphological analysis. The alternative is to put all possible forms of every word into the dictionary. But storing all possible variant forms in the dictionary is inefficient and unnecessary, because many variant forms can be derived by simple spelling rules, such as in Hindi plurals can be formed by adding -ए or -ओं to the singular noun (e.g. लड़के or लड़कों from लड़का) and past tense by adding -या to the verbal root. Therefore a morphological analysis routine to handle regular variant can save the considerable storage space. Then only irregular forms need to be stored in the dictionary and root forms of the regular variants. It may be noted that morphology in Hindi is not as simple and straight forword as in English. Hindi words are more inflected than English words. The morphology for Hindi words is described below by classifying them in their respective constituents (part of speech).

2.2.1 Morphology of Noun

Hindi nouns have two forms - Direct and Oblique. In its oblique form a noun occurs with postposition. All masculine nouns, those ending in आ undergo no change in the plural. But there are some exceptions to this. So Hindi masculine nouns have two classes.

1. In this class masculine nouns ends in आ in their singular direct, -ए in their singular oblique and in plural direct and -ओं in their plural oblique form.

Example.

	Singular	Plural
Direct	लड़का (laraka)	लड़के (larake)
Oblique	लड़के (larake)	लड़कों (larako)

2. This class contains no distinct ending in their direct singular, direct plural and oblique singular, but they add- ओं to their plural oblique forms.

Example.

	Singular	Plural
Direct	घर (ghar)	घर (ghar)
Oblique	घर (ghar)	घरों (gharo)

This class of nouns ending in -ई shorten this vowel to -इ and insert -य before the oblique plural termination -ओं. Same way nouns ending in -ऊ shorten this vowel to -उ before the oblique termination.

Example.

	Singular	Plural
Direct	आदमी (adami) चाकू (chakoo)	आदमी (adami) चाकू (chakoo)
Oblique	आदमी (adami) चाकू (chakoo)	आदमियों (adamiyin) चाकुओं (chakoon)

Like masculine nouns, Hindi feminine nouns also have two classes.

1. This class contains feminine nouns ending in - ई in their singular direct and singular oblique, - इयां in their plural direct and -इयों in their plural oblique forms.

Example.

	Singular	Plural
Direct	लड़की (laraki)	लड़कियां (larakiyan)
Oblique	लड़की (laraki)	लड़कियों (larakiyon)

2. This class contains the Hindi feminine noun ending in -ए in their plural direct and in -ओं in their plural oblique, while singular direct and singular oblique end in any sound except -ई, -इ, -इयां.

Example.

	Singular	Plural
Direct	पुस्तक (puatak)	पुस्तकें (pustaken)
Oblique	~~पुस्तक (pustak)~~	पुस्तकों (pustakon)

All Hindi nouns are third person.

2.2.1.2 Morphology of Pronoun

A pronoun is a word used instead of a noun. Hindi pronouns can be divided into following classes :

1. Personal pronouns
2. Demonstrative pronouns
3. Relative pronouns
4. Interrogative pronouns

Hindi personal pronouns are divided into three classes, according to three persons and each class has singular and plural forms.

Example.

	Singular	Plural
I person	मैं	हम
II person	तू,तुम,आप	तुम, आप
III person	यह, वह	ये, वे

"आप" is an honorific personal pronoun used to address an elderly or a respected person or of equal status but without familiarity or acquaintance.

Declension of personal pronouns can be formed by adding case-signs to them. For example, personal pronouns followed by पर are given below.

Example.

Singular		Plural	
Direct	oblique	Direct	oblique
मैं	मुझ पर	हम	हम पर
तू	तुझ पर	तुम	तुम पर
आप	आप पर	आप	आप पर
यह	इस पर	ये	इन पर
वह	उस पर	वे	उन पर

Further sequences of Hindi personal pronouns followed by postposition को may optionally be replaced by single word contractions;

मुझ	+	को	becomes	मुझे
तुम	+	को	becomes	तुझे
इस	+	को	becomes	इसे
उस	+	को	becomes	उसे
हम	+	को	becomes	हमें
तुम	+	को	becomes	तुम्हें
इन	+	को	becomes	इन्हें
उन	+	को	becomes	उन्हें

Thus the sentence उनको बुलायें is equivalent to उन्हें बुलायें. There is no contraction of आप + को.

The third person pronouns यह(this), वह(that), ये(these), वे(those) -- in addition to serving as personal pronouns also function as Demonstrative pronouns. The oblique form of यह is इस and वह is उस, ये is इन and वे is उन. These forms can be further inflected, as illustrated above.

Relative pronouns are used in complex sentences to join one clause to another(subordinate) clause. They have direct and oblique forms. जो is the singular direct and plural direct. It's singular oblique is "जिस" and plural oblique is "जिन".

Interrogative pronouns are used in interrogative sentences. कौन(who) is the singular direct and plural direct of this pronoun. Its singular oblique forms is किस(which) and plural oblique is किन(whom). The indefinite pronoun कोई has oblique form किसी. Pronoun कुछ does not have an oblique form.

2.2.1.3 Morphology of Adjective

An adjective qualifies a noun. Hindi adjectives generally agree with noun in number, gender and case. There are two kinds of adjectives - Uninflected and inflected. Only Inflected adjectives change on account of gender and number of a noun qualified by it. Uninflected adjectives require no agreement with noun.

Masculine adjectives end in -आ in singular direct and in -ए

in plural direct, singular oblique and plural oblique. But adjectives in feminine gender end in -ई in singular direct, and remain same as singular direct in other feminine forms.

Example.

Masculine

	Singular	Plural
Direct	अच्छा	अच्छे
Oblique	अच्छे	अच्छे

Feminine

	Singular	Plural
Direct	अच्छी	अच्छी
Oblique	अच्छी	अच्छी

Hindi adjectives are classified as

1. Pronominal adjectives
2. Qualitative adjective
3. Numeral adjectives

Almost all the pronouns may be used as adjectives when they qualify nouns following them. Then they are called Pronominal adjectives.

Qualitative adjectives describe the condition or quality of a thing. For example हरा, सुन्दर etc.

Numeral adjectives are of several kinds. They may indicate definite number as एक, दो or indicate indefinite number as कम,

अधिक, कुछ etc. They may also indicate the quantity of a thing. When numerals are used in ordinal form they get inflected according to the number, gender and case of the noun to which they modify.

All ordinal numeral adjectives ending in -वाँ like पाचवाँ have their corresponding feminine and oblique forms such as पाचवीं and पाचवें. They are inflected like masculine adjectives ending in -आ.

The sense of superlative and comparative degree is denoted by some of the case-signs such as से, को, में etc.

Adjectives other than those ending in -आ do not change in the feminine. For example adjective सुन्दर have the same feminine and masculine form.

2.2..1.4 Morphology of Verb

Verb is a word that tells or asserts something about some person or thing. Hindi verbs are affected by the distinction of voice, mood, tense, gender, number, person and case. The voices are two : Active and Passive. The moods are four : Indicative, Subjunctive, Imperative and Infinitive. Hindi has three tenses : Present, Past and future. They are further divided into fifteen tenses. They are grouped into two classes- Tenses from the root and Tenses from the participle. Hindi participles are three : Imperfect, Perfect and Conjunctive. Twelve tenses are formed by means of these participles, and remaining three are from verbal root. A verbal root is that basic form which remains unchanged in various formations of the verb.

Group I - Tenses from verbal root

1. Contingent future (सम्भाव्य भविष्य)
2. Absolute future (समान्य भविष्य)
3. Imperative future (प्रत्यक्ष भविष्य)

Group II - Tenses from participle

1. Indefinite Imperfect		(सामान्य संकेतार्थ)
2. Present	,,	(सामान्य वर्तमान)
3. Past	,,	(अपूर्ण भूत)
4. Contingent	,,	(सम्भाव्य वर्तमान)
5. Presumptive	,,	(सन्दिग्ध वर्तमान)
6. Past contingent	,,	(अपूर्ण संकेतार्थ)
7. Indefinite Perfect		(सामान्य भूत)
8. Present	,,	(पूर्ण वर्तमान)
9. Past	,,	(पूर्ण भूत)
10. Contingent	,,	(सम्भाव्य भूत)
11. Presumptive	,,	(सन्दिग्ध भूत)
12. Past Contingent	,,	(पूर्ण संकेतार्थ)

Imperfect participle is formed by adding to the verbal root the syllable "ता". And perfect participle is formed by adding syllable "आ" to the verbal root. Infinitive is formed by adding "ना" to the verbal root. Hindi verbs are found in dictionary in it's infinitive form.

Example.

Root	Imperfect	Perfect	Infinitive
डर	डरता	डरा	डरना
खा	खाता	खाया	खाना

The distinction of number and gender is generally maintained. However, the distinction of gender is not found in the contingent and the imperative future. Everywhere the second and third person singular terminations are same and the first and third person plural terminations are also same.

Contingent future tense is formed by adding terminations given below to the verbal root.

Example.

Persons	I	II	III
Sing.	ऊं	ए	ए
Plur.	एं	ओ	एं

Terminations of imperative future tense and contingent future are identical except in second person singular, which is formed simply by adding अ to the verbal root or by use of verbal root only. Imperative denotes commands, request, entreaty and prohibition. Infinitive forms of verb are also used in both second person singular and plural. For examp. तुम पढ़ना.

Absolute future is formed by simply adding गा, to the contingent future terminations. And the plural form of गा, is गे and feminine is गी.

Example,

Person	I		II		III	
	Mas	Fem	Mas	Fem	Mas	Fem
Sing.	ऊँगा	ऊँगी	एगा	एगी	एगा	एगी
Plur.	एंगे	एंगी	ओगे	ओगी	एंगे	एंगी

Imperfect tense represents an action as not completed and perfect as completed. As indefinite forms do not refer to any particular time, present, past or future, they are called indefinite. In these forms simple ता, आ, या are added to the root, e.g., खाता and खाया are indefinite forms of खा (to eat). No auxiliary is used in indefinite imperfect and indefinite perfect tenses. Other ten tenses are formed by the use of auxiliary verb होना.

Example.

	Singular masculine			Plural masculine		
	I	II	III	I	II	III
Pres Imp	हूँ	है	है	हैं	हो	हैं
Past Imp	था	था	था	थे	थे	थे
Cont.Imp	होऊँ	हो	हो	हों	हो	हों
Presum imp	हूँगा	होगा	होगा	होंगे	होगे	होंगे
Past Cont Imp	होता	होता	होता	होते	होते	होते

The entries for perfect tense (*present as well as past*) are the same as given in case of imperfect tense (only the form of the root verb changes). For feminine gender inflection आ is replaced by ई and inflection ए is replaced by ई throughtout in all tenses.

There is another form of Hindi verbs called causal form or causative form. In Hindi the verb undergoes some change to give its causal form. Hindi causal verbs are of two types - First causal and Second causal.

First causals are simply formed by adding आ to the verbal root. This causal expresses imediate causation. Second causals are formed by adding वा to the verbal root. This causal expresses mediate causation of the act or state of the verb. After adding above syllables to the root, the root is called causal root. Perfect, imperfect and infinitive forms of causal verb are formed simple by adding या, ता, ना to the causal root. How one can find different forms of the causal verbs can be seen in morphology of causal verbs.

Example.

	Root	I causal	II causal
1.	पक	पकाना	पकवाना
2.	कट	कटाना	कटवाना

2.2.2. Agreement Rules

Generally the relationships of words in a Hindi sentence take the following forms:

1. Relation of verb with the subject or nominative.

2. Relation of verb with the object.

3. Relation of pronoun with noun.

4. Relation of the genetive with the word to which it is related.

5. Relation of adjective with the word qualified by it.

2.2.2.1 Agreement of verb with the nominative or subject

When the nominative is in uninflected form the verb agrees with it in gender, person and number. For example, in the sentences

1. मोहन जाता है.   2. सीता जाती है.

जाता है is third person, singular number, masculine gender and present tense of verb जाना. Verb जाती है is third person, singular number, masculine gender and present tense form of जाना.

Verb is not inflected by gender of nominative in presumptive future. For example, in the sentences

1. राम आवे.   2. सीता आवे.

आवे is the presumptive form of verb आना.

When there are two or more nominatives and all are of the same gender, person and number then verb takes plural form of the same gender.

Example.

राम, श्याम और गोपाल एक ही शिक्षक से पढ़ते हैं.

Here राम, श्याम and गोपाल are three nominatives of same number, gender and person. And verb पढ़ते हैं is masculine gender, plural number and third person (it may also be first person) and present tense form of verb पढ़ना.

When there are two nominative differing in gender the verb generally takes masculine form.

Example.

नरेश और उसकी पत्नी सिनेमा देखने गये हैं.

Here nominative नरेश and उसकी पत्नी are differing in gender. The verb गये हैं is third person, masculine gender, plural number and present tense form of verb जाना.

But when there are several nominatives, some masculine and some feminine, the verb agrees with the nominative nearest to it. Example.

आजकल मेरे घर में चार पुरुष, दस बच्चे और सात स्त्रियाँ आयीं हैं.

In this sentence verb आयीं हैं is third person, feminine gender, plural number and present tense form of आना.

2.2.2.2 Agreement of verb with the object

When ने is used after the nominative, the transitive verb agrees with the object. For example,

1. लड़के ने यह तस्वीर देखी है.
2. लड़के ने ये तस्वीरें देखी हैं.

In first sentence, verb देखी है is singular number, feminine gender, third person and present tense and object तस्वीर is third person ,singular number and feminine gender. But in second sentence the number of both object (तस्वीरें) and verb (देखी हैं) becomes plural.

When the object is with its case-sign को, the verb takes third person masculine singular form.

Example.

1. लड़के ने चित्र को देखा है.

2. लड़कों ने चित्र को देखा है.

Here in both the sentences verb देखा है is singular number, third person, masculine gender and present tense form of verb देखना and the object चित्र is masculine gender, third person and singular number.

2.2.2.3 Agreement of pronoun with noun

The pronoun agrees with the noun for which it comes, as regards person and number. For example, in the sentence

राम ने कहा कि मैं जाऊँगा.

the nominative राम is third person, singular number and pronoun मैं is first person singular number.

2.2.2.4 Agreement for genetive with the word to which it is related

The form of genetive case-sign depends on the gender and number of the word following, to which it is related in different senses. For example, in the sentences

1. राम का बच्चा. 2. राम के बच्चे. 3. राम की बच्ची.

का is masculine gender and singular number, के is masculine gender and plural number and की is feminine gender and singular number possessive case-sign. बच्चा is masculine gender and singular number, बच्चे is masculine gender and plural number and बच्ची is feminine gender and singular number.

2.2.2.5 Agreement of adjective with the word qualified by it

Adjective agrees with the noun it modifies, as regards gender and number. There are some adjectives which have no corresponding feminine forms. They are treated as adjectives of common gender. Examples,

1. अच्छा लड़का.	2. अच्छे लड़के.	3. अच्छी लड़की.
4. सुन्दर लड़का.	2. सुन्दर लड़के.	3. सुन्दर लड़की.

The adjective अच्छा ~~is~~ masculine gender and singular number, अच्छे is masculine gender and plural number and अच्छी is feminine gender and singular (plural) number. The adjective सुन्दर is common gender.

2.3 POSTPOSITION

A postposition is a word placed after a noun or a pronoun to show in what relation the person or thing denoted by it, stands in regard to something else. Hindi postpositions are like English prepositions. Since they are placed after a noun or pronoun, they are called postpositions. Postpositions are also called Parsrg. Postpositions are of two types, simple and compound. Simple postpositions consist of only one word, like से, के, में etc. Compound postpositions consist of two or more words. These postpositions are used for many different functions, ranging from the specification of case relation to temporal and spatial relations of different kinds. The first element of compound postposition is almost always के(ke), or की(ki). For example, के लिए, के बारे में, की तरफ etc.

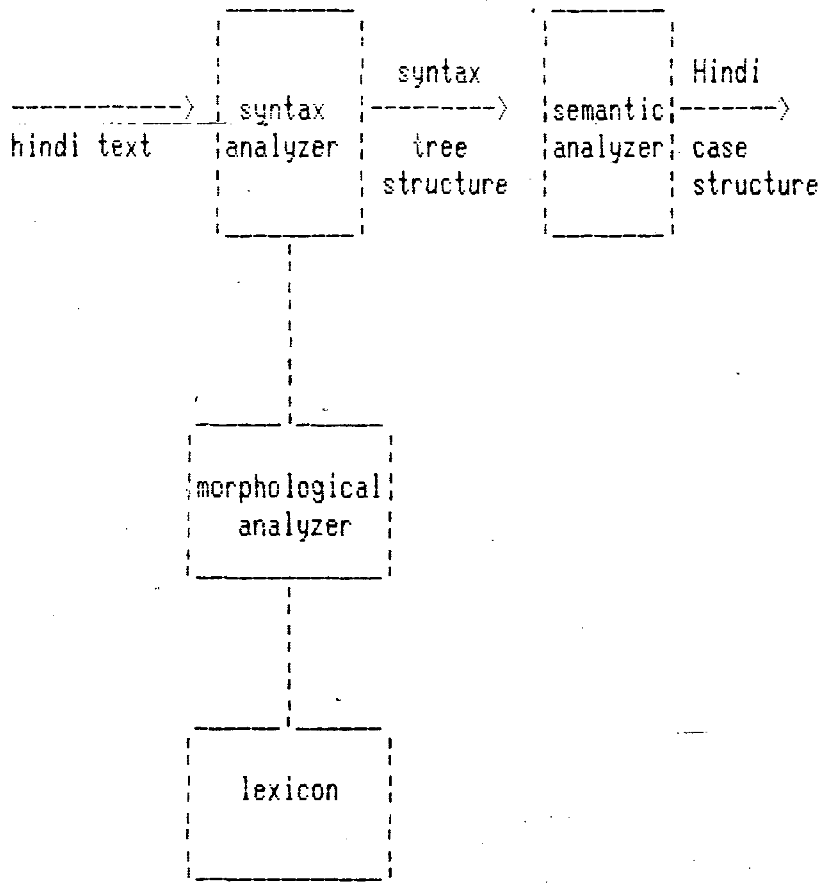
CHAPTER 3

DESIGN METHODOLOGY

As described in previous chapter, the structure of a Hindi sentence is governed by construction rules and agreement rules. In construction rules only the constituents (parts of speech) are considered whereas in agreement rules the form of these constituents are analyzed. For example, in the sentence मैं घर जाता हूँ. मैं is the subject of the sentence and घर जाता हूँ is the predicate of the sentence. This sentence is a valid Hindi sentence. As it has a valid construction, i.e. subject followed by the predicate. Similarly, तुम घर जाते हो is also a valid Hindi sentence. The difference in the form of the root verb जाना, however, in the first sentence (जाता) and in the second sentence (जाते) is due to the agreement rules which must be adhered to (refer to chapter 2). A sentence is grammatically correct if it is formed using construction and agreement rules of the given language. One can design a grammar where both the construction and the agreement rules are applied in parallel. One can also design a grammar where construction and agreement rules are applied in sequential order. The second kind of grammar turns out to be simpler for implementation. In this type of grammar the construction grammar is normally referred to as syntax grammar and the agreement grammar is referred to as morphology grammar (for performing morphological analysis). As pointed out in chapter 2 the second type of grammar needs lesser number of rules and lesser entries in the lexicon than the first type of grammar. The grammar adopted in this work is of second type. It has separate syntax analysis phase and separate morphological analysis phase.

A natural language sentence is a meaningful unit, an entity, in principle indivisible and complete in itself. Therefore, at least in principle, it should be possible to bring out the meaning of the sentence while checking its structure, i.e. syntax analysis and semantic analysis are done in parallel. Syntax analysis and semantic analysis, however, for the sake of simplicity and ease are done in separate phases which operate almost in a sequential manner. The semantic analyzer takes its input from the syntax analyzer. In the present work the syntax analyzer and the semantic analyzer are independent logical units.

Lexicon is designed basically as a data structure. Each entry in a lexicon has a word with its syntactic and semantic attributes. Lexicon can sometimes be supported by a set of procedures. Developing a lexicon in itself is a major task. Preparing a lexicon may take several man-years. In our work the function of lexicon is substituted by manual operations. Both the syntax and semantic phases depend heavily on lexicon for their fuctions. The schematic diagram of the Hindi sentence analyzer which has been adopted in the present work is given in figure below.



The subsequent sections of this chapter describe the grammar and methodology used in each phase of the sentence analyzer.

3.1 SYNTAX ANALYZER

The syntax analyzer, as already pointed out, has two phases; the parser and the morphological analyzer. The parser checks the type and the order of constituents in a sentence. A given type of a constituent may have various valid forms (refer to chapter 2). For example, लड़का, लड़के and लड़कों are three valid forms of constituent type noun. The morphological analyzer checks the validity of the form of the given constituent type. We have designed separate grammar for parser and morphological analyzer.

3.1.1 Parser

The syntax of a language can be defined by a set of rules called production rules or rewrite rules. There may exist more than one set of production rules, i.e. more than one grammar formalism may describe the given language, that is to say, there does not always exist a unique grammar for a given language. For analysing Hindi sentence the grammar which we have adopted is described by the following rules :

```
<sentence>   ::== <adverb>* <noun-phrase> <verb-phrase>
      <noun-phrase> ::== <modifier> <noun-phrase>
      <noun-phrase> ::== <noun ¦ pronoun>
      <modifier>    ::== <adverb>* <adjective>
      <adjective>   ::== <simple-adjective>  ¦
                         <relative-adjective> ¦
                         <noun> <prasarg>
     <relative-adjective> ::== <noun> <parsarg-ka>
     <verb-phrase>   ::==  <noun-phrase>* <verbs>¦
                           <simple-adjective> <verbs> ¦
                           <adverb> <verbs>
   <verbs>  ::==  <Link-verb> ¦ <main-verb>  ¦  <main-verb>
                  <help-verb>
   <main-verb> ::== <saral-dhatu> ¦ <samasik-dhatu> ¦
                    <complex-verb>
```

⟨complex-verb⟩ ::== ⟨ kriya-mool⟩ ⟨ kriya-kar⟩

⟨kriya-mool⟩ ::== ⟨noun⟩ ¦ ⟨adjective⟩ ¦ ⟨kriyangi⟩

Star symbol represents that the particular category (nonterminal) may occur zero or more times.

The grammar rules stated above contain nonterminal symbols only. The terminal symbols for the grammar reside in the lexicon. The lexicon, as already stated, is hand implemented, that is to say the system requests the user to feed the terminal symbols, whenever needed. For a fully automated system the lexicon, however must be available on line.

3.1.2 Morphological Analyzer

The function of the morphological analyzer is to provide the root form and attributes of a given word, e.g., the word लड़के has its attributes plural, masculine and root word लड़का. The morphological analyzer also provides the word form if the root of the word and its attributes are given. It also provides the list of attributes that hold good for a given root word and for the given final word form of the root word. The attributes of the word are normally referred to as morphological attributes.

In the present work, the morphological analyzer is modular in the sense that there are separate modules for noun, pronoun, adjective, adverb, verb and helping verb. The structure of each type of module is described in the following subsections.

3.1.2.1 Morphology of Noun

The root morpheme of the noun changes its form depending on the number, gender and case. In this module the data structure used is :

[transformed word, rootword, case, number, gender].

3.1.2.2 Morphology of Pronoun

The transformation of pronoun depends on number, gender, person and case. The data structure used in this module is :

[transformed word, rootword, case, number, gender, person, subcat].

Where subcat denotes the sub-category to which the pronoun belongs to, e.g. यह has the sub-category demonstrative pronoun.

3.1.2.3 Morphology of Adjective

The morphology of adjective depends on number and gender only. The data structure used for this module is :

[transformed word, rootword, number, gender].

3.1.2.4 Morphology of Verb

The morphology of verb depends on voice, tense, mood, number, gender and person. The data structure used for this module is:

[transformed word, rootword, voice, mood, tense, number, gender, person].

3.2 SEMANTIC PARSER

In the present work the semantic parser analyzes the sentence upto Kaarak (case) level. It does not analyze the sentence upto its deep semantic level.

In Hindi, kaaraks are determined mainly by parsarg associated with nominals in a sentence. A parsarg, however, may determine more than one kaarak and a kaarak may be determined by more than one parsarg. The following conditions must be satisfied for proper kaarak analysis :

1. Every nominal in the sentence must satisfy a kaarak restriction.
2. A nominal in the sentence can satisfy only one kaarak restriction.
3. All the obligatory kaarak restrictions of the verb in ~~the~~ sentence must be satisfied.

Inspite of the above kaarak restrictions, more than one interpretation of a given sentence are possible. This ambiguity can be removed only using contextual information. This is not in the scope of the present work.

The semantic parser takes as its input, the output of the syntax parser. The output of the semantic parser is a kaarak assignment list with the verb and modality.

CHAPTER 4

IMPLEMENTATION

The programming language Turbo Prolog has been chosen as the vehicle for the implementation. As pointed out earlier, Hindi sentene can be analyzed using DCG, which is directly supported by Turbo Prolog. Prolog has the advatages of backtracking, automatic reduction and pattern matching, which are very useful in Natural language processing applications. Modular programming facility provided by Prolog is also adopted here. This facility helps in writing several small modules instead of writing a big module which becomes very cumbersome in editing and error corrections. To link different modules they are projected in a single project file, named with extension PRJ. The name of the project file is included in all the modules so that they can share a single symbol table to reduce the size of code arry to make compilation successful. Otherwise it is not possible to compile a large program which exceeds the available code array size of memory.

List processing is widely used in implementation of parser for the grammar described in chapter 2. In implementation of morphological analyzer, string handling is used widely. There shouid be some way to break the whole string into words, so that the syntactic category of each word can be determined and then they can be grouped together to determine further syntactic categories. This must be done before the text is inputted to syntax analyzer. This process of tokenizing a string is sometimes called lexical analysis.

Our implementation is comprised of following modules.

1. Module handling the character strings.
2. Module handling the syntax
3. Module handling the morphology
4. Module handling the semantics

4.1 MODULE HANDLING THE CHARACTER STRINGS

The module named "strtolt.pro" converts a string into list. It reads a character at a time and inserts it into a list of characters. It keeps on reading characters and inserting them into the list until a space character or a punctuation character is encountered. This function is performed by a predicate

restword(char, char, list).

As soon as the space or any punctuation character comes, it converts the list of characters into a string or word. This is done by a predicate named

charlist_charstr(list,str)

When a word is read it is inserted into a list of strings as its head. Then the predicate

restsent(char,str,list)

reads the rest of the sentence following the same process as described to read a word. This process of reading a word continues until the value of character read is carriage return (enter key). But the character just before the carriage return must be a terminating character. A list of terminating characters is defined by the predicate named

last_list(L)

where L is a list of terminating characters

L = [',' , '|' , '.' , '?'].

The output or the list of string is then passed to the parser as its input. This module can read its input either directly from the keyboard or from a file.

4.2 MODULE HANDLING THE SYNTAX

This module is a syntax analyzer for a simple Hindi declarative sentence. All other modules are being accessed at different levels from this module. List proessing is particularly, used widely in this module. List generaated by previous module also contains punctuation characters as its elements. To delete all punctuation characters from the list a procedure is required. So the predicate named

delete_punch(list,newlist)

perform this function. Then the newlist containing no puctuation marks is passed to the analyzer. The analyzer takes a word at a time from the list to determine its syntactic category and other informations To do this function a predicate

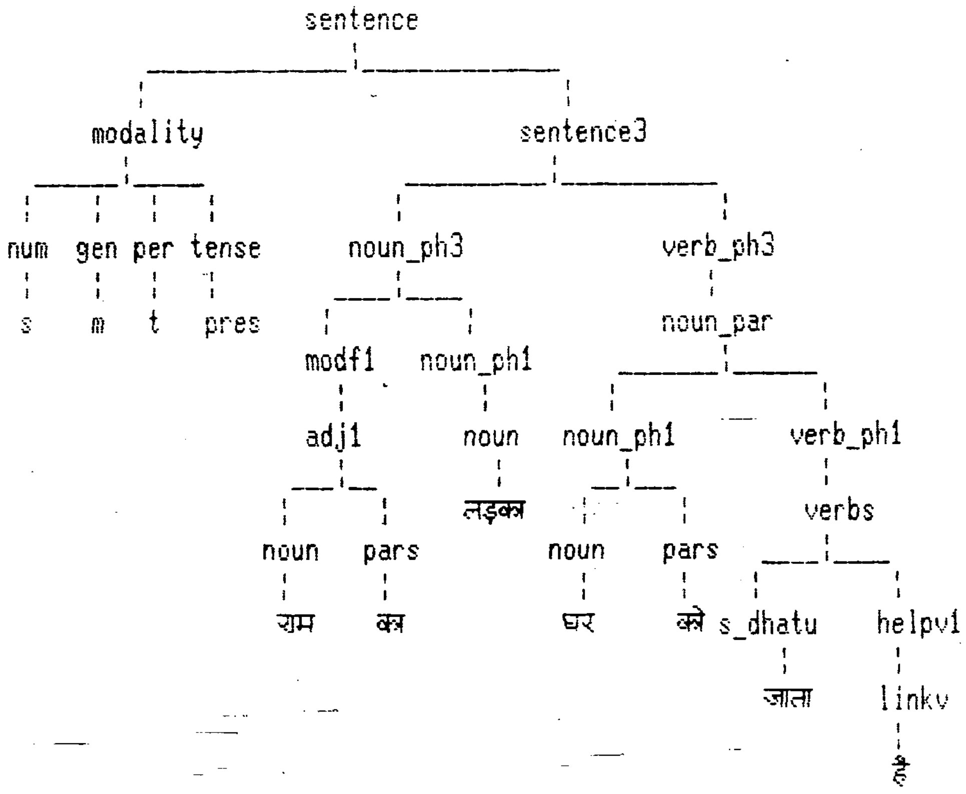
first(list,str,restlist)

is written, which takes the first element of list for analysis and generates a list of remaining elements for further analysis. This module is our main module. The tree structure is generated with the help of compound object delaration facility provided by the Prolog.

The mame of this module is "project.pro" The tree structure for a simple declarative Hindi sentence

राम का लड़का घर को जाता है.

is given as :



The representation of this tree structure in Prolog implementation is as :

```
sentence  (modality(num(s), gen(m), pers(t), tense(pres)), sentence3(
(noun_ph3(modf1(adj1(noun(राम), pars(का))), noun_ph1(noun(लड़का))),
verb_ph1  (noun_par  (noun_ph1 (noun(घर), pars(को)), verb_ph1(verbs
(s_dhatu(जाता), helpv1(linkv(है))))))).
```

4.3 MODULE HANDLING THE MORPHOLOGY

String manipulation is used very widely in this module to perform the morphological analysis. This string handling is done with the ~~help of Prolog built~~ in predicate

concat(string1,string2,string3).

This module is accessed from the parser, which passes a word or morpheme as an input to it. This module determines the root word and the morphological attributes of the input word. Lexicon is accessed from this module to determine whether the word belongs to Hindi. If the word is free morpheme the lexicon is directly accessed without performing any morphological analysis. The morphological attributes of the free morpheme are determined directly from the lexicon.

Morpholoical analysis is implemented in four different modules. These modules differ in their attributes. They are as follows :

4.3.1 Module to handle the morphology of noun

The name of this module is "morpnoun.pro". The predicate used to perform the function of this module (i.e. the morphological analysis of the noun) is

morp_noun(string, string, string, num, gen,case),

where num, gen and case are of type reference string.

4.3.2 Module to handle the morphology of pronoun

The name of this module is "morppron.pro". This module differs from the previous module that it determines the person of a pronoun. In our implementation pronouns are not stored in lexicon, i.e. it does not access the lexicon. The predicate used in this module is

morp_pronoun(string, string, string, num, gen, case, pers),

where pers is also a reference string as num, gen and case.

4.3.3 Module to handle the morphology of adjective

The name of this module is "morpadjt.pro". This module only determines the number and gender. The predicate in this module is

morp_adjt(string, string, string, num, gen).

4.3.4 Module to handle the morphology of verb

The name of this module is "morpverb.pro". The predicate used in this module is

morp_verb(string, string, string, voice, mood, num, gen, pers, tense, particp~~le~~),

where tense is also a reference string as num gen. But others are simple strings.

4.4 MODULE HANDLING THE SEMANTICS

The module named "case.pro" consists of a case analyzer. It accesses its input from the syntax module ("analyzer.pro"). It checks all the nounphrases and their associated parsargs to determine their cases. It gives as output a case list with modality.

In addition to all these modules, there are some ~~more~~ programs.

A program named "modality.pro" performs the agreement between the number, gender, person and tense of different constituents. The file named "glodoman.pro" defines a global domain which included in a file named "glopred.pro". "This glopred.pro" file is included in all the modules because it defines the global predicate.

CHAPTER 5

CONCLUSION

The present work is a very modest attempt in analysis of simple declarative Hindi sentences. The complex noun-phrases and complex and compound sentences have not been considered. This is primarily due to the limitation of time. The results of the analysis of Hindi sentences is illustrated with the help of examples shown in Appendix B. The work, however, can be very easily extended as the design of the analyzer is based on modular approach. To extend the work, the extension of the grammar given in the present work is required. The lexicon has been manually implemented here. It can be automated and provided online. Developing a lexicon itself is a big task. Processing of a natural language is a complex task and every work done in this field will always require further extension. This must continue.

```
 ****************** APPENDIX A ************

%***************** PROJECT.PRO MODULE ***********
% ************* Program for analysing a HINDI sentence ************** %

project "hindi"
include "glopred1.pro"
include "lexicon1.pro"
include "case1.pro"
predicates
          check_sent(sentence)
          check_sent1(sentence,lt)
          noun_phrase(npnode,lt,lt,lt,num,gen,pers)
          verb_phrase(vpnode,lt,lt,lt,tens,num,
                                             gen,pers)
          p_noun(tnode,lt,lt,lt,num,gen,pers)
          pars_ne(tnode,lt,lt,lt)
          modif(mdnode,lt,lt,lt,num,gen)
          adverbp(advnode,lt,lt,lt)
          adverb(advnode,lt,lt,lt,lt2,lt2,lt,lt,lt)
          adject(adjnode,lt,lt,lt,num,gen)
          s_adj(tnode,lt,lt,lt,num,gen)
          r_adj(tnode,lt,lt;lt,num,gen)
          pars_n(tnode,lt,lt,lt,num,gen)
          verbs(vnode,lt,lt,lt,tens,num,gen,pers)
          main_verb(vnode,lt,lt,lt,tens,num,gen,pers)
          link_verb(tnode,lt,lt,lt,tens,num,gen,pers)
          mainvb(vnode,lt,lt,lt,tens,num,gen,pers)
          saral_dhatu(vnode,lt,lt,lt,tens,num,gen,pers)
          samasik(vnode,lt,lt,lt,tens,num,gen,pers)
          complexv(vnode,lt,lt,lt,tens,num,gen,pers)
          pre_verb(tnode,lt,lt,lt)
          verb_kar(tnode,lt,lt,lt)
          kriyangi(tnode,lt,lt,lt)
          modal_aux(tnode,lt,lt,lt,tens,num,gen,pers,num)
          help_verb(vnode,lt,lt,lt,tens,num,gen,pers)
          pars_v(tnode,lt,lt,lt,num,gen)
          noun_phrase1(npnode,lt,lt,lt,num,gen,pers)
          noun_ph_ty1(npnode,lt,lt,lt,num,gen,pers)
          adv_adj(adnode,lt,lt,lt,num,gen)
          hyphen(t,n)
          chk_kar(t)
          chk_kri(t)
          delete_punch(lt,lt)
          %write_sent(lt)
          %write_tree(sentence)
          proper_n
          more_n
          re_try
          read_gn(c,c)
          go

    goal
          go.

    clauses

          go:-
                check_sent(S),!,nl,nl,
                write("SYNTAX ANALYSIS OF THIS "),
                write("SENTENCE IS AS FOLLOWS :"),nl,
                write(S),nl,nl,
                %write_tree(S),nl,nl,
                semant(S,Sem),
                write("CASE ANALYSIS IS AS FOLLOWS :"),nl,
                write(Sem),nl,nl,
                write("DO U WANT TO ANALYZE"),
                write(" MORE SENTENCES ANS(y/n)?"),nl,
                read_again,nl,
                go,!.
                %write_tree(S),!.
          go:-
                write("SENTENCE IS NOT CORRECT"),nl,nl,
                proper_n,
                write("DO U WANT TO ANALYZE"),
                write(" MORE SENTENCE ANS(y/n)?"),nl,nl,
                read_again,
                go,!.

          % ************ Sentence inputting starts from here ************ %
          check_sent(S):-
                write("START INPUTTING A SENTENCE :"),nl,
                read_input(L1),nl,nl,
                delete_punch(L1,L),
                check_sent1(S,L).
                %write_sent(L).

          % ************ Sentence checking starts from here ************* %
          check_sent1(S,L):-
                adverbp(S1,L1,L2,L),
                append(L1,L2,L),
               noun_phrase(S2,L3,L4,L2,N,G,P),
               append(L3,L4,L2),
               verb_phrase(S3,L4,[],L4,T,N1,G1,P1),
               %N1 = N,G1 = G,P1 = P,
               match_n(N,N1),
               match_g(G,G1),
               match_p(P,P1),
               S11 = mod(numb(N),gend(G),persn(P),
                                        tense(T)),
               S = sentence(S11,sent1(S1,S2,S3)),!.

       check_sent1(S,L):-
               noun_phrase(S1,L1,L2,L,N,G,P),
               append(L1,L2,L),
               verb_phrase(S2,L2,[],L2,T,N1,G1,P1),
               %write("\n N=",N," G=",G," P=",P),nl,
               %write("N1=",N1," G1=",G1," P1=",P1),nl,
               match_n(N,N1),
               match_g(G,G1),
               match_p(P,P1),
               S11 = mod(numb(N),gend(G),persn(P),
                                       tense(T)),
               S = sentence(S11,sent3(S1,S2)),!.

/*     check_sent1(S,L):-
               noun_phrase(S1,L1,L2,L,N,G,P),
               append(L1,L2,L),
               adverbp(S2,L3,L4,L2),write("\n done"),nl,
               append(L3,L4,L2),
               verb_phrase(S3,L4,[],L4,T,N1,G1,P1),
               write(N1," ",N," ",G1," ",G," ",P1," ",P),nl,
               match_n(N,N1),
               match_g(G,G1),
               match_p(P,P1),
               S11 = mod(numb(N),gend(G),persn(P),
                                        tense(T)),
               S = sentence(S11,sent2(S1,S2,S3)),!.
*/
       % ************ Checking for noun phrase starts from here ***
       noun_phrase(S,L1,L2,L,N,G,P):-
               modif(S1,L11,L3,L,N1,G1),
               noun_phrase(S2,L22,L2,L3,N,G,P),
               match_g(G,G1),
               match_n(N,N1),
               append(L11,L22,L1),
               S = noun_ph3(S1,S2).
       noun_phrase(S,L1,L2,L,N,G,P):-
```

```
        p_noun(S1,L11,L3,L,N,G,P),
        pars_ne(S2,L22,L2,L3),
        append(L11,L22,L1),
        S = noun_ph2(S1,S2).
noun_phrase(S,L1,L2,L,N,G,P):-
        p_noun(S1,L1,L2,L,N,G,P),
        S = noun_ph1(S1).

% **************** Checking for noun starts from here *************** %
p_noun(S,L1,L2,L,N,G,P):-
        first(X,L2,L),
        morp_noun(X,_,_,_,N,G,P),
        L1 = [X],
        S = noun(X).

% *************** Checking for pronoun starts from here ************* %
p_noun(S,L1,L2,L,N,G,P):-
        first(X,L2,L),
        morp_pronoun(X,_,_,_,N,G,P),
        L1 = [X],
        S = pnoun(X).

% ************* Checking for.modifier of a noun or pronoun starts from her
modif(S,L1,L2,L,N,G):-
        adject(S1,L1,L2,L,N,G),
        S = modf1(S1).
modif(S,L1,L2,L,N,G):-
        adverbp(S1,L11,L3,L),
        adject(S2,L22,L2,L3,N,G),
        append(L11,L22,L1),
        S = modf2(S1,S2).

% **************** Checking for adverb phrase—sarts from here *************
adverbp(S,L1,L2,L):-
        adverb(S,[],L1,_,[],_,_,L2,L).

% ************** Checking for  an adverb starts from here ************* %
adverb(S,L1i,L1,L11,Lapi,Lap,L22,L2,L):-
        first(X,L22,L),
        lex(X,"adverb",_,_,_,_,_),
        append(L1i,[X],L11),
        append(Lapi,[adv(X)],Lap),
        adverb(S,L11,L1,_,Lap,_,_,L2,L22).
adverb(S,L1i,L1,L11,Lap,Lap,L22,L2,L):-
        not(Lap=[]),
        L2 = L,
          L22 = L,
          L1 = L1i,
          L11 = L1i,
          S = advp(Lap).

% ************** Checking for adjective starts from here **************** %
adject(S,L1,L2,L,N,G):-
          s_adj(S1,L1,L2,L,N,G),
          S = adj(S1).
adject(S,L1,L2,L,N,G):-
          r_adj(S1,L1,L2,L,N,G),
          S = adj(S1).
adject(S,L1,L2,L,N,G):-
          p_noun(S1,L11,L3,L,_,_,_),
          pars_v(S2,L22,L2,L3,N,G),
          append(L11,L22,L1),
          S = adj1(S1,S2).

% *************** Checking for simple adjective starts from here ************
s_adj(S,L1,L2,L,N,G):-
          first(X,L2,L),
          morp_adj(X,_,_,_,N,G),
          L1 = [X],
          S = sadj(X).

% *************** Checking for relational adjective starts from here *******
r_adj(S,L1,L2,L,N,G):-
          p_noun(S1,L11,L3,L,_,_,_),
          pars_n(S2,L21,L2,L3,N,G),
          append(L11,L21,L1),
          S=radj(S1,S2).

% *************** Checking for parsarg (post position) starts here ********
pars_n(S,L1,L2,L,N,G):-
          first(X,L2,L),
          morp_parsarg(X,"gn",N,G),
          L1=[X],
          S=par(X).

% ***************** Checking for verb phrase starts from here **************
verb_phrase(S,L1,L2,L,T,N1,G1,P1):-
          verbs(S1,L1,L2,L,T,N1,G1,P1),
          S = verb_ph1(S1).
verb_phrase(S,L1,L2,L,T,N1,G1,P1):-
          adv_adj(S1,L11,L3,L,N,G),
          verbs(S2,L22,L2,L3,T,N1,G1,P1),
          match_g(G,G1),
          match_n(N,N1),
          append(L11,L22,L1),
          S = verb_ph2(S1,S2).
verb_phrase(S,L1,L2,L,T,N1,G1,P1):-
          noun_ph_ty1(S1,L11,L3,L,_,_,_),
          verb_phrase(S2,L21,L2,L3,T,N1,G1,P1),
          append(L11,L21,L1),
          S = verb_ph3(S1,S2).

% *************** Checking for noun phrase as a constituent of verb phrase **
 noun_ph_ty1(S,L1,L2,L,N,G,P):-
          noun_phrase1(S,L1,L2,L,N,G,P).
 noun_ph_ty1(S,L1,L2,L,N,G,P):-
          noun_phrase(S,L1,L2,L,N,G,P).
 noun_ph_ty1(S,L1,L2,L,N,G,P):-
          noun_phrase(S1,L11,L3,L,N,G,P),
          pars_v(S2,L22,L2,L3,_,_),
          append(L11,L22,L1),
          S = noun_ph4(S1,S2).

 noun_phrase1(S,L1,L2,L,N,G,P):-
          p_noun(S1,L11,L3,L,N,G,P),
          pars_v(S2,L22,L2,L3,_,_),
          append(L11,L22,L1),
          S = noun_ph2(S1,S2).

 % *************** Checking for simple adjective or adverb comes just before _..
 adv_adj(S,L1,L2,L,N,G):-
          s_adj(S1,L1,L2,L,N,G),
          S = adj2(S1).
 adv_adj(S,L1,L2,L,N,G):-
          adverbp(S1,L11,L12,L),
          s_adj(S2,L22,L2,L12,N,G),
          append(L11,L22,L1),
          S = adaj(S1,S2).
 adv_adj(S,L1,L2,L,_,_):-
          adverbp(S1,L1,L2,L),
          S = advp2(S1).

 % *************** Checking for verb starts here *************** %
 verbs(S,L1,[],L,T,N,G,P):-
          help_verb(S,L1,[],L,T,N,G,P).

 verbs(S,L1,[],L,T,N,G,P):-
          main_verb(S,L1,[],L,T,N,G,P)..
```

```
verbs(S,L1,[],L,T,N,G,P):-
        main_verb(S1,L11,L3,L,_,N,G,P),
        help_verb(S2,L21,[],L3,T,N1,G1,P1),
        match_n(N,N1),%write("\n",N,N1),
        match_g(G,G1),%write("\n",G,G1),
        match_p(P,P1),%write("\n",P,P1),
        append(L11,L21,L1),
        S = verb(S1,S2).

% **************** Checking for main verb and compound verb starts her ****
main_verb(S,L1,L2,L,T,N,G,P):-
        mainvb(S,L1,L2,L,T,N,G,P).

/*      main_verb(S,L1,L2,L,T,N,G,P):-
        mainvb(S1,L11,L3,L,T,N,G,P),
        saral_dhatu(S2,L21,L2,L3,T,N,G,P),
        append(L11,L21,L1),
        S = conjuct(S1,S2).
*/
% *************** Checking for simple main verb starts here ***************
mainvb(S,L1,L2,L,T,N,G,P):-
        saral_dhatu(S,L1,L2,L,T,N,G,P).

mainvb(S,L1,L2,L,T,N,G,P):-
        samasik(S,L1,L2,L,T,N,G,P).

mainvb(S,L1,L2,L,T,N,G,P):-
        complexv(S,L1,L2,L,T,N,G,P).

% *************** Checking for complex verb starts here **************** %
complexv(S,L1,L2,L,_,_,_,_):-
        pre_verb(S1,L11,L3,L),
        verb_kar(S2,L21,L2,L3),
        append(L11,L21,L1),
        S = comlx(S1,S2).

% ************** Checking for KRIYAMOOL starts here **************** %
pre_verb(S,L1,L2,L):-
        p_noun(S,L1,L2,L,_,_,_);
        adject(S1,L1,L2,L,_,_),
        S = prev(S1);
        kriyangi(S,L1,L2,L).

% ************** Verb kriyakar ~~is~~ checked here ***************** %
verb_kar(S,L1,L2,L):-
        first(X,L2,L),
        chk_kar(X),
        L1 = [X],
        S = kar(X).

% *************** Samasik verb is checked here **************** %
samasik(S,L1,L2,L,T,N,G,P):-
        first(X,L2,L),
        hyphen(X,M),
        frontstr(M,X,S1,S2),
        morp_verb(S1,_,_,_,_,T,N,G,P,_),
        concat("-",S3,S2),
        morp_verb(S3,_,_,_,_,N1,G1,P1,_),
        match_g(G,G1),
        match_n(N,N1),
        match_p(P,P1),
        L1 = [X],
        S = samas(X).
hyphen(S,0):-
        concat("-",_,S).
hyphen(S,N):-
        frontchar(S,_,S1),
        hyphen(S1,N1),
        N = N1 + 1.

% ************** Kriyangi is checked here ***************  %
kriyangi(S,L1,L2,L):-
        first(X,L2,L),
        chk_kri(X),
        L1 = [X],
        S = kriya(X).


% *************** Checking for main verb starts here *************** %
saral_dhatu(S,L1,L2,L,T,N,G,P):-
        first(X,L2,L),
        morp_verb(X,_,_,_,_,T,N,G,P,_),
        L1 = [X],
        S = s_dhatu(X).

% *************** Checkig for helping verb ******************** %
help_verb(S,L1,L2,L,T,N,G,P):-
        link_verb(S1,L1,L2,L,T,N,G,P),
        S = helpv1(S1).

~~help_verb~~(S,L1,L2,L,T,N,G,P):-
        modal_aux(S1,L11,L3,L,_,N1,G1,P1,_),
        link_verb(S2,L21,L2,L3,T,N,G,P),
        match_g(G1,G),
        match_n(N1,N),
        match_p(P1,P),
        append(L11,L21,L1),
        S = helpv2(S1,S2).

% *************** Checking for Modal aux verbs ***************** %
modal_aux(S,L1,L2,L,_,_,_,_,_):-
        first(X,L2,L),
        lex(X,"mod_aux",_,_,_,_,_),
        %morp_verb(X,_,_,_,_,T,N,G,P,_),
        L1 = [X],
        S = mod_aux(X).

% *************** Checking for linking verb starts here *************** %
link_verb(S,L1,[],L,T,N,G,P):-
        first(X,[],L),
        morp_lk_verb(X,_,_,T,N,G,P),
        L1 = [X],
        S = link_v(X).

% *************** Checking for parsarg (post position) starts here **********
pars_v(S,L1,L2,L,N,G):-
        first(X,L2,L),
        member(Case,["lc","ab","dt","ac"]),
        morp_parsarg(X,Case,N,G),
        L1 = [X],
        S = par(X).

% *************** Checking for parsarg (post position) starts here **********
pars_ne(S,L1,L2,L):-
        first(X,L2,L),
        member(Case,["ag","ac","dt"]),
        morp_parsarg(X,Case,_,_),
        L1 = [X],
        S = par(X).

% **************** Kriyakars are checked here **************** %
chk_kar(X):-
        member(X,["कर","किया","की","होगा",
                  "हो","होगी","दे","पड",
                  "करना","होना","देना"]).

% **************** Kriyangi words are checked here **************** %
chk_kri(X):-
        member(X,["मालूम","पेश","मना",
% *************** Punctuation marks from a list are deleted here ***********
delete_punch(L1,L2):-
        punch_list(L11),
        last_list(L22),
        union(L11,L22,L3),
        minus(L1,L3,L2).

/*      write_sent(L):-
                convls(L,A),
                openappend(xoutput,"sentence.data"),
                writedevice(xoutput),
                write(A),nl,
                closefile(xoutput).

        write_tree(S):-
                openappend(xoutput,"senttree.dat"),
                writedevice(xoutput),
                write(S),nl,
                closefile(xoutput).
*/
        read_input(L):-
                openwrite(xoutput,"temp.inp"),
                writedevice(xoutput),
                readln(X),
                write(X),
                write(13),
                closefile(xoutput),
                read_input1(L).

        read_input1(L):-
                openread(xoutput,"temp.inp"),
                readdevice(xoutput),
                read_in(L),
                closefile(xoutput).

        first(X,[],[X]):-!.
        first(X,L,[X|L]).

        proper_n:-
                write("Did U use any PROPER NOUN "),
                write("if yes ANS(y/n)?"),nl,nl,
                read_again,
                more_s,nl,
                write("Trying your sentence again"),nl,
        re_try,nl,!.
proper_n.

more_s:-
        write( "Give proper noun"),nl,
        readln(Noun),
        write("Gender(m/f): "),nl,
        readchar(Gn),
        read_gn(Gn,Gend),
        str_char(Gender,Gend),
        Gen = Gender,
        assertz(lex_n(Noun,"noun","proper",
                      Gen,"s","t")),
        save("lexicon2.pro",mybase),
        more_n.

more_n:-
        write("Have used more than one "),
        write("PROPER NOUNS in the sentence Ans(y/n)?"),nl,
        read_again,
        more_s,!.
more_n.

re_try:-
        read_input1(L1),
        delete_punch(L1,L),
        check_sent1(S,L),nl,
        write("SYNTAX ANALYSIS OF THIS "),
        write("SENTENCE IS AS FOLLOWS :"),nl,
        write(S),nl,nl,
        semant(S,Sem),
        write("CASE ANALYSIS IS AS FOLLOWS :"),nl,
        write(Sem),nl,nl,!.
re_try:-
        write("\n SENTENCE IS STILL NOT CORRECT"),nl,
        write(" there is some other reason for it"),
        nl,nl.

read_gn(Gn,Gend):-
        member(Gn,['स','श']),
        Gend='m',!.
read_gn(Gn,Gend):-
        member(Gn,['ि','इ']),
        Gend='f',!.
read_gn(Gn,Gend):-
        member(Gn,['m','f']),
                Gend=Gn.




%********* Strint to list converter *********

project "hindi"
include "glopred1.pro"
predicates
        readword(c,t,c)
        restsent(t,c,lt)
        lastword(t)
        single_char(t)
        in_word(t)
        restword(c,lc,c)


clauses
/* read a sentence*/
        read_in([W|Ws]):-
                readchar(C),
                readword(C,W,C1),
                restsent(W,C1,Ws).
/*given a word and a character after it.read i rest of the sentence.*/
        restsent(W,_,[]):-
                frontchar(W,C3,_),
                str_char(C,C3),
                lastword(C),
                !.
        restsent(_,C,[W1|Ws]):-
                        readword(C,W1,C1),
                        restsent(W1,C1,Ws).
/*read in a single word given an initial character and remembering what character ca
        readword(C,W,C1):-
                str_char(C3,C),
                single_char(C3),
                !,
                charlist_charstr([C],W),
                readchar(C1).
        readword(C,W,C2):-
                str_char(C3,C),
                in_word(C3),
                !,
                readchar(C1),
                restword(C1,Cs,C2),
                charlist_charstr([C¦Cs],W).
        readword(_,W,C2):-
                readchar(C1),
                readword(C1,W,C2).
        restword(C,[C¦Cs],C2):-
                str_char(C3,C),
                in_word(C3),
                !,
                readchar(C1),
                restword(C1,Cs,C2).
        restword(C,[],C).
/* Punctuation characters*/
        single_char(C):-
                punch_list(L),
                member(C,L).
        in_word(C):-
                member(C,["अ","आ","ा","इ","ि","ई",
                "ी","उ","ु","ऊ","ू","ए","े","ऐ",
                "ै","ओ","ो","औ","ौ","ं","ः","ऋ",
                "ृ","क","ख","ग","घ","ङ","च","छ",
                "ज","झ","ञ","ट","ठ","ड","ढ","ण",
                "त","थ","द","ध","न","प","फ","ब",
                "भ","म","य","र","ल","व","श","ष",
                "स","ह","क्ष","त्र","ज्ञ","ँ","","ॅ",
                "-","0","1","2","3","4","र्","ँ",
                "5","6","7","8","9","्"]).
                punch_list(L):-
                L = [",",";",":","?","!",".","।"].
        last_list(L):-
                L = [".","!","?","।"].
        lastword(C):-
                last_list(L),
                member(C,L).

        convls(L,A):-
                convls1(L,A,"").

        member(Head,[Head¦_]).
        member(Head,[_¦T]):-
                member(Head,T).
        append([],L2,L2).
        append([X¦L1],L2,[X¦L3]):-
                append(L1,L2,L3).
        charlist_charstr([],"").
        charlist_charstr([T¦L],S):-
        charlist_charstr(L,S1),
        frontchar(S,T,S1).

str_charlist("",[]).
str_charlist(S,[T¦L1]):-
        frontchar(S,T,S1),
        str_charlist(S1,L1).

union([],X,X).
union([X¦R],Y,Z):-
        member(X,Y),
        !,
        union(R,Y,Z).
union([X¦R],Y,[X¦Z]):-
        union(R,Y,Z).
minus([],_,[]).
minus([X¦R],Y,[X¦Z]):-
        not(member(X,Y)),
        !,
        minus(R,Y,Z).
minus([_¦R],Y,Z):-
        minus(R,Y,Z).

convls1([L¦T],A,B):-
        concat(B,L,C),
        concat(C," ",D),
        convls1(T,A,D).
convls1([],A,A).

concaten(V,Vr,Con,Sufix):-
        bound(V),
        concat(Vs,Con,V),
        concat(Vs,Sufix,Vr),!.
concaten(V,Vr,Con,Sufix):-
        bound(Vr),
        concat(Vs,Sufix,Vr),
        concat(Vs,Con,V).

read_again:-
        readchar(Char),
        member(Char,['y','ब','भ']),!.
```

```
%******* MODULE TO CHECK **************
% ****** MORPHOLOGY OF NOUN ************
% ******* "MORPNOUN.PRO" MODULE ********
project "hindi"
include "glopred1.pro"
include "lexicon1.pro"

clauses
        morp_noun(V,V,_,_,Nm,Gn,Prs):-
                lex_n(V,"noun",_,Gn,Nm,Prs),
                !.

        % ******** FORMATION OF FEMININE NOUNS *******
        % ****** To check nouns like "दुल्हा" *********

        morp_noun(V,Vr,Con,Case,"s","f","t"):-
                member(Con,["ी","िन"]),
                concaten(V,Vr,Con,"ा"),
                lex_n(Vr,"noun",_,"m","s",_),
                member(Case,["nm","ac","gn","ab"]),!.

        % To check nouns like "धोबिन" ***********
        morp_noun(V,Vr,"िन",Case,"s","f","t"):-
                concaten(V,Vr,"िन","ी"),
                lex_n(Vr,"noun",_,"m","s",_),
                member(Case,["nm","ac","dt","ab","gn"]),!.

        % ****** To check nouns like "नाइन" ********
        morp_noun(V,Vr,"इन",Case,"s","f","t"):-
                concaten(V,Vr,"इन","ई"),
                lex_n(Vr,"noun",_,"m","s",_),
                member(Case,["nm","ac","dt","ab","gn"]),!.

        % ******* To check nouns like "सोनारनी" ***********
        morp_noun(V,Vr,Con,_,"s","f","t"):-
                member(Con,["नी","णी"]),
                concat(Vr,Con,V),
                lex_n(Vr,"noun",_,"m","s",_),!.

        % ******* To check nouns like "नागिन" **********
        morp_noun(V,Vr,Con,_,"s","f","t"):-
                member(Con,["िन","िण"]),
                concat(Vr,Con,V),
                lex_n(Vr,"noun",_,"m","s",_),!.

        % ******** To check nouns like "पत्नी" **********
        morp_noun(V,Vr,"त्नी",_,"s","f","t"):-
        concaten(V,Vr,"[illegible]","[illegible]"),
        lex_n(Vr,"noun",_,"m","s",_),!.

% ******** To check nouns like "पंडिताणी" ********
morp_noun(V,Vr,Con,_,"s","f","t"):-
        member(Con,["आनी","आणी","ानी","ाणी"]),
        concat(Vr,Con,V),
        lex_n(Vr,"noun",_,"m","s",_),!.

% ****** To check nouns like "देवी,पुत्री" ***********
morp_noun(V,Vr,"ी",Case,"s","f","t"):-
        concat(Vr,"ी",V),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["nm","ab","ac",
                          "dt","gn"]),!.

% *********** DECLENSION OF NOUNS ********
% *******  To check nouns like "घोड़े" *******
morp_noun(V,Vr,"े",Case,_,"m","t"):-
        concaten(V,Vr,"े","ा"),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["dt","ag","ab","gn",
                "lc","vc","nm","ac"]),!.

% ******* To check nouns like "कुएं" ********
morp_noun(V,Vr,"ए",Case,_,"m","t"):-
        concaten(V,Vr,"ए","आ"),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["dt","ag","ab","gn",
                   "lc","vc","nm","ac"]),!.

% ******** To check nouns like "धुआँ" ********
morp_noun(V~~,Vr,~~"एं",Case,_,"m","t"):-
        concaten(V,Vr,"एं","आं"),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["dt","ag","ab","gn",
                     "lc","vc","nm","ac"]),!.

% ******** To check nouns like "बनियों" ********
morp_noun(V,Vr,"ें",Case,_,"m","t"):-
        concaten(V,Vr,"ें","ां"),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["dt","ag","ab","gn",
                 "lc","vc","nm","ac"]),!.

% ******** To check nouns like "लड़कियाँ" *******
morp_noun(V,Vr,Con,"nm","p","f","t"):-
        bound(V),
        member(Con,["ियाँ","ियां"]),
        concaten(V,Vm,Con,"ी"),
        morp_noun(Vm,Vr,"ी",_,"s","f","t"),!.

% ******** To check nouns like "लड़कियां"(VICE VERSA) *******
morp_noun(V,Vr,Con,"nm","p","f","t"):-
        bound(Vr),
        member(Con,["ियाँ","ियां"]),
        morp_noun(Vm,Vr,"ी",_,"s","f","t"),
        concaten(V,Vm,Con,"ी"),!.

% ******** To check nouns like "रोटियाँ" *********
morp_noun(V,Vr,Con,"nm","p","f","t"):-
        member(Con,["ियाँ","ियां"]),
        concaten(V,Vr,Con,"ी"),
        lex_n(Vr,"noun",_,"f","s",_),!.

% ******** To check nouns like "विधियां" ***********
morp_noun(V,Vr,Con,"nm","p","f","t"):-
        member(Con,["याँ","यां"]),
        concat(Vr,Con,V),
        lex_n(Vr,"noun",_,"f","s",_),!.

% ******** To check nouns like "भौंवें" *******
morp_noun(V,Vr,"वें","nm","p","f","t"):-
        concat(Vr,"वें",V),
        lex_n(Vr,"noun",_,"f","s",_),!.

% ******** To check nouns like "बातें,घटाएं,घटायें" *******
morp_noun(V,Vr,Con,"nm","p","f","t"):-
        member(Con,["एं","यें","ें"]),
        concat(Vr,Con,V),
        lex_n(Vr,"noun",_,"f","s",_),!.

% ******** To check nouns like "बिल्लियों" *******
morp_noun(V,Vr,"ियों",Case,"p","f","t"):-
        bound(V),
        concaten(V,Vm,"ियों","ी"),
        morp_noun(Vm,Vr,"ी",_,"s","f","t"),
        member(Case,["dt","ag","ab","gn",
                                    "lc"]),!.

% ******** To check nouns like "बिल्लियों"(VICE VERSA)
morp_noun(V.Vr,"ियों",Case,"p","f","t"):-
        bound(Vr),
        morp_noun(Vm,Vr,"ी",_,"s","f","t"),
        concaten(V,Vm,"ियों","ी"),
        member(Case,["dt","ab","gn","ag",
                                  "lc"]),!.

% ******** To check nouns like "धोबियों" *******
morp_noun(V,Vr,"ियों",Case,"p","f","t"):-
        concaten(V,Vr,"ियों","ी"),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["ac","dt","gn","ab",
                                  "lc"]),!.

% ******** To check nouns like "विधियों" *******
morp_noun(V,Vr,"यों",Case,"p",Gen,"t"):-
        concat(Vr,"यों",V),
        lex_n(Vr,"noun",_,Gen,"s",_),
        member(Case,["dt","ab","gn","ag",
                                  "lc"]),!.

% ******** To check nouns like "नाइयों" *******
morp_noun(V,Vr,"इयों",Case,"p",Gen,"t"):-
        concaten(V,Vr,"इयों","ई"),
        lex_n(Vr,"noun",_,Gen,"s",_),
        member(Case,["dt","ab","gn","ag",
                                  "lc"]),!.

% ******** To check nouns like "रातों,पुस्तकों" *******
morp_noun(V,Vr,"ों",Case,"p",Gen,"t"):-
        concat(Vr,"ों",V),
        lex_n(Vr,"noun",_,Gen,"s",_),
        member(Case,["ac","lc"]),!.

% ******** To check nouns like "कुत्तों" *******
morp_noun(V,Vr,"ों",Case,"p","m","t"):-
        concaten(V,Vr,"ों","ा"),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["ac","lc"]),!.

% ******** To check nouns like "धोबिनों" *******
morp_noun(V,Vr,"ों",Case,"p","f","t"):-
        bound(V),
        concat(Vm,"ों",V),
        morp_noun(Vm,Vr,_,_,"s","f","t"),
        member(Case,["dt","ab","ag","gn",
                                   "lc"]),!.     % ******** To check noun

% ******** To check nouns like "धोबिनों" *******
morp_noun(V,Vr,"ों",Case,"p","f","t"):-
        bound(Vr),
        morp_noun(Vm,Vr,_,_,"s","f","t"),
        concat(Vm,"ों",V),
        member(Case,["dt","ab","ag","gn",
                                   "lc"]),!.     % ******** To check noun
```

```
% ******** To check nouns like "देवताओं" *******
morp_noun(V,Vr,"ताओं",Case,"p","m","t"):-
        concat(Vr,"ताओं",V),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["ac","gn","ab",
                                "dt","lc"]),!.

% ******** To check nouns like "जोरुओं" *******
morp_noun(V,Vr,"ुओं",Case,"p","f","t"):-
        concaten(V,Vr,"ुओं","ू"),
        lex_n(Vr,"noun",_,"f","s",_),
        member(Case,["ac","ag","gn","ab"]),!.

% ******** To check nouns like "पिताओं" *******
morp_noun(V,Vr,"ओं",Case,"p",Gen,"t"):-
        concat(Vr,"ओं",V),
        lex_n(Vr,"noun",_,Gen,"s",_),
        member(Case,["gn","ab","ac","ag"]),!.

% ******** To check nouns like "चिड़ियां" *******
morp_noun(V,Vr,"ँ","nm","p","f","t"):-
        concat(Vr,"ँ",V),
        lex_n(Vr,"noun",_,"f","s",_),!.

% ******* FOR GENERAL PURPOSE NOUNS *********
% ******* to check nouns like "देवता,नम्रता" ********
morp_noun(V,Vr,"ता",Case,"s",Gen,"t"):-
        concat(Vr,"ता",V),
        lex_n(Vr,"noun",_,Gen,"s",_),
        memoer(Case,["ab","gn"]),!.

% ********* To check nouns like "चिड़िया" *********
morp_noun(V,V,"िया","nm","s","f","t"):-
        concat(_,"िया",V),
        lex_n(V,"noun",_,"f","s",_),
        !.

% ******** To check nouns like "डेरा,,इच्छा,झोला,धुआँ" ********
morp_noun(V,V,Con,Case,"s",Gen,"t"):-
        member(Con,["आ","आँ","ा","ाँ"]),
        concat(_,Con,V),
        lex_n(V,"noun",_,Gen,"s",_),
        member(Case,["nm","ac","lc"]),!.

% ******** To check nouns like "मति,कवि" *******
morp_noun(V,V,Con,Case,"s",Gen,"t"):-
```

```
        member(Con,["इ","ई","ी","ि"]),
        concat(_,Con,V),
        lex_n(V,"noun",_,Gen,"s",_),
        member(Case,["nm","ac","lc"]),!.

% ******** To check nouns like "ईश्वरत्व" ********
morp_noun(V,Vr,Con,Case,"s","m","t"):-
        member(Con,["त्व","घ"]),
        concat(Vr,Con,V),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["ac","dt","ab",
                          "gn","lc"]),!.

% ********* To check nouns like "मधु,कल्लेऊ,चड़ाओ" *********
morp_noun(V,Vr,Con,Case,"s","m","t"):-
        member(Con,["उ","ऊ","ओ","औ","व",
                    "उ","ु","ू","े","ै"]),
        concat(Vr,Con,V),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["ac","dt","ab",
                      "gn","lc"]),!.

% ******** To check nouns like "लड़कपन" ******
morp_noun(V,Vr,Con,Case,"s",Gen,"t"):-
        member(Con,["पन","पना","पा"]),
        concat(Vr,Con,V),
        lex_n(Vr,"noun",_,Gen,"s",_),
        member(Case,["ac","dt","ab",
                    "gn","lc"]),!.

% ******* To check nouns like "जलज" ********
morp_noun(V,Vr,"ज",Case,"s","m","t"):-
        concat(Vr,"ज",V),
        lex_n(Vr,"noun",_,"m","s",_),
        member(Case,["ac","dt","gn"]),!.
```

******** "MORPPRON.PRO" MODULE **********

```
project "hindi"
include "glopred1.pro"
predicates
        pn_morph(t,num,num,num,num,gen,pers)
clauses
        %  ********** pronoun morphology starts from here ***********
        morp_pronoun(Pn,_,_,_,N,G,P):-
                pn_morph(Pn,_,_,_,N,G,P),!.

        morp_pronoun(Pn1,_,_,_,N,G,P):-
                member(Pp,["ने","का","के","की","को","से"]),
                !,
                concat(Pn,Pp,Pn1),
                pn_morph(Pn,_,_,_,N,G,P).

        pn_morph("मैं","Pers pron",_,Case,"s",
                                             _,"fst"):-
                member(Case,["nm","ag"]),!.

        pn_morph("मुझे","Pers pron",_,Case,"s",
                                           _,"fst"):-
                member(Case,["ac","dt"]),!.

        pn_morph("मुझ","Pers pron",_,Case,"s",
                                           _,"fst"):-
                member(Case,["ab","lc","dt"]),!.

        pn_morph("मेरा","Pers pron",_,"gn","s",
                                        "m","fst"):-!.

        pn_morph("मेरी","Pers pron",_,"gen","s",
                                       "f","fst"):-!.

        pn_morph("हम","Pers pron",_,Case,
                                   "p",_,"fst"):-
                member(Case,["nm","ag","dt","ab",
                                        "lc","ac"]),!.

        pn_morph("हमें","Pers pron",_,Case,
                                "p",_,"fst"):-
                member(Case,["ac","dt"]),!.

        pn_morph("हमारा","Pers pron",_,"gen",
                               "p","m","fst"):-!.

        pn_morph("हमारी","Pers pron",_,"gen",
                               "p","f","fst"):-!.

        pn_morph("तू","Pers pron",_,Case,"s",
                                       _,"sec"):-
                member(Case,["nm","ag"]),!.
```

```
pn_morph("तुझ","Pers pron",_,Case,"s",
                              _,"sec"):-
        member(Case,["ac","dt","lc","ab"]),!.

pn_morph("तुझे","Pers pron",_,Case,"s",
                              _,"sec"):-
        member(Case,["ac","dt"]),!.

pn_morph("तेरा","Pers pron",_,"gn",
                   "s","m","sec"):-!.

pn_morph("तेरी","Pers pron",_,"gn",
                  "s","f","sec"):-!.

pn_morph("आप","Pers pron",_,Case,"s",
                              _,"sec"):-
        member(Case,["nm","ag","ac",
                     "dt","gn","lc"]),!.

pn_morph("तुम","Pers pron",_,Case,
                     "p",_,"sec"):-
        member(Case,["nm","ag","ac",
                    "dt","ab","lc"]),!.

pn_morph("तुम्हें","Pers pron",_,Case,
                     "p",_,"sec"):-
        member(Case,["ac","dt"]),!.

pn_morph("तुम्हारा","Pers pron",_,"gn",
                  "p","m","sec"):-!.

pn_morph("तुम्हारी","Pers pron",_,"gn",
                     "p","f","sec"):-!.

pn_morph("यह","Demo pron","Prox","nm",
                              _,_,"t"):-!.

pn_morph("इस","Demo pron","Prox",Case,
                 "s", _,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("इसे","Demo pron","Prox",Case,
                        "s",_,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph("ये","Demo pron","Prox","nm",
                        "p",_,"t"):-!.

pn_morph("इन्हें","Demo pron","Prox",
                 Case,"p",_,"t"):-
        member(Case,["ac","dt"]),!.
```

```
pn_morph("इन्हीं","Demo pron","Prox",
            Case,"p",_,"t"):-
        member(Case,["ac","dt","gn"]),!.

pn_morph("इन","Demo pron","Prox",
               Case,"p",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("इन्हों","Demo pron","pox",
                "ag","p",_,"t"):-!.

pn_morph("वह","Demo pron","Remote",
              "nm",_,_,"t"):-!.

pn_morph("उस","Demo pron","Remote",
                Case,"s",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("उसे","Demo pron","Remote",
                Case,"s",_,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph(V,"Demo pron","Remote",
                "nm","p",_,"t"):-
        member(V,["वे","वो"]),!.

pn_morph("उन","Demo pron","Remote",
                Case,"p",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("उन्हें","Demo pron","Remote",
                Case,"p",_,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph("उन्हीं","Demo pron","Remote",
                Case,"p",_,"t"):-
        member(Case,["ac","dt","gn"]),!.

pn_morph("उन्हों","Demo pron","Remote",
                "ag","p",_,"t"):-!.

pn_morph("जो","Rel pron",_,"nm",_,_,
                              "t"):-!.

pn_morph("जिस","Rel pron",_,Case,"s",
                              _,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.
```

```
pn_morph("जिसे","Rel pron",_,Case,"s",
                                  _,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph("जिन्हें","Rel pron",_,Case,
                            "p",_,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph("जिन","Rel pron",_,Case,
                            "p",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("जिन्हों","Rel pron",_,"ag",
                            "p",_,"t"):-!.

pn_morph("सो","Corel pron",_,"nm",_,
                                  _,"t"):-!.

pn_morph("कौन","Intrg pron","Defnt",
                     "nm","s",_,"t"):-!.

pn_morph("किस","Intrg pron","Defnt",
                       Case,"s",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("किसे","Intrg pron","Defnt",
                       Case,"s",_,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph("किन","Intrg pron","Defnt",
                     Case,"p",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("किन्हों","Intrg pron","Defnt",
                       "ag","p",_,"t"):-!.

pn_morph("किन्हें","Intrg pron","Defnt",
                       Case,"p",_,"t"):-
        member(Case,["ac","dt"]),!.

pn_morph("किन्हीं","Intrg pron","Defnt",
                       Case,"p",_,"t"):-
        member(Case,["ac","dt","gn"]),!.

pn_morph("कोई","Intrg pron","Indefnt",
                    "nm","s",_,"t"):-!.

pn_morph("किसी","Intrg pron","Indefnt",
                    Case,"s",_,"t"):-
        member(Case,["ac","dt","ag","gn"]),!.

pn_morph("क्या","Intrg pron","Indefnt",
                    "nm","s",_,"t"):-!.

pn_morph("क्सहे","Intrg pron","Indefnt",
                    Case,"p",_,"t"):-
        member(Case,["ac","dt","gn"]),!.

pn_morph("कुछ","Intrg pron","Indefnt",
                                _,_,_,"t"):-!.

% ************ parsarg morphology starts from here *************
morp_parsarg("को",Case,_,_):-
        member(Case,["ac","dt"]),!.

morp_parsarg(P,"lc",_,_):-
        member(P,["में","पर","तक","पे",
                                "तलक"]),!.

morp_parsarg("का","gn","s","m"):-!.

morp_parsarg("के","gn","p","m"):-!.

morp_parsarg("की","gn",_,"f"):-!.

morp_parsarg("ने","ag",_,_):-!.

morp_parsarg("से","ab",_,_):-!.

% ************* pronoun parsarg matching starts here ************
pro_par_match(Pro,Par):-
        pronounlist(L1),
        find_pos(Pro,L1,N),
        parsarglist(L2),
        find_pos(Par,L2,N).

% ************* find position of an element in the list of list ******
find_pos(P,L,1):-
        first(A,_,L),
        member(P,A),!.

find_pos(P,L,N):-
```

```
        first(A,L1,L),
        bound(A),
        find_pos(P,L1,N1),
        N = N1 + 1.

% ******* A cross list of Pronouns *****
pronounlist(L):-
        L = [["मैं"],["आप","हम","तुम"],
             ["मुझ"],["इन","उन","जिन",
             "किन","इन्हीं","उन्हीं",
             "किन्हीं"],["इस","उस","जिस",
             "किस","किसी"],["इन्हों",
             "उन्हों","जिन्हों","किन्हों"]].

% ******** A cross list of Parsargs ******
parsarglist(L):-
        L = [["","ने"],["","को","ने","से",
              "में","पर","पें"],["को","से",
              "में","पर","पें"],["का","के",
              "की","को","में","पर","से",
              "पें"],["का","के","की","से",
              "में","पर","को","पें","ने"],
*/*********** "MORPADJT.PRO" MODULE *************

project "hindi"
include "glopred1.pro"
include "lexicon1.pro"
clauses
        % ******* Predicate to check the inflection *****
        % *********** of an adjective ***********
        morp_adj(V,V,_,_,Nm,Gn):-
                lex(V,"adject",_,_,Gn,Nm,_),!.

        morp_adj(V,V,Con,"nm",Num,Gen):-
                member(Con,["आ","ा"]),
                concat(_,Con,V),
                lex(V,"adject",_,_,Gen,Num,_),!.

        % ******* To check adjective like "काले" ******
        morp_adj(V,Vr,Con,Case,Num,"m"):-
                member(Con,["े","ए"]),
                member(Sufx,["ा","आ"]),
                concaten(V,Vr,Con,Sufx),
                lex(Vr,"adject",_,_,_,Num,_),
                member(Case,["ac","dt","ab",
                   "ag","gn","lc"]),!.

        % ******* To check adjective like "काली" ******
        morp_adj(V,Vr,Con,_,_,"f"):-
                member(Con,["ी","ई"]),
                member(Sufx,["ा","आ"]),
                concaten(V,Vr,Con,Sufx),
                lex(Vr,"adject",_,_,_,_,_),!.

        % ******* To check adjective like "दुखिया" ******
        morp_adj(V,Vr,Con,_,Num,"f"):-
                member(Con,["िया","इया"]),
                member(Sufx,["ी","ई"]),
                concaten(V,Vr,Con,Sufx),
                lex(Vr,"adject",_,_,_,Num,_),!.

        % ******* To check adjective like "बायाँ" ******
        morp_adj(V,V,Con,"nm","s","m"):-
                member(Con,["आँ","ाँ"]),
                concat(_,Con,V),
                lex(V,"adject",_,_,_,_,_),!.

        % ******* To check adjective like "बायें" ******
        morp_adj(V,Vr,Con,Case,_,"m"):-
                member(Con,["ें","एं"]),
                member(Sufx,["ाँ","आँ"]),
        concaten(V,Vr,Con,Sufx),
        lex(Vr,"adject",_,_,_,_,_),
        member(Case,["ac","dt","ab",
                  "ag","gn","lc"]),!.

% ****** To check adjective like "दायीं" ******
morp_adj(V,Vr,Con,_,_,"f"):-
        member(Con,["यी","ई"]),
        member(Sufx,["ाँ","आँ"]),
        concaten(V,Vr,Con,Sufx),
        lex(Vr,"adject",_,_,_,_,_),!.

% ******** To check adjective like "दसवाँ" ********
morp_adj(V,Vr,"वाँ","nm","s","m"):-
        concat(Vr,"वाँ",V),
        lex(Vr,"adject",_,_,_,_,_),!.

morp_adj(V,Vr,"वें",Case,_,"m"):-
        concat(Vr,"वें",V),
        lex(Vr,"adject",_,_,_,_,_),
        member(Case,["ac","dt","ab",
                "ag","gn","lc"]),!.

morp_adj(V,Vr,"वीं",_,_,"f"):-
        concat(Vr,"वीं",V),
        lex(Vr,"adject",_,_,_,_,_),!.

% ******* To check superlative,comparative ******
% ******* like "श्रेष्ठकर प्रियतम पुण्यतर" *********
morp_adj(V,Vr,Con,"nm",Num,Gen):-
        member(Con,["कर","तर","तम"]),
        concat(Vr,Con,V),
        lex(Vr,"adject",_,_,Gen,Num,_),!.

% ***** To check adjectives like "तीनों" ******
morp_adj(V,Vr,"नों",Case,"p",Gen):-
        concat(Vr,"नों",V),
        lex(Vr,"adject",_,_,Gen,_,_),
        member(Case,["ac","dt","ab","ag",
                          "gn","lc"]),!.
% ****** To check adjective like "गुना" ******
morp_adj(V,Vr,"गुना",_,Num,Gen):-
        concat(Vr,"गुना",V),
        lex(Vr,"adject",_,_,Gen,Num,_),!.
```

```
****** "MORPVERB.PRO MODULE" ************

project "hindi"
include "glopred1.pro"
include "lexicon1.pro"
clauses
        % ****** CLAUSES TO CHECK MAIN VERBS *******

        morp_verb(V,V,_,_,_,Tns,Nm,Gn,Prs,_):-
                lex(V,"verb",_,Tns,Gn,Nm,Prs),!.

        % ****** To check verb infinitive like "करना,जाना" ********
        morp_verb(V,Vr,"ना","act","imp",_,
                            _,_,_,"infinitve"):-
                concat(Vr,"ना",V),
                lex(Vr,"verb",_,_,_,_,_).

        % ****** To check verbs like "लीजियेगा,दीजियेगा" ************
        morp_verb(V,Vr,Con,"act","Subjt",
                 "Imp fut",_,_,"sec",_):-
                member(Con,["ीजियेगा","ीजिये",
                          "ीजियो","ीजिएगा","ीजिए"]),
                member(Vs,["ल","द"]),
                concaten1(V,Vr,Con,"े",Vs),
                lex(Vr,"verb",_,_,_,_,_).

        % ******** To check verbs like "कीजिएगा" *********
        morp_verb(V,Vr,Con,"act","sjt",
                    "Imp fut",_,_,"sec",_):-
                member(Con,["ीजियेगा","ीजिये",
                       "ीजियो","ीजिएगा","ीजिए"]),
                concaten1(V,Vr,Con,"र","क्"),
                lex(Vr,"verb",_,_,_,_,_).

        % ******** To check verbs like "पीजियेगा" ******
        morp_verb(V,Vr,Con,"act","sjt",
                    "Imp fut",_,_,"sec",_):-
                member(Con,["जियेगा","जिये","जियो",
                               "जिएगा","जिए"   ]),
                concat(Vr,Con,V),
                lex(Vr,"verb",_,_,_,_,_).

        % ******* To check verbs like "खाइयेगा" ******
        morp_verb(V,Vr,Con,"act","ipf",
                   "Imp fut","p",_,"sec",_):-
                member(Con,["इयेगा","ियेगा","इएगा",
                                   "िएगा"]),
          concat(Vr,Con,V)
          lex(Vr,"verb",_,_,_,_,_).

% ******** To check verbs like "खाइये","खाइए" ********
morp_verb(V,Vr,Con,"act","sjt",
              "Imp fut","p",_,"sec",_):-
          member(Con,["इये","िये","इए","िए",
                                   "इयो","ियो"]),
          concat(Vr,Con,V),
          lex(Vr,"verb",_,_,_,_,_).

% ********* Different forms of verb "जाना" *********
% ********* in the past tense **************
morp_verb("गया","जा","गया","act","ind",Tens,
                        "s","m",_,"pf"):-
          lex("जा","verb",_,_,_,_,_),
          member(Tens,["Ind perf","prs perf",
          "pt perf","cnt perf","prsm perf",
          "pt cnt perf"]).

morp_verb("गये","जा","गये","act","ind",Tens,
                   "p","m",_,"pf"):-
          lex("जा","verb",_,_,_,_,_),
          member(Tens,["Ind perf","prs perf",
          "pt perf","cnt perf","prsm perf",
          "pt cnt perf"]).

morp_verb("गयी","जा","गयी","act","ind",Tens,
                   "s","f",_,"pf"):-
          lex("जा","verb",_,_,_,_,_),
          member(Tens,["Ind perf","prs perf",
          "pt perf","cnt perf","prsm perf",
          "pt cnt perf"]).

morp_verb("गयीं","जा","गयीं","act","ind",Tens,
                   "p","f",_,"pf"):-
          lex("जा","verb",_,_,_,_,_),
          member(Tens,["Ind perf","prs perf",
          "pt perf","cnt perf","prsm perf",
          "pt cnt perf"]).

% ******* To check verbs like "पाऊंगा,पाऊंगी" *********
morp_verb(V,Vr,Co,"act","ipf","abs",
              "s","m","fst",_):-
          member(Co,["ऊंगा","ूंगा"]),
          concat(Vr,Co,V),write(Vr),nl,
          lex(Vr,"verb",_,_,_,_,_).

morp_verb(V,Vr,Co,"act","ipf","abs",
                 "s","f","fst",_):-
          member(Co,["ऊँगी","ूँगी"]),
          concat(Vr,Co,V),
          lex(Vr,"verb",_,_,_,_,_).

% ******** To check verbs like "आएगा,सोएगी" ******
morp_verb(V,Vr,Co,"act","ipf","abs",
                 "s","m",Pers,_):-
          member(Co,["एगा","येगा","ेगा"]),
          concat(Vr,Co,V),
          lex(Vr,"verb",_,_,_,_,_),
          member(Pers,["t","sec"]),
          !.

morp_verb(V,Vr,Co,"act","ipf","abs",
                 "s","f",Pers,_):-
          member(Co,["एगी","येगी","ेगी"]),
          concat(Vr,Co,V),
          lex(Vr,"verb",_,_,_,_,_),
          member(Pers,["t","sec"]).

morp_verb(V,Vr,Co,"act","ipf","abs",
                 "p","m",Pers,_):-
          member(Co,["एंगे","येंगे","ेंगे"]),
          concat(Vr,Co,V),
          lex(Vr,"verb",_,_,_,_,_),
          member(Pers,["fst","t"]).

morp_verb(V,Vr,Co,"act","ipf","abs",
                 "p","f","t",_):-
          member(Co,["एंगी","येंगी","ेंगी"]),
          concat(Vr,Co,V),
          lex(Vr,"verb",_,_,_,_,_).

% ******** To check verbs like "पाओगे" ******
morp_verb(V,Vr,Co,"act","ipf","abs",
                 "p","m","sec",_):-
          member(Co,["ओगे","ोगे"]),
          concat(Vr,Co,V),
          lex(Vr,"verb",_,_,_,_,_).

morp_verb(V,Vr,Co,"act","ipf","abs",
                 "p","f","sec",_):-
          member(Co,["ओगी","ोगी"]),
          concat(Vr,Co,V),
         lex(Ur,"verb",_,_,_,_,_).

% ****** To check verbs like "लिखूँ" ******
morp_verb(U,Ur,Co,"act","sjt",Tens,
               "s",_,"fst",_):-
         member(Co,["ऊँ","ूँ"]),
         concat(Ur,Co,U),
         lex(Ur,"verb",_,_,_,_,_),
         member(Tens,["cnt future","ipf"]).

morp_verb(U,Ur,Co,"act","sjt",Tens,
                    "s",_,Pers,_):-
         member(Co,["ए","े"]),
         concat(Ur,Co,U),
         lex(Ur,"verb",_,_,_,_,_),
         member(Tens,["cnt future","ipf"]),
         member(Pers,["sec","t"]).

morp_verb(U,Ur,Co,"act","sjt",Tens,
               "p",_,Pers,_):-
         member(Co,["एं","ें"]),
         concat(Ur,Co,U),
         lex(Ur,"verb",_,_,_,_,_),
         member(Tens,["cnt future","ipf"]),
         member(Pers,["fst","t"]).

morp_verb(U,Ur,Co,"act","sjt",Tens,
               "p",_,"sec",_):-
         member(Co,["ओ","ो"]),
         concat(Ur,Co,U),
         lex(Ur,"verb",_,_,_,_,_),
         member(Tens,["cnt future","ipf"]).

% ******* Markers of Imperfect participle ********
% ******* Ex. "गिरता,चलते,गाती,हँसती" *******
morp_verb(U,Ur,"ता","act","indive",Tens,
             "s","m",_,"ipf"):-
         concat(Ur,"ता",U),
         lex(Ur,"verb",_,_,_,_,_),
         member(Tens,["Ind ipf","prs ipf",
         "pt impf","cnt impf","prs impf",
         "pt cnt impf"]).

morp_verb(U,Ur,"ती","act","indive",
           Tens,_,"f",_,"ipf"):-
         concat(Ur,"ती",U),
         lex(Ur,"verb",_,_,_,_,_),
         member(Tens,["Ind impf","prs impf",
         "pt impf","cnt impf","prs impf",
         "pt cnt impf"]).

morp_verb(U,Ur,"ते","act","indive",
           Tens,"p",_,_,"ipf"):-
         concat(Ur,"ते",U)
```

```
        lex(Vr, verb ,_,_,_,_,_),
        member(Tens,["Ind impf","prs impf",
        "pt impf","cnt impf","prs impf",
        "pt cnt impf"]).

/*      morp_verb(V,Vr,"तीं","act","indive",
                    Tens,"p","f",_,"ipf"):-
        concat(Vr,"तीं",V),
        lex(Vr,"verb",_,_,_,_,_),
        member(Tens,["Ind impf","prs impf",
        "pt impf","cnt impf","prs impf",
        "pt cnt impf"]).
*/
        % ***** Perfect participle forms ********
        % *********** of the verb "क्रिया" *********
        morp_verb(V,Vr,"िया","act","ind",
                        Tens,_,_,_,"pf"):-
        concateni(V,Vr,"िया","र","क"),
        lex(Vr,"verb",_,_,_,_,_),
        member(Tens,["Ind perf","Pers perf",
        "pt perf","cnt perf","prsm perf ",
        "pt cnt perf"]).

        % ****** Perfect forms of the "दिया,पिया" ********
        morp_verb(V,Vr,"िया","act","ind",
                        Tens,_,_,_,"pf"):-
        member(Sufix,["े","प"]),
        concateni(V,Vr,"िया",Sufix,_),
        lex(Vr,"verb",_,_,_,_,_),
        member(Tens,["Ind perf","Pers perf",
        "pt perf","cnt perf","prsm perf ",
        "pt cnt perf"]).

        morp_verb(V,Vr,Con,"act","indive",
              Tens,"s","m",_,"pf"):-
        member(Con,["यीं","ई","ईया",
                          "पं"]),
        concat(Vr,Con,V),
        lex(Vr,"verb",_,_,_,_,_),
        member(Tens,["Ind perf","prs perf",
              "pt perf","cnt perf",
        "prsm perf","pt cnt perf"]).

        morp_verb(V,Vr,Con,"act","indive",
              Tens,"s","m",_,"pf"):-
        member(Con,["या","आ","T"]),
        concat(Vr,Con,V),
        lex(Vr,"verb",_,_,_,_,_),
        member(Tens,["Ind perf","prs perf",
                  "pt perf","cnt perf",
             "prsm perf","pt cnt perf"]).

morp_verb(V,Vr,Con,"act","indive",
             Tens,"s","f",_,"pf"):-
             member(Con,["यीं","ईं","ीं"]),
             concat(Vr,Con,V),
             lex(Vr,"verb",_,_,_,_,_),
             member(Tens,["Ind perf","prs perf",
                  "pt perf","cnt perf",
             "prsm perf","pt cnt perf"]).

morp_verb(V,Vr,Con,"act","ind",
             Tens,"p","m",_,"pf"):-
             member(Con,["ये","ए","े"]),
             concat(Vr,Con,V),
             lex(Vr,"verb",_,_,_,_,_),
             member(Tens,["Ind perf","prs perf",
                  "pt perf","cnt perf",
             "prsm perf","pt cnt perf"]).

% ****** Clauses to get the root of verb "होना" start here *****
morp_hp_verb(V,"हो",Con,"act","sjt";
                    Tens,"s",_,"fst",_):-
             concat("हो",Con,V),
             member(Con,["ऊँ","ँ"]),
             member(Tens,["cnt fut","ipf fut"]),
             !.
             morp_hp_verb(V,"हो",Con,"act","sjt",
                    Tens,"p",_,Pers,_):-
             concat("हो",Con,V),
             member(Con,["एं","वें","यें","ं"]),
             member(Tens,["cnt fut","ipf fut"]),
             member(Pers,["t","fst"]).

morp_hp_verb(V,"हो",Con,"act","sjt",
                    Tens,"s",_,Pers,_):-
             concat("हो",Con,V),
             member(Con,["ए","वे","य",""]),
             member(Tens,["cnt fut","ipf fut"]),
             member(Pers,["sec","t"]).

morp_hp_verb(V,"हो",Con,"act","sjt",
                    Tens,"p",_,"sec",_):-
             concat("हो",Con,V),
             member(Con,["ओ",""]),
             member(Tens,["cnt fut","ipf fut"]),
             !.

morp_hp_verb(V,Vr,Con,"act","ipf",
             "ipf fut","s",_,"sec",_):-
```

```
          member(Con,["ूजियो","ूजियेगा",
                      "ूजिये","ूजो","ूजे"]),
          concaten1(V,Vr,Con,"ने",_),
          lex(Vr,"verb",_,_,_,_,_).

morp_hp_verb(V,"हो",Con,"act","ipf",
           "abs fut","s","m","fst",_):-
          concat("हो",Con,V),
          member(Con,["ऊंगा","ूँगा"]),
          !.

morp_hp_verb(V,"हो",Con,"act","ipf",
           "abs fut","s","f","fst",_):-
          concat("हो",Con,V),
          member(Con,["ऊंगी","ूँगी"]),
          !.

morp_hp_verb(V,"हो",Con,"act","ipf",
           "abs fut","p","m",Pers,_):-
          concat("हो",Con,V),
          member(Con,["एंगे","वेंगे","ँयगे",
                      "ँगे"]),
          member(Pers,["fst","t"]).

morp_hp_verb(V,"हो",Con,"act","ipf",
           "abs fut","p","f",Pers,_):-
          concat("हो",Con,V),
          member(Con,["एंगी","वेंगी","ँयगी",
                      "ँगी"]),
          member(Pers,["fst","t"]).

morp_hp_verb(V,"हो",Con,"act","ipf",
             "abs fut","s","m",Pers,_):-
          concat("हो",Con,V),
          member(Con,["एगा","वेगा","येगा",
                      "गा"]),
          member(Pers,["sec","t"]).

morp_hp_verb(V,"हो",Con,"act","ipf",
             "abs fut","s","f",Pers,_):-
          concat("हो",Con,V),
          member(Con,["एगी","वेगी","येगी",
                      "गी"]),
          member(Pers,["sec","t"]).

morp_hp_verb(V,"हो",Con,"act","ipf",
       "abs fut","p","m","sec",_):-
          concat("हो",Con,V),
          member(Con,["ओगे","गे"]),
          !.

morp_hp_verb(V,"हो",Con,"act","ipf",
        "abs fut","p","f","sec",_):-
           concat("हो",Con,V),
           member(Con,["ओगी","गी"]),
           !.

     morp_hp_verb(V,"हो","ता","act","ind",
                    Tens,"s","m",_,"ipf"):-
           concat("हो","ता",V),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).

     morp_hp_verb(V,"हो","ती","act","ind",
                    Tens,_,"f",_,"ipf"):-
           concat("हो","ती",V),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).

     morp_hp_verb(V,"हो","ते","act","ind",
                    Tens,"p","m",_,"ipf"):-
           concat("हो","ते",V),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).

/*   morp_hp_verb(V,"हो","तीं","act","ind",
                    Tens,"p","f",_,"ipf"):-
           concat("हो","तीं",V),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).
*/
     morp_hp_verb(V,"हो","ुआ","act","ind",
                    Tens,"s","m",_,"pft"):-
           concaten1(V,"हो","ुआ","ो",_),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).

     morp_hp_verb(V,"हो","ुई","act","ind",
                    Tens,"s","f",_,"pf"):-
           concaten1(V,"हो","ुई","ो",_),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).

     morp_hp_verb(V,"हो","ुए","act","ind",
                    Tens,"p","m",_,"pf"):-
           concaten1(V,"हो","ुए","ो",_),
           member(Tens,["idf impf","prs impf",
                               "pt impf"]).

     morp_hp_verb(V,"हो","ुईं","act","ind",
                    Tens,"p","f",_,"pf"):-
           concaten1(V,"हो","ुईं","ो",_),
           member(Tens,["idf impf","prs impf",
                        "pt impf"]).

% ***** Clauses to get different attributes of verb "है" and "था"
morp_lk_verb("हूँ","act","ind",
         "pres","s",_,"fst").

morp_lk_verb("है","act","ind","pres",
                        "s",_,Pers):-
         member(Pers,["t","sec","fst"]).

morp_lk_verb("हैं","act","ind","pres",
                    "p",_,Pers):-
          member(Pers,["fst","t"]).

morp_lk_verb("हो","act","ind","pres",
                          "p",_,"sec").

morp_lk_verb("था","act","ind","pt","s",
                               "m",_).

morp_lk_verb("थी","act","ind","pt","s",
                            "f",_).

morp_lk_verb("थे","act","ind","pt","p",
                         "m",_).

morp_lk_verb("थीं","act","ind","pt","p",
                            "f",_).

concaten1(V,Vr,Con,Sufx,Vs):-
        bound(V),
        concat(Vs,Con,V),
        concat(Vs,Sufx,Vr).
concaten1(V,Vr,Con,Sufx,Vs):-
        bound(Vr),
        concat(Vs,Sufx,Vr),
        concat(Vs,Con,V).
/*      member(X,[X|_]).
        member(X,[_|Y]):-
```

```
                P1=" ".% ********** "CASE.PRO" MODULE ******************
/* **** Program to identify different cases ***** */
/* *********** involved in a sentence *********** */

include "doman1.pro"
predicates
        semant(sentence,sem)
        sent_ty(sent,propos)
        vp_ty(vpnode,caselt)
        np_ty1(npnode,case)
        check_case(tnode,npnode,case)
        append1(caselt,caselt,caselt)

clauses
        semant(sentence(Mod,Sent),Sem):-
                sent_ty(Sent,Sem1),
                Sem = sent_sem(Mod,Sem1).

        sent_ty(sent1(_,Np,Vp),Sem1):-
                vp_ty(Vp,Sem2),
                append1([karta(Np)],Sem2,Sem3),
                Sem1 = propos(Sem3),!.

        sent_ty(sent3(Np,Vp),Sem1):-
                vp_ty(Vp,Sem2),
                append1([karta(Np)],Sem2,Sem3),
                Sem1 = propos(Sem3).

        vp_ty(verb_ph1(_),[]).

        vp_ty(verb_ph3(Np,Np1),Sem2):-
                np_ty1(Np,Sem1),
                vp_ty(Np1,Sem3),
                append1([Sem1],Sem3,Sem2).

        vp_ty(verb_ph2(_,_),[]).

        np_ty1(noun_ph2(Np,Prs),Sem2):-
                Y = noun_ph2(Np,Prs),
                check_case(Prs,Y,Sem2).

        np_ty1(noun_ph1(X),Sem2):-
                Sem2 = karm(noun_ph1(X)).

        np_ty1(noun_ph3(M,A),Sem2):-
                Sem2 = karm(noun_ph3(M,A)).

        np_ty1(noun_ph4(A,Prs),Sem2):-
```

```
                Y = noun_ph4(A,Prs),
                check_case(Prs,Y,Sem2).

        check_case(par(X),Y,Ty):-
                X = "को",
                Ty = karm(Y),!.

        check_case(par(X),Y,Ty):-
                X = "से",
                Ty = karan(Y),!.

        check_case(par(X),Y,Ty):-
                X = "से",
                Ty = apdan(Y),!.

        check_case(par(X),Y,Ty):-
                member(X,["में","पर","पे","तक"]),
                Ty = adkaran(Y).

        check_case(par(X),Y,Ty):-
                member(X,["के लिए","के लिये"]),
                Ty = spdan(Y).

        append1([],L2,L2).
        append1([X¦L1],L2,[X¦L3]):-
                append1(L1,L2,L3).

/*      member(X,[X¦_]).
        member(X,[_¦Y]):-
                member(X,Y).
        */
```

```
project "hindi"
include "glopred1.pro"
clauses
        match_g(G,G1):-
                free(G),
                free(G1),!.
        match_g(G,G1):-
                G=G1,!.
        match_g(G,G1):-
                free(G1),
                bound(G),!.
        match_g(G,G1):-
                free(G),
                bound(G1),
                G=G1.
        match_g(G,_):-
                G=" ".
        match_g(_,G1):-
                G1=" ".

        match_n(N,N1):-
                free(N),
                free(N1),!.
        match_n(N,N1):-
                N=N1,!.
        match_n(N,N1):-
                free(N1),
                bound(N),!.
        match_n(N,N1):-
                free(N),
                bound(N1),
                N=N1.
        match_n(N,_):-
                N=" ".
        match_n(_,N1):-
                N1=" ".

        match_p(P,P1):-
                free(P),
                free(P1),!.
        match_p(P,P1):-
                P = P1,!.
        match_p(P,P1):-
                free(P1),
                bound(P),!.
        match_p(P,P1):-
                free(P),
                bound(P1).
```

```
match_p(P,_):-
        P=" ".
match_p(_,P1):-
```

```
        %********** " DOMAIN.PRO" **********
% ******** Global domain declaration *******

        sentence = sentence(mod,sent)
        mod = mod(numb,gend,persn,tense)
        numb = numb(num)
        gend = gend(gen)
        persn = persn(pers)
        tense = tense(string)
        sent = sent1(advnode,npnode,vpnode);sent2(npnode,advnode,vpnode);sent3(npnode,vpnode)
        %nty2 = noun_ph4(npnode);noun_ph5(npnode,tnode)
        npnode = noun_ph1(tnode);noun_ph2(tnode,tnode);
                 noun_ph3(mdnode,npnode);noun_ph4(npnode,tnode)
        mdnode = modf1(adjnode);modf2(advnode,adjnode);
                 modf(adjnode,tnode)
        adnode = advp2(advnode);adj2(tnode);
                  adaj(advnode,tnode)
        advnode = advp(lt2);advp1(advnode,tnode)
        adjnode = adj(tnode);adj1(tnode,tnode)
        tnode =  noun(t);pnoun(t);par(t);sadj(t);
                 kar(t);kriya(t);prev(adjnode);
                 link_v(t);radj(tnode,tnode);mod_aux(t)
        vpnode = verb_ph1(vnode);
                 verb_ph2(adnode,vnode);
                 verb_ph3(npnode,vpnode)
        vnode = s_dhatu(t);comlx(tnode,tnode);
               samas(t);conjuct(vnode,vnode);
               verb(vnode,vnode);helpv1(tnode);
               helpv2(tnode,tnode)
        case = karta(npnode);karm(npnode);
               karan(npnode);adkaran(npnode);
               apdan(npnode);spdan(npnode)
        caselt = case*
        propos = propos(caselt)
        sem = sent_sem(mod,propos)
```

```
%********** "GLOPRED.PRO" *********
%******************** Program for analysing a HINDI sentence ************* %
global domains
        n = integer
        lc = char*
        c = char
        t = string
        lt = t*
        lt1 = lt*
        lt2 = adv*
        adv = adv(t)
        file = xoutput
        num,gen,pers,tens = reference string
global predicates
        read_again
        first(t,lt,lt) - (o,o,i) (o,i,i)
        first(lt,lt1,lt1) - (o,o,i) (o,i,i)
        first(c,lc,lc) - (o,o,i) (o,i,i)
        nondeterm append(lt,lt,lt) - (i,i,i) (i,i,o) (i,o,i) (o,i,i) (o,o,i)
        nondeterm append(lt1,lt1,lt1) - (i,i,i) (i,i,o) (i,o,i) (o,i,i) (o,o,i)
        nondeterm append(lt2,lt2,lt2) - (i,i,i) (i,i,o) (i,o,i) (o,i,i) (o,o,i)
        nondeterm append(lc,lc,lc) - (i,i,i) (i,i,o) (i,o,i) (o,i,i) (o,o,i)
        nondeterm member(c,lc) - (i,i) (o,i)
        nondeterm member(string,lt) - (i,i) (o,i)
        nondeterm str_charlist(t,lc) - (i,o)
        charlist_charstr(lc,t) - (i,o)
        union(lt,lt,lt) - (i,i,o)
        union(lc,lc,lc) - (i,i,o)
        minus(lt,lt,lt) - (i,i,o)
        minus(lc,lc,lc) - (i,i,o)
        deleteall(t,lt,lt) - (i,i,o)
        read_in(lt) - (o)
        read_input(lt) - (o)
        read_input1(lt) - (o)
        punch_list(lt) - (o)
        last_list(lt) - (o)
        convls(lt,t) - (i,o)
        convls1(lt,t,t) - (i,o,i)
        morp_noun(t,t,num,num,num,gen,pers) - (i,o,o,o,o,o,o)
        (i,o,i,e,i,i,i) (i,o,o,o,i,i,i)
        % ********** pred for the morphology of pronoun **********
        morp_pronoun(string,num,num,num,num,gen,pers)-(i,o,o,o,o,o,o)
        % ********** pred for parsarg morphology *********
        morp_parsarg(string,num,num,gen)-(i,o,o,o) (i,o,o,i) (i,i,o,o)
        % ********** to check, the parsarg following pronoun is appropriate or not
        pro_par_match(t,t)-(i,i)
        % ********* finds position of an element in the list of list *********
        find_pos(t,lt1,n)-(i,i,o) (o,i,i) (i,i,i) (o,i,o)
```

```
% ********* cross lists of pronoun and parsarg *******
pronounlist(lt1) - (o)
parsarglist(lt1) - (o)
morp_adj(t,t,num,num,num,gen) - (i,o,o,o,o,o)
% ********** To get different attributes of main verb
nondeterm morp_verb(t,t,num,num,num,tens,num,gen,pers,num) - (i,o,o,o,o,o,o,o,c
(i,o,o,o,o,i,i,i,i,o)

% ******** To get different attributes of verb  helping "होना"
nondeterm morp_hp_verb(t,t,num,num,num,tens,num,gen,pers,num) - (i,o,o,o,o,o,o,

% ******** To get different attributes of linking verb है,था
nondeterm morp_lk_verb(t,num,num,tens,num,gen,pers) - (i,o,o,o,o,o,o) (i,o,o,i
concaten(t,t,string,t) - (i,o,i,i) (o,i,i,i)
nondeterm concaten1(t,t,string,t,t) - (i,o,i,i,o) (o,i,i,i,o) (i,o,i,i,i)
             (i,i,i,i,o)
nondeterm match_g(gen,gen) -(i,i) (o,i) (i,o) (o,o)
nondeterm match_n(num,num) -(i,i) (o,i) (i,o) (o,o)
nondeterm match_p(pers,pers) =(i,i) (o,i) (i,o) (o,o)
%lex_n(t,num,num,gen,num,pers) - (i,i,o,o,o,o)
%lex(t,num,num,tens,gen,num,pers) - (i,i,o,o,o,o,o)
```

```
%************ A sample of Lexicon *******
global database - mybase
        lex_n(t,num,num,gen,num,pers)
        lex(t,num,num,tens,gen,num,pers)

clauses
        lex_n("लड़का","noun","common","m","s","t").

        lex_n("कुआँ","noun","common","m","s","t").

        lex_n("घर","noun","common","m","s","t").

        lex_n("राजा","noun","common","m","s","t").

        lex_n("आदमी","noun","common","m","s","t").

        lex_n("चाकू","noun","common","m","s","t").

        lex_n("अध्यापक","noun","common","m","s","t").

        lex_n("कमरा","noun","common","m","s","t").

        lex_n("कुत्ता","noun","common","m","s","t").

        lex_n("केला","noun","common","m","s","t").

        lex_n("दरवाजा","noun","common","m","s","t").

        lex_n("पानी","noun","common","m","s","t").

        lex_n("फूल","noun","common","m"," ","t").

        lex_n("बच्चा","noun","common","m","s","t").

        lex_n("मित्र","noun","common","m","s","t").

        lex_n("रुपया","noun","common","m","s","t").

        lex_n("धोबी","noun","common","m","s","t").

        lex_n("विधि","noun","common","f","s","t").

        lex_n("घोड़ा","noun","common","m","s","t").

        lex_n("पुस्तक","noun","common","f","s","t").

        lex_n("पिता","noun","common","m","s","t").

        lex_n("माता","noun","common","f","s","t").
```

```
lex("[illegible]","adject"," "," ","m","s"," ").

lex("बड़ा","adverb"," "," ","m","s"," ").

lex("दो","adject","numeral"," "," "," "," ").

lex("तीन","adject","numeral"," "," "," "." ").

lex("दस","adject","numeral"," "," "," "," ").

lex("जा","verb"," ","pres"," ","s"," ").

lex("गिर","verb"," "," "," "," "," ").

lex("खा","verb"," "," "," "," "," ").

lex("पी","verb"," "," "," "," "," ").

lex("सो","verb"," "," "," "," "," ").

lex("ले","verb"," "," "," "," "," ").

lex("कर","verb"," "," "," "," "," ").

lex("दे","verb"," "," "," "," "," ").

lex("जाग","verb"," "," "," "," "," ").

lex("लिख","verb"," "," "," "," "," ").

lex("[illegible]","verb"," "," "," "," "," ")
```

```
राम जाता है.



SYNTAX ANALYSIS OF THIS SENTENCE IS AS FOLLOWS :
sentence(mod(numb("s"),gend("m"),persn("t"),tense("pres")),sent3(noun_ph1(noun("
राम")),verb_ph1(verb(s_dhatu("जाता"),helpv1(link_v("है"))))))

CASE ANALYSIS IS AS FOLLOWS :
sent_sem(mod(numb("s"),gend("m"),persn("t"),tense("pres")),propos([karta(noun_ph
1(noun("राम")))]))

DO U WANT TO ANALYZE MORE SENTENCES ANS(y/n)?

START INPUTTING A SENTENCE :
राम पुस्तक पढ़ता है.



SYNTAX ANALYSIS OF THIS SENTENCE IS AS FOLLOWS :
sentence(mod(numb("s"),gend("m"),persn("t"),tense("pres")),sent3(noun_ph1(noun("
राम")),verb_ph3(noun_ph1(noun("पुस्तक")),verb_ph1(verb(s_dhatu("पढ़ता"),helpv1(li
nk_v("है")))))))

CASE ANALYSIS IS AS FOLLOWS :
sent_sem(mod(numb("s"),gend("m"),persn("t"),tense("pres")),propos([karta(noun_ph
1(noun("राम"))),karm(noun_ph1(noun("पुस्तक")))]))



राम खाना खाता है.



SYNTAX ANALYSIS OF THIS SENTENCE IS AS FOLLOWS :
sentence(mod(numb("s"),gend("m"),persn("t"),tense("pres")),sent3(noun_ph1(noun("
राम")),verb_ph3(noun_ph1(noun("खाना")),verb_ph1(verb(s_dhatu("खाता"),helpv1(link
_v("है")))))))

CASE ANALYSIS IS AS FOLLOWS :
sent_sem(mod(numb("s"),gend("m"),persn("t"),tense("pres")),propos([karta(noun_ph
1(noun("राम"))),karm(noun_ph1(noun("खाना")))]))
```

```
वह अच्छा है.


SYNTAX ANALYSIS OF THIS SENTENCE IS AS FOLLOWS :
sentence(mod(numb("s"),gend("m"),persn("t"),tense("pres")),sent3(noun_ph1(pnoun(
"वह")),verb_ph2(adj2(sadj("अच्छा")),helpv1(link_v("है")))))

CASE ANALYSIS IS AS FOLLOWS :
sent_sem(mod(numb("s"),gend("m"),persn("t"),tense("pres")),propos([karta(noun_ph
1(pnoun("वह")))]))

DO U WANT TO ANALYZE MORE SENTENCES ANS(y/n)?


राम का लड़का घर जाता है.


SYNTAX ANALYSIS OF THIS SENTENCE IS AS FOLLOWS :
sentence(mod(numb("s"),gend("m"),persn("t"),tense("pres")),sent3(noun_ph3(modf1(
adj(radj(noun("राम"),par("का")))),noun_ph1(noun("लड़का"))),verb_ph3(noun_ph1(noun
("घर")),verb_ph1(verb(s_dhatu("जाता"),helpv1(link_v("है")))))))

CASE ANALYSIS IS AS FOLLOWS :
sent_sem(mod(numb("s"),gend("m"),persn("t"),tense("pres")),propos([karta(noun_ph
3(modf1(adj(radj(noun("राम"),par("का")))),noun_ph1(noun("लड़का")))),karm(noun_ph1
(noun("घर")))]))

DO U WANT TO ANALYZE MORE SENTENCES ANS(y/n)?
```

BIBLIOGRAPHY

1. Mary Dee Harris, Introduction to Natural Language Processing, 1985, Reston publishing comp. A prentice hall.

2. James Allen, Natural Language Understanding, The Benjamin/Cummings publishing.

3. Gerald Gazdar and Chris Mellish, Natural Language Processing in Lisp (An introduction to computational linguistics), 1989, Addition Wesley.

4. Fernando C.N. Pereira and Stuart M. Shieber, Prolog and Natural-Language analysis, 1987, CSLI stanfoord University.

5. John E. Hopcropt and Jeffrey D. Ullman, Introduction to automata theory language and computation, third edition - Addision Wesley.

6. Ralph Grishman, Computational linguistics (An introduction), 1986, Cambridge University press.

7. Semantic Predictions and Disambiguation in Natural Language Understanding (CSI Jan. 1988), T.V.Geetha and R.K. Subramanian.

8. Eduard V Popov, Talking with conputers in Natural Language, Springer-Verlag Berlin Heidelberg, 1986

9. S. H. Kellogg, A grammar of the Hindi Language, Oriental books

10. Dr. Murlidhar Shrivastava, The elements of Hindi grammar,

1987, MotiLal Banarasias.

11. Michael C. Shapiro, A primer of modern standard Hindi, 1989, MotiLal Banarasidas.

12. Ramesh chandra mahrotra and Chitranjan kar, 'Hindi Ka Naveentam Beej Vyakaran ' (in hindi), 1986, 'pahchan prakashan.

13. Suraj bhan Singh, 'Hindi Ka Vakyatamak Vyakaran' (in Hindi), 1985, Sahitya sahakar. 14. Tery Winograd, Language as a cognitive process, vol. 1, Addition-wesly.

15. David Maier and David S. Warren, Computing with Logic Logic programming with prolog, The Benjamin/cumming publishing.