1519

FROM A SIMPLE SANSKRIT SENTENCE TO VERB & KARAKA SPECIFICATIONS : An NLP Approach

Dissertation submitted to

Jawaharlal Nehru University

in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & TECHNOLOGY

49p + fig + Appendix

by

Sanjay Kumar Dhurandher

SCHOOL OF COMPUTER & SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI - 110067

INDIA

JANUARY, 1994

CERTIFICATE

This is to certify that the dissertation titled "**From a Simple Sanskrit Sentence to Verb and Karaka Specifications : An NLP Approach**" being submitted by me to Jawaharlal Nehru University, New Delhi, in partial fulfilment of the requirements for the award of the degree of Master of Technology in Computer Science & Technology, is a record of the original work done by me under the supervision of **Prof. G. V. Singh, School of Computer and Systems Sciences,** during the Monsoon Semester, 1993.

The results reported in this dissertation have not been submitted in part or in full to any other university or institution for the award of any degree.

S.K. Dhurandher
5/1/94
Sanjay Kumar Dhurandher

[illegible]
6-1-94
Prof. K. K. Bharadwaj
Dean,
School of Computer &
Systems Sciences,
Jawaharlal Nehru University,
New Delhi.

[illegible]
Prof. G. V. Singh
School of Computer &
Systems Sciences,
Jawaharlal Nehru University
New Delhi.

ACKNOWLEDGEMENT

It is my pleasure to express my deep sense of gratitude to **Prof. G. V. Singh, SC&SS, JNU,** for his gracious and valuable guidance. This dissertation would not have happened without his timely and proper guidance. I feel privileged that I worked under his supervision.

My warmest thanks also to **Prof. K. K. Bharadwaj, Dean, SC&SS,JNU,** for providing the excellent academic environment and facilities which enabled the successful completion of my project.

My sincerest thanks to my friend **D.K. Lobiyal** for reviewing, commenting and providing valuable information. My heartfelt thanks to Amit, for revealing mysteries and tricks of Prolog. Reeta Nath, Susmita, Ajay, Bhushan and my classmates were so friendly and ever so ready to cooperate. My thanks to them. I thank my very own Roshan for his friendliness, cooperation and being such a good company in those long hours in the Lab.

I also express my thanks to DoE for funding the machines on which the present work was carried out.

My special thanks to **Dr. K. Suryanarayan**, who explained me the intricacies of Paninian Grammar. At various stages of preparation of this work, I availed his expertise and valuable suggestions.

New Delhi
January, 1994.

Sanjay Kumar Dhurandher

To

My Parents

CONTENTS

CHAPTER I

NLP : AN INTRODUCTION

1.1 Natural Language Processing

1.1.1 Introduction

Understanding or producing sentences or texts in 'natural language' (i.e. the language we use every day) have become crucial elements in human-machine communication [10]. They are therefore front-line research areas in the fields of Artificial Intelligence, Computational Linguistics and Cognitive Psychology. The 'language industry' is fast developing while research work in recent years in various countries has shown that a considerable amount of work is still to be done.

The term 'natural language processing' itself covers a variety of interpretations, while at the same time the topic also goes under various other names, including 'natural language analysis', 'computational linguistics', 'linguistic processing', 'automatic language processing' and so on.

The principal areas of research in Natural Language Processing (NLP) are :-

* Developing and modelling linguistic systems.
* Conceiving and implementing models and systems of NLP.
* Evaluating such systems from the point of view of human- machine interfaces.

Natural language understanding and **Natural language generation** are, the two components of **Natural language processing** - a technology with the ambitious goal of making it easy to communicate with computers as it is with people [5]. The phrase natural language processing generally refers to typed, printed, displayed, or spoken language rather than the speech. Getting computers to understand speech is the focus of related AI technologies called Speech recognition and Speech understanding.

1.1.2 Natural Language Understanding

Compared to people, computers require a great deal of precision, completeness, exactness in communication. For example, if we want someone to bring something for us, to drink, we might get the same result by saying any of the following sentences:-

* Please get me something to drink.
* Bring me a drink.
* Got anything to drink ?
* I'm thirsty - can you get me something ?

Unfortunately, a computer does not have the same kind of linguistic flexibility.

The goal of natural language understanding research is to enable computers to understand us well enough to perform an intended appropriate action.

Developing programs to understand natural language is important in AI because a natural form of communication with systems is essential for user acceptance. Furthermore, one of the most critical tests for intelligent behaviour is the ability to communicate effectively. AI programs must be able to communicate with their human counterparts in a natural way, and natural language is one of the most important mediums for that purpose.

We say a program **understands** a natural language if it behaves by taking a (predictably) correct or acceptable action in response to the input. For example, we say a child demonstrates understanding if it responds with the correct answer to a question. The action taken need not be an external response. It may simply be the creation of some internal data structures as would occur in learning some new facts. But in any case, the structures created should be meaningful and correctly interact with the world model representation held by the program.

1.1.3 Problems in Natural Language Understanding

Developing programs that understand a natural language is a difficult problem. Natural languages are large. They contain an infinity of different sentences. One of the problems is to account for the infinite number of sentences by developing a finite number of rules. Natural languages are ambiguous. A word as well as a sentence may have several meanings. For example, the word "bear" and the sentence such as "I saw a man on the hill with the telescope" can have different meanings in different contexts. This makes the creation of programs that ''understand'' a natural language, one of the most challenging tasks in AI. It requires that a program transform sentences occurring as part of a dialogue into data structures which convey the intended meaning of the sentences to a reasoning program. In general, this means that the reasoning program must know a lot about the structure of the language, the possible semantics, the beliefs and goals of the user, and a great deal of general world knowledge.

The four problems that cause difficulties in natural language understanding are ambiguity, imprecision, incompleteness, and inaccuracy.

Ambiguity

Natural language can be ambiguous due to multiple word meanings, syntactic ambiguity, and unclear antecedents. For example, John hit Bill because he sympathized with Mary.

Who sympathized with Mary ? As in this case of syntactic ambiguity, one cannot determine the antecedent of "he" without establishing a context for the sentence.

Imprecision

Concepts often are not described with precision. One's ability to understand what is being said may rely on one's familiarity with a situation. For example, consider the following sentences:-

* I have been waiting in the doctor's office for a long time.
* The crops died because it hadn't rained in a long time.
* The dinosaurs ruled the earth a long time ago.

If you read a story that included these sentences and then someone asked you about the length of the wait in the doctor's office, you might respond that it was no longer than a few hours because you are familiar with the concepts discussed in the sentences. Without the conceptual familiarity, a computer would not be able to differentiate between the three different lengths of time represented by the same phrase.

Incompleteness

Since we expect other people to "fill in the details" when we tell them something, we often supply incomplete information. For example, Elaine Rich suggests the following story :

"John went out to a restaurant last night. He ordered steak. When he paid for it, he noticed that he was running out of money."

Did John eat steak ? Although it is not stated explicitly in the story, you probably assumed that he did; after all, why else would he have paid for it ? Your expectations of likely events in that particular situation allowed you to understand information that was not included in the text. To be able to comprehend incomplete information, a computer must possess the same kind of situational expectations [7].

1.1.4 General Approaches to Natural Language Understanding

Essentially, there have been three different approaches taken in the development of natural language understanding programs, (1) the use of keyword and pattern matching, (2) combined syntactic (structural) and semantic directed analysis, and (3) comparing and matching the input to real world situations (scenario representations).

The keyword and pattern matching approach is the simplest.

The third approach is based on the use of structures such as the frames or scripts.

The second approach is one of the most popular approaches currently being used. With this approach, knowledge structures are constructed during a syntactical and semantical analysis of the input sentences. Parsers are used to analyze individual sentences and to build structures that can be used directly or transformed into the required knowledge formats. The advantage of this approach is in the power and versatility it provides. The disadvantage is the large amount of computation required and the need for still further processing to understand the contextual meanings of more than one sentence.

One way to ensure that none of the elements of a sentence is overlooked is to conduct a thorough analysis of the sentence's syntax. This **syntactic analysis** requires some kind of

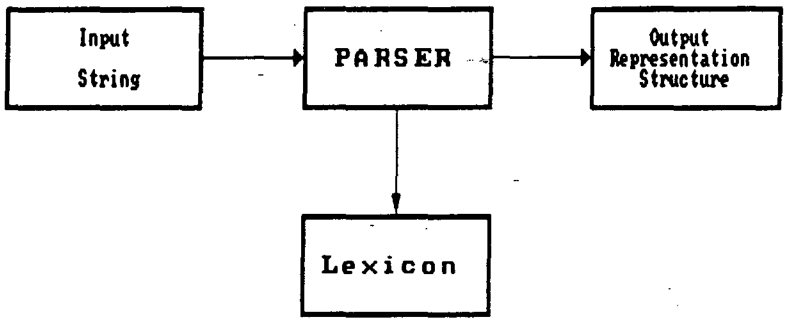
parsing technique (a method of separating a sentence into its component parts), which is the computer's equivalent of diagramming a sentence.

1.2 PARSING

Before the meaning of a sentence can be determined, the meanings of its constituent parts must be established. This requires a knowledge of the structure of the sentence, the meanings of individual words and how the words modify each other. The process of determining the syntactical structure of a sentence is known as **parsing** [6].

Parsing is the process of analyzing a sentence by taking it apart word-by-word and determining its structure from its constituent parts and subparts. The structure of a sentence can be represented with a syntactic tree or a list. The parsing process involves finding a grammatical sentence structure from an input string. When given an input string, the lexical parts or terms (root words) must first be identified by type, and then the role they play in a sentence must be determined. These parts can then be combined successively into larger units until a complete tree structure has been completed.

To determine the meaning of a word, a parser must have access to lexicon. When the parser selects a word from the input stream it locates the word in the lexicon and obtains the word's possible function and other features, including semantic information. This information is then used in building a tree or other representation structure. The general parsing process is illustrated in the figure below :-


Input String
PARSER
Output Representation Structure
Lexicon

A lexicon is a dictionary of words (usually morphemes or root words together with their derivatives), where each word contains some syntactic, semantic, and possibly some pragmatic information. The information in the lexicon is needed to help determine the function and meanings of the words in a sentence.

Basic Parsing Techniques

Parsing takes advantage of inherent regularities in natural language to ensure that the computer understands the precise function of each word in a sentence, as well as its relationship to each of the other words.

Basic parsing techniques used for analysis include the following :

Top-Down Parsing : A top-down parser begins by hypothesizing a sentence and successively predicting lower level constituents until individual preterminal symbols are written. These are then replaced by the input sentence words which match the terminal categories.

Bottom-Up Parsing : A bottom-up parser begins with the actual words appearing in the sentence and is, therefore, data driven.

1.3 OVERVIEW OF THE PROBLEM

Sentence generation and analysis are the two main issues in Natural Language Processing. Compared to generation, parsing is a complex process. The structure of a language is also an important point in this regard. Parsing languages like English, which are configurational languages is simpler compared to non- configurational languages, like Sanskrit.

Syntax is one of the most important aspects of the grammar that needs to be considered for sentence analysis.

Syntax may be described as the mode of arranging words in a sentence. Herein, questions relating to concord, government and order are to be considered. Sanskrit is mainly concerned with the first two. In our system we consider the concord of the verb with the subject. Government is considered by the *karakas.*

It is mentioned above that Sanskrit is a non-configurational language. For the same reason, the mere order of words is not of material importance. There are, however, certain constraints on this order.

The topic of the present dissertation is "From a simple Sanskrit sentence to verb and *karaka* specifications : An NLP approach". For the purpose of this work, we consider sentences comprising of one or more noun-forms and a finite verb-form. To handle this problem, two lexicons were designed, one for the verbal bases (i.e. dhatukosh) and another for nouns (i.e. pratipadikkosh). Further, detailed procedures for identifying verb-forms (i.e. reverse_tigant) and noun-forms (i.e. reverse_subant) are written.

The input to the system is a simple sentence. The system identifies, with the help of above procedures, the verb with its attributes and noun with its attributes. Further, the verb also displays all possible *karakas* that it takes. The sentence is considered to be correct if the *karakas* of the verb and the case-suffixes of the nouns match. There is a straight mapping of the case-suffixes to the *karakas.*

The system assumes and is augmented by the related work 'Sentence generation based on the verb and *karaka* specifications'. The system does not account for sentences with adjectives, adverbs, participle and passive voice.

CHAPTER II

PANINIAN GRAMMATICAL FRAMEWORK

2.1 Paninian Grammar

Panini formulated rules of Sanskrit grammar in his work called ASHTADHYAYI (literally Eight Chapters) [3]. With a finite set of rules he attempts to account for infinite variety of language forms. His work is equally well known for its style as for its contents. This style called *Sutra,* is a formal language which challenges to achieve utmost brevity in its code. Some of the techniques adopted to achieve this brevity are: (a) information chaining (*anuvritti* and particle ca), (b) concept of *iths* as control characters and operators, (c) arrangement of sounds in specific order, (d) concept of hierarchies in several layers, (e) exception handling, etc. He also details the procedures to derive the verb from verbal roots, noun forms from nominal bases and the treatment of syntax through *karaka*. These procedures have similarities to **computer program.**

The most impressive aspect of the system of Ashtadyayi is that the rules are logically interrelated and cohesively organized on sound scientific principles. Panini's system is algebraic in style and is very different from the classic Sanskrit and gives authoritative description of the Sanskrit language.

The technique of description and methodology adopted by Panini in his work, has attracted wide attention and interest in recent times, by linguists as well as computer scientists. The relevance of Sanskrit in the Indian linguistic context need hardly be emphasized.

2.2 Sanskrit Syntax

The goal of Paninian theory is to construct a theory of human natural language communication. Grammar, a part of such a theory of communication, is a system of rules that establish a relation between what the speaker decides to say and his utterance, and similarly, what the hearer hears and what meaning he extracts.

The main problem the Paninian approach addresses is how to extract *karaka* relations (which are syntactico-semantic relations) from a sentence. As it is inspired by an inflectionally rich language, it emphasizės the roles of case markers.

According to Sanskrit grammarians sentence is the minimal meaningful unit of a language. The division of a sentence into words and, of a word into base and suffixes are merely for the sake of grammatical analysis , states Bhartrhari, the grammatical-philosopher of the 5th centuary A.D [3].

The verb plays a key role and is the semantic focus of a Sanskrit sentence. A finite verbal form contains information regarding tense/mood, number, person , voice and so on. However, it may be mentioned here that Sanskrit allows purely nominal sentences.

The corner-stone of Sanskrit syntax is the notion of *Karaka*. *Karaka* is conceptual notion that mediates mapping between grammatical relations and case-suffixes.

A syntactic unit is called *'pada'* and Panini defines it as:

sup-ting antam padam P.1.4.14.

'that which ends with a *sup* suffix (i.e. inflected noun) or with a ting suffix (i.e. inflected verb) is called *pada*'.

Therefore, a Sanskrit sentence may be conceived as made up of inflected nouns and conjugated verbs.

There are six types of *Karakas* in Sanskrit. They are as follows :

1. ***KARTA*** : The most independent person /thing in the performance of an action.
2. ***KARMA*** : The object most desired (to be obtained) by the agent of action.
3. ***KARANA*** : The efficient means in the performance of an action.
4. ***SAMPRADANA*** : The recepient of the object of the verb da (to give).
5. ***APADANA*** : The static point in the action of separation.
6. ***ADHIKARANA*** : The substratum of the action performed.

Each *karaka* is expressed in terms of a case-ending. While a strict one-to-one correspondence of a *karaka* and a case-suffix may not be established, in general, it may be stated that, a particular case-suffix roughly corresponds to one *karaka*. Thus, e.g. to refer to the instrument in bringing about an action (*karana*), the third case-suffix is employed.

2.3 *Karaka* Relations

There is no straight forward mapping from *vibhakti* to semantic relation between noun groups and verbs. The key to arriving at an answer is to identify intermediate relations.

The Paninian grammar specifies a mapping from the nominals and the verb(s) in a sentence to *karaka* relations between them. Thus, these relations by themselves do not give the semantics. They specify relations which mediate between *vibhakti* of nominals and the verb form on one hand and semantic relations on the other [4]. The various levels in the Paninian model are as shown below :

Semantics	Karaka Labels	Vibhakti (case-markers)
dhruvam apaye	apadana	pancami
sadhakatanam	karana	tritya
karmana yam abhipriti	sampradana	chaturthi
adhana	adhikarana	saptami
karturipsita tama	karma	dvitiya
svatantra	karta	prathama/tritiya

Fig. 2.1

----- Semantic Level (What the speaker has in mind)

|

|

----- *Karaka* Level

|

|

----- *Vibhakti* Level

|

|

----- Surface Level (uttered sentence)

The *karaka* relations may be shown in the following manner as shown in fig. 2.1.

In all there are about six types of *karaka* relations. What the *karaka* relations do specify with respect to a verb, however, are six or so relations of nominals to that particular verb. These are sufficient for providing a mapping from *karaka* relations to semantic relations. Thus the *karakas* provide the maximum necessary information relative to a verb. For example, the *karta karaka* may get mapped to agent for one verb, and experience for another, etc. *Karaka* theory incorporates two other insights.

1. Each and every verb refers to an activity or event that can be further sub-divided into a complex of activities. Each of the sub-activities has its own semantic relations and karaka relations. For example, in the action of cooking rice, a person kindles the firewood, places the vessel on the fire, puts water and rice in the vessel and so on. Each of these

sub_activities has its own semantic relations with associated objects. Thus, for each of the sub- activities there will be appropriate *karaka* relations. For example,

(i) *ramah sthalyam odanam pacati.*

(Ram cooks rice in the pot)

(ii) *sthali pacati.*

(The pot cooks(the rice))

(iii) *odanah pacyate.*

(The rice cooks).

In (i) the speaker decides to give importance to the role of Ram, whereas in (ii) the speaker wishes to emphasize the role of pot. Even though the verb representing a main activity may be used in a sentence, the *karaka* relations specified might correspond to a particular sub- activity. This will have obvious consequences for semantic relations.

2. *Karaka* relations also depend on the concept of *Vivaksha*. *Vivaksha* refers to the speaker's viewpoint or attitude towards the activity. A sentence is not a statement of an objective activity. Rather it is the speaker's viewpoint of the world reality. Usually *vivaksha* affects the choice of verb form, which in turn affects the *karaka* relations and *vibhakti*. *Karta karaka* holds between that nominal and a verb in a sentence, whose referent is **'swatantra'** or the most independent or autonomous out of all the *karaka* nominals that are expressed by the speaker. However, it is with respect to the activity implied by the verb. In (i), out of Ram and *sthali*, it is the former which is more independent. Hence, Ram is the *karta* in (i). In (ii), in the absence of any mention of Ram, out of *sthali* and *odana* it is the former which is more independent. Hence, *sthali* is the *karta*.

According to Kaundbhatta, who further elaborates on the concept of *swatantra*, every verb in a sentence refers to an activity. Sometimes the verb can be ambiguous, in which case it may refer to several activities. When used in a sentence, it may refer to any one of its possible activities. *Karta* of a verb in a sentence is one which is the *'ashraya'* or base of the activity. In (i), the activity referred to is the act of cooking rice by Ram by placing the vessel on the fire, putting the water in the vessel etc. Thus, Ram is the *karta* of this activity. In (ii), the activity referred to is the cooking of the rice by the pot. The pot is the *karta* of this activity. In (iii), the rice is conceived as the agent in the process of cooking. Different languages make different lexical choices for the activities. English, for example, retains the same verbal root for all the three activities above.

2.4 *Karaka* to *Vibhakti* Mapping

The mapping from *karaka* level to *vibhakti* level will allow the mapping of a representation of a sentence at the *karaka* level to a representation at the *vibhakti* level.

We now give the rules in the Astadhyayi that relate these *karakas* with the case-suffixes (*vibhakti*). Only the most general rules are mentioned.

1. *pratipadikartha-linga-parimana-vacanamatre prathama*

   (P.2.3.46)

   Where the sense is that of the Crude form or where there is the additional sense of gender only, or measure only, the first case-affix is employed.

2. *karmani dvitiya* (P.2.3.2)

   When the object is not denoted by the termination of the verb, &c. i.e. the verb does not agree with it, the second case- affix is added to the word.

3. *kartrkaranayos tritiya* (P.2.3.18)

In denoting the agent or the instrument the third case- affix is employed.

4. *Chaturthi sampradane* (P.2.3.13)

In denoting the *sampradana-karaka* the fourth affix or Dative is employed after the noun.

5. *apadane pancami* (P.2.3.28)

When the *Apadana-karaka* is denoted, the fifth case-affix is employed.

6. *saptamyaadhikarane ca* (P.2.3.36)

The seventh case-affix is employed when the sense is that of location.

2.5 Paninian Theory Based Parsing

The Paninian theory discussed above can be used for building a parser. Parsing is the reverse of generation, where given a sentence a suitable semantic structure is to be assigned to it. If we build a parser based on the Paninian theory, we have to obtain a representation of a given sentence at the *vibhakti* level, using which we must obtain a representation the *karaka* level, and finally, a representation at the semantic (or mental) level.

It turns out that the Paninian theory is extremely suitable from computational point of view. It can be used in a natural manner for structuring a parser which is extremely efficient.

It is fairly obvious that one part of the parser must take care of morphology. For each word in the input sentence, a dictionary or a lexicon needs to be looked up, and associated grammatical information needs to be retrieved. The words have to be grouped together yielding nominals, verbals etc. Finally, the *karaka* relations among the elements have to be identified. The structure

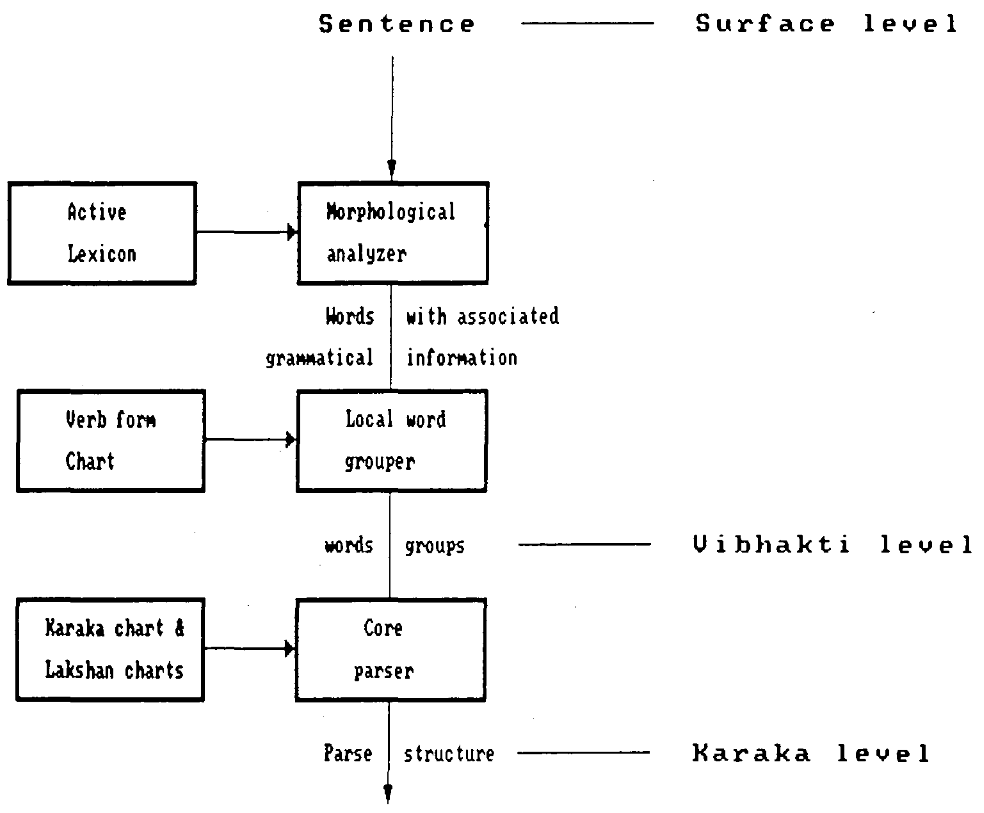


Fig. 2.2 : Structure of the Parser

of the parser is as shown in Fig. 2.2.

2.6 Morphological Process

It was mentioned that syntactic units are generated by morphological process. The procedures whereby a noun and verb are inflected are called *subant* and *tigant* respectively [9]. A brief description of the same is given below :

SUBANT

A *subant* is made up of a nominal base and a nominal suffix. The nominal base is called *pratipadik.* There are 21 nominal suffixes in Sanskrit. In all there are 7 *vibhaktis* and 3 *vachans* in Sanskrit. The nominal suffixes are arranged in 7 x 3 paradigm. The suffixes are listed as :

	ekvachan	*dvivachan*	*bahuvachan*
vibhakti			
prathama	*su*	*au*	*jas*
dvitiya	*am*	*aut*	*sas*
tritiya	*taa*	*bhyam*	*bhis*
chaturthi	*ne*	*bhyam*	*bhyas*
panchami	*nasi*	*bhyam*	*bhyas*
shashti	*nas*	*os*	*aam*
saptami	*ni*	*os*	*sup*

The nouns may be classified as sangya, sarvnam and sankhyavachak. The first varga sangya includes nouns, proper or common. The second group is that of sarvnam (pronouns), and sankhyavachak deals with numerals. The transformational rules for forming subant are different for each varga.

TIGANT

Tigant deals with verb inflexion. Panini divides all the verb roots (*dhatu*) into ten ganas. They are : *bhavadi, adadi, juhotyadi, divadi, svadi, tanadi, krayadi, churadi, tudadi, rudhadi.*

There are about 2000 *dhatus* in Sanskrit. The *bhavadi gana* is by far the largest one comprising of almost 40% of all the *dhatus*. The transformation rules acting on verbal roots are different for each gana and they are the same within a particular gana. The verbal root from a specific gana can then be a *parasmaipadi, atmanepadi or ubhaypadi.* Knowledge of *lakar* (which pertain to tense or mood of a verb) is required to specify a desired action. Sanskrit has ten lakars viz. *lot, lrit,lut.* Like English, we have three numbers or purushas in Sanskrit namely, *uttam purusha* (first), *madhyam purusha* (second) and *pratham purusha* (third). And finally we need a *purusha* to form a *tingant pada.*

For purposes of analysis, we have to take these procedures into consideration. These procedures have to be analyzed in a reverse manner for identification of word forms.

Details of the same are given in the following chapter.

CHAPTER III

SANSKRIT SENTENCE ANALYSIS

3.1 Sentence Analysis

Panini doesn't formally define the term *vakya* (sentence), but the word is used in Astadhyayi. A sentence is a series of connected words. To express the connection between the words, Panini uses different non-technical terms such as joining (1.4.59 *yoga*), syntactically connected (2.1.1 *Samaratha*), wish or desire (3.2.114 *sakanksha*) and so on.

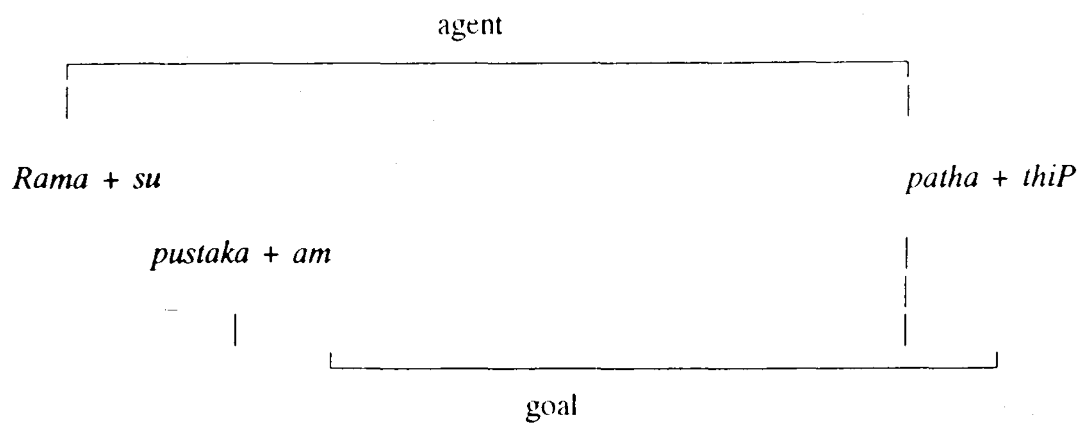
In Astadhyayi, derivation of words and their interrelation in a sentence starts from meaning. But in analyzing it, one starts with the sentence and arrives at the meaning or intention.

Before discussing about sentence analysis, formation in the Paninian system is given by example below :

Example: Let us say we want to generate the sentence 'Rama reads a book'. Here,the verb is in present tense (*varthamana kala*) and hence gets *Lat*. Rama is the agent of the action and book (*pustaka*) is the goal.

TH-6869

By 3.4.69, Lat expresses (1) agent or (2) goal or (3) the state (where the verb lacks goal). If we choose ***Lat*** to represent the agent, after transformations the final verb form is pathati. The goal can be expressed by 2.3.2 accusative case, since it is not expressed yet (*anabhihita*), (by 2.3.2) we get ***pustakam*** (*pustaka* + *am*). Verb ending already expressed agent (by 3.1.68 *karthari shap*), hence we cannot express it again. Now second or third case ending becomes applicable (by 2.3.2 and 2.3.18). But since we have to only express gender and number (by 2.3.46), the nominal stem gets the nominative case (*su*) and we form the word (rama + *su*) ramah.



Final form is : *Rama(h) pustak(am) patha(ti).*

In order to analyze the above sentence, the process starts by identifying the constituent words. By identification is meant identifying the base word as well as the case markers/verb-suffixes. Once the words are identified, the relation among the different words is established.

The Paninian grammar provides for generation of words and sentences. That is, the grammar describes the process in the forward direction, which is known and hence predictable. However, for sentence analysis and parsing, a reverse approach has to be adopted. It may be mentioned here, that Panini, nor any of the grammarians that follow him, provide us any rules in this direction. The most obvious reason being, there was no necessity, whatsoever, for such an analysis.

In the above situation, the immediate task is to develop procedures and tools for such an analysis. The advent of computers and the emergence of Artificial Intelligence, with enhanced programming paradigms, mainly logic programming, help in the realization of this goal. The most important aspects of AI, namely, inference and heuristics, and an approach based on it make such an analysis possible.

Now, for analyzing the above sentence, initially we select the first word (i.e.*ramah* here) and send it to the reverse_tigant procedure. If the word is identified as a tigant, the system returns the base along with its attributes. If it is not identified, the same is then sent to the reverse_subant procedure. If the word is identified as a subant the system returns its attributes. If it is neither a tigant nor a subant we say that the word is wrong. Similar processing is done for other words also.

A detailed description of the reverse_tigant and reverse_subant procedures are discussed in chapter 4.

3.2 Semantic Representation

Once the constituent words are identified, the *vibhaktis* are established, which are checked for syntactic compatibility. The verb lexicon contains a list of possible *karakas* it can take. If these values match with the obtained *vibhaktis* the sentence is considered as valid.

It may be mentioned here that as the work in this direction (i.e. computer analysis of Sanskrit) is in very preliminary stages, the present work may be looked as an initial step in the same direction. Further the scope of the work is limited to simple sentences(e.g. this doesn't consider adjectives, passive structures etc.), as tools are to be developed for analyzing complex sentences and so on.

3.3 *Pada* Analyzer

The process of determining the syntactical structure of the sentence is known as Parsing. It is the process of analyzing a sentence by taking it apart word-by-word and determining its structure from its constituent parts and sub-parts. When given an input string, the lexical parts, i.e. the root words must first be identified by type, and then the role they play in a sentence must be determined.

The sentence *Ramah Pustakam Pathati* contains three syntactic units, namely, *ramah*, *pustakam* and *pathati*. A program has been developed that identifies the word-forms along with the grammatical attributes that go with them. Let us say we want to analyze the word-form *ramah*. Any syntactic unit being essentially either an inflected noun or inflected verb, we need consider only the nominal-suffixes (i.e. *sup* suffixes) or verbal- suffixes (i.e. *ting* suffixes). An identification process based on the suffixal part is also an economical method, considering the fact that there are only 21 nominal suffixes and 18 verbal suffixes. It may be noted that the end part of any syntactic unit corresponds to the suffix part, either as it is, or in a modified form. In some cases, the suffix may also be totally dropped. To illustrate, consider the singular number of nominal case, which is *su*. This suffix has three possible representations, as mentioned below:

1. *su* --> *h*
2. *su* --> *am*
3. *su* --> *0*

Thus, the final ***h*** in ***ramah***, may be inferred as the modified form of *su*.

Similarly, in the word *pathati*, the end part, i.e. *ti*, may be traced to suffix *tip*, i.e. third person singular.

1. *tip* --> *ti*

The remaining part of the word, i.e. *patha*, represents the base word. For the identification of the base word the computational lexicon has to be consulted.

A major difficulty in the process of word-identification arises due to case-clashes. For example,

bhyas --> *bhyah*

may either be a plural suffix of dative or ablative case.

The above are some important issues that one needs to consider while analyzing a word-form. These ambiguities can be resolved to a great extent by the computational lexicon.

3.4 Computational Lexicon

Knowledge can be represented using an appropriate data structure, which is referred to as lexicon. Lexicon is quite essential and necessary for Natural Language Processing (NLP) systems, i.e. parser, generator, translator, etc. It is because almost all the components of NLP systems refer to lexicon in order to accomplish their task. A simple lexicon for NLP systems may contain syntactic, semantic and contextual knowledge.

The lexicon may be defined as a collection of lexemes along with their related attributes. A lexeme is considered as one word taken along with its grammatical attributes, semantic attributes, conjugate list, computational words (derived words), relations, possible surface realization, pronunciation and meaning of the word.

The system contains two lexicons, namely dhatukosh and pratipadikkosh.

Dhatukosh : There are two thousand verbs in Sanskrit. These are distributed over ten classes called *ganas*. Further, a *dhatu* can be either of *parasmaipadi* or *atmanepadi* type. The lexicon contains the list of verbs along with its attributes i.e. the *gana* and the *pada*. A list of *karakas*, both mandatory as well as obligatory are also stored with each *dhatu.*

Pratipadikkosh : The pratipadikkosh contains the list of nouns. In any language, the nouns are innumerable, and it is difficult to store them all. Hence, the lexicon contains a list of most commonly used nouns, along with their attributes. The attributes in case of a noun are its class (called *varga*) whether it is of noun, pronoun or numeral type, and gender (i.e. *ling*). Secondly, a noun can either be of masculine, feminine or neuter gender type.

CHAPTER IV

IMPLEMENTATION

In the previous chapters the theoretical framework of Paninian grammar and the issues involved in language processing are discussed. The present chapter deals with the implementation details of sentence analysis. In implementation, modular programming approach has been adopted.

The system contains three major modules. These are :

1. Lexicon
2. Analyzer
3. Main Module

The above modules contain sub-modules. The system organization for the same is as shown in fig. 4.1. There are two lexicons, namely, **Dhatukosh** and **Pratipadikkosh**. Similarly, the Analyzer consists of two sub-modules, namely, **Reverse_tigant** and **Reverse_subant**. The Analyzer is the Processing module while the lexicon is the Maintainance module. The maintainance module is accessed by the Processing module. We now describe each of these modules.

4.1 Lexicon

This system operates on dhatu and nominal data. Therefore this module consists of dhatukosh and pratipadikkosh. These are described below :

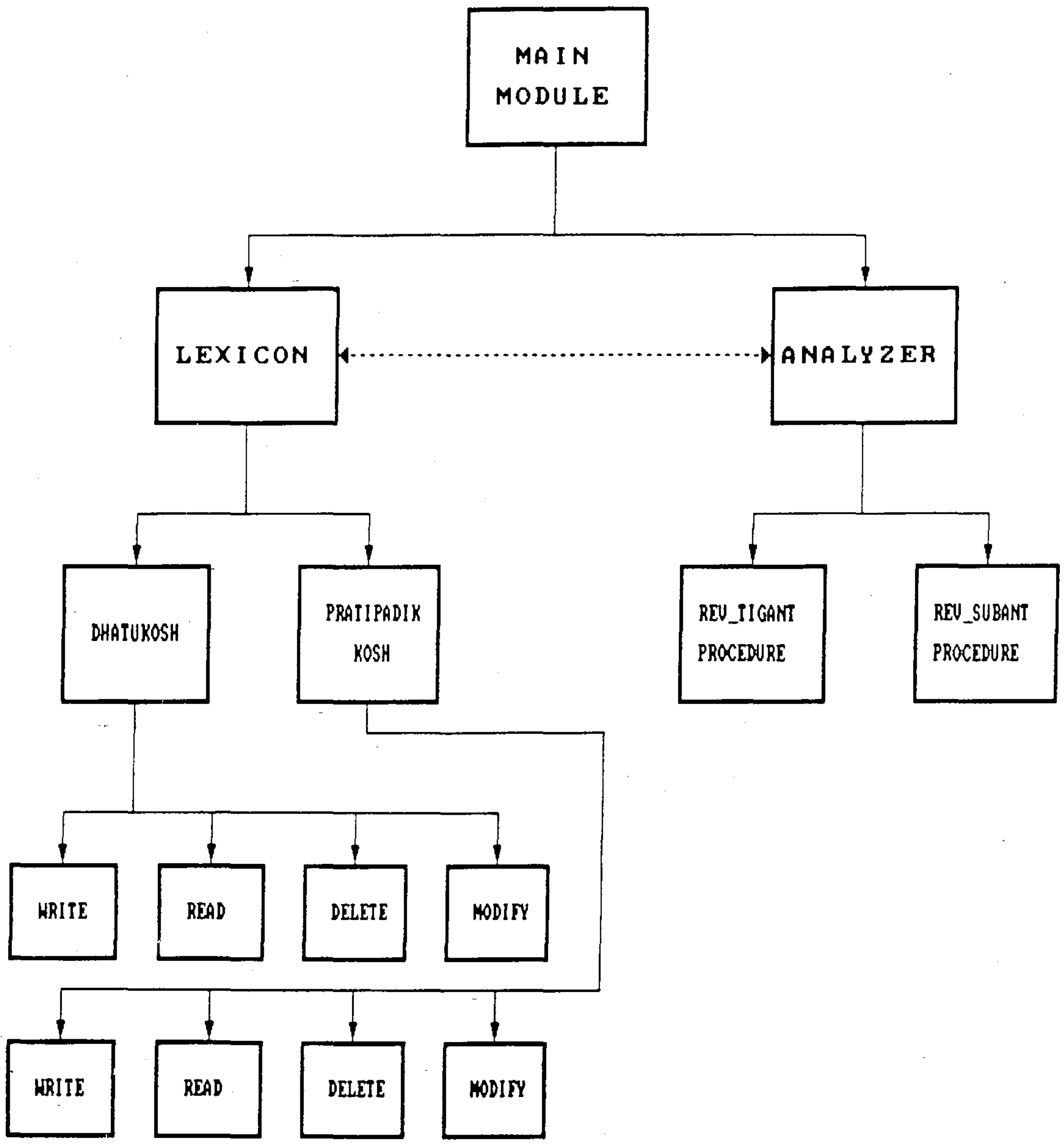


Fig. 4.1 : System Organization

4.1.1 Dhatukosh

This module consists of verb_lexicon and procedures to operate on this lexicon. In this lexicon (i.e. dhatukosh) the *dhatu* (verb root) along with its attributes is stored. These attributes are :

1. *Anyarup*
2. *Gana*
3. *Pada*
4. List of mandatory *karakas*
5. List of optional *karakas*

These attributes are stored in the lexicon through a functor mechanism of Prolog. The **data structure** used for this is as :

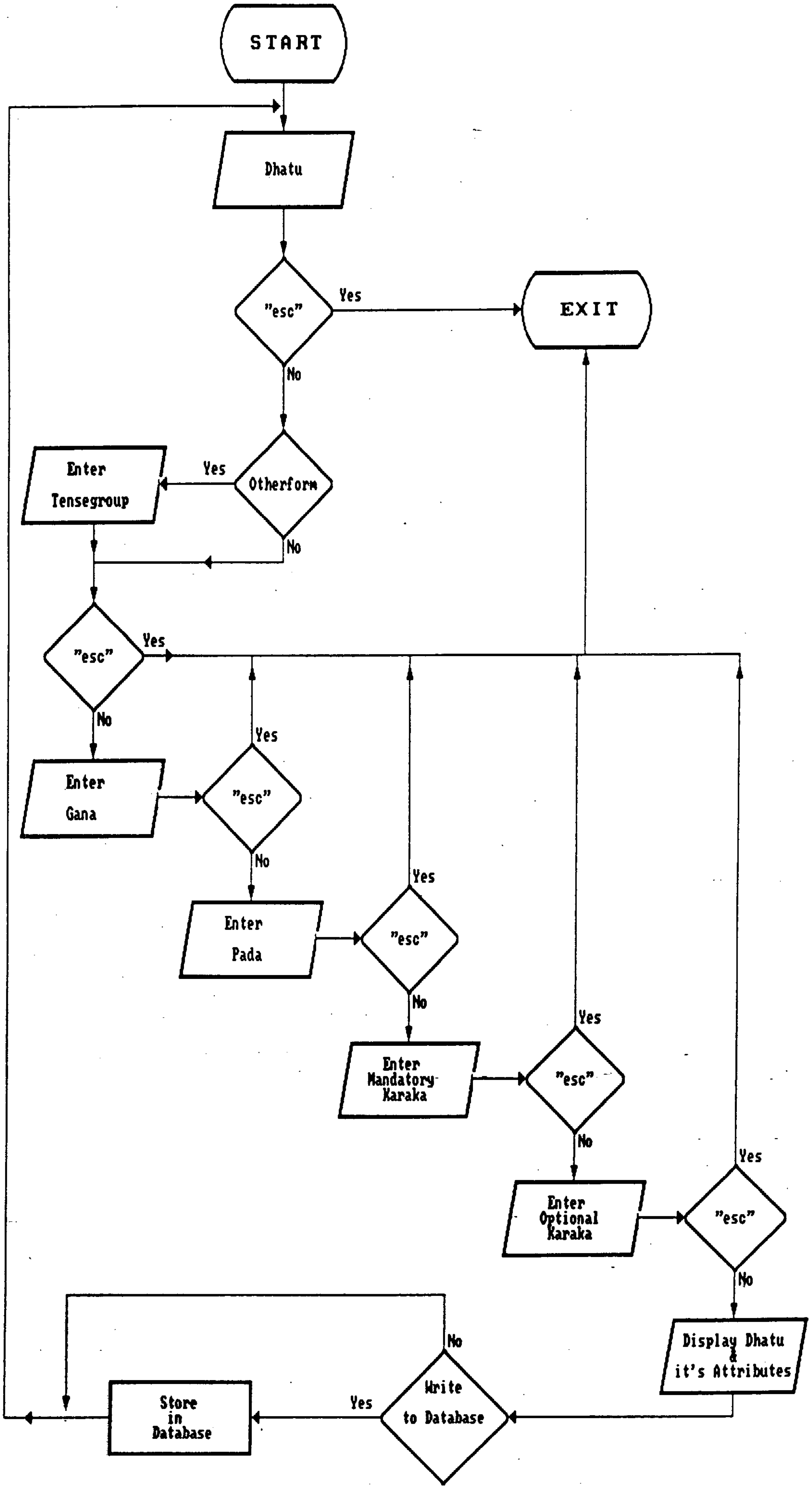
dhatu_attrib = dhatu_attr(Dhatu,Otherform,Tensegroup,Gana,Pada,

List of Mandatory Karakas,List of Optional Karakas)

This module opens with a menu displaying the procedures to create, maintain and access the dhatukosh. These procedures are **WRITE, READ, DELETE** and **MODIFY**. This menu (i.e.the main menu) appears as shown in A-1. To select any one of these procedures the user presses the <ENTER> key. These procedures are menu driven and are interactive in nature. They are described in detail below :

4.1.1.1 WRITE

To add a new ***dhatu*** to the dhatukosh the user selects the **WRITE** procedure from the main menu. By using this procedure one can store the various attributes of the dhatu in the dhatukosh.


START
Dhatu
"esc"
Yes
EXIT
No
Otherform
Yes
Enter Tensegroup
No
"esc"
Yes
No
Enter Gana
"esc"
Yes
No
Enter Pada
"esc"
Yes
No
Enter Mandatory Karaka
"esc"
Yes
No
Enter Optional Karaka
"esc"
Yes
No
Display Dhatu & it's Attributes
Write to Database
Yes
No
Store in Database

The flow chart for this procedure is shown in fig. 4.2.

Once the user has selected the **WRITE** procedure, a message asking the user to enter the *dhatu* appears on the screen (as shown in A-2). After the *dhatu* has been entered.the user is required to give the otherform (some special substitute form) of the verb, if any. In case, otherform exists then a menu appears asking the user to select the tensegroup (i.e. *lakar* group). For details see A-3.

Similarly the user is asked to select the *gana* and *pada* to which the *dhatu* belongs from their respective submenus (as in A-4 and A-5).

Once the above information is given, the user is required to input the list of mandatory *karakas*. There are six *karakas* namely, *Karta, Karma, Karana, Sampradana, Apadana, Adhikarana*, out of which the selection is made. These are selected from the submenu (as shown in A-6).

Once the user has selected the list of mandatory *karakas* another menu consisting of optional *karakas* is displayed on the monitor (as in A-7). This menu consists of the *karakas* remaining after the selection of the mandatory *karakas*.

Once this information is entered, this modules provides an option either for storing the information or to abandon it. For storing, the key "F1", and for abandoning,the key "F2" is used (as shown in A-8).

4.1.1.2 READ

To know the attributes of a *dhatu* present in the dhatukosh, the user selects the **READ** procedure for the same from the main menu. The flow chart for this procedure is shown in

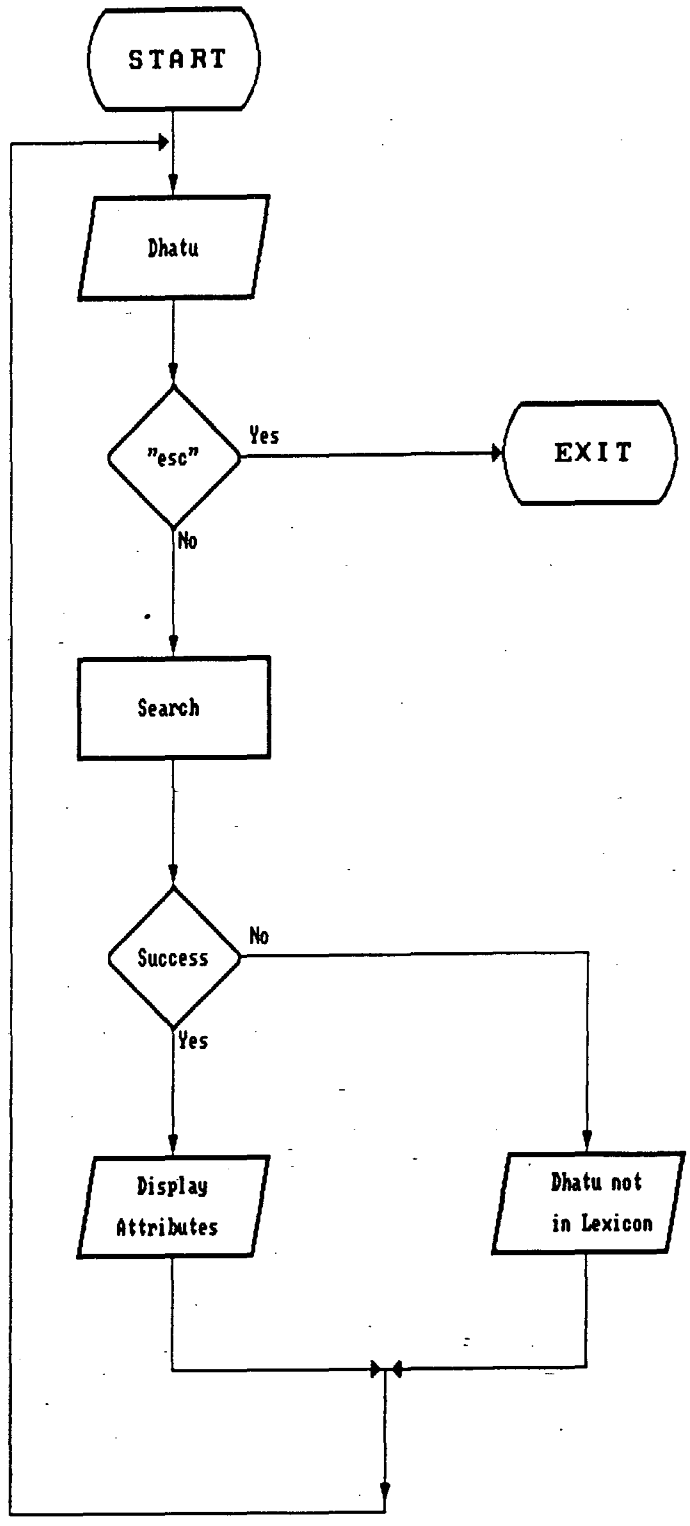


Fig. 4.3 : Flow chart for READ procedure

fig.4.3.

After selecting this procedure a message appears on the monitor asking the user to specify the *dhatu* whose attributes he/she wishes to know (as in A-9). Having entered the dhatu the attributes of the same are displayed on the screen.

4.1.1.3 DELETE

If a particular *dhatu* in the dhatukosh is not required one can remove or delete the same from the dhatukosh by selecting the **DELETE** procedure from the main menu. The flow chart for this procedure as shown in fig. 4.4.

On selecting this procedure a message appears on the screen asking to input the dhatu the user wants to delete (as in A-10).

4.1.1.4 MODIFY

The changes or modifications in a *dhatu* present in the dhatukosh can be done by selecting the **MODIFY** procedure from the main menu. The flow chart for this procedure is as shown in fig. 4.5

On selecting this procedure all the previous attributes of the *dhatu* along with a menu appears on the monitor asking the user to select which of the previous attributes is required to be changed (as shown in A-11). On selecting these attributes, one at a time, one can make the necessary modifications. Once the modifications have been made the modified form is displayed on the monitor.

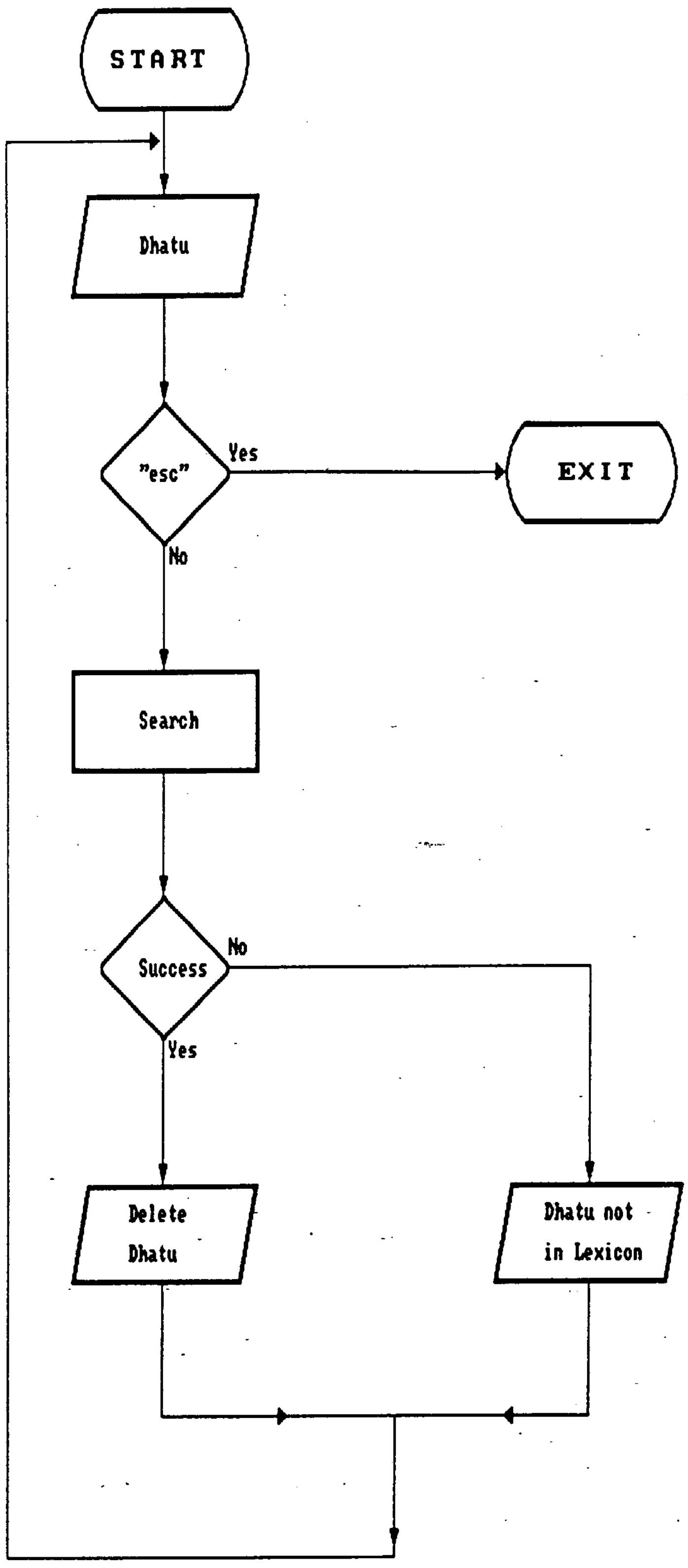


Fig. 4.4 : Flow chart for DELETE procedure

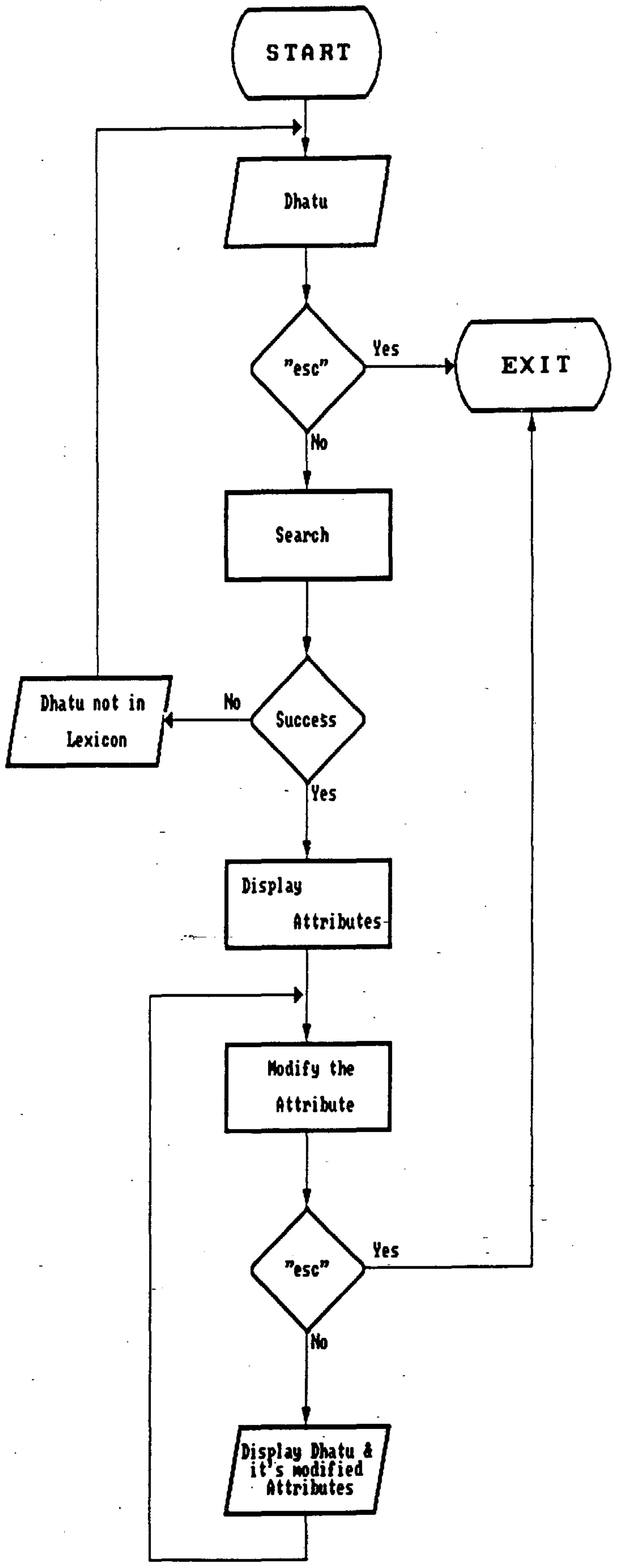


Fig. 4.5 : Flow chart for MODIFY procedure

4.1.2 Pratipadikkosh

This module consists of a nominal lexicon and procedures to operate on this nominal lexicon. In nominal lexicon (i.e. Pratipadikkosh) a *pratipadik* along with its attributes is stored. These attributes are :

1. *Varga*
2. Gender

These attributes are stored in the lexicon through a functor mechanism of Prolog. The **data structure** for this is as :

pratipadik_attrib = pratipadik_attr(Pratipadik,Gender,Varga)

This module opens with a menu displaying the procedures to create, maintain and access the pratipadikkosh. These procedures are **WRITE, READ, DELETE** and **MODIFY**. This menu (i.e.the main menu) appears as shown in A-12. These procedures are menu driven and are interactive in nature. They are described in detail below :

4.1.2.1 WRITE

If the user wants to add a new *pratipadik* to the pratipadikkosh, the **WRITE** procedure from the main menu is selected. Using his procedure various attributes of the *pratipadik* can be stored in the pratipadikkosh. The flow chart for this procedure is as shown in fig. 4.6.

Once the user has selected the **WRITE** procedure from the main menu, a message asking the user to enter the *pratipadik* appears on the screen (as shown in A-13). After the pratipadik has been entered the gender of the *pratipadik* is required. The user selects this from the submenu (as shown in A-14).

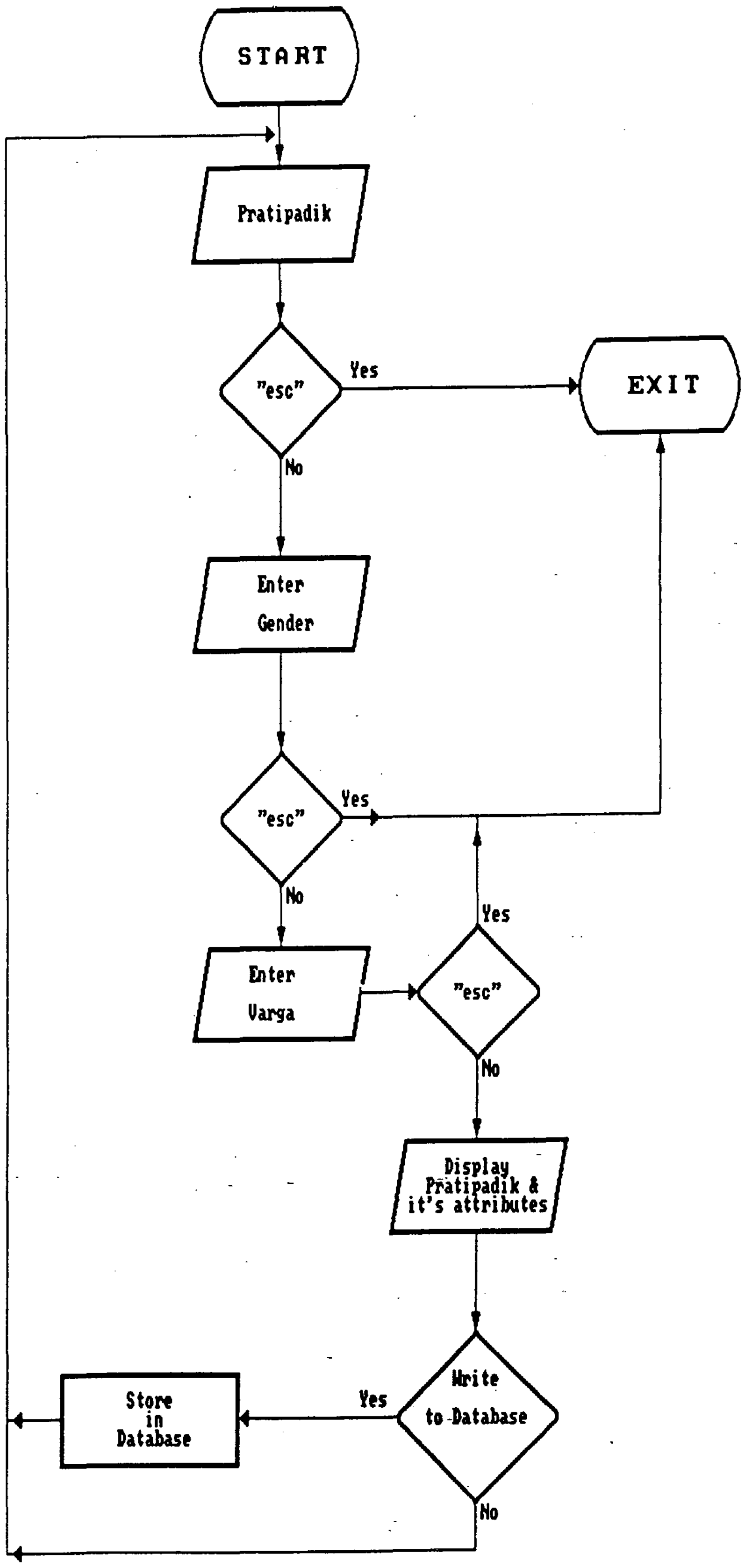


Fig. 4.6 : Flow Chart for WRITE procedure

Similarly the user is asked to select the *varga* to which the *pratipadik* belongs from the submenu (as shown in A-15).

Once the above information is entered, this modules provides an option either for storing the information or to abandon it. For storing, the key "F1", and for abandoning, the key "F2" is used.

4.1.2.2 READ

If the user wishes to know the attributes of a *pratipadik* which is present in the pratipadikkosh, the **READ** procedure from the main menu is selected. On selecting this procedure the screen appears as shown in A-16. The flow chart for this procedure is as shown in fig. 4.7.

This procedure displays the attributes of the given *pratipadik.*

4.1.2.3 DELETE

If a particular *pratipadik* present in the pratipadikkosh is not required, the user can remove or delete the same from the pratipadikkosh by selecting the **DELETE** procedure from the main menu. On selecting this procedure the screen appears as shown in A-17. The flow chart for this procedure is given in fig. 4.8.

This procedure deletes the *pratipadik* and its attributes.

4.1.2.4 MODIFY

To make modifications in a *pratipadik* (i.e. its attributes) present in the pratipadikkosh the

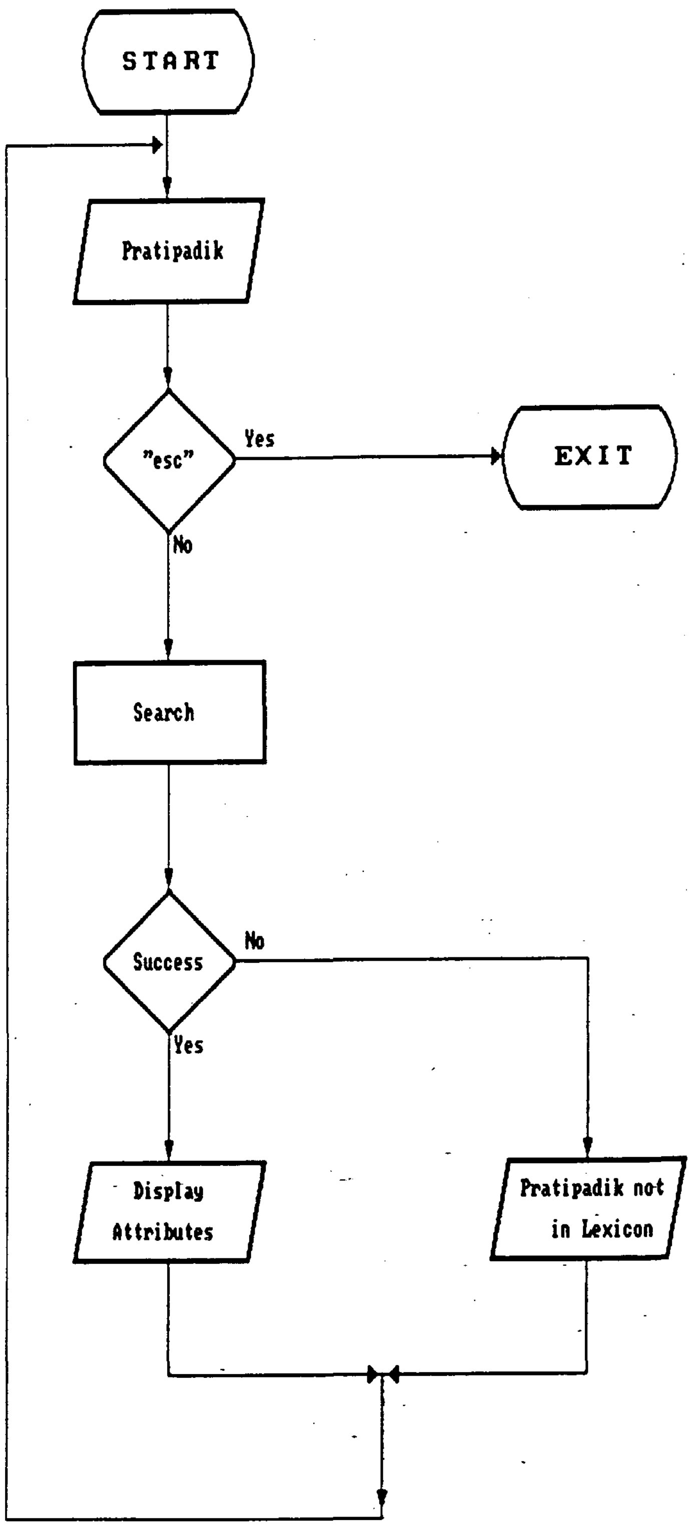


Fig. 4.7 : Flow chart for READ procedure

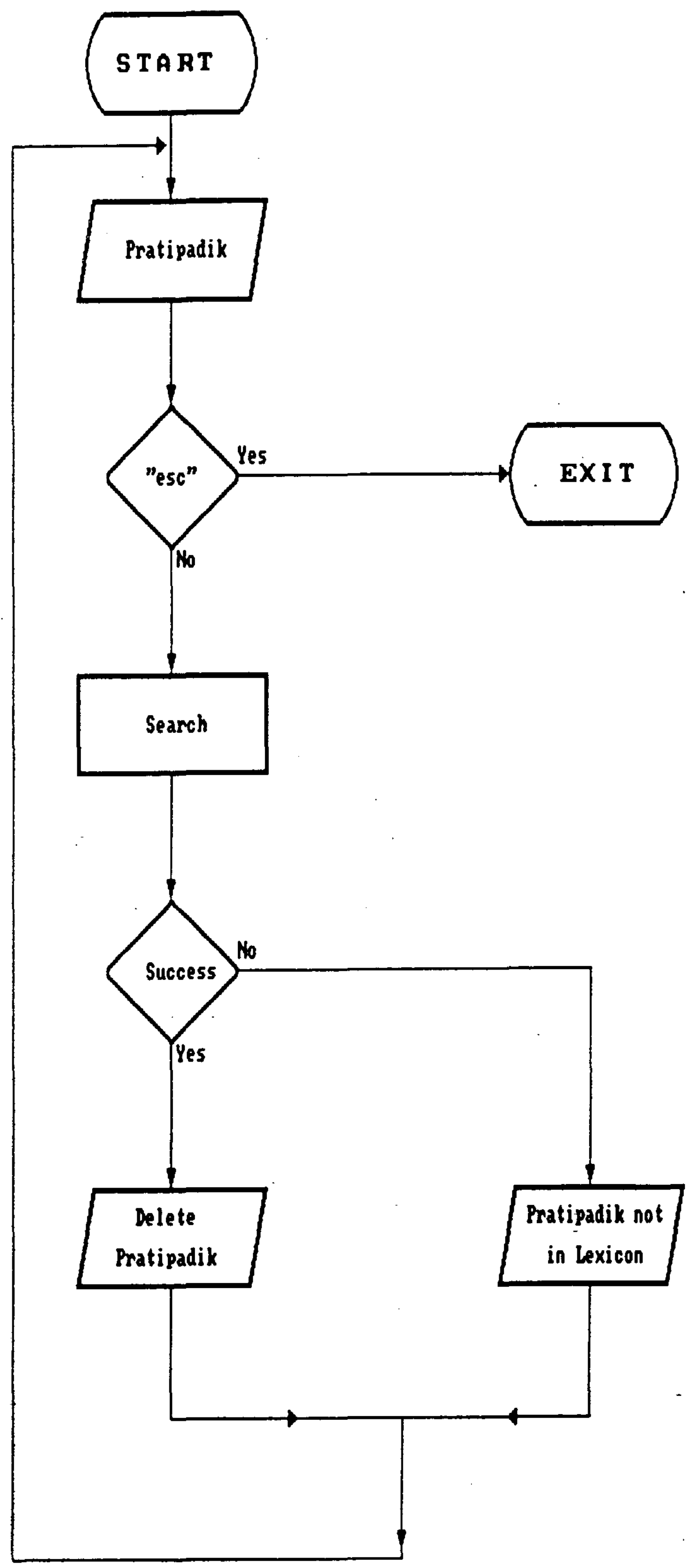


Fig. 4.8 : Flow chart for DELETE procedure

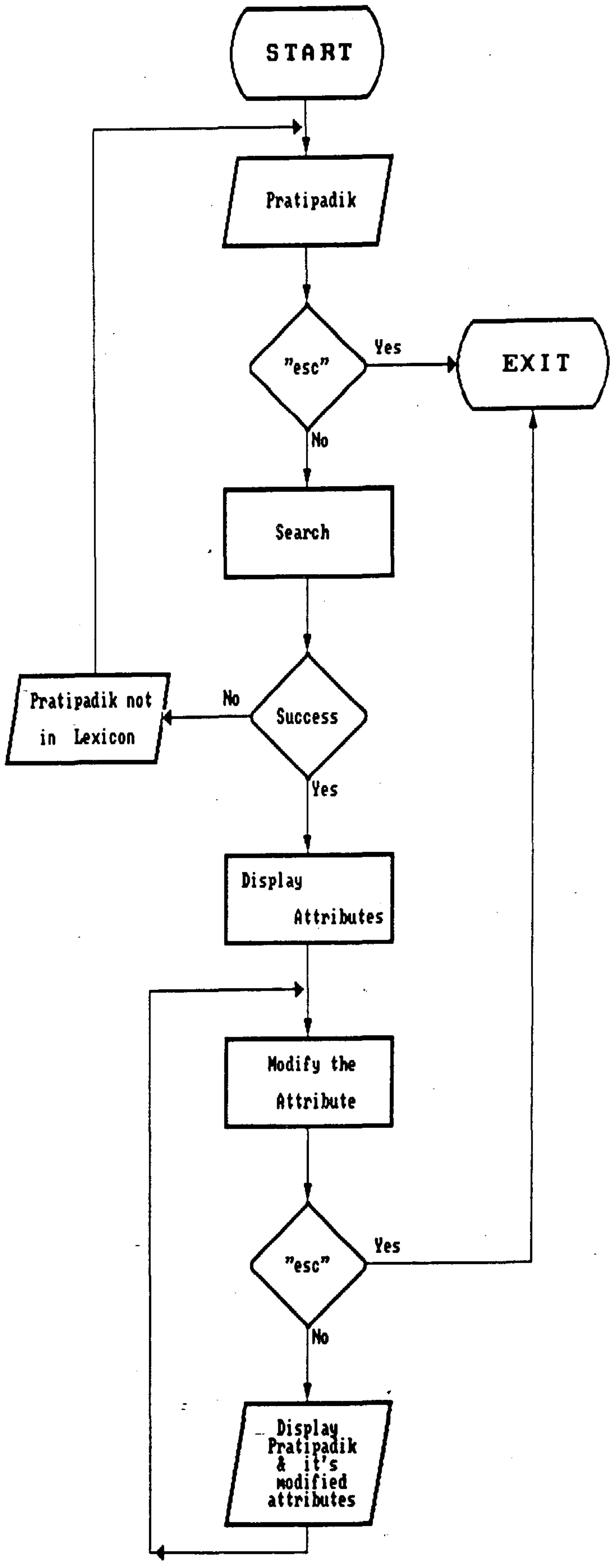


Fig. 4.9 : Flow chart for MODIFY procedure

MODIFY procedure is selected from the main menu. The flow chart for this procedure is as shown in fig. 4.9.

On selecting this procedure all the previous attributes of the *pratipadik* along with a menu appears on the monitor asking the user to select which of the previous attributes is required to be changed (given in A-18). On selecting these attributes, one at a time, one can make the necessary modifications. Once the modifications have been made the modified form is displayed on the monitor.

4.2 Analyzer

The analyzer module consists of two submodules, namely, Reverse_subant procedure and Reverse_tigant procedure. These are as described below :

4.2.1 Reverse_Subant

Here the *subant pada* is the input to the procedure. This procedure tries to identify the noun-base and the suffix. The approach is based on the heuristics of AI. The output of this procedure is

1. Noun base
2. *Varga*
3. Nominal suffix.

Sometimes there are more than one solution to the input word.

4.2.2 Reverse_Tigant

As in the case of reverse_subant procedure, here also the input to the procedure is the *tigant pada*. This procedure tries to identify the base and suffix parts, based on the heuristic approach. The output of this procedure is

1. Verbal base
2. *Gana*
3. *Pada*
4. *Lakar*
5. *Tigant* suffix.

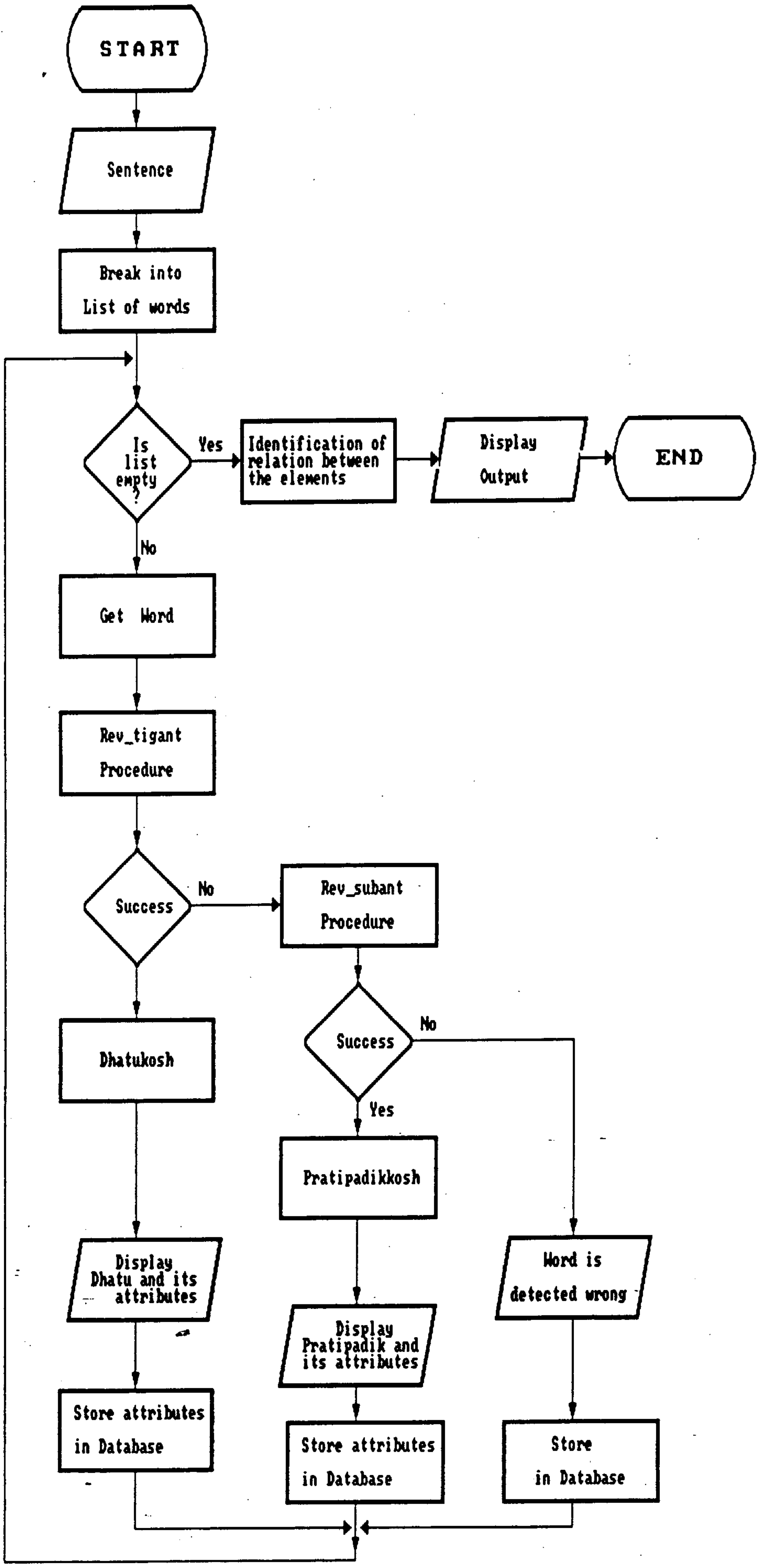
The above procedures are supported by databases wherein the bases and their related information are stored.

4.3 Main Module

Now the details of **Main Module** are described. The flow chart for this module is as shown in fig. 4.10.

The input to the main module is a sentence (as in A-19). The constituent words of the input sentence are first stored in a list. Each element of the list (i.e. the word-form) is then sent to the reverse_tigant procedure for identification. If the word is identified as a tigant pada, then the module gives the corresponding attributes of the verb. These attributes are :

1. *Dhatu*
2. *Tigant Pratyaya*
3. *Pada*
4. *Gana*


START
Sentence
Break into List of words
Is list empty ?
Yes
Identification of relation between the elements
Display Output
END
No
Get Word
Rev_tigant Procedure
Success
No
Rev_subant Procedure
Dhatukosh
Success
No
Yes
Pratipadikkosh
Display Dhatu and its attributes
Word is detected wrong
Display Pratipadik and its attributes
Store attributes in Database
Store attributes in Database
Store in Database

5. *Lakar*

Of these, the *dhatu* is selected and is sent to the dhatukosh. The dhatukosh (described above) contains the list of *karakas*. The *karakas* are further categorized as mandatory *karakas* and optional *karakas*. Every verb invariably takes its mandatory *karaka,* whereas the others are optional. The dhatukosh provides this information based on which further processing can take place.These attributes of the dhatu along with the list of *karakas* are then stored.

If the reverse_tigant procedure fails to identify the input word, it is then sent to the reverse_subant procedure. If the reverse_subant procedure identifies the word, then its corresponding attributes would be the output. These attributes are:

1. *Pratipadik*
2. *Sup Pratyaya*
3. *Vibhakti*
4. *Vachan*

These attributes are then stored by the system.

Of these, the *pratipadik* is sent to the pratipadikosh. The pratipadikkosh then gives the attributes of the same, which are :

1. *varga*
2. gender

If the reverse subant procedure also fails to identify the word, a message is flashed on the monitor stating that the input word is wrong.

A similar process is followed for remaining words of the input sentence. Once all the words in the list are identified, the task of identifying the relation among these elements remains.

In Chapter II, we mentioned that the syntactic relation of the words and their relation to verb are specified by *karakas*. At the grammatical level, these are represented by the case-suffixes. It was also mentioned there, that there is no one-to-one correspondence between the *vibhaktis* and *karakas*. In our system, we assume a straight mapping between these two. We now have the attributes of the verb and the nouns of the input sentence. The sentence is declared correct, if and only the *karakas* determined by the case-suffixes of the nouns match with the list of *karakas* taken by the verb. Thus, if a verb takes the *karma karaka* as mandatory, then a corresponding noun with third case-suffix must be present in the sentence. Otherwise, the sentence is declared wrong.

$$\text{WORDS} \dashrightarrow \underset{1}{\textit{Tigant}} + \underset{n_1}{\text{Mandatory } \textit{Subants}} + \underset{n_2}{\text{Optional } \textit{Subants}}$$

$$1 + n_1 + n_2 < \text{Total Words} < 1 + n_1 \quad \{ \text{WRONG SENTENCE} \}$$

The explanation of the formula follows as :

A sentence is declared to be wrong if the number of words it has is less than the sum of the *tigant* and the mandatory *subants*. Similarly, the sentence is also wrong if the total words it has is more than the sum of the *tigant*, the mandatory *subants* and the optional *subants*.

The above describes the implementational aspects of our system. The results of the system are given in A-20.

CHAPTER V

CONCLUSION

The present work is a modest attempt to develop an analyzer to parse simple Sanskrit sentences. The system developed is based on Paninian Grammatical Framework. The system provides user friendly environment and it is completely menu driven.

Prolog has been chosen as the language for implementation because of its in-built backtracking and pattern matching mechanism which are useful for a natural language processing system.

Though a *vibhakti* can denote more than one *karaka* and vice versa but because of the time limitation, only one-to-one mapping between *karaka* and *vibhakti* has been considered here. In addition to this, the present system does not account for sentences with adjectives, adverbs, participle and passive voice. No hierarchy of nouns is considered. Further, issues relating to semantics and pragmatics are needed to be considered.

Thus, there is enormous scope for future expansion. Once the system is completely developed it can be used in various areas of Natural Language Processing, such as teaching/learning, text understanding machine translation, database interfaces, text generation, etc. if it is complemented by the related work, namely, sentence generation based on verb and *karaka* specifications.

The input specifications and the results of the system are given in appendix.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Allen, James. Natural Language Understanding.
Menlo Park, CA : Benjamin/Cummings, 1987.

[2] Harris, Mary Dee. Introduction to NLP.
Reston : Reston Publishing Company, 1985.

[3] Katre, Sumitra M. Astadhyayi of Panini.
Delhi : Motilal Banarsidas, 1989.

[4] Kiparsky, P. Some Theoretical Problems in Panini's Grammar.
Poona : Bhandarkar Oriental Research Institute, 1982.

[5] Mishkoff, Henry, C. Understanding Artificial Intelligence.
Delhi : BPB Publications, 1986.

[6] Patterson, Dan W. Introduction to Artificial Intelligence and Expert Systems.
New Delhi : Prentice-Hall of India, 1992.

[7] Rich, Elaine. Artificial Intelligence.

New York : McGraw Hill, 1983.

[8] Sager, Naomi. Natural Language Information Processing.

Reading, Mass : Addison-Wesley, 1981.

[9] Singh, G.V., et. al., The Word-Morphology of Sanskrit, Sinha, R.M.K. (ed.), Proceedings of Computer Processing of Asian Language (CPAL-2).

New Delhi : Tata McGraw-Hill, 1992.

[10] Syseca, A G. Prolog for NLP.

John Wiley & Sons, 1985.

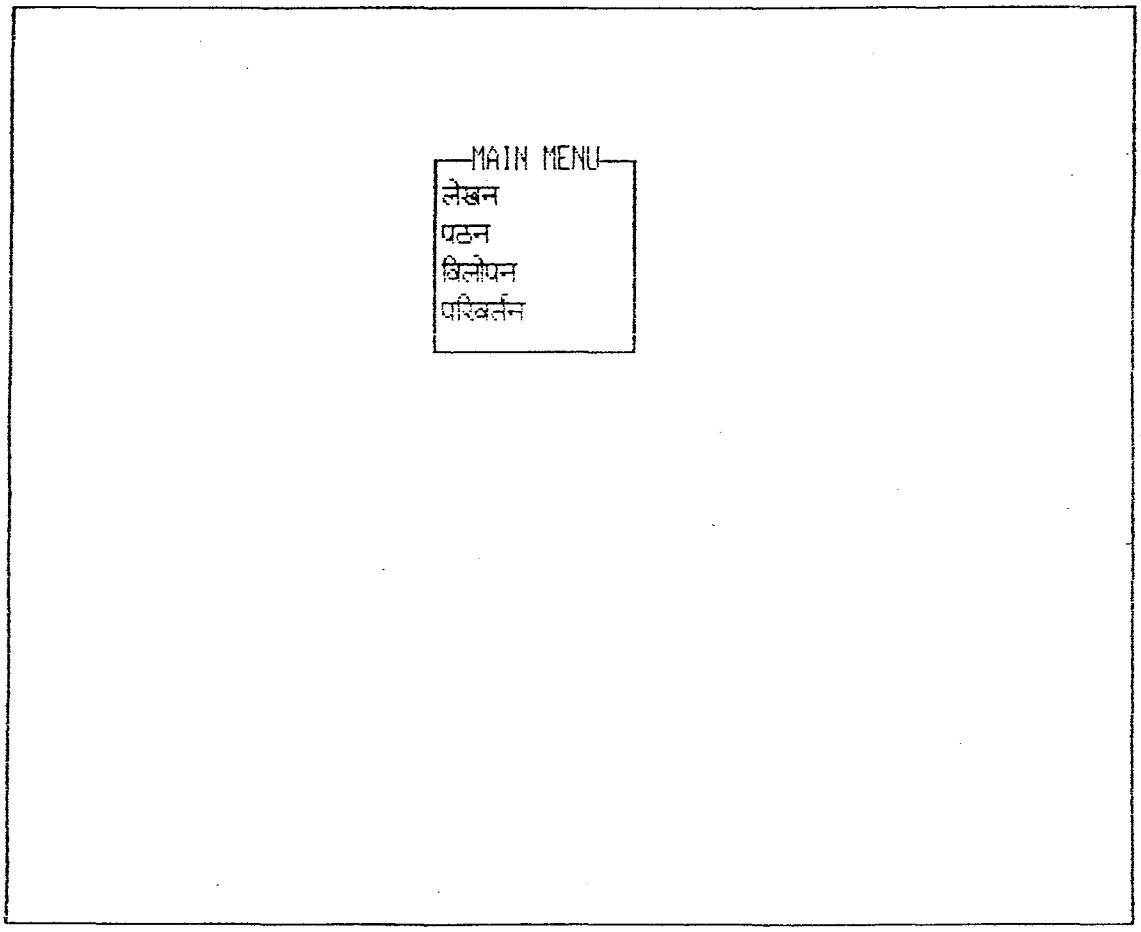
[11] Vasu, S C. The Siddhanta Kaumudi of Bhattoji Diksita.

Delhi : Motilal Banarsidas, 1982.

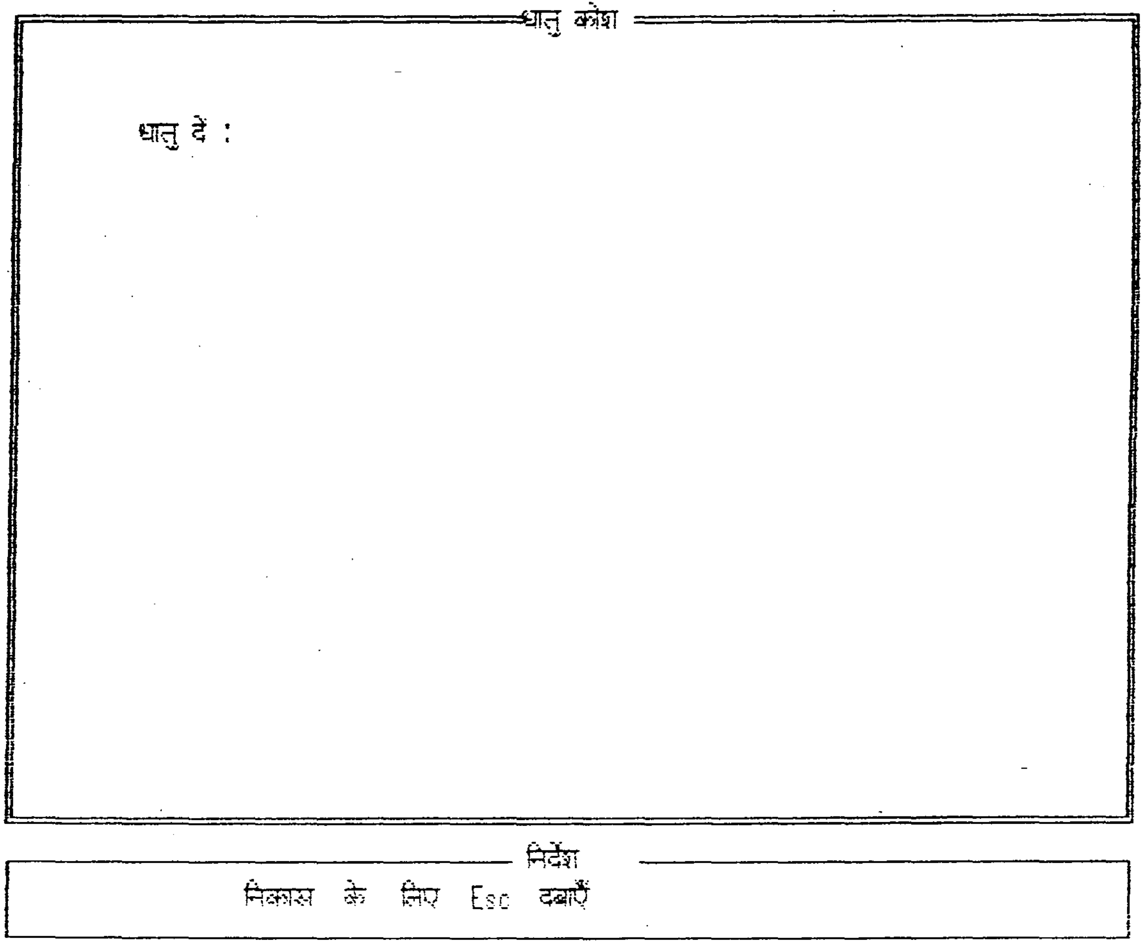
[12] Winograd, T. Language as a Cognitive Process, Volume I : Syntax

Reading, Massachusetts : Addison-Wesley, 1983.

APPENDIX



MAIN MENU
लेखन
पठन
विलोपन
परिवर्तन

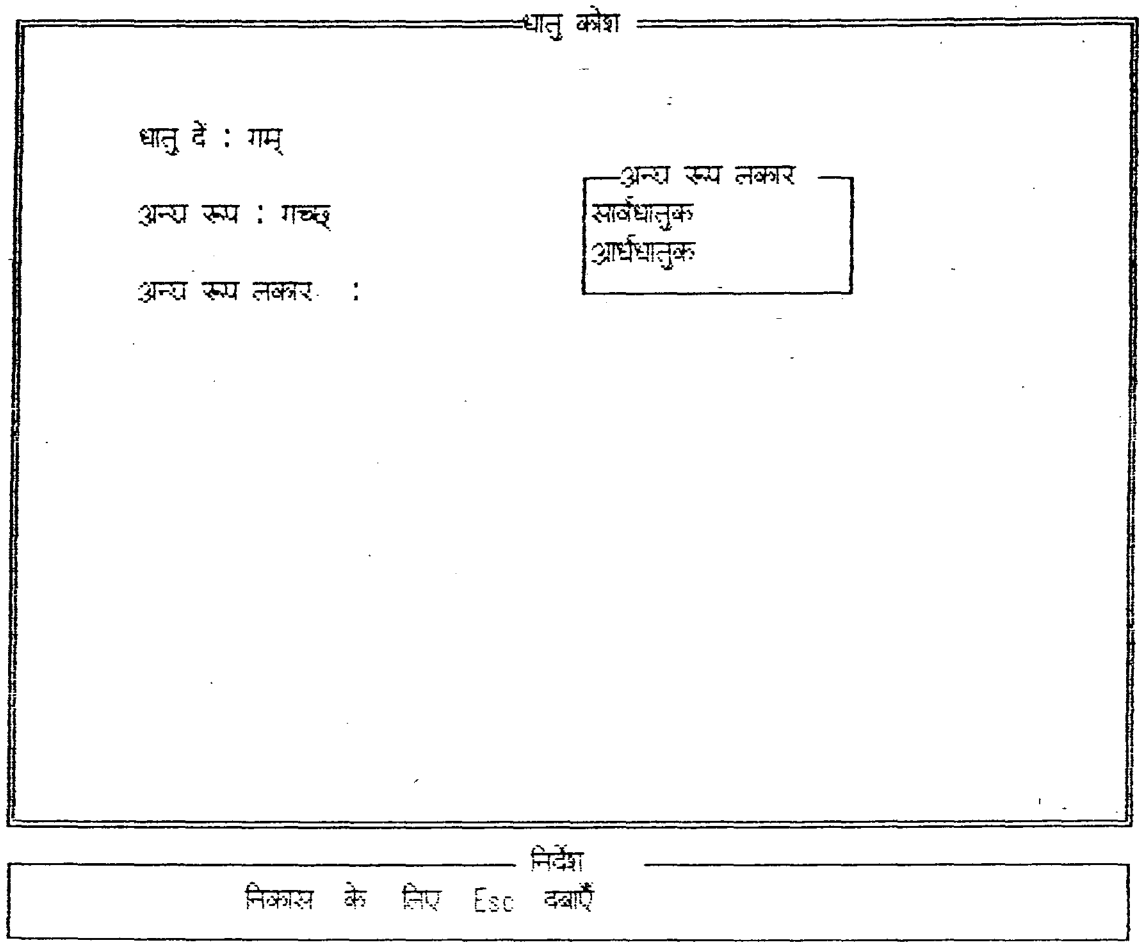

धातु कोश

धातु दें :

निर्देश

निकास के लिए Esc दबाएँ

A-2



धातु कोश

धातु दें : गम्

अन्य रूप : गच्छ्

अन्य रूप लकार :

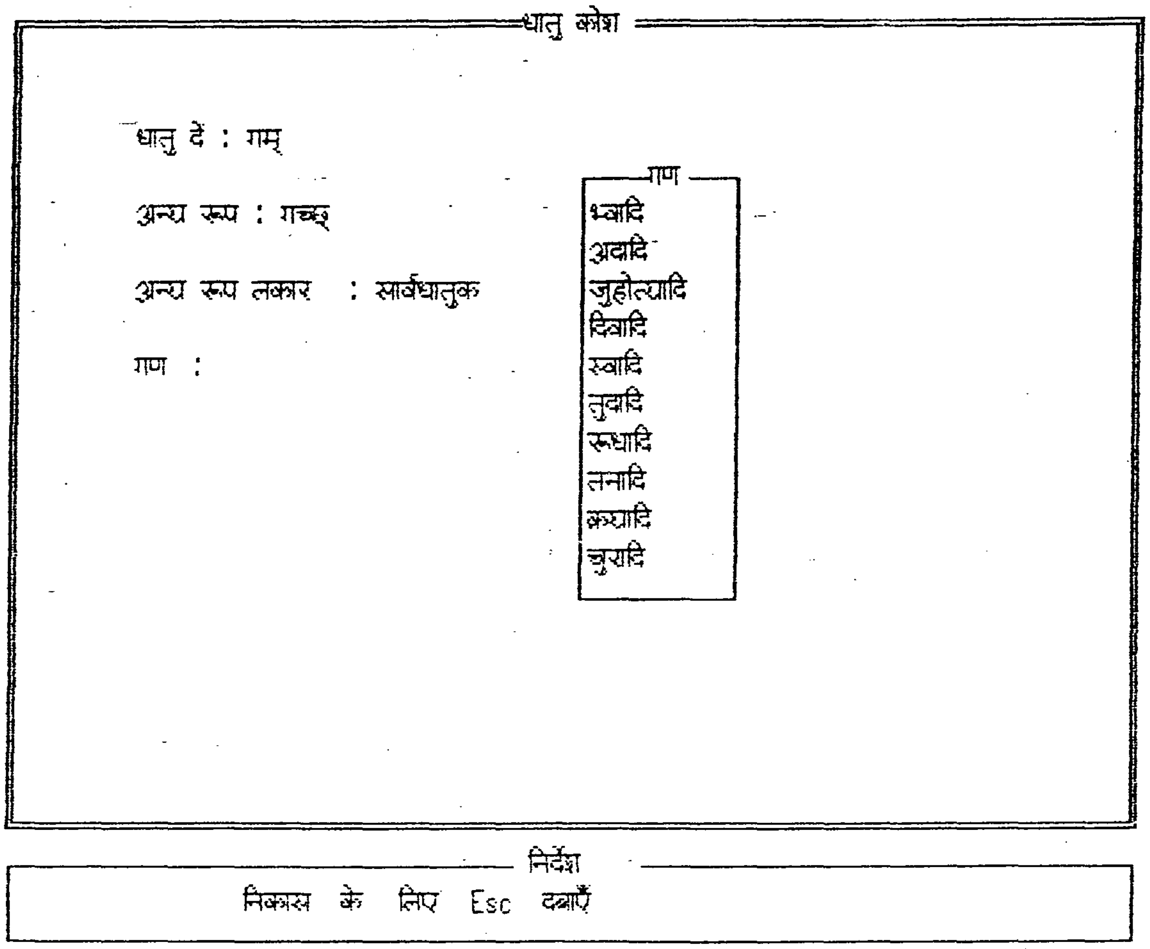
अन्य रूप लकार

सार्वधातुक

आर्धधातुक

निर्देश

निकास के लिए Esc दबाएँ


धातु कोश

धातु दें : गम्

अन्य रूप : गच्छ्

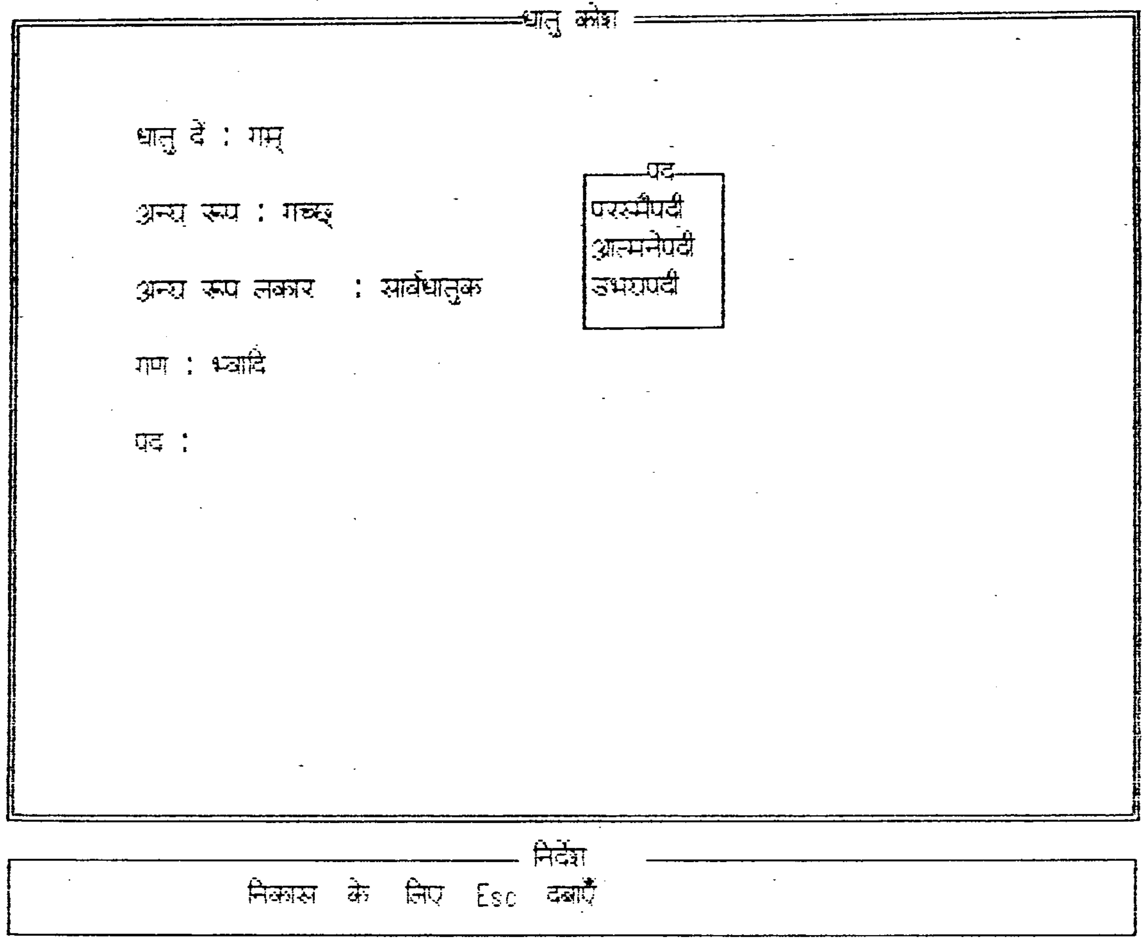
अन्य रूप लकार : सार्वधातुक

गण :

गण
- भ्वादि
- अदादि
- जुहोत्यादि
- दिवादि
- स्वादि
- तुदादि
- रूधादि
- तनादि
- क्र्यादि
- चुरादि

निर्देश

निकास के लिए Esc दबाएँ


धातु कोश

धातु दें : गम्

अन्य रूप : गच्छ्

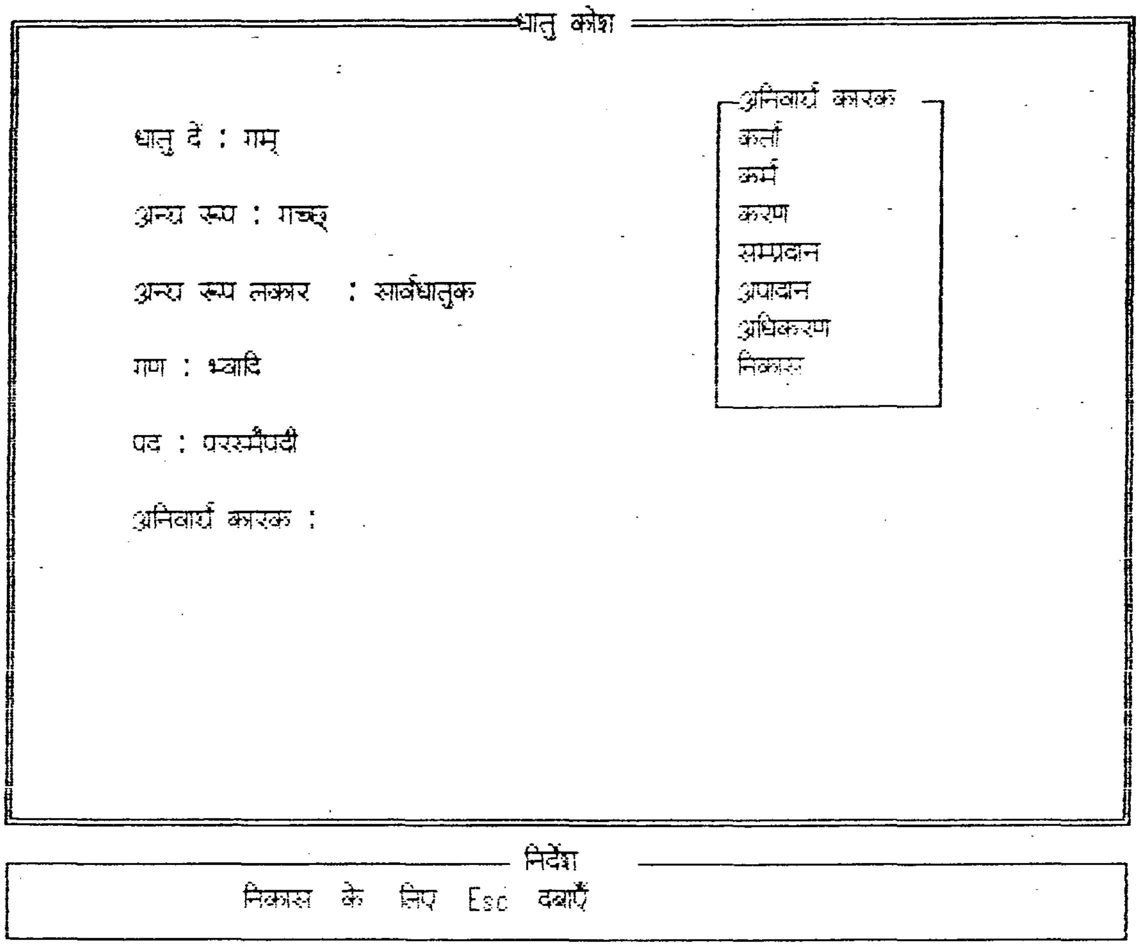
अन्य रूप लकार : सार्वधातुक

गण : भ्वादि

पद :

पद
परस्मैपदी
आत्मनेपदी
उभयपदी

निर्देश
निकास के लिए Esc दबाएँ



धातु कोश

धातु दें : गम्

अन्य रूप : गच्छ्

अन्य रूप लकार : सार्वधातुक

गण : भ्वादि

पद : परस्मैपदी

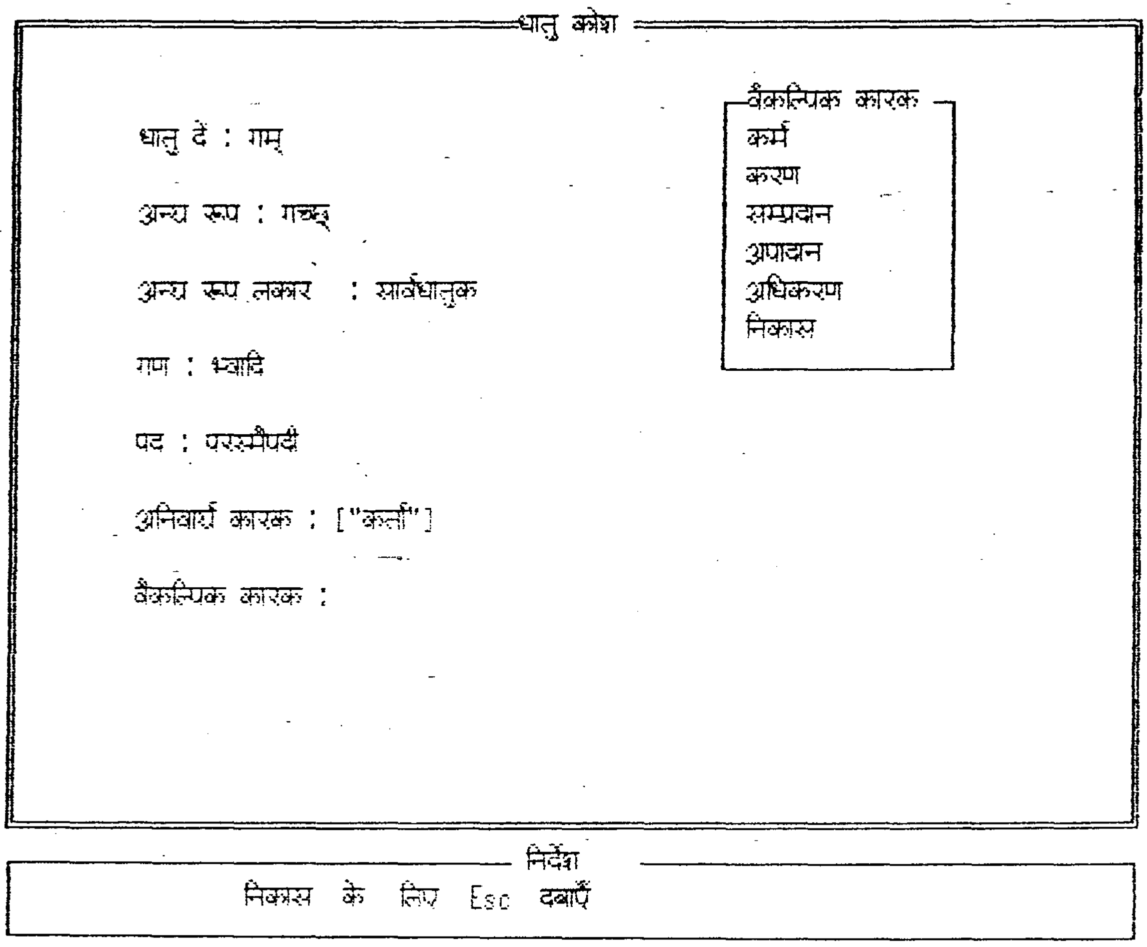
अनिवार्य कारक :

अनिवार्य कारक

- कर्ता
- कर्म
- करण
- सम्प्रदान
- अपादान
- अधिकरण
- निकास

निर्देश

निकास के लिए Esc दबाएँ



धातु कोश

धातु दें : गम्

अन्य रूप : गच्छ्

अन्य रूप लकार : सार्वधातुक

गण : भ्वादि

पद : परस्मैपदी

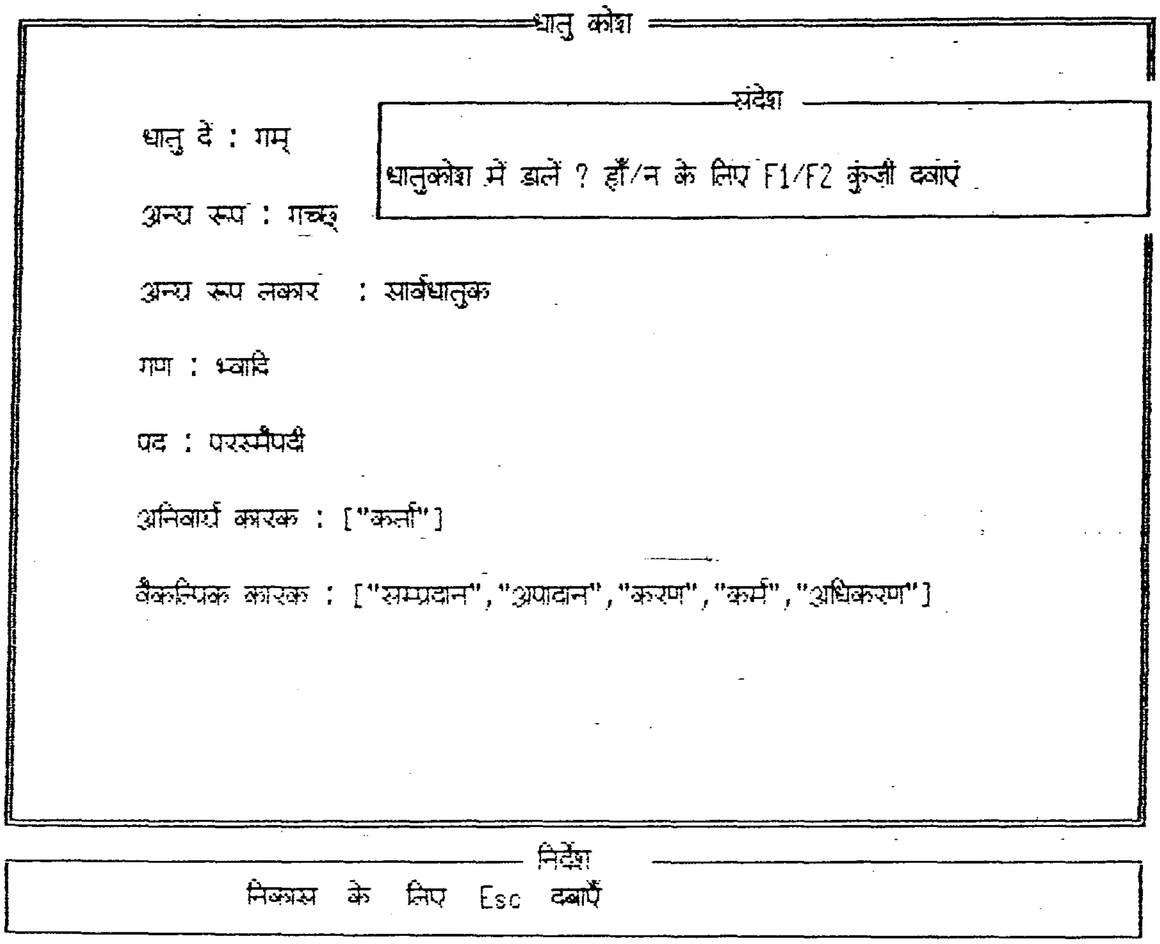
अनिवार्य कारक : ["कर्ता"]

वैकल्पिक कारक :

वैकल्पिक कारक
- कर्म
- करण
- सम्प्रदान
- अपादान
- अधिकरण
- निकास

निर्देश

निकास के लिए Esc दबाएँ



धातु कोश

संदेश

धातुकोश में डालें ? हाँ/न के लिए F1/F2 कुंजी दबाएं

धातु दें : गम्

अन्य रूप : गच्छ्

अन्य रूप लकार : सार्वधातुक

गण : भ्वादि

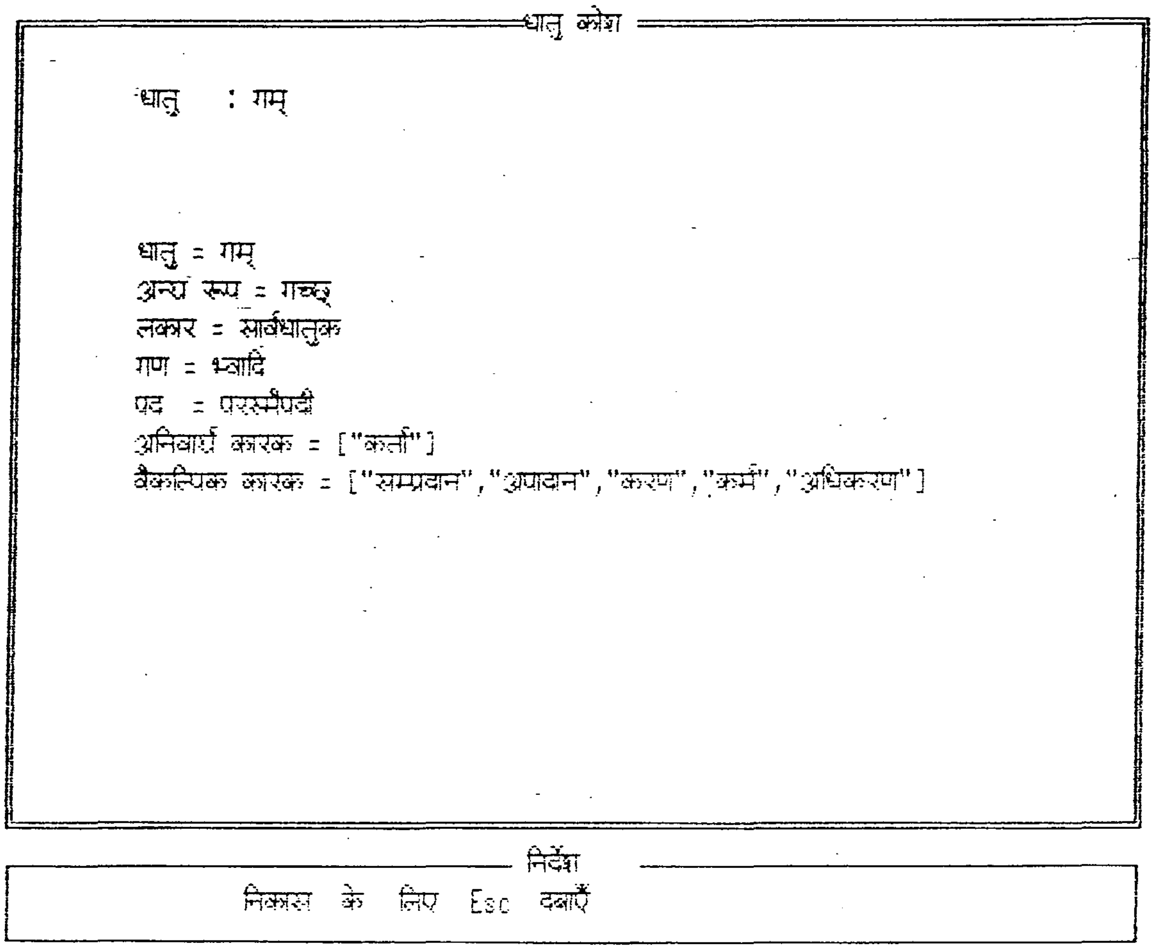
पद : परस्मैपदी

अनिवार्य कारक : ["कर्त्ता"]

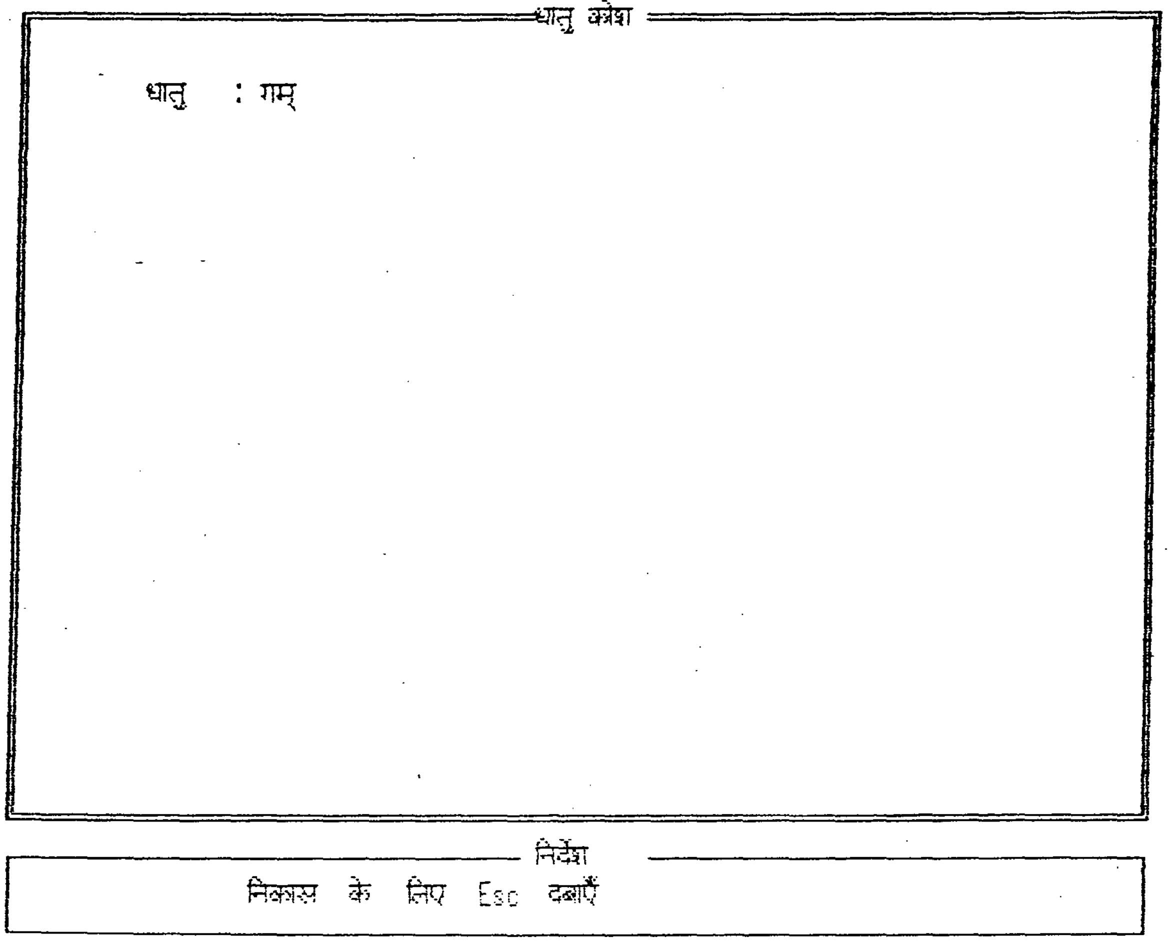
वैकल्पिक कारक : ["सम्प्रदान","अपादान","करण","कर्म","अधिकरण"]

निर्देश

निकास के लिए Esc दबाएँ



```
धातु    : गम्



धातु = गम्
अन्य रूप = गच्छ्
लकार = सार्वधातुक
गण = भ्वादि
पद  = परस्मैपदी
अनिवार्य कारक = ["कर्ता"]
वैकल्पिक कारक = ["सम्प्रदान","अपादान","करण","कर्म","अधिकरण"]
```

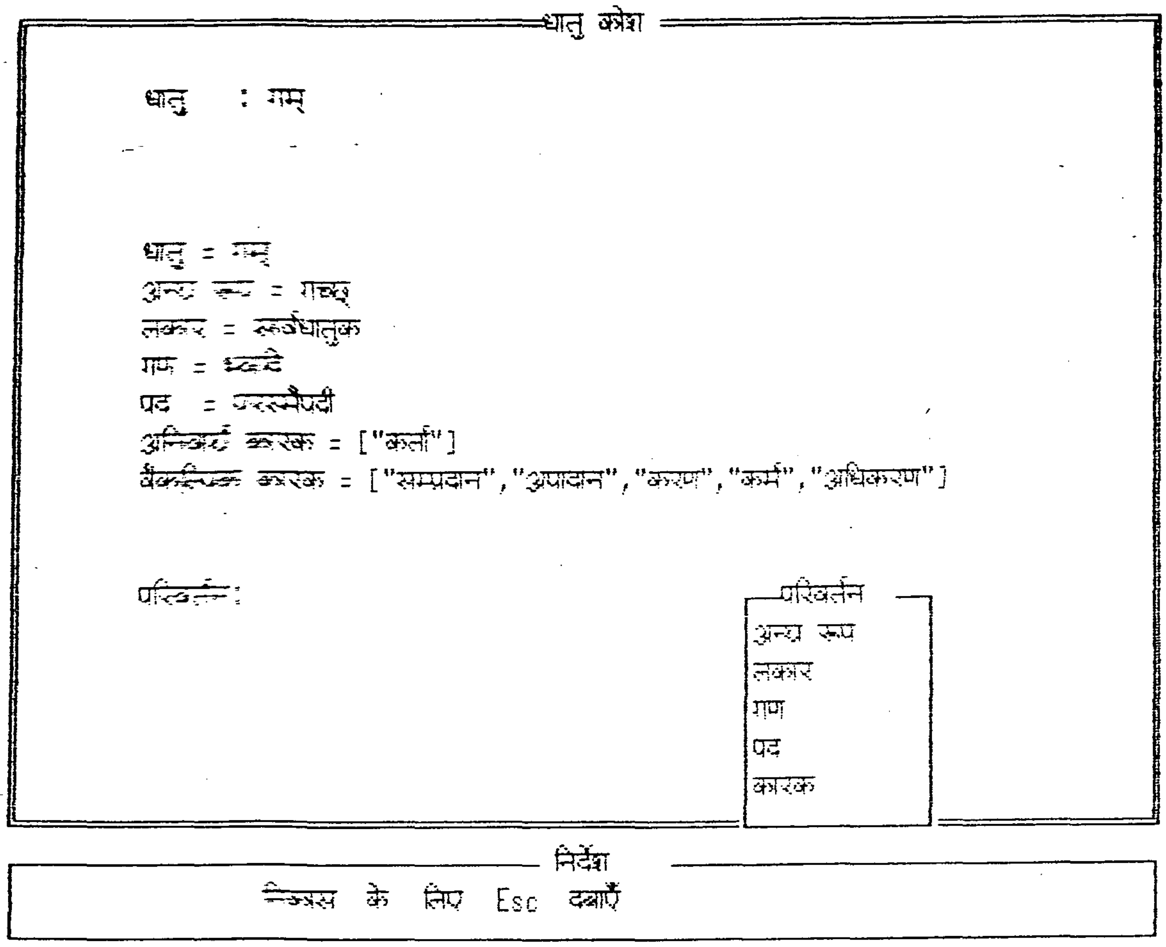


धातु प्रवेश

धातु : गम्

निर्देश

निकास के लिए Esc दबाएँ

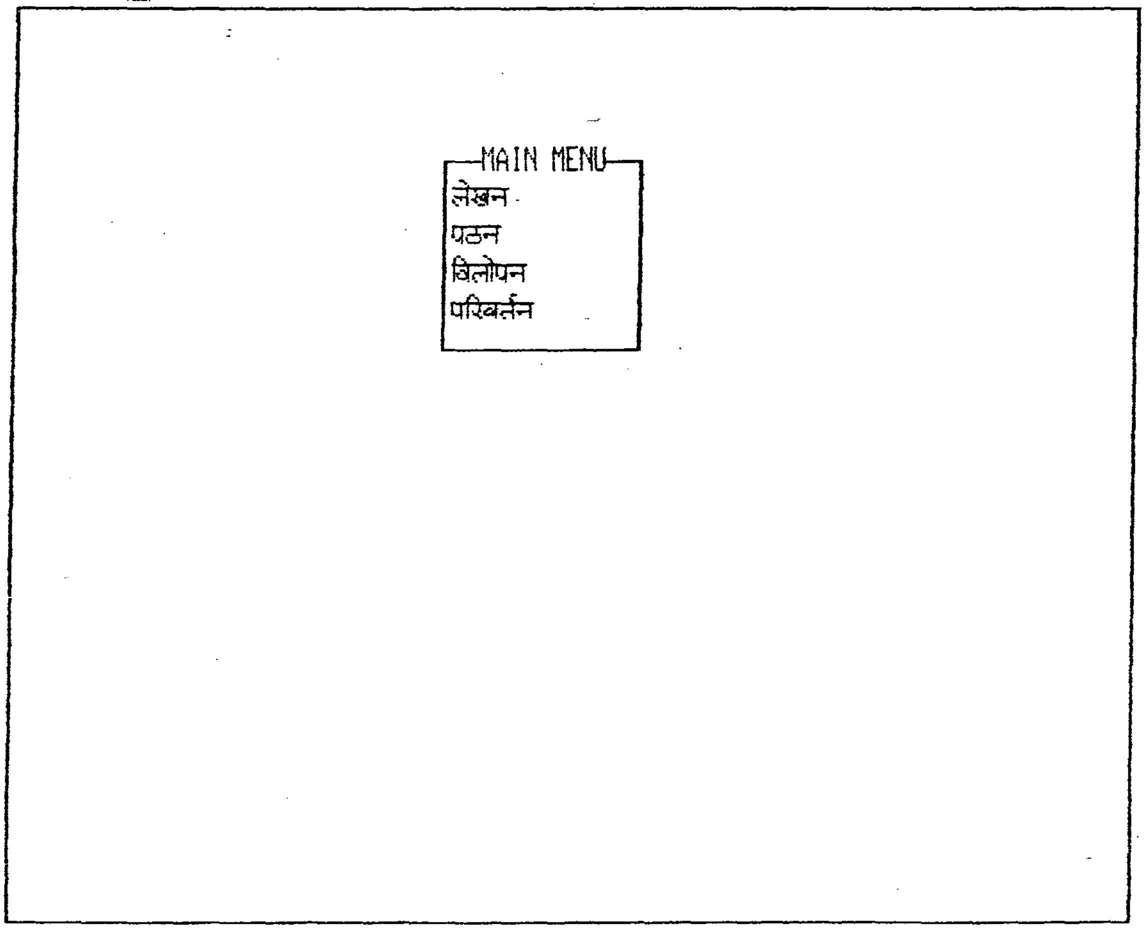


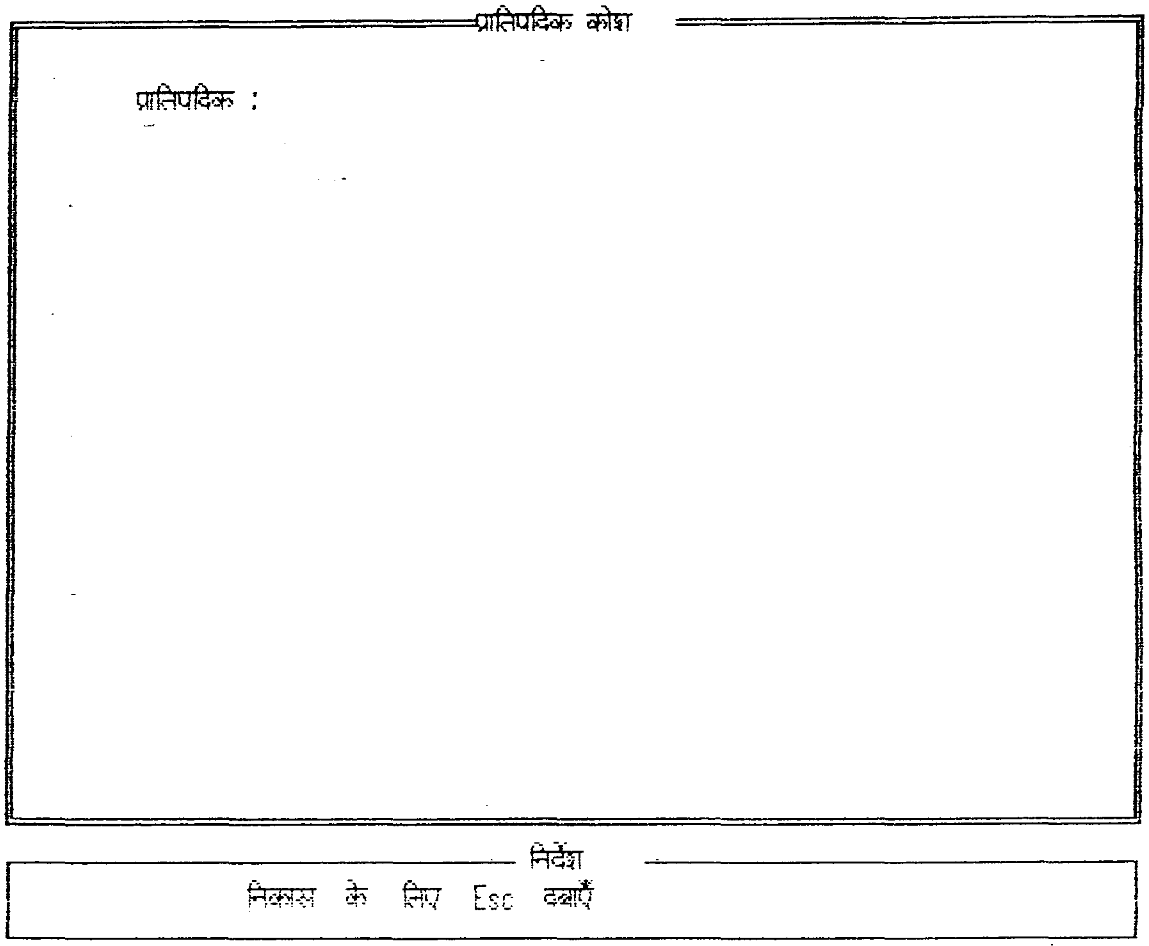
धातु कोश

```
धातु    : गम्


धातु = गम्
अन्य रूप = गच्छ्
लकार = सार्वधातुक
गण = भ्वादि
पद  = परस्मैपदी
अनिवार्य कारक = ["कर्ता"]
वैकल्पिक कारक = ["सम्प्रदान","अपादान","करण","कर्म","अधिकरण"]


परिवर्तन:
```

निर्देश

निकास के लिए Esc दबाएँ


MAIN MENU
लेखन
पठन
विलोपन
परिबर्तन

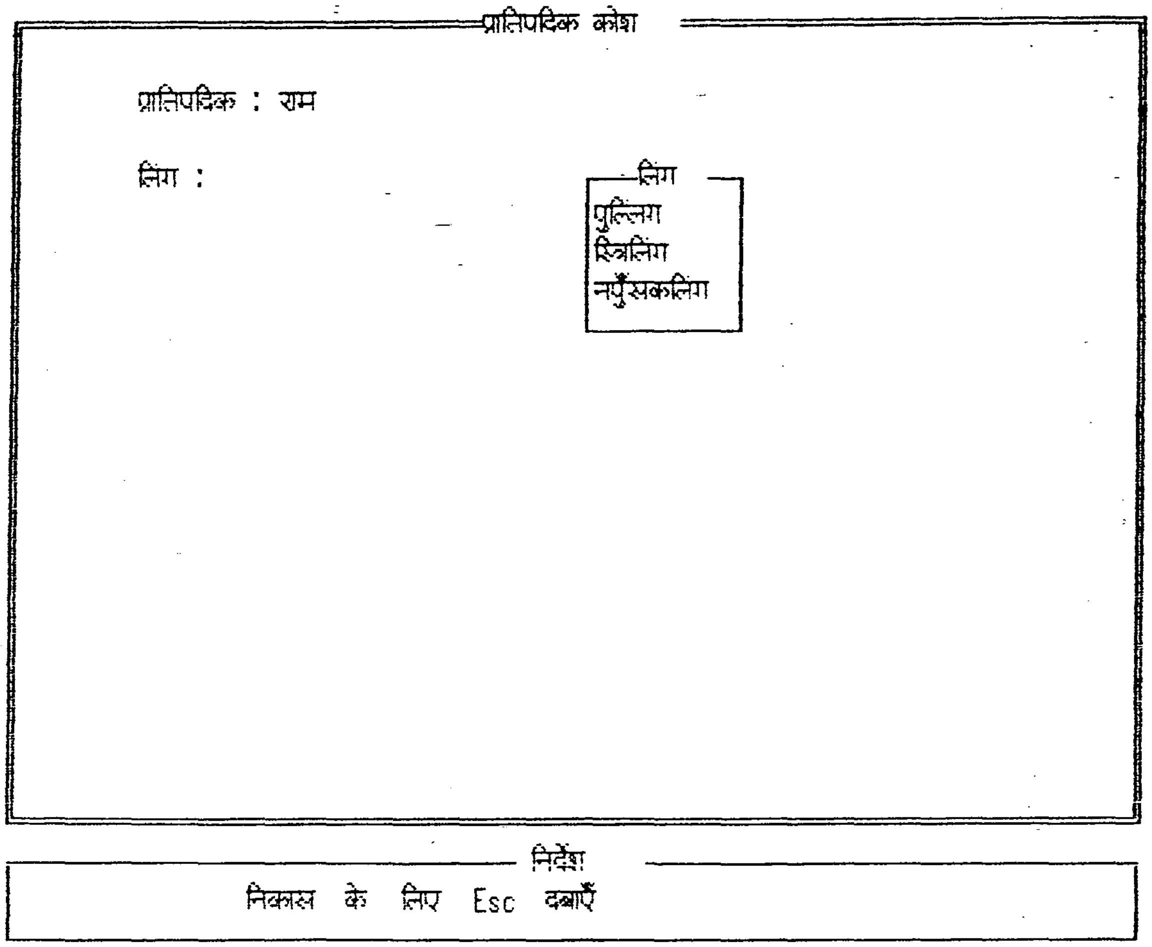


प्रातिपदिक कोश

प्रातिपदिक :

निर्देश

निकास के लिए Esc दबाएँ



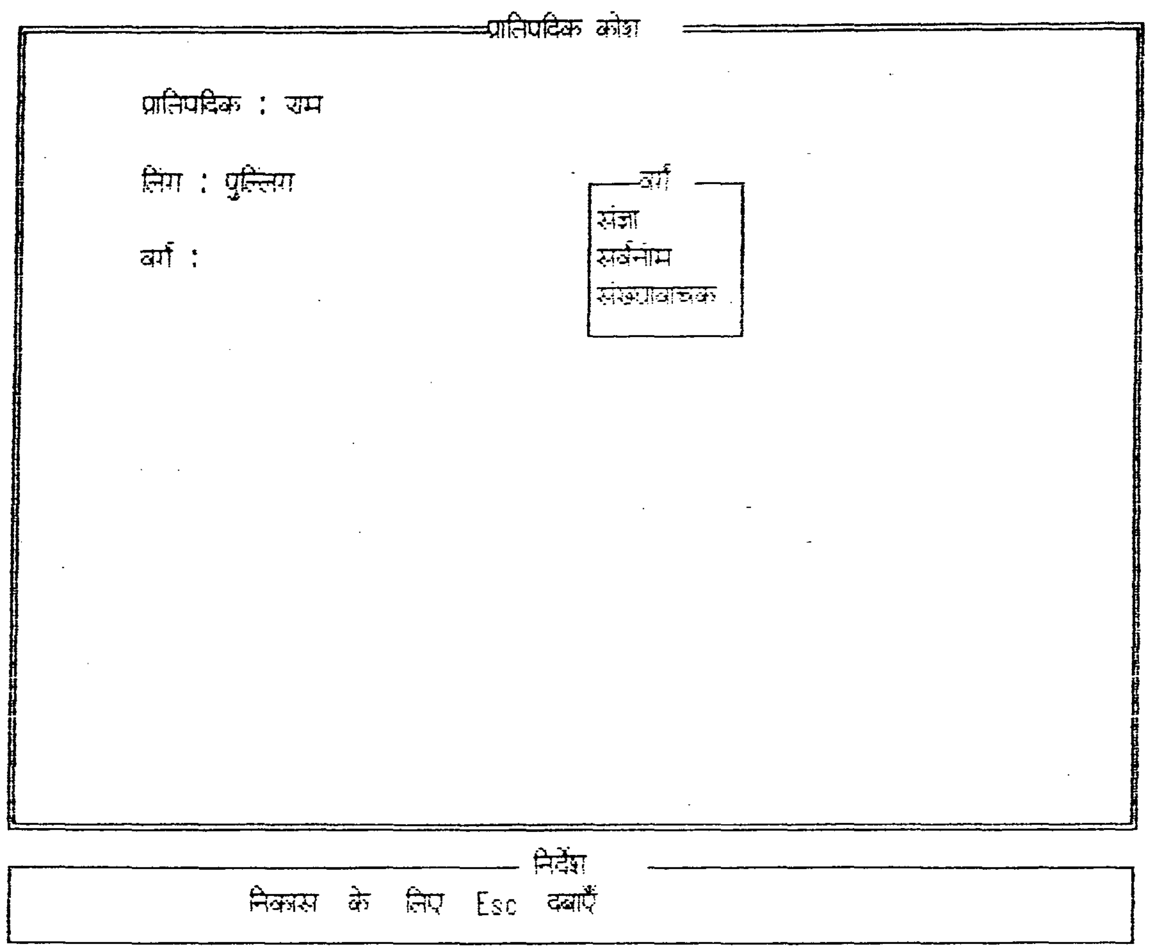
प्रातिपदिक कोश

प्रातिपदिक : राम

लिंग :

निर्देश

निकास के लिए Esc दबाएँ



प्रातिपदिक कोश

प्रातिपदिक : राम

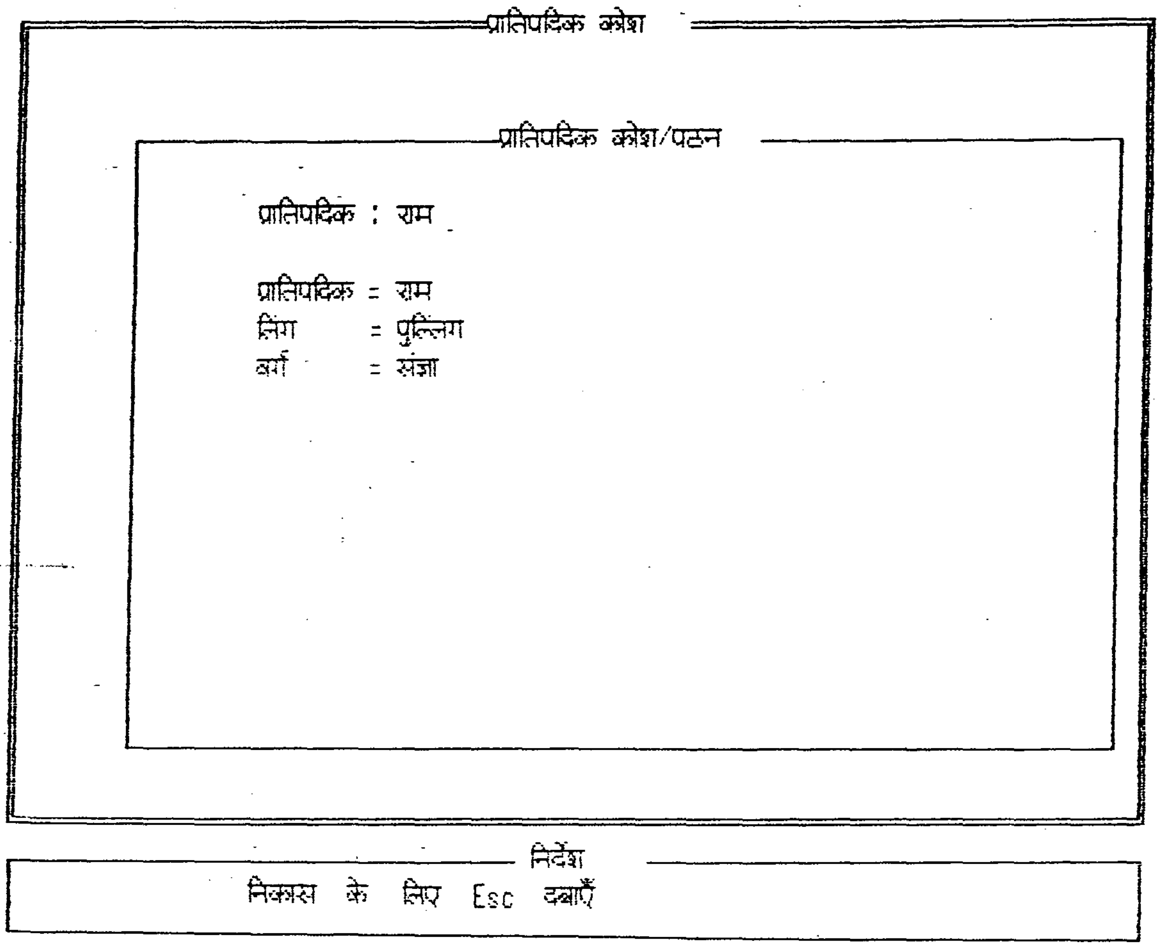
लिंग : पुल्लिंग

वर्ग :

वर्ग
संज्ञा
सर्वनाम
संख्यावाचक

निर्देश

निकास के लिए Esc दबाएँ


प्रातिपदिक कोश

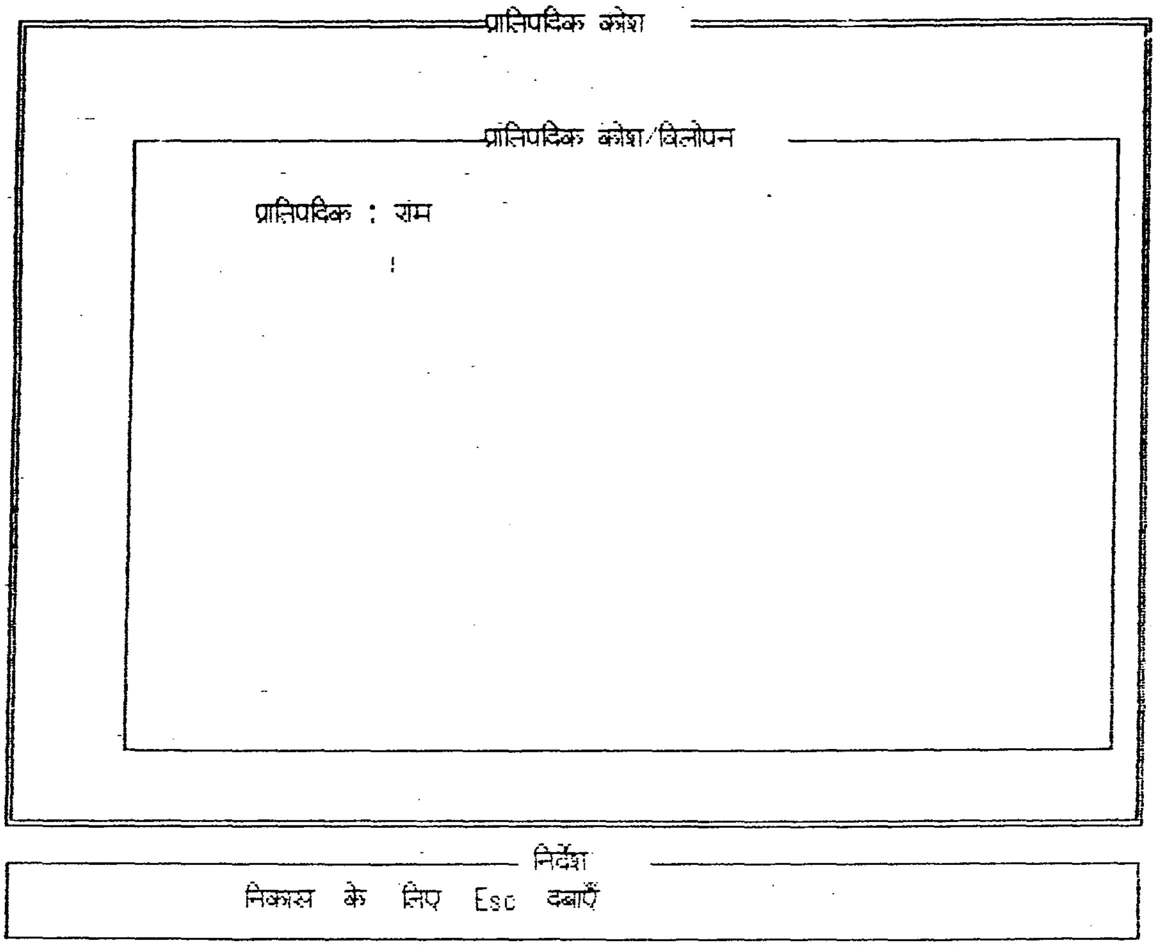
प्रातिपदिक कोश/पठन

प्रातिपदिक : राम

```
प्रातिपदिक = राम
लिंग     = पुल्लिंग
वर्ग      = संज्ञा
```

निर्देश

निकास के लिए Esc दबाएँ

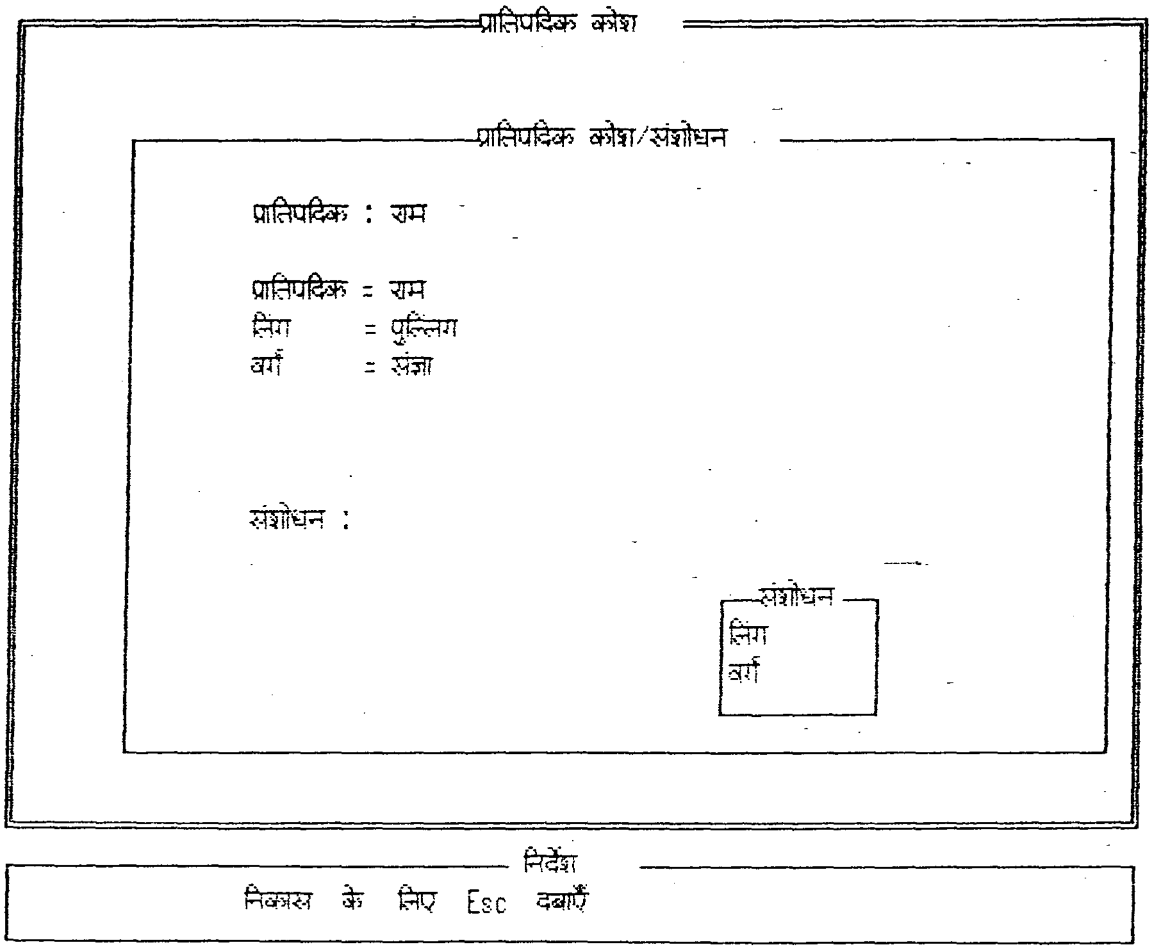

प्रातिपदिक कोश

प्रातिपदिक कोश/विलोपन

प्रातिपदिक : राम

निर्देश

निकास के लिए Esc दबाएँ



प्रातिपदिक कोश

प्रातिपदिक कोश/संशोधन

प्रातिपदिक : राम

प्रातिपदिक = राम
लिंग = पुल्लिंग
वर्ग = संज्ञा

संशोधन :

संशोधन
लिंग
वर्ग

निर्देश
निकास के लिए Esc दबाएँ

```
                    वाक्य विश्लेषण

  वाक्य : रामः ग्रामम् गच्छति
```

वाक्य विश्लेषण

```
वाक्य : रामः ग्रामम् गच्छति
शब्द = रामः
          प्रातिपदिक = राम
          लिंग = पुल्लिंग
          वर्ग = संज्ञा
शब्द = ग्रामम्
          प्रातिपदिक = ग्राम
          लिंग = नपुंसकलिंग
          वर्ग = संज्ञा
शब्द = गच्छति
          धातु = गम्
          अन्यरूप = गच्छ्
          अन्यरूप लकार = सार्वधातुक
          गण = भ्वादि
          पद  = परस्मैपदी
          अनिवार्य कारक = ["कर्ता"]
          ऐच्छिक कारक = ["सम्प्रदान","अपादान","करण","कर्म","अधिकरण"]
```