1104

# COMPUTATIONAL LEXICON: Specification and Design

*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

By

**SEYOUM MITIKU MERSHA**

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI - 110067
INDIA

# CERTIFICATE

This is to certify that the dissertation entitled "**COMPUTATIONAL LEXICON: Specification and Design**" being submitted by me to **Jawaharlal Nehru University** in partial fulfilment of the requirements for the award of the degree of **Master of Technology in Computer Science and Technology** is a record of 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 full to any other University or Institution for the award of any degree.

[illegible]

**(SEYOUM MITIKU MERSHA)**

[illegible] 5-1-94

**Prof. K.K Bharadwaj**
**Dean,**
**School of Computer &**
**Systems Sciences,**
**J.N.U., New Delhi,**
**INDIA.**

[illegible]

**Prof. G.V. Singh**
**School of Computer &**
**Systems Sciences,**
**J.N.U., New Delhi,**
**INDIA.**

# ACKNOWLEDGEMENTS

This dissertation work has not been happened without the help of many persons. I take this opportunity to express my gratitude to Prof. G.V. Singh for his valuable guidance, patience and efforts in dealing with my various technical and personal problems during the completion of this work.

I would like to thank Prof. K.K Bharadwaj, Dean, SC&SS, JNU and all the faculty members of SC & SS, JNU for providing me adequate facilities and knowledge which helped me in accomplishing this task.

I thank Department of Electronics (DoE), for using Computer facilities under the project CASTLE (Computer Aided Sanskrit Teaching and Learning Environment) sponsored by DoE.

It is my pleasure to thank Mr. D. K. Lobiyal, Mr. Pramod Gupta, Mr. Amitav Bhattacharjee, Mr. H. Nabil and Mr. Saad Ezzat, for their friendship and constant support while I was doing this project.

I would like to extend my thanks to my Ethiopan friends in Delhi, Specially, to Mr. Haile Eyesus Engida Eshet for his valuable suggestions in solving some of my linguistic difficulties.

Finally, I am extremely indebted to my wife Kidist Tadesse for her patience and moral support during this work.

**Seyoum Mitiku Mersha**

January, 1994

New Delhi

TO MY MOTHER

# ABSTRACT

This Thesis deals with the structuring of the computational lexicon, and its Software Specification and Design. It describes how the parts of speech and word features, for purely syntactic parser, and the patterns of relationships between words and word concepts, for syntactic and semantic analysis respectively, are represented in lexicon.

Under specification it sort out the problems and data flow diagram for the processes into and out of lexicon and its interfaces.

Under design it discuses the data definitions and module specification of the software in pascal like pseudo code.

As any language has enormous number of vocabularies, storing each as a separate entry will slow search activity. To enhance this the technique used is to store only morphemes (morpheme is a minimal meaningful unit which can be used independently in a language, i.e. root word).

Many words may share the same information. Storing information repeatedly along with each word will cause unnecessary space consumption. To avoid this, the lexicon is organized in such a way that the stored information can be shared between words.

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. Thus lexicon may be defined as a collection of lexemes along with its interfaces such as recognizer, generator, and grammar.

Lexicon is organized in word basis of major word classes, as follows.

i. Noun and pronoun are taken as lexeme of **NOUN** frame.

ii. Some pronoun, adjective, quantifier, and article are taken as lexeme of **ADJECTIVE** frame.

iii. Tensed verbs, tenseless verbs (infinitive) are taken as **VERB** frames.

iv. **ADVERB** frame contains adverb.

Other connectives, because of their individual properties or may be treated as part of the grammar, appear as a **LITERAL** in the lexicon.

Some words may be understood in more than one way. These elements will accordingly be represented in lexicon with the same entry but different frames to their various senses.

The relationships between words are represented by tuples. These tuples are the two words, as nodes, and the connection between them, as relational arc. Most of the feature of word are stored in lexicon in attribute form, which may be encoded by taking its first character. Case roles (Theta role) which represent the relation between an action and an object in a sentence has been encoded in the structure of lexicon under each verb as Required, Not Required or Optional.

During software design of lexicon the design approach, adopted is Structured design.

The main operations, which are considered to be applied on the Lexicon for the correctness and building of the database are **DELETE, UPDATE** and **STORE. RETRIEVE** is for looking to the content of the lexicon and **RECOGNIZER** is an interface between the users (like parser), and the operations of Lexicon. It recognizes a word, if the word is in the Lexicon, it provides a service along with other interfaces. **EXTRACT_FEATURE** is another interface, which use RECOGNIZER, for providing the necessary information of the word.

# TABLE OF CONTENTS

# CHAPTER I

## INTRODUCTION

Today there are many formal languages that the computer can understand (such as C, C++, Prolog, etc...) which serve for large class of problems. Though we already have such facilities, people want to communicate with a Computer in Natural Language. The more pertinent question is really, when is natural language, such as English, input to a computer is preferable?

Natural Language input is desirable, for example if it is necessary to use computer for retrieval of information from a text in some language. If a computer could accept that particular language input, much information now recorded only in the same language would be available for computer use without need for human translation.

A computer that understood English would be more accessible to any speaker of English. Especially for occasional use where it would not be worthwhile to train the user in specialized language.

Programming languages are process oriented. They cannot describe a problem. They are only a method for finding a solution to a problem. A Natural Language is convenient vehicle for providing a description of the problem itself, leaving the choice of processing to the problem solver accepting inputs.

"...if one can learn how to make a computer understand a natural language it is big step towards creating an artificially intelligent computer" [6].

The realization of Natural, or human language processing (NLP) system as opposed to formal languages, to handle real world problems require the accumulation of a wide range of knowledge formations and its management. Knowledge can be represented using an appropriate data structure, which is referred as a lexicon.

Lexicon is quite essential and necessary for natural language processing systems, i.e. parser, generator, translator, etc. It is because almost all the components of natural language processing systems refer to lexicon in order to accomplish their task. A simple lexicon for NLP systems may broadly contain syntactic, semantic and contextual knowledge.

Design of a lexicon is a complex activity and it involves, the data structure for the lexicon, the data to be stored in the lexicon, the size of lexicon and organization of the lexicon.

The problem of data structure depends on the data and the groups in which it is categorized. At the same time the data structure should be flexible for later up gradation of the system.

The decision about the contents of the lexicon depends on the requirements of the application. The only necessary data should be stored in the lexicon and other information should be derived from the lexicon. In any natural language, infinite number of relations exist between the objects, i.e. part_of, is_a, etc. It is not always possible to store all these relations. The stored data also depends on the grammatical framework used at syntactic and semantic level.

The size of lexicon is of very important consideration for its use and its storage. To reduce the size of the lexicon, NLP in general and inflectional languages in particular store only the free morphemes (root words) with the necessary morphological feature (bound morphemes). Free morphemes is a minimal meaningful unit which can be used independently in a language. A bound morpheme cannot be used independently but gives a meaningful word when combined with a free morpheme. The variant forms of root words can be derived by using simple spelling rules such as plurals can be formed by adding -s or -es in English language. But there are the words of irregular form which can not be derived from its root. These words need to be stored.

Organization of lexicon is an important problem in the overall design of lexicon. It deals with the management and use of the lexicon. The organization should be in such a manner that the insertion, deletion, modification and searching of an entry is fast and easy. There are several searching techniques available for fast and efficient accessing of the data. The selection of search technique depends on the application.

The problems about the lexicon discussed so far pertain only to the design of a lexicon for the NLP systems dealing with a single language. There are NLP systems such as translation systems which may require a lexicon with the entries for multiple languages. These lexicons are called multilingual lexicons. Multilingual lexicons take data from different languages so the relationship among these data may change the grouping and classification of data. This in turn brings the requirement of different data structure from that of a simple lexicon.

## OVERVIEW OF CURRENT RESEARCH ON LEXICON

Issues that arises in building relational models of lexicon and the review of current research in the development of relational lexicon are discussed as follows.

The relatíonal lexicon works at the level of individual words. Much of the information in the knowledge base consists of proposition which can be expressed in the form of relational arc. The lexeme is either an elementary lexical unit of the language, or one word taken in one well-defined sense with one set of syntactic, semantic and morphological properties. Further the lexical entry may contain :-

i. Attributes, with values assigned to them. The relevant attributes depend on the part of speech of the word.

ii. A table of appropriate arguments (cases of verbs, appropriate noun classes for adjectives, preposition, for nouns, etc.).

iii. Relational arcs to other words in lexicon.

The two types of relations; semantic relation and lexical relation play a great role in structuring of relational lexicon. Semantic relations connect concepts while Lexical relations connect words. Most models use combination of Lexical and Semantic relations. Some have chosen to work with either. Those who are building memory models concentrates on semantic relations, while those building lexicons with words and phrases as entries need lexical relations primarily. Oswald Warner has dealt with semantic in his process of building a language universal model[1]. John Sowa is building canonical graphs as a part of a comprehensive memory model, whose relation is primarily semantic[9]. On the other hand Melcuk and Calzolari are both involve in lexicography, where their models stress lexical relations[1].

The part of speech and word features in traditional dictionaries are adequate for a purely syntactic parser. But semantic analysis requires a more detailed representation of patterns of relationships between concepts.

Multiple word sense, syntactic ambiguity, implicit relationships, long range relationships problems are separately unsolvable. But an integrated theory should determine how semantic patterns for each word interact with syntax and context. Some of the proposed approach to such an integration are "projection rule for combining, Markers, Katz, and Foder [1963]", "function application, Monlague [1974]", "filling slots in templates, Wilk [1975]", "building graph structure Schanc and Rusbeck [1981]", and "conceptual graph, an integrate of all others, Sowa [1984]".

## OBJECTIVES OF THE PROJECT

As already stated above, almost all the components of Natural Language Processing systems depends on lexicon for their complete functioning. There is no standard method for description of lexicon generation and management. This depends on the application it is used for. All these problems draw the attention of researchers working in the area of NLP for the design and development of a lexicon. This work attempts to design a flexible, application independent lexicon which can later be used for any NLP application. This idea is based on the thought that most NLP systems have some common requirements, i.e. syntactic and semantic. This lexicon is as a part of NLP work being done at School of computer and systems sciences, J.N.U.

# CHAPTER II

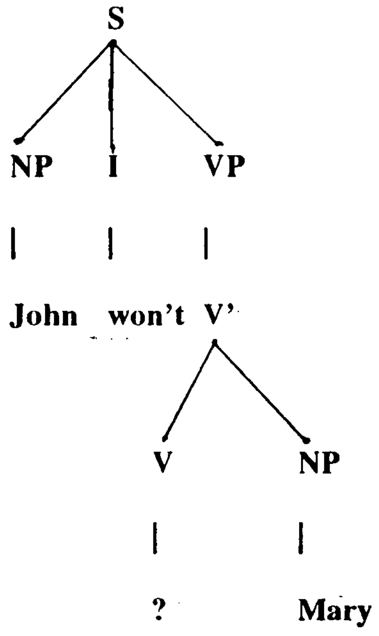
# SPECIFICATION AND STRUCTURE OF LEXICON

In a Lexicon only the necessary data shall be stored. The data to be stored in a lexicon and its format in the data structure will be described under specification and structure of a lexicon.

## 2.1 Specifications of Lexicon

Lexicon must provide us categorical information. For example our lexicon will have to tell us that cat is a Noun, or that admit can be either a Noun or a Verb and so on and so forth.

However, categorical information which dictionary entries contain must be rather more detail than is assumed by the simple portion of major classes, like Noun, Verb, Adjective and so on. If we consider the category Noun it has to be divided into a number of distinct subclasses. One important distinction is between Proper and Common nouns. Proper nouns are generally names of people(e.g. Ram), places(e.g. India), months(e.g. November), events(e.g. Deepawali), etc. They differ from common nouns in that they are not usually premodified by Determiners like 'a' and 'the' (one can not say a/the India), Whereas Common nouns are not restricted ( e.g. a/the cinema ). Common nouns can be subdivided into two distinct sets, Count and Non-count nouns. Nouns which can be used with singular determiner like a/one or a plural determiner like two/three are known as Count Nouns, hence chair is a Count Noun, since we can say both 'a chair' and 'two chairs'; but Nouns which can not be used as described above are Non Count or Mass Nouns (hence furniture is non count/mass noun , since we do not say a furniture or two furnitures). Also a singular mass Noun can be used without determiners, but not singular count noun, (e.g. we need furniture, we can't say we need chair). A mass noun can be premodified by 'much' not 'many', whereas count noun can be premodified by 'many' not 'much' (e.g. much furniture not many furniture, but many chairs). Thus our entry for 'cat' should have cat:[+N,-V,+common,+count]. But, other syntactic information should also be included in lexical entries. Let us try to examine the following structure of a sentence, **John won't ? Mary** [5].



( Where NP is noun phrase, VP is verb phrase, and I is Inflection constituent which can be filed by the modal such as can/ could/ will/ would...).

The lexicalisation principle in place of ? allows us to insert any word specified as a verb, but as we see from the following: i.e. John won't defy/ invite/ hit/ harm/ help/ like Mary, are acceptable, while John won't come/ go/ wait/ fall Mary are not acceptable. In fact some verbs in English are transitive while some are intransitive. It is not possible to detect them from the meaning of the verb, since pairs like wait and await seems to have much the same meaning, yet only the second in this context is transitive (i.e. we can say, we shall await your instructions, but not I shall wait your instruction ). There doesn't seem to be any way in which we can predict, this kind of syntactic information. It should be included in Lexical Entry for each verb.

Verbs also Permit one or more preposition, such as the following:

i. I defer to your suggestion. We cant use at, on, by, with instead of to.

ii. John waited for taxi. We can't use to, after, from, instead of for.

iii. You must abide by my decision; at, on, to, from are not acceptable instead of by.

We have to specify in the lexical entry of each verb, which kind of preposition the verb takes. e.g. 'rely' must be followed by preposition 'on'.

Another type of information which lexical entries for verb should contain is Thematic information. In the sentence, John gave Mary the book, John bears theta role Agent to the verb predicate gave, Mary bears the role Goal and the book bears the role Theme. Theta roles of verbs also contribute for the well formed of a sentence. Theta roles represent the relations between an action and objects in a sentence. Detail description of these roles are as follow:

i. **Agent:-** Instigator of some action. E.g. **John** killed Mary.

ii. **Object(patient):-** Entity under going the effect of some action. E.g. **Mary** fell over.

iii. **Experience:-** Entity experiencing some psychological state. E.g. **John** was happy.

iv. **Instrument:-** By which something come about. E.g. John wounded Henry with **knife**.

v. **Source:-** Entity from which something moves. E.g. John returned from **Paris**.

vi. **Goal:-** Entity towards which something moves. E.g. John passed the book to **Mary**.

vii. **Locative:-** Place where something is situated or take place. E.g. John hide the letter under the **bed**.

The incorporating of Thematic Functions into our model of syntax allows us to capture the similarity between different (but related) uses of the same lexical items. For example a verb such as a roll can be used both in transitive structure ( a: John rolled *the ball* down the hill), and intransitively ( b: *The ball* rolled down the hill). The italics expression clearly has a different constituent structure status in the (a) and (b) sentences. The ball is the object of the verb rolled in (a), but it is subject in (b). But in another sense the ball play the same role in both sentences as an entity undergoing motion. We can capture this role-identity by saying that the ball has the same thematic role in both sentences, which is the role THEME. The subject of murder is AGENT, that of collapse is a THEME, that of receive is GOAL, that of contain is LOCATIVE, and so on. Such role unpredictable to particular item has to be stored some where. It seems that the obvious place to represent information about an argument of an item is in the Lexicon entry under the concerned verb. That is the entry of murder will have additional information as described earlier, i.e. AGENT REQUIRED, and THEME REQUIRED.

The other information that must be stored along with the entry of a lexicon are the relationships between words and Concepts. Those relations which connect concepts are semantic relations, while those which connect words are Lexical relations. It is sometimes difficult to distinguish between the two relations. Such relations are called Lexical Semantic Relations ( semantico- syntactic ). There are infinitely many relations. But, as far as efficiency is concerned, storing all in the lexicon may not be practical. Some major lexical and lexical-semantic relations are to be selected according to the need of once goal, such as:-

**PART-WHOLE** relation : A relation which identifies something as being a segment or a portion of something else. This relation is only applicable to the words belonging to Noun class. In English it is often expressed by part of, the verb to have, and the possessive. E.g. A vehicle has a motor, A petal is part of a flower, etc.

**SYNONYM:** A relation which signals equivalence between words. In English it often appears as the verb to be. E.g. A bug is an insect ( meaning the word 'bug' is just another word for 'insect').

**TAXONOMY:** A relation showing a membership of an individual in a set or a group of individual in a large group. In English it is often expressed by a verb 'to be' and 'is a kind of'. E.g. Ants and fleas are kind of insects. It shows some hierarchical behavior.

**MAGNITUDE:** If a Lexical function MAGNITUDE is applied on word voltage the response could be high ( i.e. high voltage, but not great voltage ) and incase the word height the respond may be considerable ( considerable height ), etc.

Some of the Lexical Semantic Relation which are adopted from Chaffin and Hermand[1], with illustrated examples are as follow:

**CONTRAST:-**

i. Contrary:- old -- young, happy -- sad.

ii. Contradictory:- alive -- dead, male -- female

iii. Reverse:- attack -- defend, buy -- sell

iv. Directional:- front -- back, left -- right

v. Asymmetric_contrary:- hot -- cool, dry -- moist

vi. Pseudo_antonym:- popular -- shy

**SIMILAR:**

i. Synonymity:- car<-->auto, buy -- purchase

ii. Dimensional_similar:- smile -- laugh

iii. necessary_attribute:- bachelor -- unmarried

iv. Invited_attributes:- food -- tasty, cut -- knife

v. Action_subordinate:- talk -- lecture, cook -- fry

**CLASS INCLUSION:**

i. Perceptual_subo:- animal -- horse, flower -- rose

ii. Functional subo:- furniture--chair, tool--hammer

iii. State subo:- disease -- polio, emotion -- fear

iv. Active_subo:- game -- chess, crime -- theft

v. Geography sub:- state -- California

vi. Place:- Germany -- Hamburg, Asia -- India

**CASE RELATIONS:**

i. Agent-Action:- artist -- paint, dog -- bark

ii. Agent-Object:- baker -- bread, sculptor -- clay

iii. Agent-Instrument:- farmer-- tractor, soldier--gun

iv. Action-Recipient:- sit -- chair, hunt -- pray

v. Action-Instrument:- cut -- knife, drink -- cup

**PART-WHOLE:**

i. Functional object:- engine -- car, tree -- leaf

ii. Collection:- forest -- tree, fleet -- ship

iii. Group:- Choir -- singer, faculty -- professional

iv. Ingredient:- table -- wood, pizza -- cheese

v. Functional location:- kitchen -- stove

vi. Organization:- college -- admissions, army --corps

vii. measure:- mile -- yard, hour -- minute

Additional information to the parser, such as which type of grammar rule the word satisfies, has to be stored in the lexicon. If we consider the adjective class it has subclasses which may fall in one of the representation of grammar rules. One of the Sagers subclass AASP[7] defined as an adjective is an AASP if it occurs only with (Non-SN ) right adjunct to v OBJ ( SN an embodied or contained sentence ). The representation is as follows:

**N be Adj to V OBJ**

E.g. John is able to go. but we cant say John is able that Bill walks. The words able, fit, free, quick, etc. are in the subclass AASP. One word can be in two or more different subclasses. These classes to which the word belong has to be kept in lexicon.

For reference purposes the meaning of the word along its pronunciation has to be kept in lexicon. For example, Conjunction pronunciation is ken-junk'shen. Since it has different meanings in different uses, all possible meanings has to be stored. The major operations on a lexicon are either to build a lexicon database or require services from the lexicon. In building of lexicon the operation involved are **DELETE, STORE, and UPDATE.** While **RETRIEVE, RECOGNIZER, GENERATOR,** etc. are used for data out of database. RETRIEVE is for look up of the database, i.e., reading from the database. RECOGNIZER and GENERATOR are interfaces for recognizing the word whether it is available in the database or not, if it is available they have to give the word content to the user.

## 2.2 Ambiguity and Lexicon

Some of the sources of ambiguities in natural languages are polysomy, generality, idiomatic expressions and homonym. Polysomy of a word is the most problem in semantic description.

Many words in many sentences can be understood quite differently. It is difficult to give semantic account of such phenomena in a description of a language.

The basic distinction between the content of a sentence and its interpretation play crucial role. The content of the sentence is the inherent semantic structure of a sentence. The interpretation of a sentence is the various ways of which one and the same sentence can be understood in each unique cause of language use.

To make the distinction between them the following should be noted.

i. The inherent meaning of a word, which its full specification, is in lexicon.

ii. The possible further specification of its inherent meaning in the context of particular sentence

iii. The possible further specification in the interpretation of a sentence in a language use.

The word old is understood differently in an old man ( 'of relatively great age') and an old shoe ('not new'). It is quite reasonable to conclude old has distinct senses, though of course these senses are related. Polysomy is restricted to those words that have distinct but related senses.

Some of the words that can be understood in more than one ways will be represented in a lexicon with the same entry and corresponding distinct frames with their various senses.

A second feature of word that may give rise to the ambiguity is Generality. For example 'I brought an animal from the Zoo' can be ambiguous in language since it may arouse curiosity which animal he brought. 'To maintain child' is ambiguous between daughter and son. Finally it should be noted that generality is relative notion. Animal is more general than monkey, but monkey is still unspecified with respect to the various kinds of monkeys available.

The third possible source of ambiguity (opposite to generality) is homonymous. A lexical element is homonymous if its different senses have no relevant component in common. For example, Bank 'of a river' and bank 'place to deposit money', swallow 'to engulf' and swallow 'kind of bird ' have nothing in common.

An Idiomatic expression can be viewed as an expression; the meaning of which can not be described as a composition function of the meanings of its components and the structure. Idioms are semantically highly irregular.

## 2.3 Structure of a Lexicon

The lexicon entry is taken to be a morpheme, which is a minimal meaningful unit and can be used independently in a language. The lexeme is taken as a root word along with its grammatical attributes, semantic attributes, conjugate list, case roles, derived words, possible syntax surface realization of the word, a mark for relations, or and storage for idioms and definition. Though the entry of a lexicon is a morpheme, word phrase like full idioms has to be entered to the lexicon separately. Idiom is a phrase which can not be constructed from its constituent words by general rules and such that no constituent word retains its full meaning. For reference purpose idioms will be stored under each constituent words.

With each lexeme a frame and slot are associated. A frame may contain another frame with in itself. They may be linked together by storing the address of one in the other. All the frames are not structurally identical. One may belong to noun, verb, noun modifier, verb modifier or literal frame. Since, this lexicon is organized in word bases, words are grouped in different part of speeches. The meaning of a sentence is usually influenced by noun, verb and their modifiers. The grouping of words have been made under these four classes. Noun and Pronoun are taken to be as part of noun class. Some of Possessive pronouns, Adjective, qualifier, and article are taken as part of noun modifier class. Tensed verbs and Tenseless verbs (infinitive) are taken as part of verb class. the verb modifier class contains adverb. Those words which may not fall in none of these classes will be part of literal class. This frame may be used only as indication for existence of such word in the lexicon. The content of a lexeme: the Grammatical Attribute, Conjugate List, Derived or Computable words, Semantic Attribute, and relation are described in detail
as follows.

### 2.3.1 Grammatical Attributes:-

The grammatical attributes of the particular lexeme depends on the part of speech (P.O.S) of the root word fall. It's description is given below according to the four classes, NOUN, VERB, ADJECTIVE, and ADVERB.

_1. In case the word is NOUN, The Lexeme consists of P.O.S., Common or Proper Noun, Gender, Person, and Number. Gender tells whether the word stands for masculine, feminine or neutral. Person is for first person, second person, third person and neutral. Where Number is for singular or plural.

The structural representation is as follows:-

**WORD(**

**(P.O.S, GENDER, PERSON, NUMBER) )**

**Eg. city(**

**(noun, neutral, neutral, singular))**

_2. If the word is in VERB class its lexeme consist of Tense, Aspect, Voice, and Mood.

i. Aspect is the form of the verb indicating the type of a character of the action In English Perfect or Imperfect, while Hindi verb has numerous aspects.

a. Terminating aspects; represent act as a whole. e.g. lead sinks (general act) and I see him coming (particular act).

b. Progressive aspect: action as progressing, e.g. several books are lying on the table. etc.

ii. Voice: The form of the verb indicating relations of the subject to the action.

a. Active voice :- Indicates subject does/become something. e.g. I saw Ram, Ram goes, The boy is ill.

b. Passive voice :- Represents the subject as acted upon, thus the grammatical subject is not logical subject but logical object. e.g. The enemy was killed. I was called. It is said that..

iii. Mood:- Verb indicating the manner of the action. Hindi has three moods, namely, Imperative, Indicative, and subjective.

iv. Tenses: We consider simple tense, present tense, past tense, future tense, and etc... In english, a verb will not be modified according to the gender, person, and number. For example, in the phrases, 'He goes' and 'She goes', goes is a simple tense and has entertained both masculine and feminine. I left ..., he left.., left is a past tense which has been used for both first and second person. In Hindi, a verb is modified according to the Gender, Number,and Person of either the subject or the object, or it has reference only to the actions. Hindi verbs have three constructions.

a. Subjective construction:- The verb has the same Number, Gender, Person as a logical subject.

b. Objective constructions:- The verb has the same Gender, Number, and Person as its logical object (the person or thing to whom the action is directed.).

c. Neutral:- the verb agrees neither with the subject nor with the object in person, gender, and number, but is always placed in third person singular masculine.

The various forms discussed above Aspect, Voice, Mood, and Tense are not always independent of each other, or distinctly and individually recognizable in each verb. A single verb often represents several forms. Since most of them can be obtained at the procedure level only Tense will be considered at structural level.

The structural representation is as follows:-

**WORD(**

**(P.O.S, TENSE [GENDER, PERSON, NUMBER])** )

where [] stands for optional

_3. If a word is an ADJECTIVE:- It consists of part of speech, person, gender, and number. Part of speech indicate Adjective or Determiner or Possessive pronoun.

The structural representation is as follows:-

**WORD(**

**(P.O.S, [GENDER, PERSON, NUMBER])** )

where [] stands for optional

_4. If the word is an ADVERB:- It contains only part of speech in the grammatical attributes.

The structural representation is as follows:-

**WORD(**

**(P.O.S)** )

where [] stands for optional

### 2.3.2 Conjugate List:

Conjugate list is a storage place for those words which may be obtained by morphological processes. These words are mainly inflected form of the root word. Both inflected and root word represent the same concept. The word in the conjugate list shares the same semantic attributes and relational forms with their root word. In case of grammatical attributes they may or may not have their own value specification corresponding to their entry. Some irregular forms (those words which can not be obtained by morphological processes), such as went, will be stored under conjugate list of 'GO' and as separate entry 'WENT', in the lexicon with indication to 'GO'. This enable one to get past tense of go from the entry 'GO' and access information stored under 'GO' through entry 'WENT', to avoid redundancy.

representation of conjugate list;

```
WORD(
    (word1([tense],[gender,number,person]),
     ...
     wordN([tense],[gender,number,person]))).
```

[] shows optional.

For example, **city (cities (neutral, plural, neutral))**, Where as **go (goes ( simple tense ), going (continuous tense ), went ( past tense ) ).**

As far as modifiers are concerned, in English a word like beautiful is used for feminine while handsome is for masculine.In adverb grammatical attributes are not needed.

### 2.3.3 Derived or Computable Words:-

These words are like the words in conjugate list and are obtained by some morphological processes, but they have different senses to that of their root word. They will have distinct frames for their features and attributes storage. They share nothing with their root word. Derived word may have one of the lexeme frames depending to its part of speech. Therefore in the structure point of view at the Root word frame level the derived word, its part of speech and the address of its attributes storage place has to be specified.

Some words may have two or more different senses with in the same category, or even may lie in two different categories. All are candidate to be the root word. But the one which comes first will be the root frame and the rest will be in the derived words frames. During implementation there has to be a mechanism which specify the existence of such duplicates of words in different senses.

Their representation format is as follows:

**WORD(**

**( (word1, pos, frame reference), ....,**

**(wordN, pos, frame reference)))**

E.g. **city(citizen, noun, .24311.),** where .24311. is address of citizen frame for citizen feature and attributes.

## 2.3.4 Semantic Attributes:-

Some features and properties of a word are stored in attributes form for word concept analysis. Such information are considered as semantic attributes. For the different classes, Noun, Verb, Noun modifier, and Verb modifier, some fixed attributes slots are selected according to their class.

### 2.3.4.1. Noun semantic attributes:-

The following can be the semantic attributes of noun class:

i. **Concrete or Abstract:** It tells whether the word stands for concrete or abstract. E.g. Animal and house are concretes. While idea and stress are abstracts.

ii. **Animate or Inanimate:** If the word stands for an abstract concept, this entry will be unnecessary. For general case, since we can consider such words as inanimate concepts we may use the slot for both abstract and concrete. E.g. dog is animate and table is inanimate.

iii. **Human, Non-human mammals, Bird, or Insects:** e.g. man is human, cow is nonhuman mammal, while robin is a bird and an ant is an insect.

iv. **Self mobile or Non-self mobile:** E.g. Robot is self mobile, and File is not.

v. **Location:** Whether the thing that the word represent is in Air, Water, Earth, or Nowhere. E.g. Moon is on air, Fish is in water, Man is on the earth, and thought is nowhere.

vi. **Countable or Mass:** The count / mass distinction in language reflects the difference between discrete and continuous whole. E.g. Pen, hole, are countable, while water is indicated by volume.

vii. **Color:** blue, white, red, etc. E.g. Sky is blue and Blood is red.

viii. **Range or Point:** A point of time is represented by second, while Range of time by day.

ix. **Measurement:** The entity the word represent may be measured in Count, Weight, Height, Volume, Temperature. Where temperature can be described in actual value, high, low, and medium.

### 2.3.4.2 Verb Semantic Attributes:-

Types of verbs are classified into action, state and process. In our semantic attributes we organize them into Basic Acts, Semantic Features, and Case Roles.

_1 **Basic Acts:** We categorize basic acts into Human part act, sense act, and state act. These acts will be further subdivided into Human Part Act, Sense Act and State of Act.

i. **Human Part Act** is divide into Hold, Move, Speak, Ingest and Excrete. E.g. Hold: grasp, throw; Move: crawl, walk, run; Speak: talk, whisper, say; Ingest: eat, drink; Excrete: vomit, bleed, sweat.

ii. **Sense Act** is divided into Contact, Experience and Recall. E.g. Contact: see, hear, touch; Experience: feel, consider; Recall: remember;

iii. **State of Act** is divided into Appear, Be, Die, Transfer, State of Change and Build. For example, Appear: emerge, born; Be: is, are; Die: perish, dry; Transfer: give, take, bring; State of Change: win, fail, grow; Build: make, create;

There could be some un-categorized acts. One or more basic acts may be correspond to a verb in a lexicon. In the structure the number of subdivision could be left open so that it is flexible for addition of newly recognized acts. Basic act categorization of verb is useful in finding the semantic relationship between object and processes.

## 2. Semantic Features:

i. **Transitive, Intransitive, or Ditransitive:-** Transitive verb takes a direct object, while intransitive doesn't. E.g. My brother is ill(intransitive). The king appointed Mohan a minister(transitive). Some verbs may be used as both cases. E.g. 'to play a game' is transitive, while 'to play' is intransitive. Ditransitive verb take an indirect object and a direct object. E.g. To give a book to Mohan.

ii. **Action Center (Agent, Object, or Instrument):** It tells whether the action depends around the agent, object, or some combination of them. For example, Agent: speak, Object: pick, and Instrument: cut.

iii. **Action Movement (Positive, Localized, or Still):** The movement involved in action is either positive, localized or there is no movement. E.g. Help is positive movement. Dance is localized movement. Whereas think is still movement.

iv. **Direction (Up, Down, Forward, Backward, Left, Right, Around, Along, Centralized, or, No direction):** The direction involved in action is specified in discrete categories. E.g. Up: climb; Down: slip; Forward: walk; Backward: return; move can be used as example for forward, backward ,left, and right. Along: cooperate, spin can be for centralized or no direction.

v. **Distance (Near or Away):** The action causes the whole system to be moved near or away from the present location. E.g. Near: Join, Away: depart.

vii. **Action With (Love, Anger, Compulsion, or No emotion):** E.g. Love: kiss; Anger: rebuke; Compulsion: shut-up, NO EMOTION: sleep.

vi. **Reason (Internal, or External):** Whether the action is occurred due to the internal reasons or on the instigation of the external agency. E.g. Internal: eat; External: push.

vii. **Purpose (Attract, Protect, React, Recreate, Necessity):** E.g. Attract: laugh, Protect: define, React: excite, Recreate: build, Necessity: sweat.

_3. **Case Roles:** Theta role represent the relation between an action and an object in a sentence.

i. **Expected Case Roles** specify requirement of Agent, Object, Instrument, or/and Goal, as Required, Optional, or not allowed, for a particular verb. E.g. for verb 'get', Agent, Object and Instrument are Required, while Source , Goal, and Locus are Optional.

ii. **Expected Preposition** (or Postposition in Hindi), for subject, object, etc. E.g. In English verb 'rely' expects preposition 'on', while 'get' expects preposition 'from' for the case 'source'.

iii. **Agent to Object Status Relation:** Small, Big, or Equal. whether the Agent is small, big, or equal in abstraction or physical relationship.

iv. **Expected Features(Agent,Object):** Whether the required Agent or Object for a particular verb is inanimate, human, animal, bird, or insect. E.g. 'get' expects human as

Agent and Object.

v. **Expected Forms of(Agent, Object, or Instrument):** Liquid, Solid(hard, soft, or powder), 'Gas, Fire, or Ether.

### 2.3.4.3 Adjective Semantic Features:-

_1. **Quality (Appearance, Behavior and Motion):**

i. **Appearance(High, medium, or Low):** E.g. High: beautiful, Medium: good, Low: ugly.

ii. **Behavior (High, Medium, Low, or Zero),** E.g. High: smart, Medium: active, Low: lazy, Zero: dull.

c. **Motion (High, Medium, Low, or Zero):** E.g. High: very fast, Medium: fast, Low: slow, Zero: still.

_2. **Quantity:** Measurement(Number, Length, Volume, Weight, temperature, Pressure) as complete, high, medium, low, zero or value.

_3. Belongs to(Noun, First person, Second person, Third person pronoun). E.g. First person pronoun: mine, second person pronoun: your, Third person pronoun: theirs

### 2.3.4.4 Adverb Semantic Features:

The semantic of adverb is explained under Manner, Time qualifier and Place qualifier.

_1. **Manner (High, Medium, Low):** E.g. Medium: well, High: very, Low: badly,

_2. **Time Qualifier ( Complete, High, Low, Medium, Zero, Value):** E.g. Low: slowly, Zero: instantly, Value: 5 seconds

_3. **Place Qualifier (Complete, High, Low, Medium, Zero, Value):** E.g. Medium: far, Low: near, Zero: on spot, Value: 5 meters away _4. **Attribute Qualifier (Complete, High, Low, Medium, Zero):** E.g. Complete: all, High: mostly, Medium: very, Low: few, Zero: nowhere

## 2.3.5 Relations:

The following which are illustrated by example are the strategies used to converted recognized relations into the network.

i. For equivalent relation 'R'( i.e. reflexive, symmetric and transitive), a connection can be made so that a closed path can be obtained considering the words as nodes.

E.g.1. If W0,W1,W2 are words and W0 R W1, W0 R W2, and W1 R W2, then, the data representation of this relation will be as Fig. 1a.

Fig. 1.

a. **W0:R<address of W1>, W1:R<address of W2>, W2:R<address of W0>**

$$\begin{tikzcd}
\text{[W0 -]} \arrow[r] & \text{[W1 -]} \arrow[r] & \text{[W2 ]} \arrow[ll, bend left]
\end{tikzcd}$$

(Synonymity lies in such representation).

b. If 'R' is transitive and not symmetric the path will be without circuit. In such case the data representation will be as follows.

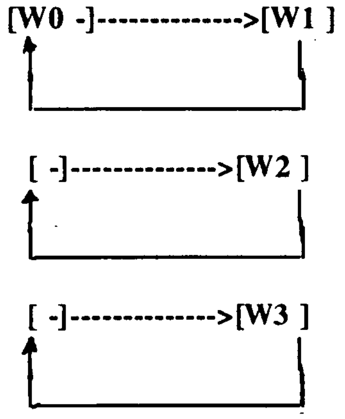
W0:R<address of W1>, W1:R<address of W2:R<null>

**[W0 -]-------->[W1 -]-------->[W2]**

(Taxonomy lies in such representation)

c. If 'R' is symmetric not transitive, and if a word has one to many relation then the relation network is as follows. If R relates W0 to W1, W2 and W3, then

**W0:R<address of W1,W2,W3>, W1:R<address of W0>, W2:R<address of W0>**



(Antonyms are part of such representation)

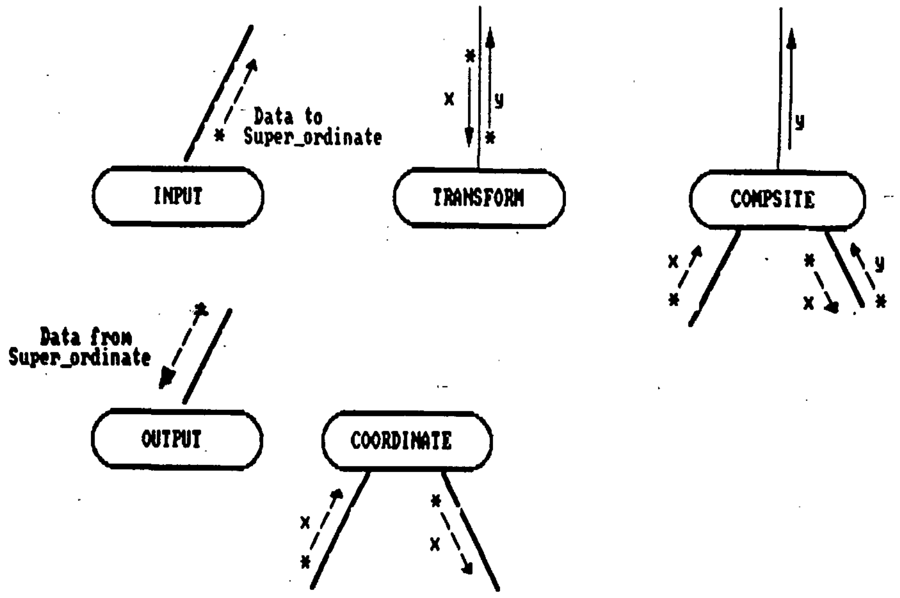
Some relations may have the combination of the above representations.

# CHAPTER III

## DESIGN METHODOLOGY

Design methodology is guideline to aid the designer during design process. The two Design processes are Structured Design and Object-Oriented design. The structured design is an approach which concentrates on the functional aspects of the problem and is based on the concept of Functional Abstraction, while Object-Oriented approach is based on the concept of Data Abstraction[2].

Structured Design Methodology (SDM) views every software system as having some inputs which are converted into the desired outputs by the software system. The software is viewed as a transformation function that transforms the given inputs into the desired outputs. In this work the goal is to produce design for lexicon software system that consist of many modules. These modules are **INPUT, OUTPUT, TRANSFORMATION, and COORDINATE MODULES.** Their pictorial representation are as follows:



In Design Methodology there are Four activities involved.

1. Restating all the problems in the Data Flow Graph* (DFG). The DFG shows the major transforms that the software will have, and how the data will flow through different transforms.

2. If once the DFG is ready, the next step is to identify the highest abstract level of input and output. The Most Abstract Input (MAI) data elements are those which can be recognized by starting from the physical inputs and traveling towards the outputs in the data flow graph, until the data elements are reached that can no longer be considered as incoming data element. Similarly we identify the Most Abstract Output(MAO), by starting from the outputs in the data flow and traveling towards the input. These are the data elements, most removed from the actual outputs but still be considered as outgoing.

3. Set MAIN module which is a coordinate module. Those modules to the left of MAI are INPUT modules, between MAI and MAO are transform modules, and to the right of MAO are OUTPUTS modules.

4. Since each of the above specified modules may have a lot of processing to do, we simplify by factoring each to different modules that distribute the task of the module.

* *DFD description and the diagram is to be shown in the next page.*

## 3.1 Data Flow Diagram and Structure Chart

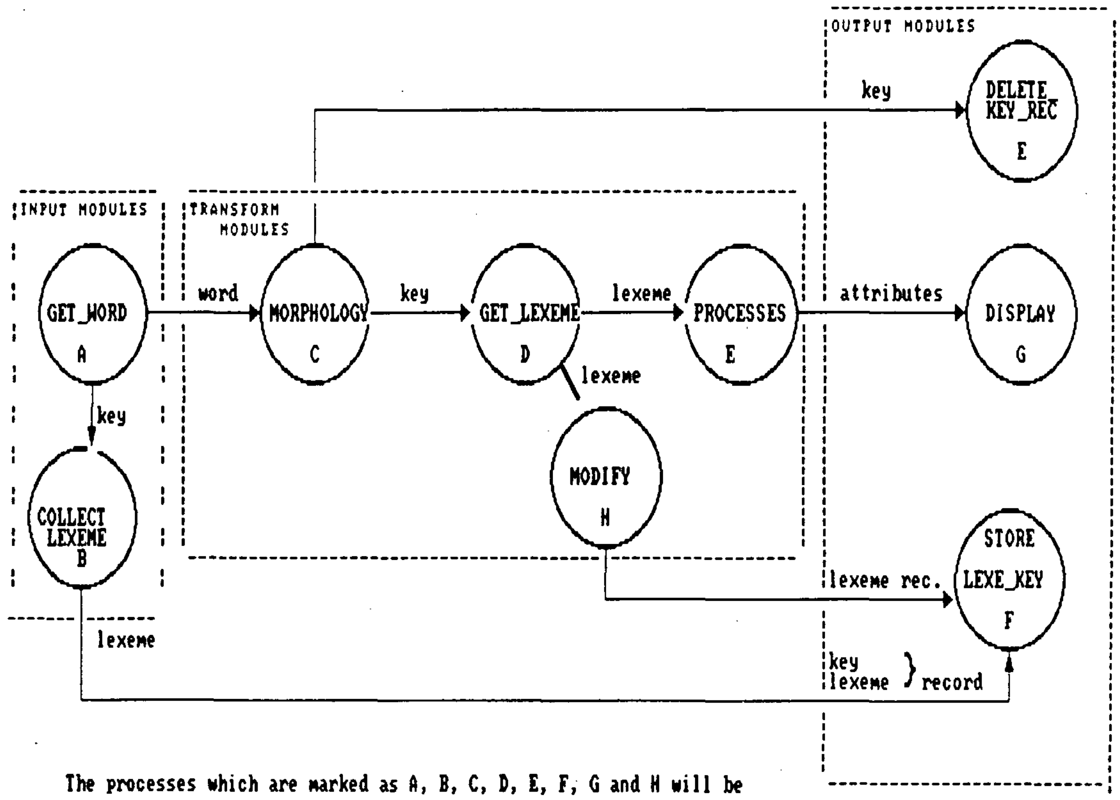
Data Flow Diagrams(DFD) are commonly used during problem analysis. DFDS are very useful in understanding a system and can be effectively used for partitioning during analysis. A DFD shows the flow of data through a system. The system may be an organization, a software system or hardware, etc. It shows the movement of data through the different transformations or processes in the system. The processes are shown by named circles, and data flows are represented by named arrows entering or leaving the circles. A named oval rectangle is used for representing a source or sink (originator or consumer of data).

In Structured Design Methodology, the design is represented by Structure Charts. The structure of a program is made up of the modules of that program together with the interconnections between modules. Every computer program has a structure, and given a program its structure can be determined. The Structure Chart of a program is a graphic representation of its structure. In the Structure Chart the modules are represented in oval like box. An arrow from module A to another module B represents that module A invokes module B. B is called subordinate module of module A, and A is called super ordinate module of module B. The arrow which is received by B is an input and the parameter returned by B is an output of module B. Fig.I, represents the Data Flow Diagram for the system while Fig.II (fig.1 to 6 ) represent the Structure Chart of the System.

FIG. I

DATA FLOW DIAGRAM OF THE SYSTEM

(DFD)



The processes which are marked as A, B, C, D, E, F, G and H will be converted into modules in the state chart of the system. The following are some of the possible data flow path using the above bubbles.

i. For RETRIEVAL (READING) the Data flow will be as follows:

[A ➡ C ➡ D ➡ E ➡ G]

ii. For DELETE the Data flow is as follows:

[A ➡ C ➡ E]

iii. For UPDATE the data flow will be as follow:

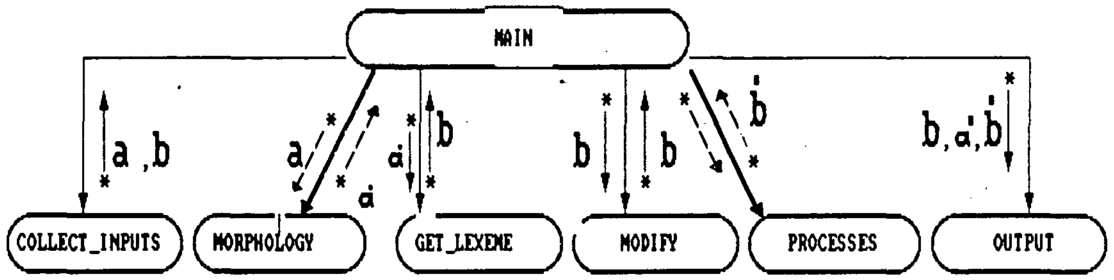
[A ➡ C ➡ D ➡ H ➡ F]

iv. For STORE the data flow is as follow:

[A ➡ B ➡ F]

FIG. II. STRUCTURAL CHART FOR THE SYSTEM

MAIN MODULE coordinates the Data Flow between INPUT, TRANSFORM and OUTPUT MODULES. The pictorial representation is as follow.



a is key, word.

ȧ is key record, b is for lexeme, and ḃ is lexemes attributes.

* b in collect_input is only when the system is used for storeing information.

FIG. 1. The task of COLLECT_INPUTS is divided into sub tasks to different modules as follows.

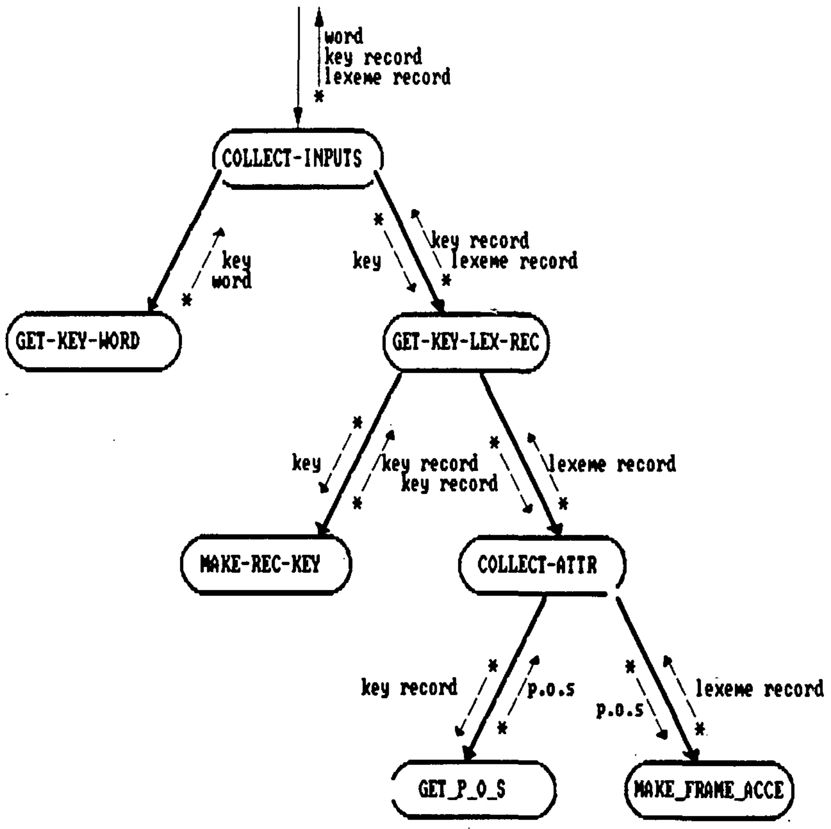


FIG. 2. The task of MORPHOLOGY MODULE is distributed to differnt subordinate modules, as shown below.

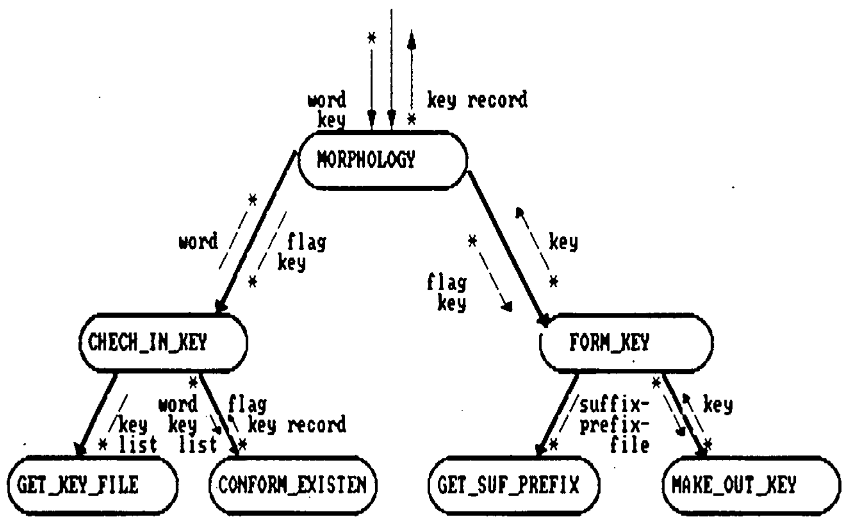


FIG. 3. The task of GET_LEXEME MODULE is distributed among other modules, as shown below.

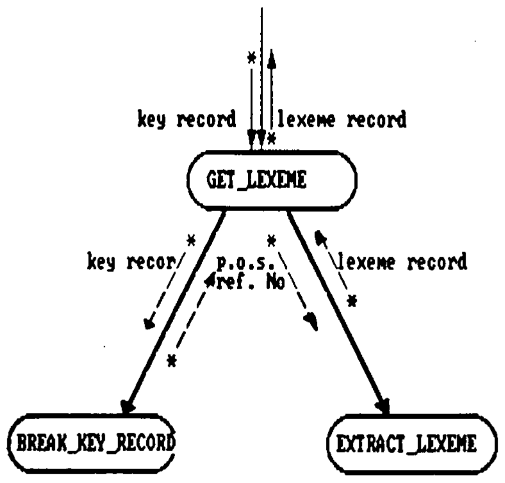


FIG. 4. Broken lines which connect the modules indicates only one sub task will be performed in one processe time.

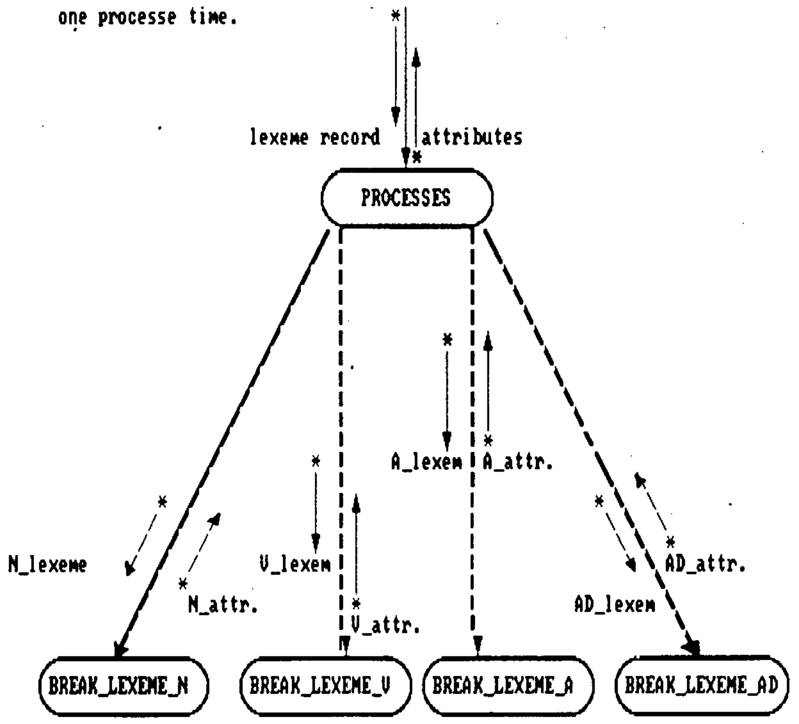


FIG. 5.

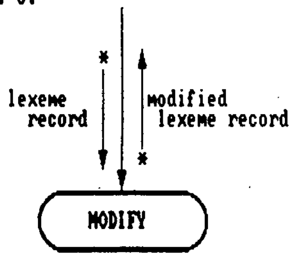
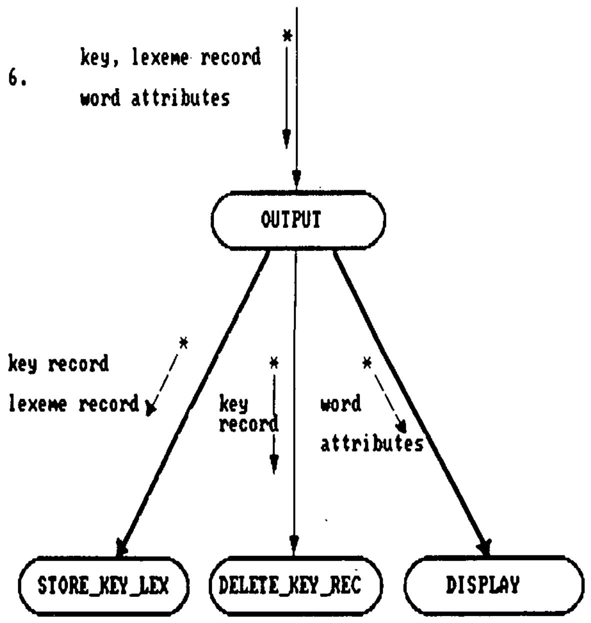


FIG. 6.



## 3.2 Data Definition

All the contents, which have to be kept in the structure of lexicon are encoded and have the following form of data definition.

**Word** is a letter, or groups of letters, which represent a unit of a language. **Key** is a morpheme which is taken as a root word.

**a. word, key: string**

**Key_rec** is a record which contains a key, the root word part of speech and its entry number in the key file.

```
b.  TYPE  Key_rec = RECORD
            pos: CHAR
            key: string
           ref_no: integer
        END
```

**N_gramm_attr** is a record which contains categorization and sub-categorizations of a key word in NOUN frame.

```
c.  TYPE N_gramm_attr = RECORD
          class,  {pos:< Noun/ Pronoun>}
          common_proper,    { <Common/ Proper> }
          person,         {<First/  Second/ Third>}
          gender,         {<Masculine/ Feminine>}
          number : CHAR;  {<Singular/  Plural>}
      END
```

**N_semantic** is a record for semantic representation of NOUN frame which has been described under the structure of lexicon.

```
d.  TYPE N_semantic = RECORD
        conc_abst,   {<Concrete/ Abstract>}
        anim_inan,   {<Animate/ Inanimate>}
        human_nonM,{<Human/non_h_Mammal/Bird/Insect>}
        self_mobile,  {<+/ ->}
        location,  {<Air/ Earth/ Water/ Nowhere>}
        countable_mass,  {<Count/ Mass>}
        range_point,    {<Range/ Point>}
        measurement : CHAR;{<Count, Weight/ Height/
                        Volume/  temperature(H/ L/ M>}
    END
```

**N_conjugate** list contains, those words which can be obtained by morphological processes and share the same semantic attributes with that of their root word, and their grammatical categories.

```
e.  TYPE N_conjugate_list = ARRAY string of
                RECORD
                  w_rd: word
                  person,
                  gender,
                  number : CHAR;
                END
```

**R_tion** is a list of recognized relations along with record of addresses of words to which a particular word under consideration is related.

```
f.   TYPE   R_tion = ARRAY OF RECORD
                relation_number: INTEGER
                rel1, ... rel_max : addresses
              END
```

**N_lexeme** is a record which represent the NOUN frame which has been discussed under structure of a lexicon. surface realization tells which grammatical rule should be inferred.

```
g.   TYPE N_lexeme = RECORD    { NOUN FRAME }
             root_word :key
             gra_n : N_gramm_att
             sem_n : N_semantic
             con_n : N_conjugate_list
             rel_n : R_tion
             drw_n : derived_words
             sur_n : INTEGER {surface_realization }
             mean  : ARRAY OF string
           END
```

Similar way of data definition has been followed for the rest frames.

**V_gramm_att** is a record for grammatical attributes of VERB frame.

```
h.   TYPE V_gramm_att = RECORD
             reg_irr,  { < Regular/ Irregular> }
             aspect,   { < Perfect/ Imperfect> }
             tense, {<Simple/ Past/ Future>}
```

```
          gender, { in case of english, not needed}

          number, { in case of english, not needed}

          person: CHAR;{in case of english not needed}

         END

i.   TYPE basic_acts = RECORD

          human,  {<Hold/Move/Speak/Ingest/Excrete>}

          sense,{<Contact/Experience/Recall>}

          state_act:CHAR;{<Appear/ Be/ Die/ Transfer/

                        state_of_Change/builD>}

         END

j.   TYPE semantic_feature = RECORD

          tran_intr,  {<Transitive/ Intransitive>}

          AOM_beneficiary, {<Agent/ Object/ Mutual>}

          continue, {<Continuous/Sudden/Intermittent>}

          action_place,  {<Air/ Earth/ Water>}

          action_center, {Agent/ Object/ Instrument}

          action_aim, { Individual/ Societal}

          AO_speed, { High/ Medium, Low/ Zero}

          action_movement,{Positive/Localized/Still}

          direction,

          distance_caused,  {< Near/ Away>}

          action_with,{<Love/Anger/Compulsion/ - >}

          reason,        {<External/Internal>}

          purpose:  CHAR; {<Attract/ Protect/ React/
```

```
                    rEcreate/ Necessity>}
        END
```

**k. TYPE expected_case_role = RECORD**

```
        object_RON, {Required/ Optional/ Not required}
        agent_RON, {Required/ Optional/ Not required}
        instrumen_RON,{Required/ Optional/ Not required}
        subject_RON, {Required/ Optional/ Not  required}
        agent_to_object,  {Small/ Big/ Equal}
        agent_expect, {Inanimate/ Animal/ Bird/ Insect}
        Object_expect,{Inanimate/ Animal/ Bird/ Insect}
        agent_forms,  {Hard/ Soft/ Powder}
        object_forms, {Hard/ Soft/ Powder}
        instrument_forms: CHAR;   {Hard/ Soft/ Powder}
        expected_preposition:string
      END
```

**l. TYPE V_conjugate = ARRAY OF string of record**

```
        tense, number, gender,
        person : CHAR;
      END
```

**m. TYPE V_lexeme = RECORD        { VERB FRAME }**

```
        root_word :key
        gra_v : V_gramm_att
        sem_v : V_semantic_feature
        bas_n : basic_act
        role  : expected_case_role
        con_v : V_conjugate
        rel_n : Relation
        drw_v : derived_words
        sur_v : INTEGER {surface_realization }
        mean  : ARRAY OF string
      END
```

**n. TYPE quality: record**

```
        appearance,  { High/ Medium, Low/ }
        behavior,    { High/ Medium, Low/ Zero}
        in_motion,   { High/ Medium, Low/ Zero}
      END
```

Quantity, quality, and a_lexeme are for ADJECTIVE frame.

**o. TYPE quantity: record**

```
      measurement,{ Number/ Length/ Weight/ Volume/
                  Temperature/ Pressure}
       measu_value: CHAR,{High/ Medium/ Low/ Zero/
                       Complete/ Value}
    END
```

**p. TYPE A_lexeme = RECORD { ADJECTIVE FRAME }**

```
        root1: string
        pos:CHAR, {Adjective/ Determiner}
        qual:quality,
        meas: measure,
                belong_to: CHAR, {Noun/ First person/  second
person/Third person pronoun}
                con_A : A_conjugate
                rel_A : Relation
                drw_A : derived_words
                sur_A : INTEGER {surface_realization }
                mean: string
            END
```

**q. TYPE AD_lexeme : RECORD**

```
                manner
                time_qualifier
                place_qualifier
                attribute_qualifier
                con_Ad : conjugate
                rel_Ad : Relation
                drw_Ad : derived_words
                sur_Ad : INTEGER {surface_realization }
                mean:  string
            END
```

## 3.3 Module Specifications

Specifications of modules which has been shown in the structural charts are given below.

**_0: MAIN**

INPUTS: KEY_FILE, LEXEME_FILE, SUF_PREFIX_FILE

OUTPUTS: NONE

SUBORDINATES: COLLECT_INPUTS

MORPHOLOGY

GET_LEXEME

MODIFY

PROCESSES

OUTPUT

PURPOSE: It coordinates the data flow of Input, Transform and output modules.

**_1. COLLECT_INPUTS**

INPUTS: FROM KEY_BOARD

OUTPUTS: WORD, KEY RECORD

SUBORDINATES: GET_KEY_WORD

GET_KEY_LEX_REC

PURPOSE: It collects inputs from the terminal or key board for its super-ordinate module.

**_2. MORPHOLOGY**

INPUTS: WORD, KEY

OUTPUTS: KEY_RECORD

SUBORDINATES: CHECK_IN_KEY

FORM_KEY

PURPOSE: Given a word as an input it will make out the possible key word for the lexicon entry. It looks for a word in a lexicon, if not found it modify the word until the word is found or no further modification is possible.

**_3. GET_LEXEME**

INPUTS: KEY_RECORD

OUTPUTS: LEXEME RECORD

SUBORDINATE: BREAK_KEY_RECORD

EXTRACT_LEXEME

PURPOSE: Takes key record as an input, it will produce the appropriate modified or unmodified lexeme as an output.

**_4. PROCESSES**

INPUTS: LEXEME RECORD

OUTPUTS: INDIVIDUAL ATTRIBUTES OF LEXEME

SUBORDINATES: BREAK_LEX_N

BREAK_LEX_V

BREAK_LEX_A

BREAK_LEX_AD

PURPOSE: It gives the individual attributes of lexeme for different applications.

**_5. MODIFY**

INPUTS: LEXEME

OUTPUTS: LEXEME

SUBORDINATES:

PURPOSE: Accept lexeme record to be modified and make necessary changes.

**_6. OUTPUT**

INPUTS: KEY_RECORD, LEXEME_RECORD

OUTPUTS:NONE / printed output

SUBORDINATES: STORE_KEY_LEX

DELETE_KEY_REC

DISPLAY

PURPOSE: Update lexeme and key file, and provides printed information.

**_1.1 GET_KEY_WORD**

INPUTS: FRAME TERMINAL BOARD

OUTPUTS: KEY, WORD

SUBORDINATES: NONE

PURPOSE: collect input from the terminal to form string

**_1.2 GET_KEY_LEXEME_RECORD**

INPUTS: KEY

OUTPUTS: LEXEME_RECORD, KEY_RECORD

SUBORDINATES: COLLECT_ATTRIBUTES

MAKE_RECORD_KEY

PURPOSE: collects attributes of key into lexeme

**_2.1 CHECK_IN_KEY_FILE**

INPUTS: WORD

OUTPUTS: FLAG

SUBORDINATES: GET_KEY_FILES

CONFORM_EXISTENCE

PURPOSE: Produce Checks whether the word exist in key file or not.

**_2.2 FORM_KEY**

INPUTS: WORD, SUFFIX, PREFIX

OUTPUTS: KEY

SUBORDINATES:GET_SUF_PREFIX_FILE

MAKE_OUT_KEY

PURPOSE: It produces the possible key from a word as lexicon entry .

**_3.1 BREAK_KEY_RECORD**

INPUTS: KEY_RECORD

OUTPUTS: POS, REF.NO

SUBORDINATES:NONE

PURPOSE: Produces part of speech and reference number from key record so that we can extract the lexeme record.

**_3.2 EXTRACT_LEXEME**

INPUTS: POS, REF.NO

OUTPUTS: LEXEME_RECORD

SUBORDINATES:

PURPOSE: From the set of lexeme it extracts the lexeme which is referenced by the given key word

**_4.1 BREAK_LEXEME_N**

INPUTS: N_LEXEME RECORD

OUTPUTS: ATTRIBUTES

SUBORDINATES:NONE

PURPOSE: It passes attributes of noun class to the main for application purpose.

## _4.2 BREAK_LEXEME_V

INPUTS: V_LEXEME RECORD

OUTPUTS: ATTRIBUTES

SUBORDINATES:NONE

PURPOSE: It passes attributes of verb class to the main for application purpose.

## _4.3 BREAK_LEXEME_A

INPUTS: A_LEXEME RECORD

OUTPUTS: ATTRIBUTES

SUBORDINATES:NONE

PURPOSE: It passes attributes of adjective class to the main for application purpose.

## _4.4 BREAK_LEXEME_Ad

INPUTS: Ad_LEXEME RECORD

OUTPUTS: ATTRIBUTES

SUBORDINATES:NONE

PURPOSE: It passes attributes of adverb class to the main for application purpose.

## _6.1 STORE_KEY_LEXEME

INPUTS: KEY_RECORD, LEXEME_RECORD

OUTPUTS: NONE

SUBORDINATES: NONE

PURPOSE: It include the lexeme and key record into their corresponding files.

## _6.2 DELETE_KEY

INPUTS: KEY_RECORD

OUTPUTS: NONE

SUBORDINATES: NONE

PURPOSE: it remove the key record from the key file. The removal of corresponding lexeme record will be done only when

considerable number of key are deleted.

## _6.3 DISPLAY

INPUTS: LEXEME ATTRIBUTES

OUTPUTS: decoded information on screen

SUBORDINATES: NONE

PURPOSE: It decodes the attributes and displays the information on the screen.

## _1.2.1. MAKE_REC_KEY

INPUTS: KEY,POS,KEY_NO.

OUTPUTS:KEY RECORD

SUBORDINATES:NONE

PURPOSE: It attaches entry number, length of the record lexeme to the key that may help for referencing it later on.

## _1.2.2 COLLECT_ATTR

INPUTS: KEY_RECORD

OUTPUTS: LEXEME_RECORD

SUBORDINATES: GET_P_O_S

MAKE_FRAME_ACCE

PURPOSE: Accepts the key word and gives the key record and lexeme record for the storage purpose in the database in the corresponding files.

## _2.1.1. GET_KEY_FILE

INPUTS: KEY_RECORDS from key file

OUTPUTS: B+tree / KEY LIST

SUBORDINATES: NONE

PURPOSE: from key file, it forms list of key records or B+ tree depending on the search strategy adopted.

## _2.1.2. CONFORM_EXISTENCE

INPUTS: WORD, B+ tree/ KEY_LIST LEXEME, ATTRIBUTES

OUTPUTS: FLAG

SUBORDINATES:NONE

PURPOSE: It checks whether the word is in key list or in B+ tree or not.

## _2.2.1. GET_SUF_PREFIX_FILE

INPUTS: SUFFIX,PREFIX FILE

OUTPUTS: LIST OF SUFFIX,PREFIX

SUBORDINATES: NONE

PURPOSE: From suffix prefix file, it Extract the record entry number, which helps to remove it from the key file.

## _2.2.2. MAKE_OUT_KEY

INPUTS: WORD, list of SUFFIX, PREFIX

OUTPUTS: KEY

SUBORDINATES: NONE

PURPOSE: By removing suffix and/or prefix, by breaking compound words, etc. it forms possible key word or words.

## _1.2.2.1 GET_P_O_S

INPUTS: KEY RECORD

OUTPUTS: P.O.S {part of speech}

SUBORDINATES: NONE

PURPOSE: It provide the part of speech of the key word.

### _1.2.2.2 MAKE_FRAME_ACCE

INPUTS: P.O.S, attributes

OUTPUTS: LEXEME RECORD

SUBORDINATES: NONE

PURPOSE: It provide the frame for accepting the attributes. After collecting the necessary attributes it provide the lexeme

## 3.4 Detailed Design

Once a module is precisely specified, the internal logic for a module that will implement the given specification can be decided. Formal and Informal ways are used to specify the modules. The Functionality of the modules most of the time are specified in Natural language, in some places pseudo code of formal language are used. The purpose of Detailed Design is to avoid confusion, if the coding is not done by the designer.

### 3.4.1 Main Module

This module is the core of the lexicon software. It manages the flow of data to / from Input modules, Transform modules and Output modules. Its pseudo code will be as follows:

```
PROC MAIN()
    BEGIN (* main *)
        OPEN(key file)
OPEN(lexeme file)
OPEN(suf_prefix file)
PRINT "uPDATE
      dELETE
      rETRIEVE
      sTORE"
READ "option_1"
COLLECT_INPUTS
MORPHOLOGY
IF key exist THEN
     BEGIN
     IF option_1 = r THEN
          BEGIN
              GET_LEXEME
              PROCESSES
              OUTPUT
          END
     ELSE IF option_1 = d THEN
              OUTPUT
          ELSE IF option_1 = s THEN
        (*  report it is already in the file *)
                ELSE IF option_1 = u THEN
                     BEGIN
                         GET_LEXEME
```

file is read till the end is reached. Each of the process in the file may be involved in a number of message passing transactions , either sending data to another process or receiving data from other processes. The **QUEUE GENERATOR** analyses each of the process and for every process it makes queues listing all the transaction names encountered in the process, and their characteristics i.e. whether it is a sending operation or a receiving operation and if it is a sending operation then the identity of the receiver. However the identity of the receiving process is not stored which will be explained later.

In the previous chapter various features of the Concurrent C language has been discussed. It can be seen there, that the Concurrent C program consists of three main sections. One is the process specification section, where various attributes of the process is described, among these are the parameters that the process will take, the transaction in which it will enter, the processes to which it will send data. As seen in the examples given there that in the process specification section no identity of the sending process is given when a process is receiving any transaction. So, in developing the queues in **QUEUE GENERATOR** for receiving process only the transaction details have been included. Similarly it can be seen that the sending process doesnot have the transaction name, the specification of it consists of only the identity of the receiving process. Analysing the process body of the Concurrent C program it can be seen that

```
_1. PROC  COLLECT_INPUTS
          BEGIN
           IF option_2 <> i
              GET_KEY_WORD
              IF option_1 = s THEN
                 GET_KEY_LEXEME_RECORD
           END
_2. PROC  GET_KEY_LEXEME_RECORD
          BEGIN
            MAKE_KEY_RECORD
            COLLECT_LEXEME
          END
_3. PROC   GET_KEY_WORD
        BEGIN
        (* collect input from terminal and form string *)
        END
_4. PROC   MAKE_KEY_RECORD
          BEGIN
        (* pack key, pos, entry number of key, length of
        the record lexeme, if variable record is used *)
          END
_5. PROC   COLLECT_LEXEME
          BEGIN
           GET_POS
        MAKE_FRAME_ACCEPT
      END
```

**_6. PROC GET_POS**

```
    BEGIN
      READ inputs from terminal by setting queries
    END
```

**_8. PROC MAKE_FRAME_ACCEPT**

```
    BEGIN
    create node according to pos and accept inputs
    END
```

### 3.4.3 Transform Modules

Transform module as its name implies transform data into some other form. Most of the computational modules fall in this category. The pseudo codes for this is as follows:

**_1. PROC MORPHOLOGY**

```
    BEGIN
      REPEAT
        CHECK_IN_KEY
         IF NOT flag THEN
           FORM_KEY
         UNTIL key is found or no further
               modification is possible
      END
```

```
_2. PROC   CHECK_IN_KEY
           BEGIN
               GET_KEY_FILE
               CONFORM_EXISTENCE_EXTRACT
           END
_3. PROC   GET_KEY_FILE
           BEGIN
               form b+ tree
           END
_4. PROC   CONFORM_EXISTENCE_EXTRACT
           BEGIN
               IF key in b+ tree THEN
                     BEGIN flag <-- T
                           extract key record
                     END
               ELSE flag <-- F
           END
_5. PROC   FORM_KEY
           BEGIN
               IF NOT flag THEN
                  BEGIN
                    GET_SUF_PREFIX
                    MAKE_OUT_KEY
                  END
           END
```

```
_6. PROC  GET_SUF_PREFIX
          BEGIN
         (* get list of suffix and prefix L1 and L2
            respectively *)
          END
_7. PROC  MAKE_OUT_KEY
          BEGIN
             (* make suffix or prefix of the word *)
             IF suffix_word IN L1 THEN
                  key <--- word - suffix + x
               (* e.g. x= 'y' if suffix removed is
                "ies", etc. *)
             ELSE IF prefix_word IN L2 THEN
                  key <--- word - prefix
            ELSE IF (* compound word break and check both
                  words in lexicon *)
             ELSE (* report failure *)
            END
_8. PROC  GET_LEXEME
          BEGIN
            BREAK_KEY_RECORD
            EXTRACT_LEXEME
          END
```

```
_9. PROC  BREAK_KEY_RECORD

        BEGIN

          (* break key_record into part of speech,key

          entry number, length of lexeme record and key,

          if variable record is used *)

        END

_10. PROC  MAKE_FRAME_ACCEPT

        BEGIN

         (*  calculate  actual address and  retrieve  the

            record into the appropriate pos frame *)

        END

_11. PROC  PROCESSES

        BEGIN

            CASE pos  OF

            noun: N_BREAK_LEXEME

            verb: V_BREAK_LEXEME

            adjective: A_BREAK_LEXEME

            adverb: AD_BREAK_LEXEME

        END

_12. PROC  N_BREAK_LEXEME

        BEGIN

            RETURN  (* grammatical record,

                    semantic record,

                      morphological list,
                relations addresses, and
                derived words record (pos, address
                and length of record)  *)
          END
_13. PROC  V_BREAK_LEXEME
        BEGIN
          RETURN  (* grammatical record,
            basic acts,
            semantic record,
            conjugate list,
            relations addresses,
            derived words record ( pos, address and
                        length of record ) *)
        END
_14. PROC  A_BREAK_LEXEME
        BEGIN
          RETURN  (* quality record,
                quantity record,
                morphological list,
                relations addresses, and
                derived words record ( pos,
                address and length of record) *)
        END
```

**_15. PROC AD_BREAK_LEXEME**

```
BEGIN
   RETURN  (* semantic record,
            morphological list,
            relations addresses, and
            derived words record ( pos,
            address and length of record) *)
END
```

### 3.4.4 Output Modules

These modules obtain information from their super ordinate module and then pass it on to their subordinate. They are typically used for outputing data to the environment. The Output modules take the output produced and prepare it for better presentation or make appropriate changes.

**_1. PROC OUTPUT**

```
BEGIN
  IF option_1 = 's' THEN
    STORE_KEY_LEX_RECORD
  ELSE option_1 = 'd' THEN
    DELETE_KEY_RECORD
  ELSE option_1 = 'r' THEN
    DISPLAY
END
```

**_2. PROC STORE_KEY_LEX**

```
        BEGIN
       (* include key record into key file lexeme
         recorded into lexeme file *)
        END
```
**_3. PROC DELETE_KEY_RECORD**

```
        BEGIN
          (* remove  key from b+ tree and the key file
             release the space occupied by key and
             lexeme to the system *)
        END
```
**_4. PROC DISPLAY**

```
        BEGIN
  (* decode all attributes of the word and display on screen *)
        END
```

## 3.5 Interfaces

Interfaces facilitate a working environment between the user of the software, such as Parser, and the Lexicon. Some of the interfaces which have to be design during the implementation of the Design of Lexicon has been given as Recognizer, Feature Extractor, and Extracting Relation.

### 3.5.1 Recognizer

A word is considered to be in a lexicon if it is found in the key file and one of the following condition is fulfilled. If a word found in key file is the same as key word then this is named as surface level check up (i.e. morphological process/es has not been done on the word). If the word is either in Conjugate List or Derived word, then this is named as deep level check up.

The pseudo code for recognizer is as follows:-

```
BEGIN ( recognizer )
   word <-- GET_WORD
   key <-- MORPHOLOGY
     IF key in b+ tree THEN
        BEGIN
          force <-- false
          lexeme <-- GET_LEXEME ( key )
        WHILE NOT force DO
           BEGIN
             IF (key = word)
               OR ( word in lexeme.conjugate )
               OR ( word in lexeme.derived_word )
               THEN BEGIN
                 flag <-- true (* indicate existence *)
                 force <-- true
                 END
             ELSE IF lexeme.derived_word subset word THEN
               lexeme <-- extract (* address of derived_word *)
               ELSE BEGIN
```

```
                force <-- true
                flag <-- false
              END
          END
      ELSE  flag <-- false
  END (*recognizer*)
```

### 3.5.2 Extract feature

Generation of word attributes goes along with recognizer. If a word is recognized by the lexicon it is possible to extract its attributes along with the process. The extracted attributes can be used directly by the user with or without decoding.

The pseudo code is as follows:-

```
  BEGIN (* extract features *)
   word <-- GET_WORD
   key <-- MORPHOLOGY
    IF key in b+ tree THEN
       BEGIN
         force <-- false
         lexeme <-- GET_LEXEME ( key )
         WHILE NOT force DO
          BEGIN
            IF key = word THEN
              BEGIN
                flag <-- true (* indicate existence *)
```

```
        force <-- true
      END
    IF ( word in lexeme.conjugate ) THEN
      BEGIN
        flag <-- true (* indicate existence *)
        force <-- true
        lexeme.grammatical <-- conjugate.grammatical
      END
    IF (word in lexeme.derived_word) THEN
      BEGIN
        flag <-- true (* indicate existence *)
        force <-- true
        lexeme  <--  (* extract lexeme  of
                         derived word *)
END
ELSE
  IF ( lexeme.derived_word subset word )
  THEN BEGIN
    lexeme <-- (extract address of derived_word)
    lexeme.grammatical<--lexeme.conjugate.grammatical
    flag <-- true  (* indicate existence *)
    force <-- true
END
ELSE BEGIN
```

```
            force <-- true
            flag <-- false
          END
        END
      ELSE  flag <-- false
   END
END (* extract feature *)
```

### 3.5.3 Extracting Relations

Representation assumption:

1. relation between two words is represented by a marked link between two nodes of the word and the relation that represent a syntax or the concepts. For example: RAIN --- [ MAGNITUDE ] ---> HEAVY, heavy rain, but not large rain. MAN --- [ ISA ] ---> ANIMAL

2. For any pair of words there is only one link for a distinct relation.

3. There are limited number of different relations, but their exact number is left open.

Processing assumption:

1. Relation between words are identified by searching the network from one or both nodes representing the two words.

2. According to the nature of the relations, even though the path joining the two nodes is obtained search may continue till a circuit is obtained. Such a relations are considered to be equivalent relations (i.e. reflexive, symmetric and transitive).

The algorithm enables one to find synonyms of already stored word by applying SYNONYMS relation mark. The assumption taken is synonyms relation is transitive.

```
BEGIN (* synonyms *)
   word <-- GET_WORD
   key <-- MORPHOLOGY
     IF key in b+ tree THEN
        BEGIN
          lexeme <-- GET_LEXEME ( key )
          ref_lex1 <-- address of lexeme
          flag <-- false
         WHILE NOT flag DO
            BEGIN
            IF word = key THEN
              BEGIN
                hold <-- hold + lexeme.key
                ref_rel <-- lexeme.relation_syn
                IF ref_rel = ref_lex1 THEN
                  force <-- true
                    ELSE lexeme <-- extract ( ref_rel )
               END
            ELSE
               IF word in lexeme.conjugate THEN
               key <-- word
        ELSE
          IF word in lexeme.derived_word THEN
            BEGIN
            ref_lex2 <-- address of lexeme.derived_word
            lexeme <-- extract ( ref_lex )
            WHILE ref_lex1 <> ref_lex2 DO
              BEGIN
                hold <-- hold + lexeme.key
                ref_lex2 <-- lexeme.relation_syn
                lexeme <-- extract ( ref_rel2 )
              END
          force <- true
        END
      END
    ELSE force <-- true
END (* synonyms *)
```

# CHAPTER IV

# CONCLUSION

The Lexicon which has been designed is independent of any particular application. While designing the lexicon English language is taken as a core language. But this does not put any restriction for using this design on any other language. different from English. Most of the structure features are applicable to all Natural Languages. For some languages slight and appropriate modification may be needed. The intention of this work is to make the design of Lexicon one and the same for all Natural Languages. Those parts which may or may not be needed in some languages have been kept as optional.

For complete analysis of Natural Language Process (NLP), all the components of it have to act together. Much work have been done on the other part of NLP, in School Of Computer and Systems Sciences, JNU, using Prolog Language as a tool. While designing software for lexicon working tool is considered to be Prolog language or C, for simplicity of interface. For this reason the Structured Design is adopted than object oriented approach.

A file which is a collection of records has a great role in implementation of lexicon. As a primary memory is limited in size and volatile in nature, in our design we have assumed Btree mechanism for file organization. The assumption taken while designing is the two files, key record file and lexeme record file reside in secondary storage. When the system is in use a B+

tree will be formed with the contents of the key record file, which help us for accessing the data from lexeme record file.

Since different senses of a word are stored in the lexicon, the user has to try for different possibilities during the application to resolve the sense ambiguity.

This Lexicon Design is for single language ( Mono-lingual ). But it may help in designing multi-Lingual Lexicons with a little additional efforts. This modifications and implementation of the Design will be future expansion of this work.

As it is indicated above to design general purpose lexicon, the characteristic of different natural Languages have to be studied. Incorporating some missing features for the completeness of the work is also its future expansion.

# BIBLOGRAPHY

1. Evens, M. Walton, Relational Model Of The Lexicon, Cambridge University Press, 1982.
2. Jalote, Pankaj, An Integrated Approach to Software Engineering. Narosa publishing house, 1991.
3. Harris, Marry Dee, Introduction to Natural Language Processing, Reston, A Prentice-Hall Company, 1985.
4. Minisk, Marvin, Semantic Information Processing, the MIT Press, 1982.
5. Radford, Andrew, Transformational Grammar, Cambridge University Press, 1988.
6. Rich, Elaine, Artificial Intelligence, McGraw-Hill, New York, 1983.
7. Sager, Noami, Natural Language Information Processing, Addison-welsey Publishing Company, 1981.
8. Sinha R.M.K. (ed.), Commputer Processing Of Asian Languages, 1983.
9. Sowa, John F., 1983. Conceptual Structures, Information processing In Mind And Machine, Addison-Welsey Publishing Company.