INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17” x 23” black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A dynamic model of comprehension with applications to Korean and English

Lee, Chang-in, Ph.D.

Rice University, 1988
RICE UNIVERSITY

A DYNAMIC MODEL OF COMPREHENSION WITH APPLICATIONS TO KOREAN AND ENGLISH

by

CHANG-IN LEE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

DOCTOR OF PHILOSOPHY

APPROVED. THESIS COMMITTEE:

Sydney M. Lamb, Professor of Linguistics and Semiotics, Director

James E. Copeand, Professor of Linguistics and Semiotics

Philip W. Davis, Professor of Linguistics and Semiotics

Robert Cartwright, Professor of Computer Science

Houston, Texas
May, 1988
ABSTRACT

A DYNAMIC MODEL OF COMPREHENSION
WITH ALLOCATIONS TO KOREAN AND ENGLISH

BY

CHANG-IN LEE

Most researchers working to develop a comprehension process fail to distinguish the different levels often recognized by linguists. Some, for example, have attempted to incorporate a composite form such as 'has' into one terminal symbol without addressing its internal complexity. The present study in contrast distinguishes different levels. The context-sensitive dispute is thus resolved by viewing complex structures as composites of multiple context-free layers that are interrelated by context-sensitive interfaces.

In the design of a comprehension system, this study recognizes the essential structural differences between the linguistic information and the computer procedure. Thus, the linguistic information is not incorporated into the program itself. Examples are illustrated from Korean and English.
ACKNOWLEDGEMENTS

I wish first of all to thank the members of my committee, Sydney Lamb, James Copeland, Philip Davis and Robert Cartwright, for giving of their valuable time. My special thanks go to Sydney Lamb, my advisor, who has been a patient source of advice and guidance. He has generously given his time to discuss problems and to suggest improvements in style, organization, and other matters.

I also wish to acknowledge some of my fellow graduate students in making many useful suggestions regarding English style. Especially, I thank my friend, Cindy Meyer.

Last, but not least, I am grateful to my husband and my son, Sang-ha and Young-il, without whose help this work could never have been done.
TABLE OF CONTENTS

Abstract
Acknowledgements
Table of Contents
Abbreviations

CHAPTER 1. GOALS AND THEORETICAL BACKGROUND 1

1.1 A Cognitive Approach to Language 1
1.2 A Stratified Decoding System 6
  1.2.1 Separation of the linguistic information from the computer procedure 6
  1.2.2 A Stratified Approach 7
  1.2.3 A Nception structure for a dictionary entry 9
  1.2.4 The dictionary structure 10
  1.2.5 The conceptual organization 17
1.3 System modification 18

CHAPTER 2. AN INFORMAL SURVEY OF KOREAN GRAMMAR 20

2.1 Nouns 22
  2.1.1 Simple nouns 22
  2.1.2 Pronouns 22
  2.1.3 Derived nouns 23
2.2 Adjectives 27
2.3 Participial clauses 31
2.4 Noun phrases 36
2.5 Verb morphology 36
  2.5.1 Morphological passive -Ki 38
  2.5.2 Aspect—Perfective 39
  2.5.3 Honorific Marker 43
  2.5.4 Tense 44
  2.5.5 Mood 54
2.6 Passive beyond -Ki 55
2.7 Conjunction 60
2.8 Complex aspect—Progressive 61
2.9 Noun phrase function markers 65
2.10 Clause structure 71
  2.10.1 Predicate 71
  2.10.2 Topic / Subject 72
  2.10.3 Object 73
  2.10.4 Adjuncts 73
  2.10.5 The order of clause constituents 74
APPENDIX A  Illustration of an example  215
APPENDIX B  Dictionaries  219
APPENDIX C  Programming codes  241
FOOTNOTES  302
REFERENCES  312
### ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>aAA</td>
<td>Attributive Adjective Marker</td>
</tr>
<tr>
<td>Adj-C</td>
<td>Comparative Adjective</td>
</tr>
<tr>
<td>Adj-Cl</td>
<td>Classifiable Adjective</td>
</tr>
<tr>
<td>Adj-D</td>
<td>Deictic Adjective</td>
</tr>
<tr>
<td>Adj-N</td>
<td>Numerative Adjective</td>
</tr>
<tr>
<td>ANI</td>
<td>Animal</td>
</tr>
<tr>
<td>B</td>
<td>Blocking</td>
</tr>
<tr>
<td>BP-A</td>
<td>Alienable Body Part</td>
</tr>
<tr>
<td>BP-IA</td>
<td>InAlienable Body Part</td>
</tr>
<tr>
<td>C1</td>
<td>Clause1</td>
</tr>
<tr>
<td>C2</td>
<td>Clause2</td>
</tr>
<tr>
<td>CFG</td>
<td>Context-Free Grammar</td>
</tr>
<tr>
<td>CLASER</td>
<td>Classifier</td>
</tr>
<tr>
<td>con.</td>
<td>concessive</td>
</tr>
<tr>
<td>Conc-Obj</td>
<td>Concrete Object</td>
</tr>
<tr>
<td>CONJ</td>
<td>Conjunction</td>
</tr>
<tr>
<td>CSG</td>
<td>Context Sensitive Grammar</td>
</tr>
<tr>
<td>DEC(F)</td>
<td>Formal Declarative Sentence Ending</td>
</tr>
<tr>
<td>DEC(N)</td>
<td>Neutral Declarative Sentence Ending</td>
</tr>
<tr>
<td>DEC(Pol)</td>
<td>Polite Declarative Sentence Ending</td>
</tr>
<tr>
<td>Def</td>
<td>Definite</td>
</tr>
<tr>
<td>DEF</td>
<td>Definer</td>
</tr>
<tr>
<td>DEIC</td>
<td>Deictic</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>DS</td>
<td>Declarative Sentence</td>
</tr>
<tr>
<td>E</td>
<td>Enabling</td>
</tr>
<tr>
<td>Ed</td>
<td>Edible</td>
</tr>
<tr>
<td>ED-LIQ</td>
<td>Edible-Liquid</td>
</tr>
<tr>
<td>ED-SOL</td>
<td>Edible-Solid</td>
</tr>
<tr>
<td>EPI</td>
<td>Epithet</td>
</tr>
<tr>
<td>FutPapl</td>
<td>Future Participle Marker</td>
</tr>
<tr>
<td>FutT</td>
<td>Future Tense</td>
</tr>
<tr>
<td>Gv</td>
<td>Given</td>
</tr>
<tr>
<td>HB</td>
<td>Human being</td>
</tr>
<tr>
<td>Hor</td>
<td>Honorific</td>
</tr>
<tr>
<td>Imp</td>
<td>Imperative</td>
</tr>
<tr>
<td>InDef</td>
<td>InDefinite</td>
</tr>
<tr>
<td>indic.</td>
<td>indicative</td>
</tr>
<tr>
<td>interro</td>
<td>interrogative</td>
</tr>
<tr>
<td>nAdd</td>
<td>Additive Marker</td>
</tr>
<tr>
<td>nAg</td>
<td>Agent Marker</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>nCaus</td>
<td>Causative Marker</td>
</tr>
<tr>
<td>nCom</td>
<td>Comitative Marker</td>
</tr>
<tr>
<td>nDat</td>
<td>Dative Marker</td>
</tr>
<tr>
<td>nDir</td>
<td>Directional Marker</td>
</tr>
<tr>
<td>Neg</td>
<td>Negative</td>
</tr>
<tr>
<td>nIns</td>
<td>Instrument Marker</td>
</tr>
<tr>
<td>nM</td>
<td>Noun Marker</td>
</tr>
<tr>
<td>nLim</td>
<td>Limitive Marker</td>
</tr>
<tr>
<td>nLoc</td>
<td>Locative Marker</td>
</tr>
<tr>
<td>nObj</td>
<td>Object Marker</td>
</tr>
<tr>
<td>NONED</td>
<td>NonEdible</td>
</tr>
<tr>
<td>NF</td>
<td>Noun Phrase</td>
</tr>
<tr>
<td>nPoss</td>
<td>Possessive Marker</td>
</tr>
<tr>
<td>nRol</td>
<td>Role Marker</td>
</tr>
<tr>
<td>NUM</td>
<td>Numerative</td>
</tr>
<tr>
<td>P1</td>
<td>Participant1</td>
</tr>
<tr>
<td>P2</td>
<td>Participant2</td>
</tr>
<tr>
<td>PasPap</td>
<td>Past Participle</td>
</tr>
<tr>
<td>Past</td>
<td>Past Tense</td>
</tr>
<tr>
<td>PPART</td>
<td>Particular Participant</td>
</tr>
<tr>
<td>pass-cond</td>
<td>Passive-Condition</td>
</tr>
<tr>
<td>Pl</td>
<td>Plural</td>
</tr>
<tr>
<td>PrePap</td>
<td>Present Participle</td>
</tr>
<tr>
<td>PreT</td>
<td>Present Tense</td>
</tr>
<tr>
<td>Prog</td>
<td>Progressive</td>
</tr>
<tr>
<td>Q(N)</td>
<td>Neutral Indicative Sentence Ending</td>
</tr>
<tr>
<td>Q(Pln)</td>
<td>Plain Indicative Sentence Ending</td>
</tr>
<tr>
<td>Q(Pol)</td>
<td>Polite Indicative Sentence Ending</td>
</tr>
<tr>
<td>RPastT, Rp</td>
<td>Remote Past Tense</td>
</tr>
<tr>
<td>S</td>
<td>Sentence</td>
</tr>
<tr>
<td>SE</td>
<td>Sentence Ending</td>
</tr>
<tr>
<td>Sg</td>
<td>Singular</td>
</tr>
<tr>
<td>Sp</td>
<td>Simple past</td>
</tr>
<tr>
<td>SUBCL</td>
<td>Subordinate Clause</td>
</tr>
<tr>
<td>SUBJ</td>
<td>Subject</td>
</tr>
<tr>
<td>subj-cond</td>
<td>Subject-Condition</td>
</tr>
<tr>
<td>top-cond</td>
<td>Topic-Condition</td>
</tr>
<tr>
<td>TVi</td>
<td>Tensed Intransitive Verb</td>
</tr>
<tr>
<td>TVt</td>
<td>Tensed Transitive Verb</td>
</tr>
<tr>
<td>Vi</td>
<td>Intransitive Verb</td>
</tr>
<tr>
<td>VI</td>
<td>Intransitive Verb Phrase</td>
</tr>
<tr>
<td>VP</td>
<td>Verb Phrase</td>
</tr>
<tr>
<td>vPur</td>
<td>Purposive Marker</td>
</tr>
<tr>
<td>vRe</td>
<td>Reasoning Marker</td>
</tr>
<tr>
<td>Vt</td>
<td>Transitive Verb</td>
</tr>
</tbody>
</table>
CHAPTER 1. GOALS AND THEORETICAL BACKGROUND

This dissertation is concerned with the design of a stratified comprehension system with an expandable dictionary. This overall objective may be broken down into three interrelated aims—(1) to construct a stratified language comprehension system, using syntactic tree pathways and various other devices (Chapters 3 - 5), (2) to model one means of system expansion (Chapter 6), and (3) to write a small grammar of Korean for use by the first two systems (Chapter 2).

1.1. A Cognitive Approach to Language

A Discussion of Cognitive Linguistics is provided as a guide for the organization of the linguistic information, which will be a basis for our decoding system. Cognitive linguists approach language differently from other linguists. They view linguistic data as manifestations of a cognitive system and the cognitive processes of people using language. Along with Cognitive Psychology, Artificial Intelligence (AI), and Neuro-Science, Cognitive Linguistics can be considered as a branch of Cognitive Science, whose subject matter is the study of cognition/mind/thought/intelligence. Most researchers in AI are in general agreement with the cognitive approach in Linguistics. In 1976, the generative grammarians Dresher and Hornstein wrote in their paper "On Some Supposed Contributions of Artificial Intelligence to the Scientific Study of Language" as follows:
"Workers in AI have misconstrued what the goals of an explanatory theory of language should be." (p.377)

"Current AI research into language is headed in a wrong direction and it is this research that is unlikely to contribute to a scientific theory of language." (p.322)

They bitterly criticized AI researchers because most were headed in a direction away from Generative Linguistics and were not contributing to a Chomskyan theory of language. Later, Winograd (1977:152) stated in his paper, "On Some Contested Suppositions of Generative Linguistics about the Scientific Study of Language," as follows:

"--- the currently dominant school of Chomskian Linguistics is following an extremely narrow and isolated byway of exploration. The limitations result not from the structure of language, but from a commitment to a specific arbitrary set of meta-linguistic beliefs."

Unlike generative linguists, AI researchers and cognitive linguists pay attention to the cognitive processes of people. In Cognitive Linguistics, researchers are concerned with linguistic processes such as encoding, decoding and system modification. Here we need to discuss Cognitive Linguistics further as theoretical background. We include this discussion because a dynamic grammar and a cognitive grammar are two sides of one coin; they presuppose and reinforce each other.

Lamb (1984, 1986) suggests that Cognitive Linguistics considers the 'information system' that is used by the 'individual' for production (encoding) and comprehension
(decoding). That is, it is the study which attempts to model the information system that is present in the mind of the 'typical' (not 'ideal') individual. What then is an information system? Typical communication situations involve two or more independent information systems, not necessarily of the same kind—one could be a person, the other a computer. The classic view was that the best way to build an information system was to base it on formal logic. In his book Kowalski (1979:239) defined information system as follows: "--- information system, ---, is used to refer to any collection of assumption (or beliefs) expressed in logic." But the information system that we find in language is flexible. Since this information system is not directly observable, its definition depends largely on how it is modelled. Some properties of information systems are follows:

A. An information system has diversity of expression. We can not form a message if every expression is the same.

B. This diversity is not totally unconstrained. We need pattern within diversity.

C. Within an information system, there must be connections between expression and content.

D. Ordinary human information systems are open-ended, i.e. capable of generating new signals to reflect new meanings.

Cognitive Linguistics is concerned with these properties when modelling a system, and workers in this field entertain the hypothesis that language consists of
relationships between content and expression that make possible the production and understanding of texts. Following Anderson (1983), we can assume that human high-level activities such as architecture, mathematics, computer programming, and language processing are developed through the common faculty of the brain. If this interpretation is correct, language must be a kind of representation of this common faculty of human beings. This claim suggests that the structure of language has correlation with the structure of the brain. According to many cognitive linguists, language can be viewed as consisting of relationships. This suggestion is supported by Kintsch (1970). Kintsch proposes that neurological structure centers around so-called "graphs." These are systems of nodes, corresponding to items or elements in the brain interconnected by 'links' which represent various sorts of relationships among the items.

By comparing the approach of Cognitive Linguistics with that of other types of linguistics, e.g. Generative Linguistics or Descriptive Linguistics, we can draw a clearer picture of Cognitive Linguistics. Keeping it in mind that language in Cognitive Linguistics is viewed as an information system, let us consider Figure 1.1 (cf. Lamb 1987).
The aim of Descriptive Linguistics was to describe mainly the structure of 'texts' (usually, however, not texts in general but only sentences) through the use of methods of classification and segmentation. This analysis results in a view of language as a collection of objects rather than a conglomerate of thousands of intertwined relationships. On the other hand, the main concern of Generative Linguistics was to characterize the 'grammar' of a language defined as rules for generating strings of symbols. In Generative Linguistics, linguists start with rules which are derived from observing data, and they test deductively to find out whether the rules will generate the data or not. In Cognitive Linguistics, however, we construct a model or a theory of the information system which a language user must have in order to speak and understand a language. Therefore, cognitive linguists believe that the proper domain of study is (1) the structure of the linguistic knowledge possessed by an individual who uses a language AND (2) the processes which operate with and upon the knowledge structure. Cognitive linguists use as evidence the empirical data gathered in descriptive and generative linguistics to verify this theory of language as an informa-
tion structure. Cognitive linguists see it as circular to rely on intuitions for an explanation of what goes on in the mind of a language user. Instead they rely on non-intuitive linguistic data to explain the mind. That is, Cognitive Linguistics attempts to study the mind of human beings by using textual linguistic data.

1.2. A Stratified Comprehension System

1.2.1. Separation of Linguistic Information from Computer Procedure

This study attempts to construct a system for natural language comprehension (interpretation), which is one of three basic processes operating upon the linguistic structure. Much previous research on Natural Language Processing seems not to have recognized the essential structural differences between linguistic information system and computer procedure. The design of the process of comprehension in this study aims to implement Lamb's (1964) suggestion that linguistic information should not be incorporated into the program itself, since the result would be an ad hoc procedure for the particular information rather than a general procedure. Lamb (p.38) further suggests that we can gain the following advantages by separating linguistic information and computer procedure:

(1) A linguist can write his rules as linguistic statements rather than in some programming language.

(2) Whenever it is needed, a linguist can revise his
rules without any need for reprogramming.

(3) The basic operations are written only once in the separated program.

(4) The basic operations are general enough to parse sentences written in any natural language.

The primary purpose of the work described in this study is to construct a general system for actualizing the advantages that Lamb outlines.

1.2.2. A Stratified Approach

The general comprehension process for a written language involves proceeding from graphemic representations to conceptual representations. With a goal of developing such an interpretation process, there has been continuing research in the field of linguistic automation for three decades. In the 1960's and 1970's, much of the research on the interpretation process adopted either a wholly syntactic or a wholly semantic orientation, e.g. Kuno (1965), Schank (1972). However, the present study includes three phases--morphological, syntactic, and semantic. To make a choice among these phases would deny the independent contribution of each to the overall structure. It is in keeping with the stratificational view that the three approaches represent generalizations at different levels of representation. A morphological approach (hereafter 'morphological decoding') is appropriate for the representation at the morphemic level, a syntactic approach (hereafter 'syntactic decoding') is appropriate
for the representation at the lexemic level, and the semantic approach (hereafter 'semantic decoding') is appropriate for the representation at the sememic level (i.e. to identify the relationship between a process and participants). This study aims to construct a decoding system that incorporates these three approaches.

The organization of the system is based on the assumption that the complex structure of language is a composite of multiple layers. By viewing complex structures as made up of different strata with realizational relationships between them, we can describe the tactics of each stratum as context-free. By positing context-sensitive realizational relationships between levels, each stratum is relatively simple.

This stratified view of language was formulated by cognitive linguists, e.g. Lamb (1971), Lockwood (1973), who have developed and elaborated the notions of 'stratum' (or 'level') and 'tactic patterns.' They have suggested modularity within the system by positing separate tactic patterns at each level. Owing to the modularity of each stratum, we do not have to rewrite the Morphemic Supplement, the Master Dictionary, the Lexemic Supplement I and the Sememic Supplement II, even if we convert the Lexemic Supplement II from a context-free parser (as in the present system) to a context-sensitive for the phrase structure later on. However, the Lexemic Supplement II (which must be implemented on a new parser) and the Sememic Supplement I
(which is heavily dependent on the Lexemic Supplement II) would need to be adapted to the new context-sensitive parser.

1.2.3. A Nection Structure as a Dictionary Entry

The processes for morphological, syntactic and semantic decoding suggested in this study are performed by an algorithm which searches through dictionary-like structures set up between each pair of levels (Lamb 1985, 1987). All linguistic information, which is totally separated from computer procedures, is in the form of statements which are like dictionary entries. Each such entry is an alternative notation for the nection structure shown in Figure 1.2.

![Diagram showing the structure of a nection with branches for meaning, tactic function, and expression.](image)

Fig. 1.2.

Linguistic structure is 'a system of signs'. In relational network terminology, linguistic structure is viewed as 'a system of nections'; the cognitive representation of the linguistic sign is the nection (cf. Lamb 1984, 1987). The nection has two sides, one content-oriented, the other expression-oriented. It is thus an essentially semiotic
unit. The general properties of the typical nection are shown in Figure 1.2. The dictionary-like entry uses a linear notation for the nection, for ease of both reading and writing, as well as for computer manipulation.

1.2.4. The Dictionary Structure

In constructing a general decoding system which separates linguistic information from programming procedures, the most important part is to set up appropriate, efficient and formalized dictionaries. The dictionary system within a stratified model can be viewed in either of two ways in accordance with Lamb's notion (1987: 4-4, 4-7):

I.

\[
\begin{array}{c}
\text{CONCEPTUAL REPRESENTATIONS} \\
\quad \uparrow \\
\text{SEMICON} \\
\quad \uparrow \\
\text{LEXICON} \\
\quad \uparrow \\
\text{MORPHICON} \\
\quad \uparrow \\
\text{GRAPHEMIC REPRESENTATIONS}
\end{array}
\]

Fig. 1.3.
II. **CONCEPTUAL REPRESENTATIONS**

- **SEMEMIC SUPPLEMENT**
- **LEXEMIC SUPPLEMENT**
- **MORPHEMIC SUPPLEMENT**
- **GRAPHEMIC REPRESENTATIONS**

Fig. 1.4.

Starting from the dictionary structure I in Fig 1.3, we get the dictionary structure II of Fig 1.4 by taking all of the simple, regular lexemes and putting all of their information from the Morphicon, the Lexicon and the Semicon into the Master Dictionary. The result is that the Master Dictionary includes most of the Morphicon of dictionary structure I plus some of the Lexicon and the Semicon. The reason for going to dictionary structure II is to eliminate repetition that would be present in the structure I. (Detailed discussion of the Master Dictionary is provided in section 3.3). Some readers might ask why information about semantic properties is included in the Master Dictionary, rather than in the dictionaries for the sememic level (Sememic Supplement I or II). One might also think that it is mixing levels to put semantic information in the
Master Dictionary, which also includes information about tactic functions. To consider such questions it is necessary to look more closely at the stratification of language.

Considerations of linguistic data lead us to conclude that language can be viewed as being structured in multiple layers. How many levels, then, can be posited in the linguistic structure? As Lamb claims (class notes), the number of levels is not clear-cut. It depends on our point of view. He illustrates that statement with a tree branch as a non-linguistic analogy (Figure 1.5).

If we take the point of view I, the structure appears stratified in two levels (expression and content) as early linguists, e.g. Hjelmslev (cf. Lamb 1966b), had dealt with it. If the point of view II is taken, the structure appears to be stratified in three levels (phonemic, lexico-grammatic and
sememic). On the other hand, if we take the view of point of III, a language seems to be structured in four strata (phonemic, morphemic, lexemic and sememic). Rather than make a choice between these different point of views, we can employ all of them, each where appropriate.

As far as simple lexemes are concerned, point of view I is correct, as the connections for a simple lexeme form a single nection. Figure 1.6 shows that all four strata can be spanned by just one nection.

![Diagram of Ed-Sol with Noun connections]

**Fig. 1.6.**

Therefore, the establishment of the Master Dictionary, which includes semotactic information in addition to lexical categories, is the strategy of storing information about a single nection in a single entry.

The present study adopts a version of dictionary system II and proposes a master dictionary together with six supplements for English and five supplements for Korean.
English requires two morphemic supplements, while Korean needs only one (cf. 3.2)). The number of supplements depends on the language we are dealing with. The number of supplements constructed in this study is determined by practical need, and not by theoretical considerations. For practical purposes it is convenient to split each of the supplement into two sections. In the case of English, there are six supplements and one Master Dictionary. The six supplements include two for the level of word structure (the Morphemic Supplements I and II), two for the lexemic level (the Lexemic Supplements I and II) and another two for the sememic level (the Sememic Supplements I and II). Thus, the dictionary system that the present study suggests, based on the dictionary structure II, is that shown in Figure 1.7.
Adopting the version dictionary system II shown in Fig. 1.7, in accordance with Lamb's suggestion about separation of linguistic information from the computer procedure, we propose an overall architecture of the system as shown in Figure 1.8.
Fig. 1.8. (The System Architecture)

\[ \text{Data Flow} \quad \text{Dictionary Interaction} \]
1.2.5. The Conceptual Organization

The central theme that marks the cognitive sciences during the last five years is the conceptual representation (cf. W. Kintsch, J. Miller and P. Polson 1984:306). In this study the conceptual content is constructed from ordinary context-free syntactic trees. Traditional syntactic trees are thus seen as a useful preliminary to the organization of semantic statements even for a program whose primary aim is the computation of meaning (cf. Chapter 5). The conceptual representation is not just a syntactic tree going by a different name, since conceptual representations and syntactic trees are clearly not in a relationship of one-to-one correspondence. A given function can be represented by different syntactic forms, and a given syntactic form can represent different functions, depending on context. In order to represent concepts relating to copular verb constructions, for instance, we do not identify the English be alone as the process if it is followed by an adjective, but the combination of be and the ADJ as the process. As another example, in the conceptual representation for passive constructions, we assign the subject as P-2 (the second participant or argument) by means of the rule, 'if the lexeme node 'passive' is found in the syntactic tree, identify the NP string after 'by' in the VP as P1 and identify the first NP string as P2.' The pathway strategy converts syntactic trees into provisional conceptual
representations, which are not necessarily well-formed conceptually. The connection between a process and its participants must be checked at the level of conceptual syntax. A clause which is conceptually ill-formed (e.g. % I ate the table) is rejected after comparison of the semantic properties of participants with those that a process presupposes. Just as the process for conceptual representations is carried out by keeping track of syntactic tree structure, the comparison process is carried out by keeping track of semantic tree structure.

The comprehension procedure is running successfully using small grammars of English and Korean. Examples from these two languages are shown in chapters 4 and 5. This general system is not intended to constitute a complete software system for linguistic automation.

This dissertation suggests that we can establish a general procedure for comprehension via any language for which a linguist can provide a suitable grammar. The system can be useful for a student of Linguistics who wants to test his grammar by parsing sentences. One can also apply it to developing a Machine Translation system or a Question-Answering system.

1.3. System Modification

This study describes a set of procedures for adding new information to a linguistic system. To be realistic, a
natural language understanding system must be able to acquire new lexical items including the necessary information about its conceptual properties. Without this ability to expand, a system would have to be complete from the start, which is of course impossible. This acquisition process is implemented by interacting with the user in a mode that has become customary for computers—by means of menus. The system elicits the necessary information from the user by means of interactive conversation. To reflect this process we call the system 'dynamic'; it can expand from the core lexicon to adapt to specific uses.
CHAPTER 2 AN INFORMAL SURVEY OF KOREAN GRAMMAR

Linguists have attempted to describe a great many languages over the centuries, but despite these attempts, no grammarian has succeeded in writing a 'complete' grammar. A complete grammar would be applicable to a given language and would not only cover all the observed data but also predict unobserved data. Suggesting some criteria for a grammatical description ('general, specific, inclusive, productive and efficient'), Hockett (1954:398) states that no grammar would be complete which did not meet the first four criteria above. As he notes, however, it is impossible to write a grammar that is general and inclusive. Language is too complex. Since even a single lexeme has its own syntax and the set of lexemes is constantly changing, it is impossible to write a complete grammar (cf. Lamb 1987). We may write a core grammar, but not a 'complete' grammar. This chapter therefore does not attempt an exhaustive description of the Korean grammar. The intent, rather, is to provide a sketch of the core grammar of written Korean. What we include in the present chapter touches upon most of the levels of word structure and phrase structure of written Korean.

Various theoretical aspects of Korean grammar have been studied over a long period of time. Many Korean linguists, e.g. Lee and Im (1983), Nam and Ko (1985), I. S. Yang (1972), D. W. Yang (1975), H. B. Lee (1975), Bak (1981), etc., have studied various aspects of Korean syntax and semantics. In
earlier work, Martin and Lee (1968) and Martin (1952) have also described features of Korean. All the recent analyses (in the '70's and '80's), however, have been undertaken in the context of the generative approach to language. In the past thirty years, Generative Linguistics has been predominant due to the abundant published works, beginning with Chomsky's Syntactic Structure in 1957. This approach treats a language as a mathematical object, but it has failed to account for language-using processes of human beings. A new approach, sometimes called "cognitive" or "computational" (cf. Winograd 1983) is more concerned with ordinary language-using processes. The grammatical sketch of Korean is based on a version of this new approach.

Korean is a typical SOV language. It preposes modifiers and postposes markers. Along with Chinese and Japanese, Korean is a topic-prominent language.

In this study, we classify parts of speech on the formal syntactic basis of positional distribution (Bloomfield 1933). As will be observed later in this section, we treat nominal forms which are derived from verbs or adjectives as nouns, since they occur in nominal positions. Also, we describe adjectives separately from verbs since they occur in different syntactic positions in a sentence, although they fall under the morphological form class of verbs in Korean since they can be conjugated. Unlike other verbs, however, adjectives can modify nouns,
in which case they are not conjugated. They comprise an 'adjective-verb' class. The terms 'nouns,' 'attributive adjectives,' 'predicative adjectives,' and 'verbs' (which will be used throughout this chapter) are employed on the basis of the positions in which they occur in a sentence.

2.1. NOUNS

Korean nouns can be classified into three types--simple nouns, pronouns, and derived nouns. These noun types are discussed in the following sections (2.1.1 - 2.1.3).

2.1.1. Simple Nouns

Korean nouns have no grammatical gender and are not declined for case. Instead, markers attached to nouns show the relationships of the noun to the rest of the sentence (cf 2.9). The Korean noun expresses a wide variety of different lexical meanings, some of which are assigned as semantic properties to each noun, e.g. 'Human Beings', 'Animal', etc (cf. section 5.2).

2.1.2. Pronouns

In this chapter, personal pronouns are included within the noun category as a sub-category called 'pronoun.' The so-called personal pronouns in Korean have been classified according to person and number as shown in Figure 2.1 (cf. Choi 1961:277):
<table>
<thead>
<tr>
<th>Sg.</th>
<th>polite</th>
<th>Pl.</th>
<th>honorific humble</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td></td>
<td>plain</td>
<td></td>
<td>honorific humble</td>
</tr>
<tr>
<td>1st</td>
<td>na</td>
<td>ce</td>
<td>wulitul</td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td>ne</td>
<td>tangsin</td>
<td>caney</td>
<td>nehuy(tul)</td>
</tr>
<tr>
<td>3rd</td>
<td>ii</td>
<td>ipun</td>
<td>ii tul</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kui</td>
<td>kupun</td>
<td>kui tul</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cei</td>
<td>cepun</td>
<td>cei tul</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2.1

The choice of status (honorific, humble) of a personal pronoun is determined by the age or social status of the person spoken to in relation to that of the speaker. In the case of 3rd person pronouns, the choice is determined by that of the person whom the speaker refers to. The three forms of the 3rd person pronoun differ according to spatial distance from the speaker, e.g. ii 'this person,' kui 'that person,' and cei 'that person over there or yonder.'

Korean uses demonstrative adjectives and the bound noun kes 'thing' to refer to an inanimate thing, anaphorically (e.g. ku kes 'that thing').

2.1.3. Derived Nouns

Some nouns are morphologically related to verbs or adjectives. They consist of verbs or adjectives plus a nominalizer, ki or m. The morpheme m is realized as um when occurring after a consonant. Otherwise, it is
realized as m. Since a form ending in ki occurs only in the positions where nominals occur, Park (1969:205) calls this form a 'nominalizing verb-ending.' Martin and Lee (1969:439) call the form ending in ki 'substantive'.

The nominalizers ki and m have different distribution. The nominalizer ki occurs with verb stems (cf. the list of examples), but not with adjective stems. The forms ending in ki occur 1) mostly before predicative adjectives (rarely before action verbs) and 2) especially in the positions of emphasized subject, topic or object. Consider the following examples:

(Nz:Nominalizer)

(1) thaykwento paywu-ki-ka cham elyep-supnita.
   martial art learn-Nz-nSubj very difficult-DEC(Pol.)
   'Learning a martial art is very difficult.'

(2) malha-ki-nun swip-ciman silchenha-ki-nun
   say-Nz-nTop be easy-concessive perform-Nz-nTop
   elyep-supnita.
   be difficult-DEC(Pol.)
   'Although it is easy to say, it is hard to perform.'

The nominalizer m occurs with either verb stems or adjective stems. The derived nouns with m nominalizer can be made from any verb, including the copular.

It appears from the lists below that the two suffixes have different meanings. The use of ki over m can be determined by the meaning of the suffixes. Let us consider some examples of derived nouns with the nominalizer m or ki.
<table>
<thead>
<tr>
<th>verb (stem)</th>
<th>noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>kuli-</td>
<td>kuli-m kuli-ki</td>
</tr>
<tr>
<td>'to draw'</td>
<td>'picture' 'the act of drawing'</td>
</tr>
<tr>
<td>ssu-</td>
<td>ssu-m ssu-ki</td>
</tr>
<tr>
<td>'to write'</td>
<td>'writing' 'the act of writing'</td>
</tr>
<tr>
<td>ca-</td>
<td>ca-m ca-ki</td>
</tr>
<tr>
<td>'to sleep'</td>
<td>'sleep' 'the act of sleeping'</td>
</tr>
<tr>
<td>kku-</td>
<td>kku-m kku-ki</td>
</tr>
<tr>
<td>'to dream'</td>
<td>'dreaming' 'the act of dreaming'</td>
</tr>
<tr>
<td>paywu-</td>
<td>paywu-m paywu-ki</td>
</tr>
<tr>
<td>'to learn'</td>
<td>'learning' 'the act of learning'</td>
</tr>
<tr>
<td>mek-</td>
<td>mek-um mek-ki</td>
</tr>
<tr>
<td>'to eat'</td>
<td>'eating' 'the act of eating'</td>
</tr>
<tr>
<td>el-</td>
<td>el-um el-ki</td>
</tr>
<tr>
<td>'to freeze'</td>
<td>'ice' 'the act of freezing'</td>
</tr>
<tr>
<td>chwu-</td>
<td>chwu-m chwu-ki</td>
</tr>
<tr>
<td>'to dance'</td>
<td>'dancing' 'the act of dancing'</td>
</tr>
<tr>
<td>sal-</td>
<td>sal-m sal-ki</td>
</tr>
<tr>
<td>'to live'</td>
<td>'life' 'the act of living'</td>
</tr>
<tr>
<td>al-</td>
<td>al-m al-ki</td>
</tr>
<tr>
<td>'to know'</td>
<td>'knowledge' 'the act of knowing'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>adjective (stem)</th>
<th>noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>elyep-</td>
<td>elyew-um 'difficulty'</td>
</tr>
<tr>
<td>'be difficult'</td>
<td></td>
</tr>
<tr>
<td>kippu-</td>
<td>kippu-m 'pleasure'</td>
</tr>
<tr>
<td>'be pleased'</td>
<td></td>
</tr>
<tr>
<td>etwuwp-</td>
<td>etwuwp-um 'darkness'</td>
</tr>
<tr>
<td>'be dark'</td>
<td></td>
</tr>
<tr>
<td>hayngpokha-</td>
<td>hayngpokha-um 'happiness'</td>
</tr>
<tr>
<td>'be happy'</td>
<td></td>
</tr>
<tr>
<td>(Note: Some forms such as noph-i 'height', kiph-i</td>
<td></td>
</tr>
<tr>
<td>'depth' or nelhp-i 'width' have the ending i</td>
<td></td>
</tr>
<tr>
<td>instead of m. These forms seem to be fossilized</td>
<td></td>
</tr>
<tr>
<td>idiomatic expressions.)</td>
<td></td>
</tr>
</tbody>
</table>

Even though it is hard to make a single statement about the usages of the nominalizers--ki and m, we could generalize roughly that the nominalized verb form appears more static when it is nominalized with m and more dynamic when nominalized with ki--usually the nominalizer m describes the finished product of that process, while the nominalizer ki describes the act of performing the process. (cf. the gloss of nouns with m and ki in examples).
(3) salangha-ki-nun swip-ciman heyeci-ki-nun elyep-ta.  
    *salangha-m-un   *heyeci-m-un  
    love-ING-nTop be easy-con. leave-Nz-nTop be hard-DEC(N)  
    'Loving somebody is easy, but leaving somebody is hard.'

(4) na-nun paywu-ki-lul cohaha-n-ta.  
    *m-ul  
    I-nTop learn-Nz-nObj like-PreT-DEC(N)  
    'I like learning.'

As observed in examples (3) and (4), loving or leaving somebody, or learning something involves treating the process itself as a noun. Therefore, nominalizer ki, rather than m, is appropriate. The following examples illustrate that the nominalizer m carries the static meaning, for the result of the process.

(5) na-nun ca-m-ul ca-ss-ta.  
    *ki  
    I-nTop sleep-Nz-nObj sleep-PasT-DEC(N)  
    'I got some sleep'

(6) na-nun kkwu-m-ul kkwu-ess-ta.  
    *ki  
    I-nTop dream-Nz-nObj dream-PasT-DEC(N)  
    'I had a dream.'

(7) na-nun hayngpokha-n sal-m -ul sal-ass-ta.  
    *ki  
    I-nTop happy-aAA live-Nz-nObj live-PasT-DEC(N)  
    'I had a happy life.'

The nouns cam 'sleeping,' kkwum 'dreaming,' and salm 'life,' which function as objects, are derived from their verbs ca- 'to sleep,' kkwu- 'to dream,' and sal- 'to live,' respectively. These cognate objects always occur with their underlying verbs from which they are derived as in English I sing a song, I dreamed a dream, or I saw sight.
2.2. Adjectives

Korean adjectives can occur in either of two functions, with two different syntactic positions in a sentence—predicative and attributive. Those termed 'predicative' occur with verbal endings as in (8a), and those used as 'attributive' occur before the modified noun as in (8b).

(8) a. kkoch-un yeppu-ta
    flower-nTop pretty-DEC(N)
    'Flowers/The flower are/is pretty.'

     b. yeppu-n kkoch
        pretty-aAA flower
        'pretty flowers/the pretty flower'

The label aAA indicates that the marker is attached to adjective ('a') stems to form attributive adjectives ('AA'). Attributional adjectives modify a head noun in the formation of a noun phrase. We can classify attributive adjectives into two subsets—morphologically-simple attributive adjectives and morphologically-complex attributive adjectives. An attributive adjective such as yeppu-n 'pretty' consists of two morphemes—yeppu and -n, while attributive adjectives such as say 'new', hen 'used', vele 'several' and modun 'every' are morphologically unitary. Except for this morphological difference, the two subsets share the same characteristics; they are never inflected with respect to tense, honorific system or sentence ending when occurring before nouns.

In the morphologically-complex form of the attributive
adjective, we get the marker -n if the stem ends in a vowel or -un if the stem ends in any consonant other than -l or -p. If the stem ends in -l, this -l is realized graphemically and phonemically as zero. For example, in the attributive form of an adjective kil-0, the final consonant -l is realized as zero before -u, and the marker -n rather than -un is attached, e.g. ki-n yenphil 'a long pencil.' Therefore, morphographemic rule-ordering is involved in forming the attributive adjective kin 'long':

kil 'adjective root'
kil-un 'attributive adjective'
kiφ-un The morphographeme /l/ is realized as $\emptyset$ before -u.
kiφ-φn The -n is attached after stems ending in a vowel.

If the stem ends in p which is immediately preceded by a vowel, not a consonant, we find another morphographemic alternation, between p and w (p-w): the morphographeme p is realized as w if it is immediately preceded by a vowel and followed by -u-; otherwise it is realized as p. (This morphographemic alternation can be said to be 'graphemically conditioned non-automatic alternation).

Examples:
<table>
<thead>
<tr>
<th>stem</th>
<th>predicative</th>
<th>attributive</th>
</tr>
</thead>
<tbody>
<tr>
<td>nappu-</td>
<td>nappu-ta</td>
<td>nappu-n</td>
</tr>
<tr>
<td>chakha-</td>
<td>chakha-ta</td>
<td>chakha-n</td>
</tr>
<tr>
<td>sulph-</td>
<td>sulphu-ta</td>
<td>sulphu-n</td>
</tr>
<tr>
<td>ssek-</td>
<td>ssek-ta</td>
<td>ssek-un</td>
</tr>
<tr>
<td>ccalp-</td>
<td>ccalp-ta</td>
<td>ccalp-un</td>
</tr>
<tr>
<td>kil-</td>
<td>kil-ta</td>
<td>ki-n</td>
</tr>
<tr>
<td>mel-</td>
<td>mel-ta</td>
<td>me-n</td>
</tr>
<tr>
<td>kanul-</td>
<td>kanul-ta</td>
<td>kanu-n</td>
</tr>
<tr>
<td>cil-</td>
<td>cil-ta</td>
<td>ci-n</td>
</tr>
<tr>
<td>mip-</td>
<td>mip-ta</td>
<td>miw-un</td>
</tr>
<tr>
<td>tep-</td>
<td>tep-ta</td>
<td>tew-un</td>
</tr>
<tr>
<td>chwup-</td>
<td>chwup-ta</td>
<td>chwuw-un</td>
</tr>
<tr>
<td>alumptap-</td>
<td>alumptap-ta</td>
<td>alumptaw-un</td>
</tr>
<tr>
<td>kakkap-</td>
<td>kakkap-ta</td>
<td>kakkaw-un</td>
</tr>
<tr>
<td>nelp-</td>
<td>nelp-ta</td>
<td>nelp-un</td>
</tr>
</tbody>
</table>

(Notice that in the last example nelpta, the consonant ɹ, not a vowel, precedes ꗦ, and the alternation does not occur.)

In Korean, adjectives that are pre-deictic, deictic and numerative have the morphologically-simple form. Examples:

- ce 'over there, yonder'
- i 'this'
- ku 'that'
- han 'one'
- twu 'two'
- ssey 'three'
- net 'four'
- tases 'five'
- yeses 'six'
- ilkop 'seven'

If morphotactically-simple and complex forms occur together in a sentence, the former precedes the latter--
morphotactically-simple adjectives come before complex ones. Among simple adjectives, deictic ones come before numerative ones and among complex adjectives, epithet precedes classi-
fier, just as in English (cf. Quirk and Greenbaum 1973, Davis 1987) as shown in Figure 2.2.

Fig. 2.2.

The **predicative form** of adjectives occurs with sentence endings other than an imperative ending. Adjectives with predicative function are conjugated with respect to the honorific system, tense, and the sentence ending. Some Korean linguists, e.g. Park (1969:121), call them 'descriptive verbs.' (Hor:Honorific, PasT:Past Tense, DEC(Pol):Declarative(Polite))

9. (a) ku sensayngnim-un cham yeppu-si-ess-upnita.
    that teacher-nTop very pretty-Hor-PasT-DEC(Pol)
    'That teacher was very pretty.'

(b) ku yeca-nun yeppu-ess-ta
    that woman-nTop pretty-PasT-DEC(N)
    'That woman was pretty.'

(c) Ku yeca-nun yeppu-ess-upnikka?
    that woman-nTop pretty-PasT-Q(Pol)
    'Was that woman pretty?'
Adjectives can be characterized as follows in their predicative use:

(1) Predicative adjectives occur with sentence endings other than an imperative ending. (ex. yeppu-ta)

(2) Predicative adjectives cannot occur as imperatives. (*kil-ela 'Be long', *ccalp-ala 'Be short')

(3) Unlike ordinary verbs, predicative adjectives cannot be used with aspectual markers, e.g. ko iss (progressive) or peli (perfective). This observation can be made since predicative adjectives are stative in meaning (cf. the aspect section in verb phrases)
   (*ku-nun chakha-ko iss-ta 'He is being nice.'
   *yenphil-i kil-ko iss-ta 'The pencil is being long.')

2.3. Participial Clauses

An embedded clausal-type modifier has been much discussed in studies of Korean syntax (Cook 1968; Ree 1969; I.S. Yang 1972; D.W. Yang 1975; H.B. Lee 1975). However, all of these earlier analyses describe these constructions in terms of relativization, and they are concerned with whether the constructions are the result of a process of 'promotion' followed by 'prodeletion' (Yang 1975) or one of merely a simple deletion (Yang 1972, Lee 1975) followed by subsequent reordering (Park 1981); they are mainly concerned with a supposed deep structure of the constructions. Let us observe an example from Park (1981:345) (PasPapl: Past Participle).

10 (A) thulek-i pat-un kay-ka cwuk-ess-ta.
   truck-nSubj hit-PasPapl dog-nSubj die-PasT-DEC(N)
   'The dog that the truck had hit died.'
Park claims that sentence (10A) is derived from underlying sentence (10B) by undergoing 'deletion' of NP (kay-lul) in (Ba).

(10B) a. thulek-i kay-lul pat-ass-ta.
   truck-nSubj dog-nObj hit-PasT-DEC(N)
   'The truck hit the dog.'

   b. kay-ka cwuk-ess-ta.
      dog-nSubj die-PasT-DEC(N)
      'The dog died.'

Lee and Im (1983) treat a clausal-type modifier as an 'unconjugated clause.' However, this term is not very apt since the embedded clause is inflected with respect to aspect, tense and honorific system. Let us consider the following examples:


(11) a. John-i tani ∼hakyo-nun phakoytoy-ess-ta. ∼n
   b. ∼n
   c. ∼1
   John-nSubj attend -PrePap1 school-nTop be destroyed-
   -PasPap1 PasT-DEC(N)
   -FutPap1
   'The school that John attends was destroyed.'
   attended will attend

(12) a. na-nun ce chayk-ul sa ∼haksayng-ul a-n-ta. ∼n
    b. ∼n
    c. ∼1
    I-nTop that book-nObj buy PrePap1 student-nObj know-
       PasPap1 PreT-DEC(N)
       FutPap1
    'I know the student who buys that book.
    bought will buy

(13) a. pap-ul mek ∼sonyen-i sakwa-to mek-nun-ta. ∼nun
    b. ∼un
    c. ∼ul
    rice-nObj eat PrePap1 boy-nSubj apple-nAdd eat-
    PasPap1 PreT-DEC(N)
The boy who *FutPap1 eats* (is eating) *fate will eat* rice eats the apple, too.'

We term the clausal modifier a 'participial clause', and the marker which comes at the end of the clause a 'participial marker.' Participial clauses combine with following head nouns to form noun phrases. They function as modifiers in the formation of the noun phrases. The markers -n and -l have graphemically conditioned realizations. -N and -l are realized as -un and -ul after consonants, respectively; otherwise as -n and -l.

Let us now explain why we term the embedded clause above a participial clause. As can be seen from the examples above, the tense of an embedded clause is expressed by the marker on the verb form at the end of the clause, i.e. -nun/-n/-l. However, this marker is different from the general tense marker which indicates the tense of a main clause (cf. section 2.5.4). Besides tense, the markers in braces above also indicate that the clause is not an independent clause but is rather like an English nonfinite participial clause as in 'The architect having lunch is my uncle' or 'The money wired from Korea has not arrived yet.' Since each of the embedded clauses in examples (11)-(13) contains a verb phrase which has a significant tense indicator of an embedded clause, and function like adjectives, we call them 'finite participial clauses,' rather
than 'unconjugated' or 'relative' clauses; the verbs in the
embedded clause have the characteristics both of verbs (in
taking participants) and adjectives (in modifying nouns).

Korean participial clauses are quite different from
those of English. In English, those verbs which can have
adjectival past participle forms can be interpreted as
realizations of two sememes (Barkai 1972)—a 'core' lexeme
plus some kind of a modifier, usually one of manner, degree
or instrument. Observe the following examples borrowed from
Barkai (1972:377).

(14) The murdered man had thrown a bomb into the
police station.

But not

(15) *The killed man had thrown a bomb into the
police station.

Likewise, (16) is grammatical but not (17).

(16) The polished instruments lay on the table.
(17) *cleaned

Further (18) is satisfactory, but not (19).

(18) The stolen money was taken from the robbed bank.
(19) *taken

The lexeme 'murder' is a realization of the sememe 'kill'
and another sememe 'intentionally' or 'premeditatedly';
'polish' is the realization of the sememes 'clean' and
'shiny'; 'steal' is the realization of the sememes 'take
something' and 'illegally.' These observations suggest that
the semantically complex verbs are candidates for adjectival
past participle forms in English. However, in Korean, there
are no similar restrictions on the occurrence of a form as
a past participle. Any verb (including cognitive verbs) can occur with a participial ending within a participial clause regardless of semantic consideration.

Moreover, the tense of the English 'participial clause', which Hudson (1973) calls 'reduced relative clause,' is different from that of Korean. As the examples above illustrate, the tense of a participial clause in Korean, unlike English, is realized as the marker that occurs at the end of the participial clause, and it is interpreted as being relative solely to the moment of speaking; it is not interpreted as being relative to the time referred to by the main clause. The participial markers in Korean are

present _nun (The event of a participial clause occurs at the same time as 'speech time' (ST).)
past _n (The event of a participial clause occurred before ST)
future _l (The event of a participial clause will occur after ST)

On the other hand, in English participial clauses, the imperfective aspect is compatible with a present time, which is relative either to the moment of speaking as in (20), or with a past time referred to by the main clause as in (21).

(cf. Hudson 1973:253)

(20) The people living here twenty years ago from now will have had a maid.
(21) The people living here have a maid.

Likewise, the completive aspect in the English past participial clauses is compatible with either a past time before the moment of speaking as in (22), or with a past time
before the one referred to by the main clause as in (23).

(22) Books published before 1990 will avoid additional taxes.
(23) The asparagus grew best in ground sprayed with insecticide.

Thus, in English participial clauses, the 'past versus present' distinction has a different basis from the superficially similar distinction in Korean: in English, the time in a participial form is interpreted as either relative to the moment of speaking or as relative to the time referred to by the main clause, whereas in Korean it is related solely to the moment of speaking.

2.4. Noun Phrases

Noun phrases in Korean, as in other SOV languages, consist of a noun plus, optionally, some preceding modifiers. These modifiers can be either adjectives (cf. 2.2) or adjectival clauses, i.e. participial clauses (cf. 2.3).

2.5 Verb Morphology

A Korean verb stem consists of a root together with zero or more verbal suffixes. The Korean finite verb complex consists of the following positional categories, in the order given: a verb root, voice, aspect, honorific, tense, and mood/conjunction, as illustrated in example (24).

(24) ku pun-un kyengchal-hanthey cap-hi-epeli-si-ess-ta/-ko that person-nTop police-nAg catch- 1 - 2 - 3 - 4 -5/-6 'That (honorable) person was caught by the police.'
1: Passive
2: Aspect
3: Honorific
4: Tense
5: Mood
6: Conjunction

The rest of this section describes each category of verbal suffix by positional categories—Passive, Aspect, Honorific, Tense and conjunction (in a non-final clause) or Mood (in a final clause). Consider the table in Figure 2.3. (The basis of organization will be discussed in the following sections).

```
Vr-Passive-Aspect-Honorific-Tense-Mood
```

<table>
<thead>
<tr>
<th>Positional Categories of Verbal Suffixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>-Ki</td>
</tr>
<tr>
<td>tang-ha</td>
</tr>
<tr>
<td>pat-ci-</td>
</tr>
<tr>
<td>cwung-i</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Fig. 2.3.

Several of the verb suffixes of various categories have vowel alternations -a or -e in initial position.
conditioned by the last grapheme of the preceding stem, e.g. 
-epeli, -apeli for perfective aspect (cf. section 2.5.2), 
-ass, -ess for past tense (cf. section 2.5.4). There have 
been various descriptions of these vowel alternations. 
Martin and Lee (1969), and Park (1969) describe them as 
'infinitive endings', while Eckardt (cf. Choi 1937) views 
them as 'extended stems'. Choi terms them 'euphonic vowels'. 
In the present study, we treat them as properties of certain 
suffixes. For example, the past morpheme -ess is realized 
as -ass or -ess depending on graphemic and morpho-
logical conditions. The details are given below for the 
suffixes that exhibit the alternations.

2.5.1. Morphological Passive Ki

The morpheme Ki expresses a passives function by 
morphological marking of the verb. Some Korean linguists, 
e.g. Pay (1984), treat the realizations -i-/hi-/li-/ki- 
as four different morphological passive markers. However, 
if we understand them in terms of morphological properties, 
we see that they are not four different passive markers but 
four different realizations of the same morpheme, which can 
be labelled Ki. This Ki marker has graphemically 
conditioned realizations. If the last element of the verb 
stem is a vowel, Ki is realized as i; if the last 
consonant of the verb base is a stop, the Ki is realized 
as hi; if it is a lateral l, the Ki is realized as li; 
if it is a non-stop consonant, e.g. nasal or fricative, the
realization is -ki-. Examples are:

-\textit{i-}

<table>
<thead>
<tr>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>po-/po-i-</td>
<td>'to see/to be seen'</td>
</tr>
<tr>
<td>cha-/cha-i-</td>
<td>'to kick/to be kicked'</td>
</tr>
<tr>
<td>ssa-/ssa-i-</td>
<td>'to wrap/to be wrapped'</td>
</tr>
<tr>
<td>ssu-/ssu-i-</td>
<td>'to use/to be used'</td>
</tr>
</tbody>
</table>

-\textit{hi-}

<table>
<thead>
<tr>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cap-/cap-hi-</td>
<td>'to catch/to be caught'</td>
</tr>
<tr>
<td>mwut-/mwut-hi-</td>
<td>'to bury/to be buried'</td>
</tr>
<tr>
<td>palp-/palp-hi-</td>
<td>'to tread on/to be trodden on'</td>
</tr>
<tr>
<td>mek-/mek-hi-</td>
<td>'to eat/to be eaten'</td>
</tr>
</tbody>
</table>

-\textit{li-}

<table>
<thead>
<tr>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>kal-/kal-li-</td>
<td>'to whet/to be whetted'</td>
</tr>
<tr>
<td>phal-/phal-li-</td>
<td>'to sell/to be sold'</td>
</tr>
<tr>
<td>ssel-/ssol-li-</td>
<td>'to cut/to be cut'</td>
</tr>
<tr>
<td>ppal-/ppal-li-</td>
<td>'to suck/to be sucked'</td>
</tr>
</tbody>
</table>

-\textit{ki-}

<table>
<thead>
<tr>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>an-/an-ki-</td>
<td>'to hug/to be hugged'</td>
</tr>
<tr>
<td>ssis-/siss-ki-</td>
<td>'to wash/to be washed'</td>
</tr>
</tbody>
</table>

We can summarize the realizations of the morphological passive marker \textit{KI} as shown in Figure 2.4.

\[
\text{Morphological Passive KI} \begin{cases} \text{Vowel:-i-} \\ \text{Cons.} \begin{cases} \text{Stop:-hi-} \\ \text{Lateral:-li-} \\ \text{Otherwise:-ki-} \end{cases} \end{cases}
\]

Fig.2.4.

2.5.2. Aspect—Perfective

Aspect is one of the less well described areas in the study of Korean. Youn (1986) used the term 'aspect' as one verbal category of Korean, but she did not discuss this category beyond naming the term. Koreanaspectual meanings are expressed by the simple marker \textit{peli} (perfective) and the complex markers \textit{-ko iss/-cwung i} (imperfective). This section deals with the simple morphological marker \textit{peli}
only, and the complex aspectual markers are discussed in section 2.8.

_Peli_- as an independent verb means 'to throw away,' 'to discard.' As a verbal suffix on the stem of an action verb, it denotes 'finality', 'termination' or 'completion' of the action of the verb. Examples: (Perf:Perfective)

(25) nay-ka ku congí-lul peli-ess-ta.
    I-nSubj that paper-nObj throw-PasT-DEC
    'I threw away that paper.'

(26) a. ku yeca-nun ka- peli - ess-ta.
    that woman-nTop go-Perf -PasT-DEC(N)
    'She's gone away.'

    b. nameci -lul ta mek -epeli - ess -ta.
    left overs-nObj all eat -Perf -PasT-DEC(N)
    'I have eaten up all the left overs.'

    c. nay tongsayng-i ku selyu-lul peli-epeli-ess-ta.
    my younger sister-nSubj that paper-nObj throw-Perf-PasT-DEC(N)
    'My sister has thrown away that paper.'

The forms _peli_ and _epeli_ in sentences (26a), (26b) and (26c) are different realizations of the same morpheme, _peli_, which is realized as _peli_ after the grapheme _a_ (e.g. _ka-peli-ta_ 'went away'); the morpheme _peli_ is realized as _apeli_ after the grapheme _o_ (e.g. _sso-apeli-ta_ 'finished shooting'); otherwise, it is realized as _epeli_ (e.g. _peli-epeli-ta_ 'threw away', _cwu-epeli-ta_ 'gave away', _mek-epeli-ta_ 'finished eating'). In contrast to _peli_ functioning as a main verb in the sentence (25), the marker _peli_ in the sentences (26a) and (26b) gives the meaning of 'completeness.' Example (26c) serves as a good example to demonstrate that the _peli_ has
dual function: the first \_peli in (26c) functions as the lexical main verb, and the second \_epeli in (26c) functions as an aspectual marker. The clauses (25) and (26b) can be shown in terms of stratal description as follows:

(For 25)

S: Process (peli 'throw away')
   P1 (nay 'I')
   P2 (DEIC ku 'that'
      PPART congj 'paper')
   Tense (ss 'past')

L: nay-nSubj ku congj-nObj peli-Past-SE.

M: nay-ka ku congj-lul peli-ss-ta.

G: nayka ku congj-lul peliessta.

(For 26b)

S: Process (mek 'eat')
   P2 (nameci 'left overs')
   Aspect (peli 'perfective')
   Tense (ss 'past')
   Extent (ta 'all')

L: nameci-nObj ta mek-Perfective-Past-SE

M: nameci-lul ta mek-peli-ss-ta.

G: namecijlul ta mekepeliessta.

As the representation at the sememic level illustrates, the morpheme peli in (25) and (26b) has totally different functions.

The aspectual marker \_peli is restricted in syntactic distribution. It cannot be used in the non-final clause as illustrated in (27c) and (28c). It occurs only in the final clause. The meaning of perfective aspect in a nonfinal
clause is signaled by the past tense _-ss_. (cf. Choi 1937, 
Nam 1978, and Lee and Im 1983). Consider the following 
examples borrowed from Lee and Im (1983:183-84).

(27)  
      he-nTop money-nObj hide-ceasing wife-nAg   be found  
      -Past-DEC(N)  
      'He was found by his wife while he was hiding some money.'

   b. ku-nun ton-ul swumki-ess-taka pwuin-hanthey  
      he-nTop money-nObj hide-Past-ceasing wife-nAg  
      tulkhi-ess-ta.  
      be found-Past-DEC(N)  
      'Having hidden some money, he was found by his wife.'

   c. *ku-nun ton-ul swumki-epeli-taka pwuin-hanthey  
      tulkhi-ess-ta.

(28)  
   a. tunglokha-le ka-taka salam-i manh-ase  
      register-vPur go-ceasing people-nSubj be crowded-vRe  
      tolao-ss-ta  
      come back-Past-DEC(N)  
      'I came back from registration since there were so  
      many people.' (The speaker did not finish going there,  
      he saw the long line from the distance)

   b. tunglokha-le ka-ss-taka salam-i manh-ase  
      register-vPur go-Past-ceasing people-nSubj be crowded-vRe  
      tolao-ass-ta.  
      come back-Past-DEC(N)  
      'I went to register, but I came back since there were  
      so many people.'

   c. *tunglokha-le ka-peli-taka salam-i manh-ase  
      tolao-ass-ta.

In Korean, the past tense marker _-ss_ in non-final clauses, 
as in (27b) and (28b) signals that the event _swumki_- 'hide' 
or _ka_- 'go' is completed. On the other hand, in sentences 
(27a) and (28a), the event is not accomplished--the agent
gives up the action on the way for whatever reason. If the past tense marker -se comes in the final or non-embedded clause, it indicates that the event occurred before the speech time (cf. 2.5.4).

2.5.3. Honorific Marker

Korean has three syntactic devices to express the speaker's deference towards the person referred to or being addressed—(1) a morphological device, (2) a lexical device and (3) a mood device. (For the mood device see section 2.5.5). Korean employs the suffix si to represent the speaker's deference to the referent of the subject of the sentence (cf. S. Kuno and Y.J. Kim 1985). The morpheme si has graphemically conditioned realizations; after consonants it is realized as usi, otherwise as si. When the morpheme -si occurs with other suffixes such as passive, aspect, tense and mood in the same verb, the honorific suffix -si always precedes tense and mood suffixes and follows aspect and passive suffixes, as indicated in Figure 2.5.

\[
\text{Vr-Passive-Aspect-Honorific-Tense-Mood}
\]

Fig. 2.5.

In an inflected form, the honorific suffix is not used if the subject in the sentence is inferior to the speaker. In
Korean, the speaker never honors himself, that is, the suffix -si in a verbal form does not occur when the subject and/or the topic is the speaker. The plain verbal form and the honorific one are represented in (29a) and (29b), respectively.

(29) a. apeci-ka mwul-ess-ta.  
father-nSubj ask-Past-DEC(N)  
'The father asked.'

b. apeci-ka mwul-usi-ess-ta.  
father-nSubj ask-Hor-Past-DEC(N)  
'The father asked'

2.5.4. Tense

Korean tenses are indicated by the morphemes -н (present), -ss (simple past), -ssess (remote past), and -keysə (future). This section describes each of these morphemes in terms of its realization and function. Since these morphemes occur paradigmatically in a verb form, a general statement can be made as follows: Tense suffixes come after passive, aspect, honorific suffixes and before the mood suffix (cf. Fig. 2.5).

The morpheme н has graphemically conditioned realizations--н and nun. It is realized as nun after consonants, otherwise as н. The predicate with the morpheme н is said to have the meaning of 'present' time. The function of н can thus be described as 'Present Tense.' The meaning of present tense in Korean, however, is not limited to the time of situation at a single place. It is
relatively rare for a situation to coincide exactly with the present time. Situations of this rare type do, however, occur, and of course the present tense is an appropriate form to use in locating them temporally. One set of examples falling under this rubric would be such exclama-
tions as aya 'ouch' or cekilal 'damn!', where the situation is simultaneous with the present moment. However, exclamations contain no mark of tense and no syntactic structure, and situations of this kind are rare. A more characteristic use of the Korean Present Tense n is referring to situations which occupy a much longer period of time, including future situation.

For examples:

'Universal Time' (Helz 1979, Nam & Ko 1985)
(30) Han Kang-un Seoul-lo hulu-n-ta.
    Han river-nTop Seoul-nDir flow-PreT-DEC(N)
    'The Han river flows into Seoul.'

'Habitudal'
(31) ku-nun mayil achim khephi-lul masi-n-ta.
    He-nTop every morning coffee-nObj drin-PreT-DEC(N)
    'He drinks coffee every morning.'

'Future'
(32) na-nun mayil pusun-ulo ka-n-ta.
    I-nTop tomorrow Pusan-nDir go-PreT-DEC(N)
    'I go to Pusan tomorrow.'

As can be seen from the examples above, the general uses of Korean present tense go beyond the restricted range. In sentence (30), a fact or truth which does not change over a period of time can be represented with the present marker -n; and in example (31), the present tense is also used with a meaning of habitual aspect. Sentences with habitual
meanings refer not to a sequence of situations recurring at intervals but rather to a habit, a characteristic situation that holds at all times. If we describe this in terms of Smith's (1978) notions, in habitual sentences a particular 'Event Time' is not specified. In example (31), the frequency adverbial 'every morning' is interpreted as Reference Time; moreover, the present tense -n may well continue beyond the present moment as in example (32). In the sentence, na-nun cikum nay nonmwn i cang-ul ssu-n-ta 'I am writing chapter two of my dissertation,' it is indeed true that the situation holds at the present moment, but it is not the case that the situation is restricted only to the present moment. Even if the sentence above is represented with the present tense, our knowledge of the world gives a clue that work on a chapter of a dissertation requires a considerable span of time. Thus, the present tense can refer to future events.

From these examples, it appears that our real world knowledge enables us to decide whether a situation is literally to be located just at the present moment or over a period encompassing the present moment. If a user gets as an output the conceptual representation that includes present tense, the interpretation of it depends on his knowledge--it can be interpreted as universal, habitual or future time with the aid of accompanying adverbial expressions, together with real world knowledge as represented in the conceptual system.
The morpheme ss has realizations which depend on the final element of a verb stem. It can be realized as -ass, -ess, or -yess. Although Martin and Lee (1969), and Park (1969) call the vowel alternation among -a-, -e-, and -ye an 'infinitive ending,' we treat them as properties of the past tense suffix. Martin and Lee (p.127) suggest that since we cannot find general statements for the vowel alternations we had better memorize the form outright for each verb. However, we can find some regularities for at least some stems that end in a vowel, as follows:

(a) The past morpheme -ss is realized as -ess after stems ending in u or i.
  
cwu-  cwu-ess  'give'
twu-  twu-ess  'place'
swi-  swi-ess  'rest'
masi-  masi-ess  'drink'
kitali-  kitali-ess  'wait'
kaluchi- kaluchi-ess  'teach'
(Note: If the stem ends in i, the vowel sequence i-e can be optionally realized as the graphemes ye ('vowel contraction'). We can interpret this phenomenon as follows: the Korean morphemic level does not welcome a vowel sequence broken by a morpheme boundary, while it welcomes a vowel sequence without an internal morpheme boundary (ex. ai 'baby'). Therefore, the vowel sequence i-e at the morphonic level is realized as the graphemes ye at the graphemic level).

(b) For some stems, the past morpheme -ss is realized as -ess if the stem ends in u and the vowel -u is realized as zero.
  
khu-  kh-ess  'be big'
ssu-  ss-ess  'write'
aphu-  aph-ess  'be sick'
ku-  k-ess  'to draw a line'

(c) The past morpheme -ss is realized as -ass if stems
end in _o.
  o-    o-ass  'come'
  po-   po-ass  'see'
  ko-   ko-ass  'stew'
  sso-  sso-ass  'shoot'

(Note: The vowel sequence o-a at the morphonic level
     is realized as the graphemes wa at the
     graphemic level).

(d) After the verbalizing suffix _ha, _ss is
    realized as _vess.

<table>
<thead>
<tr>
<th>Noun</th>
<th>Verb</th>
<th>Past</th>
</tr>
</thead>
<tbody>
<tr>
<td>kongpwu</td>
<td>kongpwuha-</td>
<td>kongpuha-yess</td>
</tr>
<tr>
<td>wuntongha-</td>
<td>wuntongha-yess</td>
<td>'exercise'</td>
</tr>
<tr>
<td>panucil</td>
<td>panucilha-</td>
<td>panucilha-yess</td>
</tr>
<tr>
<td>sayngkakha-</td>
<td>sayngkakha-yess</td>
<td>'think'</td>
</tr>
</tbody>
</table>

(e) Except as otherwise specified (i.e. if stems end in
    a or e), the past morpheme _ss is realized as _ss.
  ka-    ka-ss  'go'
  sa-    sa-ss  'sell'
  pha-   pha-ss  'dig'
  se-    se-ss  'stand'

On the other hand, if a stem ends in a consonant, it is
hard to formulate a consistent description. We list verbs
as type I or II, the past tense morpheme _ss being realized
as _ess and _ass, respectively.

Type I    | Type I
--------|--------
 mek-    mek-ess  'eat'
 cwuk-   cwuk-ess  'die'
 ip-     ip-ess  'wear'
 nelp-   nelp-ess  'be wide'
 pis-    pis-ess  'comb'
 iss-    iss-ess  'exist'
 eps-    eps-ess  'do not exist'
 pes-    pes-ess  'take off'
 mel-    mel-ess  'be far'
 kil-    kil-ess  'be long' (in length)
 mantul- mantul-ess  'make'
Type II  cop-  cop-ss 'be narrow
noph-  noph-ass 'be high
pokk-  pokk-ass 'roast (beans)
noh-  noh-ass 'put
al-  al-ass 'know
sal-  sal-ass 'live

(Note: Even though Park (1969) treats wess as another realization of -ss (of course he does not use this term), as mentioned before in the adjective section (see 2.2), -w is the result of morphographemic realization. Hence, the Adj. stem swip- 'be easy' is realized as swi-wess 'was easy' when the past marker is attached. The past form swi-wess should be described as swi-ess, not as swi-wess.)

The predicate with the morpheme -ss designates any action or description which has been finished before 'Speech Time.' There are two past tenses in Korean--Simple Past and Remote Past, which Martin and Lee (1969:164) call 'past-past.' Simple past tenses in Korean are formed by attaching the suffix -ss: remote-past tenses are formed by doubling the past morpheme -ss, i.e. -ss-ss. The second past suffix -ss is realized as -ess, and thus the graphemic realization of Remote Past is -ssess.

The simple past in Korean can mean a single point prior to the present moment, or an extended time period prior to the present moment. Its interpretation is often aided by an accompanying adverb in a sentence. For examples:

(33) na-nun ku-lul ecey tases si - ey mana-ss-ta.
    I-nTop he-nObj yesterday 5 o'clock-nAd meet-PasT-DEC(N) 'I met him yesterday at 5 o'clock'

(34) cikumccaci ku pyeng-ul chilyoha-l su eps-ess-ta.
    so far that disease-nObj cure-can Neg-PasT-DEC(N) 'Up to this point, the disease has been incurable.'
In the sentence (33) the simple past tense sets the situation to a single point, whereas in the sentence (34), the past tense sets the situation to the span.

However, if the single past tense _-ss is used with some verbs of cognition, such as sayngkak-ha- 'think', al- 'know;' or mit- 'believe', the interpretation can be different. The simple past with these verbs includes the present meaning. For examples:

(35) A. ne-nun swunhi-ka ney cokha i-n kes-ul al-ass-ni?
    you-nTop Swunhi-nSubj your niece be-PrePapl know-PasT-
    Q(N) 'Did you know that Swunhi is your niece?'

    B. ung, al-ass-e.
    yes, know-pasT-DEC(P1n)

(36) na-nun onul-i ku-uy sayngil i-lako
    I-nTop today-nSubj he-nPoss birthday be-Quotative

    sayngkak-ha-yess-ta.
    think-PasT-DEC(N)
    'I thought today is his birthday.'

In the examples (35) and (36), we recognize that the situation started in the past and still continues. In sentence (35B), the verb al-ass 'knew' contains the present interpretation; I still know that Swunhi is my niece. Also in sentence (36), the verb sayngkak-ha-yess 'thought' has the meaning of present; I started thinking about his birthday and I am still thinking about it.

The remote past denotes an action which was done or happened a relatively long ago, or a description which ended a relatively long time ago, as in (37). The remote past is
also used to indicate the more remote of two or more past actions as in (38):

(37) caknyen-ey na-nun mwuchek aph-ess-ess-ta.
    last year-nTemp I-nTop very be sick-RPasT-DEC(N).
    'Last year, I was very sick.'

(38) chelswu-nun ecey Pusan-ulo ka-ss-ko,
    chelswu-nTop yesterday Pusan-nDir go-PasT-CONJ

    na-nun cinan tal-ey ka-ss-ess-ta.
    I-nTop last month-nTemp go-RPasT-DEC(N)
    'Chelswu went to Pusan yesterday, and I had been there
    last month.'

The morpheme keyss 'future' has a single realiza-
tion keyss. There has been some dispute on the function
of keyss, which can be interpreted either modally or as a
future tense. The interpretation depends on the subject and
the absence/presence of tense or aspect marker in the
sentence. Consider the following examples.

(39) a. na-nun nayil hakkyo-lo ka-keyss-ta.
    I-nTop tomorrow school-nDir go-FutT-DEC(N)
    'I will go to school tomorrow'

    b. na-nun taum hakk-ey yenge-lul thaykha-keyss-ta.
    I-nTop next semester-nTemp English-nObj take-FutT-DEC(N)
    'I will take English next semester'

(40) a. ne-nun mwues-ul ha-keyss-ni?
    you-nTop what-nObj do-FutT-Q(Fln)
    'What are you going to do?'

    b. tangsin-un nayil eti-lo ka-si-keyss-upnikka?
    you-nTop tomorrow where-nDir go-Hor-FutT-Q(Pol)
    'Where would you like to go tomorrow?'

(41) a. ku ai-nun kwaynchan-keyss-upnikka?
    that baby-nTop be OK-FutT-Q(Pol)
    'Do you think the baby will be Ok?'

    b. kim sensayngnim-i cip-ey kye-si-keyss-upnikka?
       kim teacher -nSubj home-nLoc be -Hor-FutT-Q(Pol)
       'Do think the teacher Kim will be at home?'
If the marker -keyss is used with the first person pronoun na or ce (polite), it indicates the speaker's intention for the future. Also, if -keyss is used in an interrogative form with second person as in (40) and (41), the speaker is asking the addressee about his intention or opinion of the future. In the absence of other marks of tense and aspect, keyss is interpreted as a future time. The sentence nanun kakeyssta, which has no reference time (e.g. adverbs) and aspect, is interpreted as 'I am about to leave.' On the other hand, if the subject or the topic of the sentence is other than the speaker or the addressee, the declarative sentence which includes -keyss denotes an opinion or presumption about the subject or the topic regardless of any specific tense as in (42)-(44). These sentences are borrowed from Lee and Im (1983:77).

(42) kyay-tul-un cikum hanchang sinnakay nol-ko iss-they-nPl-nTop now very hard play-Prog-keyss-ta.
    Modal-DEC(N)
'At this time, they will probably be playing very hard.'

(43) Seoul-un cikum cham chwup-keyss-ta.
    Seoul-nTop now very be cold Modal-DEC(N)
'Seoul is probably very cold at this time.'

(44) i thokki-nun ecey cwuk-ess-keyss-ta.
    this rabbit-nTop yesterday die-Past Modal-DEC(N)
'(It might have been yesterday that the rabbit died.)'

As observed in sentences (42)-(44), -keyss does not represent any future meaning; it is used with the progressive
(cf. 2.8) as in (42) and with the past tense as in (44). Rather than representing future meaning, the morpheme -keyss gives the meaning of 'prediction' or 'intention.' That is, in the presence of marks of tense or aspect, keyss is interpreted as modal, i.e. prediction or intention. Therefore, we can represent its dual function as shown in Fig. 2.6.

![Diagram](keyss)

**Fig. 2.6.**

We can say that the dispute on the morpheme -keyss (i.e. future tense/modal) comes from the confusion of the same syntactic position--after the honorific and before the mood suffix. The people who exclude the form -keyss from the rank of Korean tense (e.g. Lee and Im 1983) seem to pay more attention to the modal function while the people (e.g. Choi 1937, Nam and Ko 1985) who include the suffix keyss in the tense system seem to pay more attention to the tense function. Lee and Im seem to be influenced by English linguists Leech (1971:52) and Quirk et al (1972:87) who also exclude future tense from the English tense system. Leech and Quirk et al suggest that English future tense is not inflected and refers to the world that has not yet
happened ('irrealis').

2.5.4. Mood

Mood is one of the inflectional verbal categories in Korean. Mood occurs in sentence-final position and each mood marker can be said to be the realization of two sememes in Korean (portmanteau realization). The mood suffix indicates not only the kind of sentence (e.g. declarative, interrogative, etc.) but also the level of speech (e.g. plain, polite, etc.). Values for the level of speech and for mood are simultaneously realized by the sentence-final suffix. The mood sememe indicates the speaker's deference toward the addressee, while the honorific sememe si expresses the speaker's deference toward the referent of the subject of the sentence (cf. section 2.5.3). Consider the following examples:

(45) na-nun hakkyo-lo ka-ss-upnita.
    I-nTop school-Loc go-PasT-DEC(F)
    'I went to school.'

(46) sensayngnim-i o-si-n-nta.
    teacher-nSubj come-Hor-PreT-DEC(N)
    'The teacher is coming.'

(47) sensayngnim-i o-si-n-pnita.
    teacher-nSubj come-Hor-PreT-DEC(F)
    'The teacher is coming.'

In sentence (45), the speaker expresses his deference toward the addressee (the speaker never honors himself in Korean), while in sentence (46) the speaker expresses his deference toward the referent of the subject sensayngnim 'teacher.'
If the speaker wants to express deference toward both the addressee and the subject (so-called 'superpolite.' cf. Youn 1986), he uses both the honorific marker si and the formal sentence ending as in sentence (47).

The realizations of mood can be summarized as follows on the basis of the kind of sentences and level of speech. (cf. Martin and Lee 1969, Youn 1986, Park 1969)

<table>
<thead>
<tr>
<th></th>
<th>Declarative</th>
<th>Interrogative</th>
<th>Imperative</th>
<th>Exclamatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>-ta</td>
<td>-ni</td>
<td>-la</td>
<td>-nunkuna</td>
</tr>
<tr>
<td>Polite</td>
<td>-yo</td>
<td>-yo</td>
<td>-o</td>
<td>-kunyo</td>
</tr>
<tr>
<td>Formal</td>
<td>-pnita</td>
<td>-pnikka</td>
<td>-kulye</td>
<td>-pnitakulye</td>
</tr>
</tbody>
</table>

Fig. 2.7.

Note: The graphemic yo, which functions as either declarative or interrogative (polite) sentence ending, has two intonationally different forms distinguished by prosodic elements in spoken language.

2.6. Passives beyond -Ki

Section 2.5.1 describes about the morphological passive. The passive, as has often been supposed, is not a consistent construction across languages. Passives and passive-like constructions allow for a choice of either of two participants as subject of a clause. If we take a broader view of passive, as a device for allowing P-2 (the second participant) to be the subject, we come to realize that we
need to extend the domain of passives to cover tang-ha-/tway-, pat- constructions, and the marker ci-. The tang-ha-/tway- and pat- constructions occur only with verbs derived from nouns:

<table>
<thead>
<tr>
<th>Noun</th>
<th>Active Verb</th>
<th>&quot;Passive&quot; Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>salang</td>
<td>salang-ha-</td>
<td>salang-pat-</td>
</tr>
<tr>
<td>'love,'</td>
<td>to love,</td>
<td>to be loved'</td>
</tr>
<tr>
<td>conkyeng</td>
<td>conkyeng-ha-</td>
<td>conkyeng-pat-</td>
</tr>
<tr>
<td>'respect,'</td>
<td>to respect,</td>
<td>to be respected'</td>
</tr>
<tr>
<td>chingchan</td>
<td>chingchan-ha-</td>
<td>chingchan-pat-</td>
</tr>
<tr>
<td>'praise,'</td>
<td>to praise,</td>
<td>to be praised'</td>
</tr>
<tr>
<td>chepho</td>
<td>chepho-ha-</td>
<td>chepho-tang-ha-/chepho-tway-</td>
</tr>
<tr>
<td>'arrest,'</td>
<td>to arrest,</td>
<td>to be arrested'</td>
</tr>
<tr>
<td>hisayng</td>
<td>hisayng-ha-</td>
<td>hisayng-tang-ha-/hisayng-tway-</td>
</tr>
<tr>
<td>'sacrifice, to sacrifice,'</td>
<td>to be sacrificed'</td>
<td></td>
</tr>
<tr>
<td>chimlyak</td>
<td>chimlyak-ha-</td>
<td>chimlyak-tang-ha-/chimlyak-tway-</td>
</tr>
<tr>
<td>'attack'</td>
<td>to attack</td>
<td>to be attacked'</td>
</tr>
</tbody>
</table>

The syntactic distribution of these additional passive markers is the same as -ki--before the aspectual suffix (cf. 2.5.1). Here, we need to discuss what determines the choice among the three possible expressions--pat-, tang-ha-, and tway-. We find the condition in the meaning of the verb. If the process is 'positive' to P-2, -pat is used; otherwise (i.e. 'not positive') tang-ha- or tway- is used. What then is the condition to choose tang-ha- instead of tway-? This condition is also in the meaning of verb. The processes with tang-ha- or tway- in examples above are 'harmful' or 'negative' to P-2. There is, however, a difference in the degree of harmfulness or negativeness. In the examples, hisayng-tang-ha-/hisayng-tway- 'to be sacrificed,' kangkan-tang-ha-/kangkan-tway- 'to be raped,' and chepho-tang-ha-/chepho-
twoy- 'to be arrested,' the first member of each pair reflects the meaning that the action was premeditated, and that the P-2 is more hurt in terms of negative involvement. On the other hand, the verb twoy- gives the impression that the P-1 did not plan in advance to perform some act on the patient, but the action just happened; it represents a more neutral description in terms of P-2 involvement.

Consider the following examples:

(48)

a. ku cikkong-tul-i hayko-tang-ha-yess-ta.
   hayko-t moy-ess-ta.

b. that workman-nPl-nSubj be laid off-PasT-DEC(N)
   become to be laid off-PasT-DEC(N)

'The workman was laid off (by the boss).'

If one says sentence (48a), the hearer would assume that the workers were laid off because the boss of a factory forced them to for his own benefit, even though the workers had done nothing wrong. The individuals denoted by the subject of (48a) are described as persons who were adversely affected in consequence of the event. In example (48b), however, the hearer would not have the impression that the workers denoted by the subject were adversely treated; it describes that the event just happened for whatever reasons, and now the situation is changed. Consider the following examples.

(49) Hyuntay catongcha-ka swuchwul-t moy-ess-ta.
    swuip-t moy-
    HYUNDAI car -nSubj be exported-PasT-DEC(N)
    imported

'The car 'HYUNDAI' was exported/imported.'

The process with twoy is neutral to P-2—neither positive nor negative. Based on this description of the tang-ha-/
twoy-, pat- constructions, and the morphological marker Ki (cf. 2.5.1), a relational network can be drawn as shown in Figure 2.8.

![Diagram showing derived verbs, positive, negative, neutral, tang, pat, twoy, and ha connections.](image)

Fig. 2.8.

There is also another passive construction, which uses the morpheme -ci-. The morpheme -ci has two graphemically different realizations. It is realized as ci after stems ending -v, otherwise as -eci. Consider the following examples.

I. wumciki-/wumciki-eci- noki-/noki-eci-  'to move/to begin to move' 'to melt/to begin to melt'

II. et-/et-eci- patatuli-/patatuli-eci- nukki-/nukki-eci-  'to get/to be gotten' 'to accept/to be accepted' 'to feel/to be felt'

III. kkay-/kkay-ci-  'to break/to be broken'
mantul-/mantul-eci- 'to make/to be made'
hemwul-/hemwul-eci- 'to demolish/to be demolished'

Pay (1984) excluded ci from the set of passive markers since it represents the meaning of 'beginning' or 'possibility,' especially in group I. However, closer examination reveals that ci marks 'intensity' of the resistance of the patient. For example, imagine a situation in which one tries to move a heavy rock that refuses to move. He tries again and again, and finally it begins to move. In this case, one would say wumcikiecinta 'begins to move.' Therefore, the -ci- marker reflects strong resistance of the patient. -Ci- in the second group also reflects the patient orientation. The -ci- along with the semantics of verbs indicates intense experience of the patient. This strong involvement of the patient can be clearly shown in the example of tullinta 'is listened to' and tulli-eci-nta 'is listened to.' The first one reflects the meaning of 'unconscious listening'—unfocused or background information. For example muzak in the elevator can be described in terms of teulinta. Nobody pays attention to muzak in the elevator. It is just there. However, the second one tulli-eci-nta represents the meaning of 'conscious listening'—focused or foreground information. For example, in ESL class, while students are relaxing, a story can be listened to as a part of communicative approach. This situation can be described in terms of tulli-eci-nta. Even though students are relaxing, they
are paying attention to the story. Whatever the reasons, in contrast to verbs such as 'to send,' 'to give' and 'to touch,' the verbs in the group II reflect the focused patient. Therefore, _ci_ in this group also reflects 'intensity' in terms of the involvement of the patient.

Let us now turn to verbs in group III. The verb 'break', 'make' or 'demolish' clearly shows radical alteration or affectedness of the patient. Therefore, we treat the _-ci_ construction as another passive.

Likewise, if we examine the semantics of the relation that holds between the event and the patient, we find that where a passive is possible, there is a concomitant conception of the P-2 resisting and/or being affected or altered. The passive alternative is possible if the P-2 is directly affected and altered by its involvement in the event. Therefore, we come to the conclusion that we need to expand the domain of Korean passives and include tang-ha-_/twoy-_, pat-_, and _-ci_ constructions as well as the Ki construction (see section 2.5.1).

2.7. Conjunction

The verb suffix _-ko_ occurs at the end of a non-finite clause and conjoins the two predicates, not the clauses. The construction consisting of a verb form ending in _-ko_ indicates the time sequence of the two actions/descriptions—the predicate in the _-ko_ form occurs first. Therefore, the _-ko_ construction can often be
translated as 'and' in English. The analysis is shown in
Figure 2.9. (SE: Sentence Ending)

\[
\text{Predicate - ko} \quad \text{Predicate - SE}
\]

\[
\text{Fig. 2.9.}
\]

For example:

(50) na-nun achim-ul mek-ko haky-o-lo ka-ss-ta.
I-nTop breakfast-n0bj eat-CONJ school-nDir go-PasT-DEC(N)
'I had breakfast and went to school.'

(51) yenge-nun elyw-ess-ko
English-nTop be difficult-pasT-CONJ

\[
\begin{align*}
\text{hankwukmal-un swiw-ess-ta}. \\
\text{Korean-nTop be easy-PasT-DEC(N)}
\end{align*}
\]

'English was difficult and Korean was easy.'

(52) na-nun sensayngnim-i twoy-ess-ko
I-nTop teacher-nSubj become-PasT-CONJ

\[
\begin{align*}
\text{ku-nun saepka-ka twoy-ess-ta}. \\
\text{he-nTop businessman-nSubj become-PasT-DEC(N)}
\end{align*}
\]

'I became a teacher and he became a business man.'

2.8. Complex Aspect-Progressive

The imperfective aspect is indicated by a complex
lexeme, realized as a combination of the two morphemes
ko and iss, with an intervening word-boundary. If
the verb iss 'to exist' immediately follows ko, it
denotes that the action of the verb in the ko form is
in the process of occurring ('midperformance'). This
combination, ko iss realizes the imperfective lexeme.
The imperfective form -ko iss is used frequently, as when an announcer at a sporting event describes an ongoing series of actions.

(53) senswu A-ka kong-ul kaci-ko iss-upnita
player A-nSubj ball-nObj have-Imperfective-DEC(F)
'The player A has the ball.'

Since iss is morphologically a verb stem, tense and/or level of speech ('register') suffixes occur after it. As can be observed in example (53), the imperfective marker in Korean can be used with verbs of 'long periodicity' (Davis 1987), such as salang-ha- 'love,' al- 'know,' nuKKi- 'feel,' or mit- 'believe.' For examples:

(54) na-nun enehak-ul paywu-ko iss-ta.
I-nTop linguistics-nObj learn-Prog-DEC(N)
'I am learning Linguistics.'

(55) ku-nun chayk-ul ilk-ko iss-ess-ta.
he-nTop book-nObj read-Prog-Past-DEC(N)
'He was reading a book.'

(56) swunhi-nun chelswu-lul salangha-ko iss-ess-ta.
Swunhi-nTop Chelswu-nObj love -Prog -Past-DEC(N)
'Swunhi was in love with Chelswu.'

(57) na-nun ca-ko iss-ess-ko
I-nTop sleep-Prog-Past-CONJ
ku-nun chayk-ul il-ko iss-ess-ta.
he-nTop book-nObj read-Prog-Past-DEC(N)
'I was sleeping and he was reading a book.'

The forms -ko and -ko iss are treated as representing separate lexemes; the form -ko as a simple lexeme functions as a conjunction, and the form -ko iss as a complex lexeme functions as the imperfective aspect. The first and the third -ko in example (57) function as a part of the complex lexeme -ko iss, and the second
ko functions as a conjunction. Figure 2.10 shows the relationships between ko and ko iss.

![Diagram showing relationships between ko, ko iss, and iss]

Fig. 2.10.

The semantic content of midperformance or ongoing experience can also be realized as the construction consisting of cwung 'middle' followed by the copular expression i-. Cwung is a bound noun. When cwung occurs with other nouns, it occurs with nouns from which verbs can be made by attaching the verbalizer ha 'to do'. A form in cwung followed by the copula i- means 'something is in progress or is in the midst of a certain action.' For examples:

<table>
<thead>
<tr>
<th>NOUN</th>
<th>VERB</th>
</tr>
</thead>
<tbody>
<tr>
<td>cenwha</td>
<td>'telephone'</td>
</tr>
<tr>
<td>malssum</td>
<td>'talking'</td>
</tr>
<tr>
<td>kongpwu</td>
<td>'studying'</td>
</tr>
<tr>
<td>wuntong</td>
<td>'exercise'</td>
</tr>
<tr>
<td>sayngkak</td>
<td>'thought'</td>
</tr>
<tr>
<td>hwulyen</td>
<td>'training'</td>
</tr>
</tbody>
</table>

(58) cenwha-cwung-i-ta '(He)\'s on the phone.'
    telephone-middle-be-DEC(N)

(59) malssum-cwung-i-ta '(He)\'s talking'
    talking- middle-be-DEC(N)

(60) kongpwu-cwung-i-ta '(He)\'s studying'
    study-middle-be-DEC(N)
(61) wontong-cwung-i-ta '(He)’s exercising'
    exercise-middle-be-DEC(N)

(62) sayngkak-cwung-i-ta '(He)’s thinking'
    thought-middle-be-DEC(N)

(63) hwulyen-cwung-i-ta '(He)’s training'
    training-middle-be-DEC(N)

An alternative way of expressing the constructions

(58)-(63) above are:

(58') cenwha(-lul)-ha-nun cwung-i-ta.
    telephone(-nObj)-do-PrePM middle be-DEC(N)

(59') malssum(-ul)-ha-nun cwung-i-ta.
    talking(-nObj)-do-PrePM middle be-DEC(N)

(60') kongpwu(-lul)-ha-nun cwung-i-ta.
    study(-nObj)-do-PrePM middle be-DEC(N)

(61') wungtong(-ul)-ha-nun cwung-i-ta.
    exercise(-nObj)-do-PrePM middle be-DEC(N)

(62') sayngkak(-ul)-ha-nun cwung-i-ta.
    thought(-nObj)-do-PrePM middle be-DEC(N)

(63') hwulyen(-ul)-ha-nun cwung-i-ta.
    training(-nObj)-do-PrePM middle be-DEC(N)

Based on these constructions (58)-(63'), we can suggest a
possible conceptual representation of (58') as follows:

(Process.engage-in) (P2.phone) (Aspect.Prog)

This representation is supported by the fact that the object marker *lul* can be attached after the participant
cenwha ’phone’ as in (58').

Nam and Ko (1985:310) treat the verbal marker -myense
'while' as progressive, too.

(64) ku-nun pap-ul mek-myense simwun-ul po-n-ta.
    he-nTop meal-nObj eat-while newspaper-nObj-PreT-DEC(N)
    'While he is having a meal, he reads the newspaper.'
The marker -myense is attached to verbs only in the embedded clause. If this marker is attached to a verb stem, the embedded clause functions as an adjunct and gives the English gloss 'while ---'. Therefore, we treat this marker as a kind of adverbial clausal marker, rather than as an imperfective marker.

In summary, progressive aspect in Korean is represented by either the complex lexeme -ko iese or the construction consisting of cwung 'middle' followed by і 'to be.'

2.9. Noun Phrase Function Markers

When a noun is used in a sentence, it usually has some grammatical function, such as subject, object, direct object or topic. These functions are realized by markers occurring as suffixes on the noun. These syntactic markers are morphemes, which express the relationships between the noun phrase and the other constituents in a sentence. These markers may be called enclitics since they function syntactically as independent words in that they occur with the whole noun phrases but are combined with the previous noun into one graphemic word. Each marker is given a label (see the label in parentheses below), in which the first (lower case) letter tells the form class to which it is attached (i.e. n for Noun), and the rest of the label indicates the category of the combination. Thus, nSubj represents the marker that is attached to noun phrases to
form subjects; nTop, the marker that is attached to noun phrases to form topics; nObj, the marker that is attached to noun phrases to form object; etc. Other labels based on the same principle have been used in writing a grammar (cf. The Lexemic Supplement II in Chapter 4). Markers occurring in Korean are shown in Figure 2.11.

| Subject:       | -ka       | (nSubj) |
| Topic:         | -nun      | (nTop)  |
| Object:        | -lul      | (nObj)  |
| Agent:         | -hanthey  | (nAg)   |
| Dative:        | -eykey, -kkey | (nDat) |
| Possessive:    | -uy       | (nPoss) |
| Location:      | -eyse     | (nLoc)  |
| Direction:     | -lo       | (nDir)  |
| Instrument:    | -lo       | (nIns)  |
| Role:          | -lo       | (nRol)  |
| Temporal:      | -ey       | (nTemp) |
| Causative:     | -ey       | (nCaus) |
| Comitative:    | -wa, -hako | (nCom) |
| Additive:      | -to       | (nAdd)  |
| Limitive:      | -kkaci, -man | (nLim) |

Fig. 2.11.

Some of the markers have alternating graphemic realizations. The realizations are summarized below in Figure 2.12.

<table>
<thead>
<tr>
<th></th>
<th>After V</th>
<th>After C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>ka</td>
<td>1</td>
</tr>
<tr>
<td>Topic</td>
<td>nun</td>
<td>un</td>
</tr>
<tr>
<td>Object</td>
<td>lul</td>
<td>ul</td>
</tr>
<tr>
<td>Direction</td>
<td>lo</td>
<td>ulo</td>
</tr>
<tr>
<td>Instrument Role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comitative</td>
<td>wa</td>
<td>kwa</td>
</tr>
</tbody>
</table>

Fig. 2.12.
The indirect object is marked with \texttt{eykey} (neutral) or \texttt{kkey} (polite). Among these markers, the subject marker \texttt{ka} and the topic marker \texttt{nun} signal an important distinction, which is our primary concern in what follows.

Topic in general is 'what the speaker is talking about.' Topic is usually adapted to known information between the speaker and the hearer. Logically, a topic could be any elements that identify a concern which the speaker is addressing. This phenomenon appears especially when constituents other than the grammatical subject (that is, object, time or location) appear initially, as shown in examples (65) and (66).

(65) popcon-un/*ka nay-ka mantul-ess-ta.
  popcorn-nTop/nSubj I-nSubj make-PasT-DEC(N)
  'As for popcorn, I made it.'

(66) ecey-nun/*ka pi-ka nayli-ess-ta.
  yesterday-nTop/nSubj rain-nSubj-rain-DEC(N)
  'Yesterday, it rained.'

Let us now examine the following examples (Park 1981: 235) to see what information the subject marker is adapted to.

  nTop himself-nObj mistreat-progressive-DEC(N)
  'As for Chelswu, he is mistreating himself.'

b. Chelswu-ka cakicasin-ul haktayha-ko iss-ta.
  nSubj himself-nObj mistreat-progressive-DEC(N)
  'Chelswu is mistreating himself.'

Even though both sentences (67a) and (67b) are intelligible, and the choice of \texttt{nun} or \texttt{ka} causes no detriment to the grammaticality of the utterance, there is a con-
siderable semantic difference between these two sentences. Sentence (67a) can be used in the context in which the speaker is talking mainly about Chelswu. On the other hand, sentence (67b) can be used in the broader context which is assumed to be about Chelswu and another person. The speaker attempts to focus on Chelswu in contrast to another person. The fact that the morpheme ka carries the meaning of contrast is also reflected in the following examples.

(68) a. nay-ka mant-un popcorn-nun/i ssek-ess-ta.
    I nSubj make-PM popcorn-nTop/nSubj be rotten-PasT-DEC(N)
    'The popcorn that I made was rotten.'

    b.*nay-nun . . .

(69) a. nay-ka kuli-n kulim-un/i epseci-ess-ta.
    I-nSubj draw-PM picture-nTop/nSubj disappear-PasT-DEC(N)
    'The picture that I drew disappeared.'

    b.*nay-nun . . .

As in sentence (67), either nun or i can be attached to the non-embedded sentences in examples (68) and (69). However, the i, as in sentence (68), provides another meaning--markedness or 'saliency.' Supposing that we erase the first participial clauses, we can easily perceive the difference; i's in both sentences are used to emphasize popcorn 'popcorn' or kulim 'picture'. If i is employed in examples (68) and (69), they mean that

\[
\begin{align*}
\{\text{popcorn}\} & , \text{not the other thing,} \{\text{was rotten.}\} \\
\{\text{the picture}\} & \{\text{disappeared.}\}
\end{align*}
\]

The i is also adapted to 'unknown' information as
well as 'marked' information. When the participial clauses occur with the i in examples (68) and (69) they provide the answers to the questions 'Which popcorn was rotten?' and 'Which picture was lost?', respectively. On the other hand, when the topic marker nun occurs with the participial clause, the embedded clause is used just to mention, not assert, the fact that 'I made the popcorn' or 'I drew the picture.' --Everybody knows the fact. The reason only ka, not nun is appropriate inside the participial clause (cf. (68b) and (69b)) is given as follows. The participial clause, as a relative clause in English, provides information necessary for 'identification.' It informs the hearer of something that he could not identify before. Even though the popcon 'popcorn' or the kulim 'picture' is not within the system prior to its mention, it is identifiable by the participial clause. Therefore, nav 'I' within the participial clause is not known information but a part of new information. We can then understand why only i can occur with nav 'I' in the embedded clauses of examples (70) and (71). Let us look at further examples.

(70) a. Why did you go?

b. yenghi-ka/*nun o-ass-ki ttaymwuney, na-nun/*ka 
   nSubj/nTop come-PasT because I-nTop/nSubj 
   ka-saa-ta. 
   go-PasT-DEC(N) 
   'I went because Yenghi came.'

(71) a. When did you go?

   nSubj/nTop come-PasT when na-nun/*ka ka-ssta.
'I went when Yenghi came.'

(72) a. What would make you go?

   b. yenghi-ka/*nun o-myen na-nun/*ka ka-keyss-ta.
      nSubj/nTop come-cond. I-nTop/nSubj go-FutT-DEC(N)
      'I will go if Yenghi comes.'

These examples illustrate that only ka can occur in the clauses that provide answers to the questions. This phenomenon indicates that only ka can co-occur with new information. The following examples also demonstrate that ka, not nun, reflects new information.

(73) a. nwu-ka/*nun pocon-ul mantl-ess-ni?
      who-nSubj/nTop popcorn-nObj make-PasT-Q(N)
      'Who made the popcorn?'

   b. nay-ka/*nun popcon-ul mantl-ess-ta.
      I nSubj/nTop popcorn-nObj make-PasT-DEC(N)
      'I made the popcorn.'

   c. ni-ka/*nun?
      you-nSubj/nTop
      'You?' (surprise!)

(74) a. Hankwuk-un/*ka mwues-i/*nun manhi na-ni?
      korea-nTop/nSubj what-nSubj/nTop many produce-Q(N)
      'What is the speciality of Korea?'

   b. Hankwuk-un/*ka insam-i/*nun manhi na-n-ta.
      korea-nTop/nSubj ginseng-nSubj/nTop many produce-
      PreT-DEC(N)
      'As for Korea, ginseng is the speciality.'

As is clear from examples (73)-(74), only ka triggers new information. In examples (73a) (74b) and (74c), only ka occurs with nwu 'who' in answers and expressions of surprise. The fact that only ka can provide the meaning of surprise and serve as the answer to questions shows that only ka represents new information. Therefore, the sentence would be unacceptable if the new infor-
mation nay 'I' in (73b) or insam 'ginseng' in (74b) co-occurred with nun.

Up to this point, we have demonstrated that ka does not co-occur with known information, and that it conveys unknown information and a more focus-like property ('saliency', markedness, contrast, or surprise), while nun carries only known information and can be attached to any syntactic constituent that has functions like location or time. Our conclusion receives confirmation from S.J. Hwang's (1987:125) statement that 'The general rule is to use ka for new information. She also states (p.133) that 'the topic particle nun is employed for old information that is known and shared between speaker and hearer, endophorically in verbal context and exophorically in situational context.'

2.10 Clause Structure

2.10.1 Predicate

In Korean, the predicate comprises verbs including objects. Inflected predicative adjectives are treated as verbs. A Korean clause is complete and clear with the predicate alone, e.g. kipta 'be deep,' mekessta 'ate something.' It would not be wrong to add a subject (and/or an object), but it would be superflous (if it is clear). The predicate is thus considered to be the most essential part of a Korean clause. Verbs can be distinguished as
intransitive, transitive and ditransitive, depending on the requirements for participants associated with them.

Intransitive  No Object
Transitive   Optional Object
Ditransitive Optional Direct and Indirect Object

In Korean, like Quichua (Beukema 1975), transitive and ditransitive verbs optionally require object(s).

For Examples:

(75) na-nun ka-n-ta.   (Intransitive)
     I-nTop go-PreT-DEC(N)
     'I go'

(76) A: nwu-ka os-ul sa-ss-ni?
      who-nSubj clothes-nObj buy-PasT-Q(N)
      'Who bought the clothes?'

     B: nay-ka sa-ss-ta.  (Transitive)
        I-nSubj buy-PasT-DEC(N)
        'I did.'

(77) A: nwu-ka halmeni-kkey sakwa-lul cwu-ess-ni?
      who-nSubj grandma-Dat apple-nObj give-PasT-Q(N)
      'Who gave an apple to my grandma?,'

     B: nay-ka cwu-ess-ta.  (Ditransitive)
        I-nSubj give-PasT-DEC(N)
        'I did.'

2.10.2 Topic/Subject

The subject and the topic are marked with the markers, ka and nun, respectively. All subsets of nouns including pronouns and indefinite nouns may serve as the subject of a clause.

(78) elma-ka philyoha-ni?
     how much-nSubj need-Q(N)
     'How much do you need?'

(79) eti-ka aph-si-ppnikka?
     where-nSubj be sick-Hor-Q(Pol)
     'What seems to be the problem?'
However, indefinite nouns cannot function as topic in a clause since, as mentioned in section 2.9, topic always goes with known information. For examples:

(80) *elma-nun philyoha-ni?
    how much-nTop need-Q
    'How much do you need?'

(81) *eti-nun aph-si-pnikka?
    where-nTop be sick-Hor-Q(F)
    'What seems to be the problem?'

2.10.3 Object

There are two kinds of objects, direct and indirect. Both are optional, even if the verb is transitive or ditransitive as shown in examples (80) and (81). The direct object is marked with a syntactic marker jul (nObj) and the indirect object is marked with eykey (neutral) or kkey (polite).

2.10.4 Adjuncts

Adverbs in Korean function as either adjuncts or topics. Adverbs are either lexical or formed from nouns plus some adverbial markers. Lexical adverbs are intensifiers such as cal 'well,' acwu, maywu 'very,' manhi 'much,' etc. Other types of adverbs, e.g. temporal, locative, accompaniment, etc., are formed from nouns plus some adverbial markers. Some adverbial markers, e.g. instrument, location, temporal, limitive and additive, are listed in section 2.9.
When adverbs function as adjuncts, they occur before the verbal complex (unmarked cases). In marked cases, however, (e.g. when adverbs function as topic or they are expressed as afterthoughts or emphasis), they occur at the beginning or at the end of a clause. For examples:

(82) na-nun Pusan-ulo ka-n-ta. (An Unmarked Case)  
I-nTop Pusan-Dir go-PreT-DEC(N)  
' I go to Pusan.'

(83) tosesil-eyse-nun coyonghihay-la. (Topic)  
library-Loc-nTop be quiet - Imp  
' In a library, be quiet.'

(84) na-nun ka-n-ta, Pusan-ulo. (Afterthought or  
I-nTop go-PreT-DEC(N) Pusan-nDir Emphasis)  
' I go, to Pusan.'

2.10.5 The Order of Clause Constituents

The unmarked order of the constituents can be summarized as follows.

(Topic) (Subject) (Adjuncts) Predicate [(Object) + Verb]

For Examples:

Topic + Subject + Adjunct + Predicate
(85) nayil-nun nay-ka hakkyo-eyse Yenghi-lul  
tomorrow-nTop I-nSubj school-nLoc Yenghi-nObj

manna-keyss-ta.  
meet-FutT-DEC(N)  
'Tomorrow I will meet Yenghi at school.'

Topic + Adjunct + Predicate
(86) na-nun nayil Yenghi-lul manna-keyss-ta.  
I-nTop tomorrow Yenghi-nObj meet-FutT-SE

'I will meet Yenghi tomorrow.'
Subject + Predicate [IO + DO + Ditransitive Verb]

(87) Yenghi-ka na-ekey phyenci-lul ponay-ess-ta.
    Yenghi-nSubj I-nDat letter-nObj send-PasT-SE(N)
'Yenghi sent me a letter.'

Variation in the order of the constituents is associated with differences in meaning, e.g. emphasis, afterthought, etc, but since variations are beyond the scope of this study, they are not further discussed.
Chapter 3. MORPHOLOGICAL DECODING

This chapter concerns the first step in the automatic decoding system, morphological decoding, which produces (a) combination(s) of syntactic categories. Regard to this concern, we discuss the dictionaries needed to complete the syntactic combinations. Input strings are checked to determine whether they fit the morphological arrangement that is stated in the form of dictionary entries. The procedure to recognize the input strings and to produce (a) combination(s) of syntactic categories is completed by consulting lower-level dictionaries—the Morphemic Supplements I and II, the Master Dictionary.

3.1. The Morphemic Supplement I

The Morphemic Supplement I is used for implementing the distinction between the graphemic and morphemic strata. There are discrepancies between the graphemic and morphemic strata. That is to say, a graphemic word can be analyzed into strings of morphemes. For example, the Korean graphemic word mwullisiesssupnita 'an honorable person was bitten' is specified as a string of five morphemes:

mwul - Ki - si - ss - supnita.
bite Passive Hor Past DEC (Pol)

The justification for recognizing a morphemic stratum distinct from graphemic stratum is to avoid redundancies.
Let us take an example to see what this means. As explained in the previous chapter, Korean syntax requires markers on nouns, e.g. to mark subject, possessive, object, topic, etc.

**TABLE I**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>'apple'</td>
<td>'player'</td>
<td>'water'</td>
</tr>
<tr>
<td></td>
<td>sakwaka</td>
<td>senswuka</td>
<td>mwuli</td>
</tr>
<tr>
<td>Topic</td>
<td>sakwanun</td>
<td>senswunun</td>
<td>mwulun</td>
</tr>
<tr>
<td>Object</td>
<td>sakwalul</td>
<td>senswulul</td>
<td>mwulul</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Subject</th>
<th>stem - ka/i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
<td>stem - nun/un</td>
</tr>
<tr>
<td>Object</td>
<td>stem - lul/ul</td>
</tr>
</tbody>
</table>

(Alternants whose choice depends on the preceding grapheme automatic alternants, are represented by a slash (/))

The forms in columns A and B in Table I above are partially similar, as are the forms in columns C and D. Table II shows the result of isolating the partial similarities and generalizing on the basis of them. The repetition found in Table I is the result of allowing redundancy. This redundancy is avoided by isolating the markers. Besides (1) segmenting suffixes from stems the Morphemic Supplement I allows us to take care of (2) a partial check of morphotactics on the basis of the number of suffixes that a stem can take ('a suffix-count check'), (3) graphemic alternation, and 4) irregular realizations of morphologically conditioned alternation. These functions
will be discussed in the following sections (3.1.1 - 3.1.4), and the computer data structure for the morphological information is discussed in section 3.1.5.

The Morphemic Supplement I has four sections. Portions of the Morphemic Supplement I for Korean and English are shown in Figure 3.1 for reference (cf. Appendix B for the complete Morphemic Supplement I).

### KOREAN

<table>
<thead>
<tr>
<th></th>
<th>ENGLISH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of Suffixes</strong></td>
<td><strong>List of Suffixes</strong></td>
</tr>
<tr>
<td>I HI LI KI</td>
<td>ES S</td>
</tr>
<tr>
<td>EPELI APELI PELI</td>
<td>ED D</td>
</tr>
<tr>
<td>SI USI</td>
<td>EN</td>
</tr>
<tr>
<td>ASS ESS YESS SS N NUN KEYSS</td>
<td>ING</td>
</tr>
</tbody>
</table>

2. **Irregular Forms**

<table>
<thead>
<tr>
<th>EATEN</th>
<th>EAT</th>
<th>EN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATE</td>
<td>EAT</td>
<td>ED</td>
</tr>
<tr>
<td>WENT</td>
<td>GO</td>
<td>ED</td>
</tr>
<tr>
<td>WON</td>
<td>WIN</td>
<td>ED</td>
</tr>
</tbody>
</table>

3. **Morphotactics**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( )</td>
<td>ROOT</td>
<td>( )</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( )</td>
<td>ROOT</td>
<td>( )</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( )</td>
<td>ROOT</td>
<td>( )</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( )</td>
<td>ROOT</td>
<td>( )</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( )</td>
<td>ROOT</td>
<td>( )</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( )</td>
<td>ROOT</td>
<td>(W P)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. **Homographic Suffix –s**

<table>
<thead>
<tr>
<th>Noun s</th>
<th>Noun Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vi s</td>
<td>Vi 3rd. Sg. Pres</td>
</tr>
<tr>
<td>Vt s</td>
<td>Vt 3rd. Sg. Pres</td>
</tr>
</tbody>
</table>

Fig. 3.1
The segments listed after 1 in Fig. 3.1 are the graphemic realizations of verbal suffixes that can be attached to a verb or a noun in Korean. As described in Chapter 2, a Korean verb consists of a verb base plus one or more suffixes in linear order, and noun phrases contain a marker attached to the head noun that signals the function of the noun phrase in the sentence. The verbal suffixes and noun markers in the Korean Morphemic Supplement I (see Figure 3.1) reflect the following categories, line by line:

Passive (-i, -hi, -li, -ki)
Aspect (-epeli, -apeli, -peli)
Honorific (etc.)
Tense (Past (3), Present (2) Future (1))

The suffixes in the English Morphemic Supplement I reflect the following categories, respectively:

Present 3rd Person Sg. (-es, -s)
Past (-ed, -d)
Past Participle (-en)
Progressive, Present Participle (-ing)

Considering the functions of the Morphemic Supplement I, we now examine each function, one by one.

3.1.1 The lists shown after 2 in each Morphemic Supplement represent the suffix-count check of Korean and English. In formalizing a partial check of morphotactics, the possible numbers, not the type, of suffixes
that can be put after a root are considered. If we consider number of suffixes for a verb and a noun of Korean in terms of a relational network, it can be drawn as as Figure 3.2.

![Diagram of a verb with suffixes and a noun with a marker]

Fig. 3.2

If we disregard the types of suffixes and consider only the number of suffixes that can be preceded by a root, we can interpret the networks above as "for a Korean noun base, only one suffix can be attached and for a verb base, maximally five suffixes can be attached."

**Possibility 1:** Only one suffix can be attached to a stem in the case of nouns.
(ex. nanun, nelul, etc).

**Possibility 2:** Two suffixes can be added to a verb stem--tense/honorific/passive/aspect and SE.
(ex. mek-ess-ta 'ate', mek-nun-ta 'eat', mek-keyss-ta 'will eat', ka-si-ta 'go(polite)', mwul-li-ta 'be bitten', ka-peli-ta 'go away')

**Possibility 3:** Three suffixes can be attached to a verb stem--honorific/passive/aspect, tense and SE.
(ex. ka-si-ess-ta 'went(polite)', mwul-li-ess-ta 'was bitten', ka-peli-ess-ta 'went away', mek-ko iss-ess-ta 'was eating')

**Possibility 4:** Four suffixes can be added to a verb stem--passive/aspect, honorific, tense and SE.
(ex. mwul-li-si-ess-ta
'(an honorable person) was bitten')
ka-peli-si-ess-ta
'(an honorable person) went away')

Possibility 5: Five suffixes can be attached to a verb stem--passive, aspect, honorific, tense and SE. (ex. mwul-li-epeli-si-ess-ta. '(an honorable person) has been bitten')

In English, we can list the four morphemes that can appear as inflectional suffixes after verbal bases or nominal bases. (Graphemically conditioned non-automatic alternants are presented by a colon (\(\_\))).

Possibility 1 (3rd sg. noun plural): /es/ after -st, -sh, -ch, and -x; otherwise /s/.

Possibility 2 (Preterit): /ed/ after -t or -d; otherwise /d/.

Possibility 3 (Participle): /ed/ after -t or -d; otherwise /d/ or /n/ after -e; otherwise /en/.

Possibility 4 (Gerund, Present Participle): /ing/

In the case of English, only one suffix can be attached to a base form, whatever the type of suffix. Whatever the types of suffixes, the decoding system checks how many suffixes are attached to a verb or a noun base in a given input. Thus the morphotactic rules, which are reintroduced for convenience in Figure 3.3, provide a list of the possible number of suffixes that can come after a base (see section 3.1.5 in order to see how information is represented in a data structure).
A: Information about Prefix-counter
B: Information about Prefix graphemic alternation
C: Information about Suffix graphemic alternation
E: Information about Suffix-counter

Fig. 3.3

If we specified the types, not the numbers, of suffixes, we would need to list all possible alternants of the suffixes. For example, even for a type of the past tense, we have to specify four different realizations at the graphemic level: (see section 2.5.4).

root-ass
root-ess
root-yess
root-ss

These four possible cases can be reduced to [ROOT ( ) 1] in our morphotactic rules. This notation means only a single suffix of whatever type comes after a stem. If two suffixes come together, e.g. honorific and past tense, the combined possibilities of suffix types would be eight (honorific realizations 2 \times past tense realizations 4 = 8). However, these eight possibilities also can be reduced to a single statement, [ROOT ( ) 2] in the morphotactic rules. This notation means two suffixes of whatever type are
attached after a root. If there were large numbers of additive suffixes as in Turkish, the possible combinations of additive suffixes would be cumbersome. Therefore, the number that comes at the end of the statement represents the number of suffixes possible in Korean. In English, the number of suffixes that comes after a root is only one. (see Fig. 3.1)

Here, we need to raise a question: if the Morphemic Supplement I checks the number of suffixes, not the types of suffixes, how can we check the order of verbal suffixes in Korean? If the input were *mek-ta-ess (stem-SE-Tense), it should be rejected as deviant morphotactics. The linear order of suffix types is stated in Lexemic Supplement II in which syntactic statements for clauses are contained (cf. Lexemic Supplement II section, 4.2).

3.1.2 The Morphemic Supplement I also handles graphemically conditioned alternation; that is, graphemic alternations of the stem conditioned by the following grapheme, e.g. aleumtap- aleuntaw-un 'beautiful.' The different graphemic realizations of the same morphographeme are specified in the dictionary. In Fig 3.1 the parentheses in the rules in the braces give information pertaining to the graphemic alternation of the same morphographeme. In Korean, as mentioned in Chapter 2, there is graphemic alternation between p and w (p~w).
(See the adjective section (2.2)). The information in the
parentheses means that the morphographeme \( P \) in the root is realized as \( w \) if it is immediately followed by a vowel and otherwise it is realized as \( p \). Therefore, if the alternant \(-w-\) is found in the root of the input, it is the realization of the morphographeme \( P \). So, if the input \textit{alumtaw-un} 'beautiful' (attributive adj.) is given, the grapheme \(-w-\) is related to the realize \(-P-\), and the root form \textit{alumtap-} is found in the Master Dictionary. In the case of English \textit{study}, there can be graphemic alternation between \(-i-\) and \(-y-\) (i~y). The morphographeme \( Y \) is realized as \( i \) if the surrounding environment is \(-es\) or \(-ed\), otherwise it is realized as \( y \). So, the morphotactics for a verb like \textit{study} can be written as

\([\text{STEM (I Y) 1}],\) which means if the input \textit{studi-} is found (after the suffix is detached), the final grapheme \(-i-\) is related to the morphographeme \( Y \), and the stem form \textit{study} is consulted in the Master Dictionary. If there is no graphemic alternation, i.e. if the graphemes of a root are invariant, the parentheses are left empty (see Fig. 3.1).

To make the system general, however, we also consider languages that have prefixes, e.g. English and German. The two columns (A and B) are for information about additive prefixes (see Fig. 3.1). Just as the last two columns (C and D) are for the number of suffixes and for information about graphemic alternation resulting from regressive assimilation, so the first two columns are for the number
of prefixes and for information about the graphemic alternation resulting from progressive assimilation. Since the present chapter does not deal with stems to which prefixes can be attached (e.g. Kor. *am so* 'cow' *sus so* 'bull', Eng. *incomplete, refill*, etc.), however, we leave information about them empty.

3.1.3 The Morphemic Supplement I also makes it possible to take care of Morphologically conditioned alternation. Although irregular forms are not found in Korean, a large number of irregular composites which undergo morphologically conditioned alternation are found in English. There have been various opinions on these forms among structural linguists. For example, several solutions have been proposed for *took* by structuralists. Bloch (1947) suggests that the form /took/ is an allomorph of /take/; Hockett (1954) treats /took/ as a portmanteau representation of /take/ and /-ed/; Gleason (1961) suggests that *took* is *take* plus a replacive morph /u/ ← /ey/; /took/ has also been described to be a discontinuous morph /t--k/ of *take*, plus an infixed allomorph /u/ of /ed/. In the present study, the irregular form is dealt with as a portmanteau realization that /took/ is the portmanteau realization of the morpheme /-ed/ and the morpheme /take/. This treatment is like that of Webster's dictionary. If we consult the dictionary under the entry *took*, it says that *took* is the 'past of take'.
The Morphemic Supplement I handles such irregular composites as the simultaneous realization of the two morphemes. The English Morphemic Supplement I contains the morphologically conditioned alternants and treats each alternant as two morphemes. The group after 3 in Fig 3.1 is a list of these irregular forms. Some examples are listed below.

<table>
<thead>
<tr>
<th>EATEN</th>
<th>EAT</th>
<th>EN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATE</td>
<td>EAT</td>
<td>ED</td>
</tr>
<tr>
<td>WENT</td>
<td>GO</td>
<td>ED</td>
</tr>
<tr>
<td>GONE</td>
<td>GO</td>
<td>EN</td>
</tr>
<tr>
<td>WON</td>
<td>WIN</td>
<td>ED</td>
</tr>
</tbody>
</table>

3.1.4 The last function of the Morphemic Supplement I is to take care of neutralization of morphemes, e.g. the homographic suffix -s. This morpheme has two different functions as shown in Figure 3.4.

![Diagram of plural and 3rd SG. PreT with suffix -s](image)

Fig. 3.4.

Somewhere in our dictionary, we need to include this linguistic information, and the most appropriate place for the present system is the Morphemic Supplement I, in which morphological information is contained. If we want to
decode a sentence such as *The boy sleeps*, we first look in the Master Dictionary and get the syntactic categories for the graphemic words *the* (DET) and *boy* (NOUN). Then, with the aid of the Morphemic Supplement I the graphemic word *sleeps* is analyzed into two morphemes *sleep* and *s*, and syntactic categories are assigned from the Master Dictionary to the morphemes *sleep* and *s* as 'Vi' and 'Pl'/'3rd Sg. PreT', respectively. After the syntactic categories are recognized, there are two parsing possibilities because of the two syntactic categories assigned to the suffix *-s*. This is shown in Figure 3.5.

<table>
<thead>
<tr>
<th>I.</th>
<th>DET</th>
<th>NOUN</th>
<th>Vi</th>
<th>3rd Sg. PreT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the</td>
<td>boy</td>
<td>sleep</td>
<td>s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>II.</th>
<th>DET</th>
<th>NOUN</th>
<th>Vi</th>
<th>Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the</td>
<td>boy</td>
<td>sleep</td>
<td>s</td>
</tr>
</tbody>
</table>

Fig. 3.5.

In this case, it is undesirable to assign dual syntactic categories to the suffix *-s*, and some method of distinguishing the two syntactic categories is required.

As a linguistically valid way, we provide linguistic information in the Morphemic Supplement I, that states "the morpheme *-s* after a verb should be recognized as '3rd Sg. PreT', and the morpheme *-s* after a noun should be recognized as 'Plural.' To make this possible, rather than assigning syntactic categories for the morpheme *-s* in the Master Dictionary, we wait until a syntactic category
of the base form is recognized. After a syntactic category of the base is recognized in the Master Dictionary, it is passed on to the Morphemic Supplement I. The Morphemic Supplement I then assigns syntactic categories to the morpheme -s and determines whether the morpheme has the function 'plural' or '3rd Sg. PreT,' depending on the syntactic category of the base form, which is passed from the Master Dictionary. That is, the machine does not try both syntactic categories '3rd Sg. PreT' and 'Plural.' Before the sentence is passed to the Lexemic Supplement II, which consists of grammatical statements, the syntactic category of the morpheme -s is determined. So, the procedure for decoding the input sentence above The boy sleeps will be as follows:

```
The    boy    sleeps
   MD     MD
 DET   NOUN   sleeps
           MSI
 DET   NOUN   sleep s
       MD
 DET   NOUN   Vi  s
           MSI
 DET   NOUN   Vi  PreT
```

In summary, the function of the Morphemic Supplement I is to decode from graphemic strings to morphemic strings. It does so by taking care of regular and irregular composites and graphemic alternations, and assigning syntactic categories for homographic suffixes, such as the English -s.

3.1.5 The data structure for the Morphemic
Supplement I is designed to store information about morphology—the presence/absence of prefixes/suffixes, graphemic alternation, the prefix-/suffix-count check, and irregular composites. The data structure is shown in Figure 3.6.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>status</td>
</tr>
<tr>
<td>counter</td>
<td>counter</td>
</tr>
<tr>
<td>morphographeme and realized grapheme</td>
<td>morphographeme and realized grapheme</td>
</tr>
</tbody>
</table>

Fig. 3.6.

Column A is information about prefixes and Column B is information about suffixes. The first row ('status') indicates whether the prefix/suffix is present or not, e.g. -ed, -s, etc. in English and -ss, -n, -keyss, etc. in Korean. This indication is represented by boolean values, 0 (absent) and 1 (present). The second row ('counter') is for the checking of the number of prefixes/suffixes. The final row is for graphemic alternation. If there is graphemic alternation between y~i, the morphographeme y is stored first and the realized grapheme i is stored later. Since the present study deals with suffixes only, Column A is left empty. Let us take an example with an entry of the Morphemic Supplement I in order to see how the information is stored. If the machine reads the entry

0 ( ) Root (i y) 1,
which means 'There
is graphemic alternation in suffix between \(i\sim y\),' and 'the
number of a suffix that is attached to the root is 1' (as
in studies), the machine stores the information as
shown in Figure 3.7.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 3.7.

Irregular composites are stored separately in a separate
array. When the machine reads the irregular composite *ate*
and *went* it first stores them as shown in Figure 3.8.

```plaintext
ate  went
```

Fig. 3.8

After the two realized morphemes for this irregular
composite are read, the array in Figure 3.8 is changed as
shown in Figure 3.9.

```plaintext
eat  ed  go  ed
```

Fig. 3.9

This array is passed to the next higher level in order
to deal with the anatactic phenomenon.

Before discussing the functions of the Morphemic
Supplement II, we explain the procedure for going through
the Morphemic Supplement I.

(1) Look in the Master Dictionary first for the given
string of the input. If the string is not found
in the Master Dictionary, then look in the Morphemic
Supplement I.
(The reason for consulting the Master Dictionary first will be discussed later.)

(2) A. Detach the suffix if it is in the group of suffixes listed in the Morphemic Supplement I.
B. If the input does not include any suffixes and is not found in the Master Dictionary, treat it as an irregular form. Look for it in the list of morphologically conditioned alternants.

(3) After detaching (a) suffix(es), check the number of (a) detached suffix(es) to see if it/they fit(s) to the suffix-count of the language under consideration.

(4) The segmented morphemes--bases and suffixes--are carried over to the next higher level dictionary, the Master Dictionary (Korean) or the Morphemic Supplement II (English), in preparation for the next step of decoding.

For instance, let us suppose that one of the input strings is mwullisiessta 'an honorable person was bitten'.
According to the procedure summarized above, we need to consult the Master Dictionary first to see if it contains this graphemic word. Since the Master Dictionary includes only the stem mwul-, however, we do not find the given input string there. As the second step, we search through the Morphemic Supplement I. The suffixes -ta(SE), -ess(past), -si(honorific) and -li(passive) are found in the list of suffixes in Morphemic Supplement I, and they are detached from the graphemic word, mwul-li-si-ess-ta.
As the third step, these four suffixes segmented from the stem are checked to see if they fit the suffix-count check of Korean. The four suffixes and one root are matched to one morphotactic statement, [ROOT () 4] (cf. Fig. 3.1). After this step is completed, as the forth step the segmented morphemes are taken up to the Master Dictionary
to find their matching grammatical symbols in preparation for syntactic decoding.

For English, if one of the input strings is *took*, the procedure is slightly different. An irregular form such as *took* is not found in the Master Dictionary and it does not include any suffix in the list of suffixes either. Therefore, we look for it in the list of irregular forms where we will find it decoded into two morphemes, *take* and *ed*. This decoding is checked against the English suffix-count check in the English Morphemic Supplement I. It matches the entry [0 ( ) ROOT ( ) 1] (cf. Fig. 3.1). Finally, the decoding (*take ed*) is carried over to the next higher level dictionary, the Morphemic Supplement II.

Since we search through the Master Dictionary again after the graphemic words are segmented by going through the Morphemic Supplement I, the question may be prompted why we should also search the Master Dictionary before consulting the Morphemic Supplement I. This procedure is followed in order to avoid analyzing unnecessary segments. If we consult the Morphemic Supplement I first, the Morphemic Supplement I would chop up the bases if a part of them happen to have the same form as the suffixes. For instance, if the Morphemic Supplement I is consulted first, it would segment *-i, -ki* and *-li* as passive suffixes in the process of decoding input sentences that have graphemic words such as *congi 'paper', kongki 'air* or *ppalli 'quickly*', respectively. However, the process involved in
these segmentations is not successful, and it just consumes computational time to perform the unnecessary segmentation. If we perform look-up in the Master Dictionary first, however, a graphemic word which realizes a single morpheme does not have to undergo the extra steps involved in looking through the Morphemic Supplement I. Since we consult the Master Dictionary first, the graphemic words congi, kongki and ppalli are immediately found to be NOUNs and ADV, and they do not have to go through the Morphemic Supplement I. As a result, the extra computational steps are saved, and efficiency is significantly enhanced.
3.2 The Morphemic Supplement II

3.2.1. Treatment of Anataxis in English

The rationale for establishing the Morphemic Supplement II is to treat the phenomenon of anataxis in English. The Morphemic Supplement II deals with anataxis (Lockwood 1972:28, Lamb 1966:22-23) between graphemic and morphemic levels and thus make it possible to take care of lexemes with discontinuous realizations, e.g. be >en, be >ing, as in is taken, is taking, etc. Irregular forms (e.g. ate, went, took, etc.) as well as regular composites are analyzed by the Morphemic Supplement I, and the output of this dictionary is passed up to the dictionary of the next level, called the Morphemic Supplement II. The Morphemic Supplement II then makes it possible to recognize the discontinuous realization of a lexeme as a unit on the higher level. It does so by realizing English verbal inflectional suffixes before bases ('Affix-hopping'). Since Korean has this phenomenon only at the level of phonology, e.g. alhko \ alkho 'to be sick and,' Korean does not need the Morphemic Supplement II for decoding written language. Let us consider the following stratal descriptions:

```
L: Tns Passive Vt
L: Tns be Papl Vt
M2: 3 sg. ed be en take
M1: be 3 sg. ed take en
G: was taken
```

Lexemic Supplement II
Master Dictionary
Morphemic Supplement II
Morphemic Supplement I
What are the advantages of such a realizational treatment? First, we can simplify the grammatical statements for the Verb Phrase, which are in the Lexemic Supplement II (and thus also save parsing time). If we did not treat the anatactic phenomenon somewhere in the dictionaries, we would have to repeat tense for every VI. Tense, however, occurs with the whole verb phrases, not just with the verb. For example, we would have to specify all possibilities that tense can occur as shown in statement I below:

I. \[ \begin{align*} &VP / VI \\
&VI / TVt \text{ NP} \\
&VI / TVi \\
&TVt / Vt \ Tns \text{ (for the tense that occurs after a verb, TVt / Tns Vt e.g. present and past).} \\
&TVi / Vi Tns \\
&TVi / Tns Vi \text{ (for the tense that occurs before a verb, e.g. future).} \\
\end{align*} \]

(Note: TVt and TVi stand for 'Tensed Vt' and 'Tensed Vi', respectively. Since the future, past & present tenses occupy different syntactic positions, we need a separate tense statement for every kind of verb, e.g. Vt, Vi, BIVERB, COPVERB, etc.)

However, these statements can be much simplified by recognizing anataxis. All tenses including the past and the present are put before verbs like the future tense at the deep morphemic level, and the grammatical statement for the tense is needed only once as shown in statement II below.
II. VP / Tns VI
   VI / Vt NP
   VI / Vi

The seven grammatical statements in version I are reduced to
three statements in version II.

Second, by means of the anatactic treatment ('affix-hopping') of bases and suffixes in English, we can easily
deal with discontinuous realizations of lexemes. Without
the deep morphemic representation, discontinuous morphemes
such as be >en or be >ing could not be grouped
together. If we assume that stratum M2 of description II
is missing, it would be more difficult to group be and
en as the passive as shown in stratal description I.

<table>
<thead>
<tr>
<th>Description I</th>
<th>Description II</th>
</tr>
</thead>
<tbody>
<tr>
<td>L: Aux-Be Tns Vt Papal</td>
<td>L2: Tns Passive Vt</td>
</tr>
<tr>
<td>M1: be s sell en</td>
<td>L1: Tns Aux-Be Papl Vt</td>
</tr>
<tr>
<td>G: is sold</td>
<td>M2: s be en sell</td>
</tr>
<tr>
<td></td>
<td>M1: be s sell en</td>
</tr>
<tr>
<td></td>
<td>G: is sold</td>
</tr>
</tbody>
</table>

This example demonstrates that complexity dissolves when
complex structures are treated as involving two different
strata with intervening realizations. With realizational
relationships between two levels, each stratum can be
described as context-free.

To conclude this section, the function of the Morphemic Supplement II is to treat anatactic phenomena in English morphology in preparation for syntactic decoding. As a result of this treatment, the Morphemic Supplement II makes it possible (1) to simplify the grammatical statements in the Lexemic Supplement II and (2) to recognize discontinuous realizations of a lexeme as a unit at the higher level, e.g. be >en, be >ing, have >en, etc.

3.2.2. Computational Representation for Anatactic Phenomenon

In order to represent anatactic phenomenon the procedure of swapping two morphemes is employed. The buffer in Fig. 3.9 passed from the earlier stage of decoding with the aid of the Morphemic Supplement I results in Figure 3.10 by swapping the bases and the suffixes.

```
ed  eat  ed  go
```

Fig. 3.10.
3.3. The Master Dictionary

Along with the format of the other dictionaries (cf. Chapter 1), an entry in the Master Dictionary is also a linear representation of the nection, in the form of the heading and the information. To make the Master dictionary simple, context-sensitive information is dealt with in the Morphemic Supplement I (see section 3.1.), and thus the Master Dictionary assigns syntactic codes without specifying context-sensitive information.

The Master Dictionary specifies two kinds of information, indicated in the entry for each lexical item. The first type is syntactic information; information which specifies one or more of lexical categories, e.g. NOUN, Vt, Vi. The second type information is semantic; an indication of where the signified concept resides in a semantic relational network. For example, if we locate the entry 'apple,' we can get the information NOUN as a lexical category and ED-SOL ('Edible Solid') as its semantic property. If we interpret this information in terms of nections (cf. 1.1), the information NOUN is a label for the tactic function associated with the heading 'apple,' and the semantic property label ED-SOL provides information about its meaning. That is, the heading of the dictionary entry is the Expression, a lexical category such as NOUN, VERB, etc., labels the Tactic Function and the semantic property is part of the Meaning.

The rationale of putting semological information in the
Master Dictionary is that the Master Dictionary, which includes semotactic information in addition to lexical categories, allows us to have simpler dictionary entries and to avoid copying the same lexical symbols. If we included sememic properties for each lexeme symbol in the Sememic Supplement II (which will be explained later in section 5.2), we would have to duplicate all of the morphemic words of the Master Dictionary for inclusion in the Sememic Supplement II. That is, the Master Dictionary allows us to write just

\texttt{apple NOUN Ed-Sol}

where the two identical entries would otherwise be required:

\texttt{apple NOUN}
\texttt{apple Ed-Sol}

Therefore, the general format of the Master Dictionary is as follows:

\begin{verbatim}
Morphemic Word  #, Lexical Category  Semantic Information
\end{verbatim}

'\#' represents how many lexical categories a given lexical item can have. If the morphemic word is \texttt{plant}, the number is marked as '2'--one for \texttt{NOUN} and the other for \texttt{Vt}. This notation device has a for practical motivation. It signals the amount of memory space for each entry (cf. data structure for the Master Dictionary in section 6.2).

Figure 3.11 lists some entries extracted from the Master Dictionary of Korean to illustrate its organization.
KOREAN

<table>
<thead>
<tr>
<th>Morphemic Word</th>
<th>Lexical Category</th>
<th>Semantic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANGEYMWUL</td>
<td>1 NOUN</td>
<td>8</td>
</tr>
<tr>
<td>CAP</td>
<td>1 Vt</td>
<td>10 11 12</td>
</tr>
<tr>
<td>CHANGIN</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>CHELKAPSEN</td>
<td>1 NOUN</td>
<td>5</td>
</tr>
<tr>
<td>COHAHA</td>
<td>1 Vt</td>
<td>5 6</td>
</tr>
<tr>
<td>CWU</td>
<td>1 Vt</td>
<td>8 9</td>
</tr>
<tr>
<td>MWUL</td>
<td>2 NOUN</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Vt</td>
<td>7 8</td>
</tr>
<tr>
<td>CIP</td>
<td>2 NOUN</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Vt</td>
<td>6 7 8</td>
</tr>
</tbody>
</table>

Fig. 3.11.

Semantic information is represented with numbers, which provide a pointer of cross-reference for the information. Detailed information about these properties and the strategy for using numbers is discussed in the description of the Sememic Supplementary Dictionary II (cf. 5.2).

In this section, we have not explained the function of the Master Dictionary in detail. Since its function is related to the Sememic Supplement II, we will return to the discussion pertaining to the Master Dictionary in the Sememic Supplement II section, 5.2. Roughly speaking, the function of the Master Dictionary is (1) to associate an activated morpheme (which is received from the Morphemic Supplement I or from the graphemic representation) with a grammatical category symbol in preparation for the production of a syntactic tree, and (2) to provide semological information needed for semantic decoding.
CHAPTER 4. SYNTACTIC DECODING

4.1. The Lexemic Supplement I

4.1.1. Complex Lexemes

The function of the Lexemic Supplement I is to (1) allow complex lexemes to be treated as units and (2) allow recognition of lexemes which have discontinuous realizations. The meaning of formal constructions such as black bird (vs. blackbird) or green house (vs. greenhouse) can be predicted from the meanings of their constituents plus the meaning of the construction. We find, however, constructions whose meaning is not predictable from the meaning of their constituents. Conklin (1962) discusses such forms (e.g. poison oak, black-eyed Susan, jack-in-the-pulpit, park bench, atom bomb, etc.), calling them "complex lexemes." Lamb (1987) calls them "phrasal lexemes." This section introduces some difficulties in the computation of complex lexemes in general (4.1.1) and proposes a notational system for representing complex lexemes (4.1.2).

There has been extensive research, e.g. Finnin (1980), Leonard (1984), Gershman's (1979) Noun Group Parser (NGP), in the field of linguistic automation about the noun sequence constructions under the title of 'noun-noun modifier' or 'nominal compounds'. PHRAN (PHRasal ANalyzer), implemented by Arens, Granacki & Parker (1987), has concentrated on handling the recognition and partial analysis
of such constructions, and has achieved some results. The main difficulties recognized by most people in this field of research can be summarized as follows:

(1) It is hard to determine the length of the complex lexeme. One must make sure that just the first noun or verb is not taken to constitute the NP or VP, ignoring the words that follow (e.g. Rice University, call up, etc).

(2) It is hard to determine the semantic properties in the case of the noun sequence.

4.1.2 Notations for Representing Complex Lexemes

The Lexemic Supplement I is a list of complex lexemes. A part of the Lexemic Supplement I is listed in Figure 4.1 for reference.

<table>
<thead>
<tr>
<th>KOREAN</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bg</td>
<td>Ed</td>
<td>LC</td>
<td>SI</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>HANKWUK (0)</td>
<td>CHWULSIN</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>KWENTHWU (0)</td>
<td>KYUNGKI</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>KUM (0)</td>
<td>MEYDAL</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>SEYKYEY (0)</td>
<td>CHOYCHO</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>SUNGMA (0)</td>
<td>SENSEWU</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>TONG (0)</td>
<td>MEYDAL</td>
<td>-1</td>
<td>NOUN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENGLISH</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bg</td>
<td>Ed</td>
<td>LC</td>
<td>SI</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>CALL (1)</td>
<td>UP</td>
<td>-1</td>
<td>Vt</td>
</tr>
<tr>
<td>0</td>
<td>COHEN (0)</td>
<td>HOUSE</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>GET (3)</td>
<td>INTO</td>
<td>-1</td>
<td>Vi</td>
</tr>
<tr>
<td>0</td>
<td>IBM (0)</td>
<td>PC</td>
<td>-1</td>
<td>NOUN</td>
</tr>
<tr>
<td>0</td>
<td>RICE (0)</td>
<td>UNIVERSITY</td>
<td>-1</td>
<td>NOUN</td>
</tr>
</tbody>
</table>

(Bg: Beginning of a complex lexeme
Ed: End of a complex lexeme
LC: Lexical Category
SI: Semantic Information)

Fig. 4.1.

Concerning the first difficulty summarized above (i.e. determination of the length of a complex lexeme), we treat a
complex lexeme as a unit, not as a combination of morphemes. Since the complex lexeme is treated as a unit, it belongs as a whole to its respective syntactic and semantic categories. The notation requires us to put arbitrary numbers 0 and -1 at the beginning and end of a complex lexeme. The number 0 signals that it is the beginning of a complex lexeme and the number -1 signals that it is the end of a complex lexeme. With regard to the second difficulty, we consult the last noun in the sequence in order to assign semantic properties for the noun sequence. In most cases, the semantic property of noun sequences can be inferred largely from the last noun in the sequence. Just as we assign syntactic and semantic information for simple lexemes in the Master Dictionary (cf. 3.3) so we assign the same information for complex lexemes in the Lexemic Supplement I. The assignment of semantic information is required in order to provide a correct interpretation of the meaning of the given input.

Let us now concern with the computational procedure for going through the Lexemic Supplement I with an example. When the machine reads the graphemic word string from the input unit, graphemic word by graphemic word, and it notes the address of each morpheme in the Master Dictionary. The address is marked as a number. Using addresses in the Master Dictionary is like using a library indexing scheme. In a library, books are similarly arranged according to a numbering system. Naturally, finding the book in the
index does not immediately provide you with the book. Rather it gives you the reference number of the book. Likewise, an input sentence is referenced by numbers. For example, if the input is *I like the IBM PC*, the machine reads the five graphemic words one by one, and it notes the addresses of each word. Let us suppose that the graphemic word *I* comes 20th in the Master Dictionary, the word *like* comes 40th, the word *the* 60th, *IBM* 23rd, and *PC* 50th. Then the machine saves the address numbers in the buffer, one by one. A buffer for these input strings (in numbers) appears as shown in Figure 4.2.

```
  0  0  0  0  0
20 40 60 23 5C
```

Fig. 4.2.

The reason for reading the input *IBM PC* first as two separate units is that each can occur separately in other strings (ex. *The PC is mine* or *I am working in the IBM company*). The network in Figure 4.3 shows these interlocking relationships:

![Interlocking network diagram](image)

Fig. 4.3.

After saving the address number of each graphemic word, the buffer is passed to the Lexemic Supplement I. If the two
graphemic words are found as a sequence in the Lexemic Supplement I, then the machine combines the two input strings into one and remembers the address number of the complex lexeme in the Lexemic Supplement I. For example, if the IBM PC is found in the fifth position in the Lexemic Supplement I, the machine rearranges the buffer as shown in Figure 4.4.

![Figure 4.4](image)

In the buffer, each number represents I, like, the and IBM PC, respectively. What, then, do the numbers '0' and '1' (indicated by the arrow) mean? These numbers represent the boolean values, true (1) and false (0). If the input string is found in the Master Dictionary, it is marked as 0, if the input string is found in the Lexemic Supplement I, it is marked as 1. Why do we set up these numbers in the buffer? This data structure is made to avoid confusion. Without it the machine cannot identify whether the word comes from the Master Dictionary or from the Lexemic Supplement I. That is to say, the computer will not know whether the number 5 stands for the fifth word in the Master Dictionary or the fifth word in the Lexemic Supplement I. If the 5th word in the Master Dictionary is boy, instead of IBM PC, the computer could set up the buffer to contain the input string
I like the boy. But, this is not the sentence that we want to decode. Therefore, the number '0' means that the word comes from the Master Dictionary, and the number '1' means that the word comes from the Lexemic Supplement I.

What then does the number that appears within a complex lexeme stand for? The number shown in braces in Lexemic Supplement I above signifies the maximum number of morphemes that can possibly appear internally within the complex lexeme. For example, the number 0 in the brackets means that no morpheme can intervene within the complex lexemes. That is, the English complex lexeme, IBM PC or Rice University does not allow any morpheme to come between IBM and PC or Rice and University. The number 1 in the braces means that the maximum number of 12 morphemes that possibly intervene in the complex lexeme call up as in I ed call her up/I ed call up my frined. The number 3 in the braces means that the maximum number of morphemes that possibly intervene in the complex lexeme get into trouble is 3 as in He ed get his little sister into trouble (after parsing through the Morphemic Supplement II). By assigning the maximum number of morphemes which can come within a complex lexeme, discontinuous constituents can be structured (cf. Winograd 1983:136). We would like to treat examples (a) and (b) below as containing a single lexical item call up.

a. He called up his girl friend.
b. He called her up twice a week.

If we simply treat called and up as two unrelated
words, the grammar fails to reflect the structure in which they are really a single unit. To make the structural description reflect this connection, the Lexemic Supplement I allows discontinuous constituents to form a complex lexeme and allows some morphemes to intervene between the discontinuous constituents. So, if a string contains morphemes that intervene in a single lexical item like call up or get into trouble, we enter the number of intervening morphemes in the space between the brackets. In decoding a sentence such as He called her up, called is segmented as call and -ed, and the suffix -ed is placed before the base call at the deep morphemic level (cf. 3.1 and 3.2). Accordingly one intervening morpheme (her) can be found within the discontinuous constituent call up. Thus the number 1 is inserted between call and up. The intervening morpheme is extracted from the constituent call her up, and the rest of it (call up) is treated as a single lexical item functioning as a transitive event in a sentence. By the same token, in the phrase got his little syster into trouble, the irregular composite got is analyzed as get and ed by the Morphemic Supplement I, and the tense is placed before the base by the Morphemic Supplement II. Thus, the number of intervening morphemes is marked as 3 (i.e. his little syster). The rest of the phrase (get into trouble) is treated as a single unit functioning as an intransitive event in a sentence.
We have attempted to solve the two main difficulties raised in the treatment of complex lexemes. To overcome these difficulties, complex lexemes are treated as a unit, not a combination of morphemes. In assigning semantic properties for the noun sequence one consults the semantic properties of the last noun. In order to represent the complex lexeme some specific notations are required. The notations require that one put arbitrary numbers 0 and -1 in order to signal the boundary of the complex lexemes. Besides addressing this, the notations also make it possible to treat discontinuous constituents by assigning the number of morphemes that intervene in a complex lexeme.
4.2. Lexemic Supplement II

The Lexemic Supplement II provides syntactic rules, which direct the phrase structure analysis of the input string. The syntactic decoder is operated upon syntactic rules in order to produce (a) syntactic tree(s).

This section (4.2) consists of three sub-sections. The first sub-section (4.2.1) provides a discussion of Context-Free Grammars since the Lexemic Supplement II is a CFG. The second sub-section (4.2.2) deals with the algorithm of the syntactic parser with a simplified example, and the third sub-section (4.2.3) illustrates some syntactic outputs produced by the grammar.

4.2.1 Context-Free Grammars (CFGs)

There have been many disputes among linguists working on linguistic automation concerning the constraint of a syntactic grammar. Some linguists in this field, e.g. Welin (1979), suggest that the grammar should be motivated by a linguistic model which is not context-free. Welin claims that a context-free grammar is not sufficient to describe or analyze the whole range of syntactic constructions which occur in natural language texts, e.g. the phenomenon of agreement (concord), or government in English. On the other hand, other linguists, e.g. G.R. Sampson (1983), G. Gazdar (1985) and Graham, Harrison & Ruzzio (1980) argue for a context-free grammar, and Gazdar (1985:82) states:

"The issue of the status of Natural Languages (NLs) with respect to the context-Free Languages (CFLs) and Context-
Free Phrase Structure Grammars (CF-PSGs) is not resolved, . . . , all the published arguments seeking to establish that NLs are not CFLs, or that CF-PSGs constitute the appropriate formal theory of NL grammars, are completely without force. But it does have as a consequence that computational linguists should not just give up on CF-PSGs."

Also, Sampson (1983:158-160) states that the correct parsings of natural sentences are ones which can be produced relatively quickly. He cites Earley (1970) as the basic reference to show that context-free grammars save parsing time.

Among the different types of a stratified grammar, the Lexemic Supplement II employs a part of it, which is called CFG. The choice is motivated by theoretical and practical considerations. By using Lamb's stratified approach, the context-sensitivity dispute is resolved by viewing complex structures as composites of multiple layers (context-free) that are interrelated by context-sensitive interfaces. Thus we can treat the complex structure in a series of less complex layers such that each layer is relatively simple. Many workers in the field of linguistic automation fail to clearly separate out the different levels often recognized by linguists. They often attempt to use just 'words' as terminal symbols, for instance. The consequence of failure to stratify is excessive complexity, which dissolves if complex structures are seen as composites of multiple layers, each of level is relatively simple. Viewing complex structures as being made up of different strata with realizational relationships between
them enables us to describe each level as context-free. Dictionary entries of Morphemic Supplements (I & II) and the Lexemic Supplements include a treatment of stratal discrepancies when necessary.

Ex 1. M2: ed take Morphemic Supplement II
    M1: take ed Morphemic Supplement I
    G: took

Ex 2. L': Tns Passive Vt Lexemic Supplement II
    L: PreT be en Vt Master Dictionary
    M': s be en take Morphemic Supplement II
    M: be s take en Morphemic Supplement I
    G: is taken

At the lower level, the context-sensitive structures are analyzed as context-free constructions (cf. Morphemic Supplement I for the suffix _s).

Besides this theoretical consideration, as a practical consideration, CFGs are relatively transparent and their nature is well understood, thanks to multiple applications in natural language understanding. Before we discuss context-free grammars, let us briefly take up the definition of CFGs. CFGs may be contrasted with 'context-sensitive' grammars (CSFs). A context-free grammar is a finite set of rules each of the form 'A → B,' where 'A' is a single symbol drawn from a set of non-terminal symbols (i.e. symbols invented for use in the formal grammar) and
'B' is a finite sequence of symbols which may include non-terminal symbols and/or terminal symbols (i.e. elements of the vocabulary of the language to be defined). In the use of a CFG for producing strings, beginning with the initial symbol or distinguished symbol (usually S for 'sentence'), one continues to write symbols as permitted by the rules until one is left with a string containing only terminal symbols. Any sentence permitted by a context-free grammar will be associated with a tree-structure in a fairly obvious way: each node of the tree, with the branches below it, will correspond to one application of a rule. It is claimed that a CFG can generate a syntactic tree which is useful in computing the semantic organization.

We need to notice here that it does not cause any problems in the general decoder whether either CFGs or CSGs are employed. The output of the Lexemic Supplement II, whether the syntactic rules are stated in the form of the CFG or the CSG, is a syntactically parsed tree. The output of a syntactic structure tree, whether it is produced by a CFG parser or a CSG parser, is further decoded by the upper level dictionaries, e.g. Sememic Supplement I and II.
4.2.2. An Algorithm for the Syntactic Parser.

The algorithm used in syntactic decoding implements the Active Chart method which originated from Kaplan's General Syntactic Processor (1973) and was elaborated by Winograd (1983).

The parsing strategy used for the process of syntactic decoding is a mixture of the TOP-DOWN and the BOTTOM-UP strategies. The reason for employing this dual strategy is to achieve certain efficiencies of both TOP-DOWN and BOTTOM-UP parsing, and to enhance technical efficiency. Basically, a TOP-DOWN procedure begins by expanding rules for the desired top-level structure (usually a sentence), sees what constituents would be needed to make it up, looks for rules for those constituents, and in this way proceeds down the tree structure until it reaches strings of graphemic words. The Lexemic Supplement II and the Sememic Supplement I are organized in a top-down fashion. As opposed to such TOP-DOWN parsers, a BOTTOM-UP parser begins with the graphemic words and reduces them to the root 'S.' It takes an input sentence, replaces the terminal symbols by their grammatical categories, and strings of categories by other categories. It proceeds up the tree structure until it is able to combine constituents coverting the entire input into a single structure labeled with the 'distinguished symbol.' The same structures are recognized by TOP-DOWN and BOTTOM-UP parsing, but the amount of work done and the nature of the
working structures are quite different. In general, a TOP-DOWN strategy avoids putting together combinations of input strings that would not fit into the overall structure of the sentence, while a BOTTOM-UP parser never tries building up structures that will fail. The Morphemic Supplements I & II and the Lexemic Supplement I are organized in a bottom-up fashion. The Active Chart parser defines an algorithm that combines the advantages of these two strategies: Only do what is relevant, and don't do anything more than once.

Let us turn our attention to how an Active Chart Parser actually works. The chart can be visualized as a network of 'vertices' representing points in the sentence, linked by 'edges' representing constituents. An edge names a constituent that begins and ends at the vertices it connects. The parser executes three procedures: (1) propose edges, (2) combine active edges, and (3) combine complete edges. Each procedure is explained one by one.

(1) Propose Edges
To propose edges means to expand non-terminal symbols, e.g. S / NP VP, NP / NP2, etc. The propose procedure does not propose a non-terminal symbol more than once. Therefore, we can avoid recursive problems. For example, among two possibilities for NP rules (NP -> Det NP2, NP -> NP2), if the first rule matches the given input and once the NP2 in the first rule is proposed, the second NP rule is not proposed again.

(2) Combine Active Edges
To combine active edges means to combine two edges iff (if and only if) the label (e.g. NP, VP, etc) of the two edges is same AND the starting vertex number of an entering edge is the same as the ending vertex number of an active edge which still has a remainder.
e.g. Entering edge : NP → NP2 (label = NP)
Active edge : S → NP VP (label = NP)
New edge : S → NP VP (label = VP)
(Note: In the case of a complete edge, which does not have any remainder, the label is the symbol on the left-hand side. On the other hand, in the case of an active edge, label is the symbol to be proposed).

(3) Combine Complete Edges
To combine complete edges means to combine two edges iff an active edge is complete and the ending vertex number of an entering edge is the same as the starting vertex number of an active edge.
e.g. Entering edge: NP → NP2
Active edge: NP2 → NOUN
New edge: NP → NP2

We now follow an active chart parse of the sentence I go, using the provisional grammar in Figure 4.5.

\[
\begin{align*}
S & / NP \ VP \\
NP & / NP2 \\
NP2 & / NOUN \\
VP & / VERB \\
\end{align*}
\]

Fig. 4.5.

If the input sentence is English I go, the chart starts out containing only the edges corresponding to the individual words and their lexical categories, as in Figure 4.6.

![Diagram](image)

Fig. 4.6.

The initialization process first produces the chart of Fig. 4.6, then proposes the symbol S at the first vertex. The proposal produces a new active edge for each rule whose
left-hand side is the symbol being proposed. Both the starting vertex and ending vertex of the edge are the position at which it is being proposed, and the remainder is the entire right-hand side of the rule. In the grammar in Fig. 4.5, there is only rule for S, and the initialization creates a single pending edge, (S → NP VP).

Step 1: Entering (S → NP VP)

Since there is only one rule for NP, a single pending edge is created, (NP → NP2). At this point, there is nothing to be combined since the symbol of the remainder (NP2) does not match a complete edge that was created for the morpheme I. A new proposal of NP2, however, is created at vertex 1.

Step 2: Entering (NP → NP2)

A pending edge is created from proposing NP2 at vertex 1, (NP2 → NOUN). At this point, there is something to be combined since the symbol of the remainder (NOUN) matches a complete edge that was created for the morpheme I. As a result of combining the proposed edge with the complete edge NOUN, the new active edge NP2 → NOUN is created.

Step 3: Entering (NP2 → NOUN)

This newly entered complete edge creates another complete edge NP → NP2.
Step 4: Entering \( \_\text{NP} \rightarrow \_\text{NP2} \_2 \)

By combining the newly entered complete edge with continuing active edge \( \_\text{S} \rightarrow \_\text{NP} \_\text{VP} \), a pending edge is created \( \_\text{S} \rightarrow \_\text{NP2} \_\text{VP} \).

Step 5: Entering \( \_\text{S} \rightarrow \_\text{NP2} \_\text{VP} \)

A pending edge is created from proposing VP at vertex 2, \( \_\text{VP} \rightarrow \_\text{VERB} \).

Step 6: Entering \( \_\text{VP} \rightarrow \_\text{VERB} \)

This newly entered edge matches a complete edge that was created for the morpheme \( \text{go} \). As a result of combining the active edge with the complete edge \_\text{VERB} \), the newly complete edge \( \_\text{VERB} \rightarrow \_\text{VP} \) is created.

Step 7: Entering \( \_\text{VP} \rightarrow \_\text{VERB} \_3 \)

This newly entered edge is combined with continuing active edge \( \_\text{NP2} \rightarrow \_\text{VP} \), resulting in another complete edge \( \_\text{NP2} \rightarrow \_\text{VP} \_3 \). Finally the given input succeeds and the syntactic parser produces a syntactic tree.
4.2.3. Examples of Syntactic Outputs.

This section gives some examples of the syntactic output produced by the subsystem consisting of all of the components up to and including the Lexemic Supplement II. The labels, e.g. nSubj, nTop, nObj, etc., which are used by Lamb (1958) and Beukema (1975) in their dissertations, are employed in the Lexemic Supplement II. In the label, first (lower case) letter tells the form class to which it is attached (i.e. n for Noun), and the rest of the label indicates the category of the combination (i.e. Subj. for Subject).
'I ate some apples'

Output:

>> nanun sakwalul mekessta.

**<The parsed list>**

(S (DS (C1 (NP (NP2 (NOUN NA)))
   (nM (nTop NUN))))
 (VP (ADJUNCT EMPTY))
 (VI (NP (NP2 (NOUN SAKWA)))
   (nM (nObj LUL))))
 (Vt MEK)))
 (vV (Tns (Past ESS))))))))
(SE (Dec-N TA))))
'My younger-sister is pretty.'

Output:

>> nay tongsayngun yepputa.

**<The parsed list>**

+ (S (DS (C1 (NP (NP2 (UADJ (UAdj NAY)))))
  (NP2 (NOUN TONGSAYNG))))
  (nM (nTop UN)))
  (VP (ADJUNCT EMPTY))
  (VI (Adj YEPPU))
  (vV EMPTY))))
  (SE (Dec-N TA))))
c. nay tongsayng i yenghi eykey yeppu n koyangi lul cu ess ta.
my younger sister nSubj Yenghi nMDat be-pretty AtTH cat nObj give past SE-N

Output:

>> nay tongsayngi yenghieykey yeppun koyangilul cu esssta.

**(The parsed list)**

(S (DS (C1 (NP (NP2 (URDJ (URDJ NRY))))
  (NP2 (NOUN TONGSAEYNG)))))
  (nM (nSubj I)))
  (VP (ADJUNCT (ADV (DatADV (NP2 (NOUN YENGHI))))
    (nMDat EYKEY))))
  (ADJUNCT EMPTY))
  (VI (NP (NP2 (Att-Adj (Adj YEPPU))
    (AtTH N)))
    (NP2 (NOUN KOYangi)))
  (nM (nObj LUL)))
  (Vt O Ju))
  (vV (Tns (Past ESS)))))
(S (Dec-N TA))))
'Yenghi likes the player who won the gold medal in swimming.'
Output:

>> yenghinun swuyengseyse kum meytallul ttan senswulul cohahanta.

**<The parsed list>**

(S (DS (C1 (NP (NP2 (NOUN YENGHI))
   (nM (nTop NUN)))))
   (VP (ADJUNCT (ADV (LADV (NP2 (NOUN SWUYENG)))
   (nAdL EYSE))))
   (ADJUNCT EMPTY))
   (VI (NP (NP2 (PAPL-CL (C1 (VP (ADJUNCT EMPTY))
   (VI (NP (NP2 (NOUN KUM MEYTAL)))
   (nM (nObj LUL)))))
   (Vt TTA))
   (vv EMPTY)))))
   (Pap1M (PasPM N) ) ) )
   (NOUN SENSBU))
   (nM (nObj LUL)))))
   (Vt COHHA))
   (vv (Tns (PreT N)))))))
   (SE (Dec-N TA))))
   (SE (Dec-N TA))))
   (SE (Dec-N TA)))))
The boy eating rice is my younger-brother.

Output:

`>> papul meknun sonyeni nay tongsayng ita.`

**<The parsed list>**

(S (DS (C1 (NP (NP2 (PAPL-CL (C1 (VP (ADJUNCT EMPTY)))
(VI (NP (NP2 (NOUN PAP))))
(nM (nObj UL))))
(Vt MEX))
(vV EMPTY)))(PapLM (PrePM NUN)))
(NOUN SONYEN))
(nM (nSubj I))))
(VP (ADJUNCT EMPTY))
(VI (COMPLEMENT (NP2 (UADJ (UAdj NAY)))
(NP2 (NOUN TONGSAYNG)))
(Id-Vb I)))
(vV EMPTY)))(SE (Dec-N TA))))
'I ate some rice and Yenghi ate some apples.'

Output:

>> nanun papul mekko yenghinun sakwalul mekessta.

**<The parsed list>**

(S (DS (C1 (NP (NP2 (NOUN N)))))
  (nM (nTop NUN)))))
  (VP (ADJUNCT EMPTY))
  (VI (NP (NP2 (NOUN PAP))))
  (nM (nObj UL)))))
  (Vt MEK)))
  (vV EMPTY))))))
(CONJ (Conj KO)))
(C2 (C1 (NP (NP2 (NOUN YENGHI))))
  (nM (nTop NUN)))
  (VP (ADJUNCT EMPTY))
  (VI (NP (NP2 (NOUN SAKWA))))
  (nM (nObj LUL)))))
  (Vt MEK)))
  (vV (Tns (Past ESS)))))
(Se (Dec-N TA))))
'Pretty Yenghi was bitten by the snake.'

Output:

>> yeppun yenghika paymhanthey mwulliessta.

**<The parsed list>**

(S (DS (Cl (NP (NP2 (Att-Adj (Adj YEPPU)))
  (AttM N))))
  (NP2 (NOUN YENGHII)))
  (nM (nSubj KA))))
  (VP (ADJUNCT (ADV (AGADV (NP2 (NOUN PAYM)))
    (nAdAG HANTHEY))))
  (ADJUNCT EMPTY)))
  (VI (Vt MWUL))
  (PASSIVE (HI LI)))))
  (vV (Tns (Past ESS))))))
  (SE (Dec-N TA))))
ENGLISH

a. The salesman sold the IBM PC.
   Def Noun ed sell Def Noun Noun
   DET NP2 Past Vt DET Noun
   NP Tns NP2 NP
   VI VP
   CL

Output:

>> The salesman sold the IBM PC.

**<The parsed list>**

(S (CL (NP (DET (Def THE))))
   (NP2 (NOUN SALESMAN)))
   (VP (Tns (past ED)))
   (VI (Vt SELL))
   (NP (DET (Def THE)))
   (NP2 (NOUN IBM PC)))))))

(DS EMPTY)))
b. The IBM PC was sold by the salesman.

Def Noun Noun ed be en sell nPPAG Def Noun

DET NOUN Past Aux-Be papl Vt DET NP2

NP2 Tns

NP

PASSIVE

NP

AGPP

VI

PP

VI

VP

CL

Output:

>> The IBM PC was sold by the salesman.

**<The parsed list>**

(S (CL (NP (DET (Def THE)))
  (NP2 (NOUN IBM PC)))
  (VP (Tns (past ED)))
    (VI (VI (PASSIVE (Aux-Be BE))
        (papl ED)))
      (Vt SELL))
    (PP (AGPP (nPPAG BY))
      (NP (DET (Def THE)))
        (NP2 (NOUN SALESMAN))))))))

(DS EMPTY)))
c. The girl is pretty.
   Def Noun be Adj
   DET NP2 Pres COPVERB COMPLEMENT
   Tns
   VI
   VP
   CL

Output:

>> The girl is pretty.

**<The parsed list>**

(S (CL (NP (DET (Def THE)) (NP2 (NOUN GIRL)))) (VP (Tns (pres S))) (VI (COPVERB BE)) (COMPLEMENT (ADJ PRETTY))))

(DS EMPTY))
Output:

>> The pretty girl gave me a book.

**<The parsed list>**

(S (CL (NP (DET (Def THE))))
   (NP2 (ADJ PRETTY))
   (NP2 (NOUN GIRL))))
(S (VP (Tns (past ED)))(VI (BIVERB GIVE))
   (NP3 (NP (NP2 (NOUN ME))))
   (NP4 (NP (DET (InDef A))))
   (NP2 (NOUN BOOK)))))))
(DS EMPTY)))
Output:

>> The pretty little girl gave a book to me.

**<The parsed list>**

(S (CL (NP (DET (Def THE)))
   (NP2 (ADJ PRETTY))
   (NP2 (ADJ LITTLE))
   (NP2 (NOUN GIRL)))))
   (VP (Tns (past ED)))
   (VI (VI (BIVERB GIVE))
   (NP (DET (InDef A)))
   (NP2 (NOUN BOOK)))
   (PP (DATPP (nPPDAT TO))
   (NP (NP2 (NOUN ME))))))

(DS EMPTY)))
f. The tall player who won the gold medal drinks beer.

Output:

>> The tall player who won the gold medal drinks beer.

**<The parsed list>**

(S (CL (NP (DET THE)))
  (NP2 (ADJ TALL))
  (NP2 (NOUN PLAYER))
  (RC (RP WHO))
    (SUBCL (VP (VI (TVt (Vt WIN)))
      (Tns (past ED)))
    (NP (DET THE)))
  (NP2 (ADJ GOLD))
    (NP2 (NOUN MEDAL)))
  (VP (VI (TVt (Vt DRINK)))
    (Tns (pres S)))
  (NP (NP2 (NOUN BEER))))
(DS EMPTY))
g. I liked the boy who will like my cat.

Output:
>> I liked the boy who will like my cat.

**<The parsed list>**

(S (CL (NP (NP2 (NOUN I)))))
  (VP (Tns (past D)))
  (VI (Vt LIKE))
  (NP (DET (Def THE)))
  (NP2 (NOUN BOY))
  (RC (RP WHO))
  (SUBCL (VP (Tns (fut WILL))))
  (VI (Vt LIKE))
  (NP (NP2 (ADJ MY))
   (NP2 (NOUN CAT)))))

(DS EMPTY)
h. We loved Korea and they loved America.

Output:

>> We loved Korea and they loved America.

**<The parsed list>**

(S (DS (C1 (NP (NP2 (NOUN WE)))))
 (VP (Tns (past D)))
 (VI (Vt LOVE))
 (NP (NP2 (NOUN KOREA)))))))

(CONJ AND))

(C2 (C1 (NP (NP2 (NOUN THEY))))
 (VP (Tns (past D)))
 (VI (Vt LOVE))
 (NP (NP2 (NOUN AMERICA))))))))))}
Output

>> He is eating lunch.

**<The parsed list>**

\( (S (CL (NP (NP2 (NOUN HE))))
   (VP (Tns (pres S)))
   (VI (PROGRESSIVE (Aux-Be BE))
     (prog ING)))
   (Vt EAT))
   (NP (NP2 (NOUN LUNCH))))))
(\(DS \text{ EMPTY})\))
This section has concentrated on the function of the Lexemic Supplement II, along with the algorithm of the syntactic parser. The Lexemic Supplement II consists of context-free phrase structure rules, which direct the phrase structure analysis of the input string. The syntactic decoder produces a syntactic tree structure for the input sentence, which is then processed by the semantic decoder to compute the relevant semological information. Syntactic rules for Korean and English are included in Appendix B.
CHAPTER 5. SEMANTIC DECODING

5.1. The Sememic Supplement I

Our discussion in the preceding chapters focusses on the procedure for producing syntactic trees from strings of graphemic words. As mentioned, the procedure for morphological and syntactic decoding is carried out by consulting the Morphemic Supplements I & II, the Master Dictionary, and the Lexemic Supplements I and II. If a natural language understanding system is to serve either as a model of human linguistic ability or as a practical front-end for an intelligent computer system, it must, however, at some stage, deal with the 'meaning' of sentences. It is well known that a properly conceived automatic decoding system should have procedures for constructing and representing semological information. This process would involve building a semantic structure for the representation at the conceptual level. The main concern in this chapter is to represent the conceptual content in a modular system. Modularity involves the analysability of a program or system into (interacting) parts, and the nature of the relationship between the parts. Phrase structure and conceptual structure, roughly syntactic and semantic levels, are treated separately in the system. The essential question is how to achieve modularity without mixing linguistic information with computer procedure. An answer is that the output result of syntactic decoding completed at an earlier stage of the
decoding process can serve as input to the semantic level. Traditional syntactic trees are thus seen as a useful preliminary to the organization of semantic statements. We therefore need not regard them as redundant constructs even for a program whose primary aim is the computation of meaning. The syntactic trees produced by the lower-level dictionaries are converted into semantic representations. The dictionary immediately above the Lexemic Supplement II, called the Sememic Supplement I, enables us to decode the output of the syntactic parsing tree and thus to provide a semantic structure tree. In order to achieve semantic comprehension, this study proposes two dictionaries at the sememic level. The reason of constructing two dictionaries at the sememic level, e.g. Sememic Supplements I and II, is based on practical motivation, not on any linguistic requirement.

The Sememic Supplement I consists of information pertinent to the representation of a specific input string at the conceptual level. This information takes the form of a list which keeps track of the possible syntactic pathways for identifying a specific input string. Therefore, the function of the Sememic Supplement I is to identify the role relationships with respect to a specific event in a general way and to illustrate the organization of a proposition, e.g. process, P1, P2, Tense, etc.

It is worthwhile to mention that the symbol used in this study for a 'concept' is a 'lexemic' symbol.
The same symbol used to represent a lexeme is used arbitrarily to represent the corresponding concept as well. In the conceptual stratum there are unique concepts. The concept is realized as a specific lexeme in a given language. Instead of using a gloss from a common language to represent a concept this study borrows lexemes of each language as symbols for concepts. Therefore, Korean lexemic symbols are used for Korean conceptual representations, and English lexemic symbols are used for English conceptual representations.

In order to make our problem and its solution more concrete, let us designate a simple syntactic tree as an input and compute the desired output. After processing the input sentence The salesman sold the IBM PC through Morphemic Supplements I & II, the Master Dictionary, and Lexemic Supplements I and II, we derive the following syntactic tree as the output:
It is now necessary to derive a conceptual representation from the syntactic tree that identifies the NOUN within the first NP string as P1 (the first participant or argument), the NOUN within the second NP string as P2 (the second participant), the main verb within the VP as Process, etc. How can such identification be made by the computer? One possible approach is to keep track of the pathways of nodes appearing in the parsing tree, and cutting them as necessary. The terminal symbols below the slanted lines carry semantic labels such as process, participants, etc. As Figure 4.1 shows, salesman
is connected to the NOUN node within NP2 which is in turn connected to NP. This NP is then attached to C1 (Clause 1), which in turn is connected to DS (Declarative Sentence) and S (Sentence). There is more than one type of syntactic construction that can be recognized as representing P1, P2, Process, etc., just as more than one entry can recognize non-terminal symbols in the Lexemic Supplement II, e.g. NP, NP2, VI, etc. Each terminal symbol has a syntactic pathway in the tree leading from the distinguished symbol. The syntactic pathway activated for identifying the input sentence above is shown in Table I:

<table>
<thead>
<tr>
<th>Lexeme</th>
<th>Pathway</th>
</tr>
</thead>
<tbody>
<tr>
<td>sell</td>
<td>S.DS.C1.VP.VI.Vt</td>
</tr>
<tr>
<td>salesman</td>
<td>S.DS.C1.NP.NP2.NOUN</td>
</tr>
<tr>
<td>IBM PC</td>
<td>S.DS.C1.VP.VI.NP.NP2.NOUN</td>
</tr>
<tr>
<td>ed</td>
<td>S.DS.C1.VP.Tns</td>
</tr>
</tbody>
</table>

TABLE I

Each syntactic pathway corresponds to a specific semotactic function. Thus each of the lexemes sell, salesman, IBM PC, and -ed has a specific semotactic function, Process, P1, P2 and Tense, respectively. These function labels can be correlated with the pathways. Therefore, Table I above can be expanded as Table II:
Besides this information, the Sememic Supplement I needs to indicate whether a participant is known or unknown to the hearer. In English, this information is carried in part by the choice of article. The semantics of English articles involves a scale that ranges from NEW to GIVEN, i.e. NEW - COMPUTABLE - RECOVERABLE - GIVEN. (cf. Copeland and Davis 1980, 1983, Davis 1987). For manageability, this study, however, intends to represent only the two extreme cases, Given and New. At the level of conceptual representation, this study is concerned with whether an input argument is completely known or completely unknown to the hearer, excluding distinctions that falls between these two extremes, i.e. Computable and Recoverable. In the same way that we identify the process and the participants in a proposition, the status of Givenness or Newness of a participant can be identified by keeping track of its syntactic pathways as follows:

\[
\text{GV} = \text{S.DS.C1.NP.DET.Def} \quad \text{or} \quad \text{S.DS.C1.VP.VI.NP.DET.Def}
\]

\[
\text{NEW} = \text{S.DS.C1.NP.DET.InDef} \quad \text{or} \quad \text{S.DS.C1.VP.VI.NP.DET.InDef}
\]
After we derive the participants along with their status as Given or New and the process from the syntactic structure tree, we can draw a semantic structure tree as shown in Figure 5.2.

![Diagram](image)

Fig. 5.2.

Let us take an example from Korean to further demonstrate the function of the Sememic Supplement I. We produce the syntactic tree as shown in Fig. 5.3 for the sentence *Yenghinun yepputa* 'As for Yenghi, she is pretty,' by using the appropriate rules in the Lexemic Supplement II.
An adjective, if used predicatively in Korean (cf. 2.2), functions as Process at the level of conceptual representation. Also in the English structure of copular verb I am hot, the process should be represented as be and hot, not as the copular verb be alone.

As also mentioned in section 2.9, the semantics of the subject and the topic markers in Korean are quite different from each other, and the distinction of subject and topic needs to be specified at the level of the conceptual representation. This distinction is made by giving a condition like 'topic-cond' or 'subject-cond.' These conditions are interpreted such that 'if the topic (nun) or subject (ka) marker is found in a syntactic structure tree, a certain command will be executed.' In the command 1 below, the execution to assign the same P1 to the topic
is completed if and only if the preceding condition is met (if the topic marker is found). By the same method used to identify the participants and the process in English, we can identify the participants and the process for the Korean sentence *Yenghinun yepputa*, using the following command:

```
Command 1
Process = S.DS.C1.VP.VI.Adj
P1 = S.DS.C1.NP.NP2.NOUN
TOPIC = @topic-cond P1
```
(The marker '@' means 'if the following condition is met'.)

The semantic tree structure derived from the syntactic pathways for the Korean sentence above is shown in Fig. 5.4 below.

```
S
   /\  \
  /   \  
Process P1 = TOPIC
    /\     \    
   yeppu Yenghi
```

Fig. 5.4.

Based on the two examples above, some readers might conclude that a conceptual representation, which converts a syntactic tree structure into concepts, is just a syntactic pathway going by a different name. In the Sememic Supplement I in our system, however, 'conceptual representation' is not simply another label for a syntactic pathway. Conceptual representations and syntactic pathways do not stand in a relationship of one-to-one correspondence. In order to represent concepts of copular verb
constructions, for instance, we do not identify the English copular verb *be* alone as the Process. For the conceptual representations of the copular verb constructions, the entries for the Sememic Supplement I make it possible to indicate the ADJ and *be*, not the copular verb *be* alone, as the Process. Besides this copular verb constructions, the Sememic Supplement I entries provide additional indication that the conceptual representations are not simply syntactic pathways using different labels. In the conceptual representation for passive constructions, we give some constraints in the Sememic Supplement I for assigning the P2, not the P1, as the Subject. (See section 5.1.2)

Let us now discuss how the Sememic Supplement I treats a sequence of ADJ's. Our strategy for keeping track of syntactic nodes and cutting them where necessary can cause some problems for a construction which contains several ADJ's. The easiest way to present the problem is to give an example and discuss it. From an input sentence such as *Those two little players won the gold medal*, we produce the syntactic tree structure shown in Fig. 5.5.
If we use the syntactic path $S \cdot D S \cdot C I \cdot N P \cdot N P 2 \cdot A D J$ to identify modifiers (those, two, little), it recognizes only the first ADJ those as a modifier, but not the other ADJ's, two and little. However, we want to represent all three ADJ's those, two and little asModifiers (or Describers). That is to say, at the level of the conceptual representation, the syntactic node NP2 needs to be represented as shown in Fig. 5.6.
To represent a sequence of ADJ's as Describers at the level of the conceptual representation, the Sememic Supplement I contains the following command:

$$\text{DES} = \text{C1.NP.NP2.}.^*\text{ADJ.}^.$$ 

The symbol '*' stands for the recursive node of the NP2, e.g., and the symbol '^' following a variable (an item) stands for its immediately preceding node (or a parent node). Here, the variable is ADJ and its preceding node is NP2. For example, the command 'B.' means 'the parent node of B.' In the following syntactic tree, the parent node of B is A:

If we use a command such as C1.NP.NP2.*.ADJ without the symbol '^', the syntactic tree is cut right after the first ADJ node, with the result that the Describer is represented as those, only. By assigning the symbolism '^', we give a command to the machine that means 'cut the parent node of
the ADJ node,' in other words, cut the first NP2 node.

Let us consider Figure 5.7.

```
    C1
     |  
    NP  
     |   
    NP2  
      |  
     ADJ  
      |  
     NP2  
      | 
     ADJ  
      | 
    NP2
    /|
   Adj-D Adj-N Adj-C NOUN P1
   /
those two little players
```

Fig. 5.7.

Within this tree, the status of the NOUN players must also be identified. If, in an effort to identify the Describer, we give a command to cut the syntactic tree at the first NP2, then the NOUN node, which is connected to player, will be included within the scope of the Describers. However, the Noun should be excluded from the domain of the Describer. This treatment of the NOUN can be achieved by an entry for the P1. The Sememic Supplement I includes the entry for the P1 `C1.NP.NP2.*.NOUN', which directs us to cut the syntactic node right after the NOUN node as shown in Figure 5.8.
Since the Sememic Supplement I is based on the assumption that a single syntactic node has a single function in a single syntactic tree structure, the NOUN player is recognized only as the P1, not as the Describer. The command for the Describer C1.NP.NP2.*.ADJ.* cuts the syntactic tree at position A, and the command for the P1 C1.NP.NP2.*.NOUN cuts the syntactic tree at position B as shown in Fig. 5.9.

Therefore, the Describer is represented just as those, two and little, excluding the NOUN player.

The three ADJ's, those, two and little, however, need
to be further specified; each describer has a different function at the level of semantics. The Adj-D *those* has the function 'Deictic,' the Adj-N *two* has the function 'Numerative,' the Adj-C *little* has the function 'Epithet' and the Adj-Cl *gold*, the function 'Classifier.' This specification can be represented by subcategorizing ADJ's. We treat ADJ as a non-terminal symbol in the Lexemic Supplement II, and this non-terminal symbol, 'ADJ' can be subcategorized into various kinds of Adj's:

\[
\begin{align*}
\text{ADJ} / \text{Adj-D} & \quad \text{(Demonstrative Adjective)} \\
\text{ADJ} / \text{Adj-N} & \quad \text{(Numerative Adjective)} \\
\text{ADJ} / \text{Adj-C} & \quad \text{(Comparative Adjective)} \\
\text{ADJ} / \text{Adj-Cl} & \quad \text{(Classifiable Adjective)}
\end{align*}
\]

(Note: Adj's that can have comparative forms are represented as 'Epithet' at the level of conceptual representation. (cf. Davis 1987, Quirk and Greenbaum 1973, Halliday 1976))

Since we specify various kinds of Adj's as terminal symbols, a syntactic tree contains nodes such as Adj-D, Adj-N, etc. as shown in Fig. 5.8. or 5.9. Therefore, different functions of the ADJ's can also be identified by keeping track of syntactic nodes. These nodes are produced by syntactic decoding. Consider the semantic tree in Fig 5.10 and entries for the semotactic functions, DEIC, NUM, etc.
Entries for specification of the ADJ's

#Adjectives  F DEIC NUM EPI CLASER
DEIC = F.NP2.*.ADJ.Adj-D (DEICTIC)
NUM = F.NP2.*.ADJ.Adj -N (NUMERATIVE)
EPI = F.NP2.*.ADJ.Adj -C (EPITHET)
CLASER = F.NP2.*.ADJ.Adj-C1 (CLASSIFIER)
(Note: The symbol 'F' represents the preceding nodes of NP2, S.DS.C1.NP. This strategy of substitution is discussed later in the section 5.1.3)

Finally, for the input sentence Those two little players won the gold medal, the conceptual representation is:

>> Those two little players won the gold medal.

***< Conceptual Representation >**

(S (CL1 (PROCESS WIN)
 (SUBJ (DEIC THOSE)
 (NUM TWO)
 (EPI LITTLE))
 (PLURAL S)
 (PPART PLAYER))
(PART1 (DEIC THOSE)
 (NUM TWO)
 (EPI LITTLE))
 (PLURAL S)
 (PPART PLAYER))
(PART2 ( GV )
 (CLASER GOLD)
 (PPART MEDAL))
(Tense ED)))
Up to this point, we have demonstrated the function of the Sememic Supplement I with some examples, and have shown that the conceptual representation is not simply another label for a syntactic pathway. The following three sub-sections demonstrate how the Sememic Supplement I deals with active, passive and complex sentences, respectively.
5.1.1 The way in which the Sememic Supplement I deals with active sentences is discussed above. Our approach is to keep track of the hierarchy of the syntactic constituents produced by the Lexemic Supplement II. The conceptual representations of some active sentences are shown below.

KOREAN

a. nanun sakwalul cohahanta. 'As for me, I like apples.'

>> nanun sakwalul cohahanta.

**< Conceptual Representation >**

(S (CL1 (PROCESS COHAHA)
 (TOPIC (PPART NA))
 (PART1 (PPART NA))
 (PART2 (PPART SAKWA))
 (Tense N)))

b. nayka cipulo kakeyssta. 'I will go home.'

>> nayka cipulo kakeyssta.

**< Conceptual Representation >**

(S (CL1 (PROCESS KA)
 (SUBJ (PPART NAY))
 (PART1 (PPART NAY))
 (Tense KEYSS)
 (DIR (PPART CIP)))))
c. yenghika halmenieykey sakwa lul cwuessta.
    'Yenghi gave an apple to the grandma.'

>> yenghika halmenieykey sakwalul cwuessta.

**< Conceptual Representation >**

(S (CL1 (PROCESS CWU)
    (SUBJ (PPART YENGGHI))
    (PART1 (PPART YENGGHI))
    (PART2 (PPART SAKWA))
    (PART3 (PPART HALMENI))
    (Tense ESS)))


d. ku salami yeppun yenghilul chassta.
    'That man kicked the pretty Yenghi.'

>> ku salami yeppun yenghilul chassta.

**< Conceptual Representation >**

(S (CL1 (PROCESS CHA)
    (SUBJ (DEIC KU)
         (PPART SALAM))
    (PART1 (DEIC KU)
         (PPART SALAM))
    (PART2 (EPI YEPPU N)
         (PPART YENGGHI))
    (Tense SS)))

(Note: Notice that the attributive adjective yeppun
'pretty' is represented as Epithet and the
demonstrative UAdj ku 'that' is represented
as Deictic.)
e. nay tongsayngun kapeliessta.
    'My younger brother/sister has gone.'

>> nay tongsayngun kapeliessta.

**< Conceptual Representation >>

(S (CL1 (PROCESS KA)
    (TOPIC (PDEIC NAY)
        (PPART TONGSAYNG))
    (PART1 (PDEIC NAY)
        (PPART TONGSAYNG))
    (Tense ESS)
    (Aspect PELI)))

f. nanun cipayse papul mekko issessta.
    'I was having meal at home.'

>> nanun cipayse papul mekko issessta.

**< Conceptual Representation >>

(S (CL1 (PROCESS MEK)
    (TOPIC (PPART NA))
    (PART1 (PPART NA))
    (PART2 (PPART PAP))
    (Tense ESS)
    (Aspect KO ISS)
    (LOC (PPART CIP)))))
ENGLISH

a. He likes you.

>> He likes you.

**< Conceptual Representation >**

(S (CL1 (PROCESS LIKE)
     (SUBJ (PPART HE))
     (PART1 (PPART HE))
     (PART2 (PPART YOU))
     (Tense S)))

b. The pretty girl walks to school.

>> The pretty girl walks to school.

**< Conceptual Representation >**

(S (CL1 (PROCESS WALK)
     (SUBJ (GV )
             (EPI PRETTY)
             (PPART GIRL))
     (PART1 (GV )
             (EPI PRETTY)
             (PPART GIRL))
     (Tense ES)
     (DIR (PPART SCHOOL))))

c. I gave a book to the boy.

>> I gave a book to the boy.

**< Conceptual Representation >**

(S (CL1 (PROCESS GIVE)
     (SUBJ (PPART I))
     (PART1 (PPART I))
     (PART2 (NEW )
             (PPART BOOK))
     (PART3 (GV )
             (PPART BOY))
     (Tense ED)))
d. The girl gave the boy a book.

>> The girl gave the boy a book.

**< Conceptual Representation >**

(S (CL1 (PROCESS GIVE)
  (SUBJ (GV )
   (PPART GIRL))
  (PART1 (GV )
   (PPART GIRL))
  (PART2 (NEW )
   (PPART BOOK))
  (PART3 (GV )
   (PPART BOY))
  (Tense ED)))

Note: Sentences (c) and (d) have the same conceptual representation, but different syntactic realizations. The difference in discourse (e.g. Rheme or Focus) is beyond the scope of this study.

e. I called her up.

>> I called her up.

**< Conceptual Representation >**

(S (CL1 (PROCESS CALL UP)
  (SUBJ (PPART I))
  (PART1 (PPART I))
  (PART2 (PPART HER))
  (Tense ED)))

f. He gets into trouble.

>> He gets into trouble.

**< Conceptual Representation >**

(S (CL1 (PROCESS GET INTO TROUBLE)
  (SUBJ (PPART HE))
  (PART1 (PPART HE))
  (Tense S)))
g. He is eating my lunch.

>> He is eating my lunch.

**< Conceptual Representation >**

```
(S (CL1 (PROCESS EAT)
   (SUBJ (PPART HE))
   (PART1 (PPART HE))
   (PART2 (DEIC MY)
     (PPART LUNCH))
   (Tense S)
   (Aspect BE ING))))
```

h. He has gone.

>> He has gone.

**< Conceptual Representation >**

```
(S (CL1 (PROCESS GO)
   (SUBJ (PPART HE))
   (PART1 (PPART HE))
   (Tense S)
   (Aspect HAVE EN))))
```
5.1.2. The Sememic Supplement I also deals with passives by using the same strategy of keeping track of pathways in the syntactic trees. In English passives, the optional NP string occurring as object of by in the VP should be identified as P1, and the NP string that comes immediately under C1 in a syntactically parsed tree should be identified as P2. The reversed word order of P1 and P2 can be correctly identified by giving a condition to the Sememic Supplement I. In English, the condition can be stated as follows: 'If the lexeme node passive (which is expanded into be plus en) is found in the syntactic tree, identify the NP string after by in the VP as P1 and identify the first NP string as P2.' In Korean, the condition can be stated as follows: 'If the passive morpheme is found in the syntactic tree, identify the NP string in adverbial (i.e. agent) phrases as P1 and identify the first NP string as P2.' For example, if the input sentence is 

\text{tongsayngun paymhanthey mwulliessta} 'My younger-sister was bitten by the snake,' P1 and P2 are identified with the aid of the following dictionary entries, which corresponds to the Ordered-Or node in relational networks. In the Ordered-Or node, the left line is activated if the condition given is satisfied, i.e. if the condition 'passive' is found, then the left-line which realizes the P1 is activated:
pass-cond = S.DS.C1.VP.VI.PASSIVE
Process = S.DS.C1.VP.VI.Vt  \( \text{(mwul-)} \)
P1 = @pass-cond  S.DS.C1.VP.ADJUNCT.ADV.AGADV.NP2  \( \text{(paym)} \)
P2 = @pass-cond  S.DS.C1.NP  \( \text{(nay tongsayng)} \)
TOPIC = @top-cond  P2  \( \text{(nay tongsayng)} \)
Tense = S.CL.VP.vV.Tns  \( \text{(-ess-)} \)

(The marker '@' means 'if the following condition is found.')

The semantic structure tree produced by using the command above is shown in Figure 5.11.

```
Fig. 5.11.
```

Output:

`>> nay tongsayngi paymhanthey mwulliessta.`

**< Conceptual Representation >**

```
(S (CL1 (PROCESS MWUL))
 (SUBJ (PDEIC NAY)
   (PPART TONGSAYNG))
 (PART1 (PPART PAYM))
 (PART2 (PDEIC NAY)
   (PPART TONGSAYNG))
 (Tense ESS)))
```
Some further examples of passives are shown below:

**KOREAN**

a. Yenghika Seouleyse ku namcahanthey caphiessta.
   'Yenghi was arrested by the man in Seoul.'

>> yenghika seouleyse ku namcahanthey caphiessta.

**<< Conceptual Representation >>**

(S (CL1 (PROCESS CAP)
   (SUBJ (PPART YENGI))
   (PART1 (DEIC KU)
     (PPART NAMCA))
   (PART2 (PPART YENGI))
   (Tense ESS)
   (LOC (PPART SEOUL))))

b. cipulo kacamaca, nanun halmenihanthey ankiestsa.
   'As soon as I went home, I was hugged by my grandma.'

>> cipulo kacamaca nanun halmenihanthey ankiestsa.

**<< Conceptual Representation >>**

(S (CL2 (PROCESS AN)
   (TOPIC (PPART NA))
   (PART1 (PPART HALMENI))
   (PART2 (PPART NA))
   (Tense ESS)
   (MTIME (CL1 (PROCESS KA)
     (DIR (PPART CIP)))))

(MTIME stands for 'Momentary Time', and this temporal clause is embedded in the CL2)
c. ku sonyennun Yenghihanthey chaissta.
   'The boy was kicked by Yenghi.'

>> ku sonyenun yenghihanthey chaissta.

**< Conceptual Representation >**

(S (CL1 (PROCESS CHA)
   (TOPIC (DEIC KU)
       (PPART SONYEN))
   (PART1 (PPART YENGGHI))
   (PART2 (DEIC KU)
       (PPART SONYEN))
   (Tense ESS)))

ENGLISH

a. The boy was arrested by the police.

>> The boy was arrested by the police.

**< Conceptual Representation >**

(S (CL1 (PROCESS ARREST)
   (SUBJ (GV )
       (PPART BOY))
   (PART1 (GV )
       (PPART POLICE))
   (PART2 (GV )
       (PPART BOY))
   (Tense ED)))

b. The IBM PC was sold by the salesman.

>> The IBM PC was sold by the salesman.

**< Conceptual Representation >**

(S (CL1 (PROCESS SELL)
   (SUBJ (GV )
       (PPART IBM PC))
   (PART1 (GV )
       (PPART SALESMAN))
   (PART2 (GV )
       (PPART IBM PC))
   (Tense ED)))
5.1.3. Complex Sentences are also handled by the Sememic Supplement I. If the P1 or P2 contains an embedded clause, e.g. relative clauses in English and participial clauses in Korean, the complex participant needs to be specified further for the conceptual representation. Let us suppose that we get the provisional conceptual representation below for the Korean sentence swuyengeyse kum meyalul ttan senswuka pal mokul ppiessta 'The player who got the gold medal in swimming has sprained his ankle.'

(Process. ppi) 'sprain'
(P1. swuyengeyse kum meyalul ttan senswu)
    'the player who got the gold medal in swimming'
(P2. pal mok) 'ankle'
(Tense. _ess-) '-ed'

Here, the P1 swuyengeyse kum meyalul ttan senswu needs to be specified further since it contains another complete clause at the sememic level--the embedded clause illustrates another organization of a proposition. The participial clause swuyengeyse kum meyalul ttan senswu 'The player who won the gold medal' comes in the mode of 'definer' or 'identifier' for the P1. Since the definer has another clause structure, we can draw the semantic structure tree shown in Fig. 5.12.
If the English input sentence is The tall player who won the gold medal in swimming drinks beer, the semantic structure tree would be same as the one above. The different syntactic realization (i.e. the definer comes after the participant head in English) does not play any significant role for the representation of the conceptual level.

Keeping in mind that we use an algorithm to keep track of the path taken in constructing a syntactic tree structure, we can identify Describer, P1, P2, Process, etc. within the Definer in the same way. To demonstrate this possibility, let us look at the syntactic parsing output of the Korean and English sentences above:
swuyengeye Kum meytalul ttan senswuka pal novul ppiesta.
'The player who won the gold medal in swimming has sprained
his ankle.'

**<The parsed list>**

(S (DS (C1 (NP (NP2 (PAPL-CL (C1 (VP (ADJUNCT (ADV (LADV (NP2 (NOUN SWUYENG)))
  (nM (nObj UL)))))
  (VT (PPI))
  (PAPLM (PASM N))))
  (nM (nObj RA))))
  (VP (ADJUNCT EMPTY))
  (VI (NP (NP2 (NOUN PLAYER)))
  (RC (RP WHO))
  (SUBCL (VP (Tns (past ED)))
  (VI (VT WIN))
  (NP (DET (Def THE)))
  (NP2 (ADJ (Adj-C GOLD)))
  (NP2 (NOUN MEDAL)))))))

(VP (Tns (pres S)))
  (VI (VT DRINK))
  (NP (NP2 (NOUN BEER)))))))

ENGLISH

The tall player who won the gold medal in swimming drinks beer.

**<The parsed list>**

(S (DS (C1 (NP (DET (Def THE)))
  (NP2 (ADJ (Adj-C TALL)))
  (NP2 (NOUN PLAYER))
  (RC (RP WHO))
  (SUBCL (VP (Tns (past ED)))
  (VI (VT WIN))
  (NP (DET (Def THE)))
  (NP2 (ADJ (Adj-C GOLD)))
  (NP2 (NOUN MEDAL))))))

(VP (Tns (pres S)))
  (VI (VT DRINK))
  (NP (NP2 (NOUN BEER)))))))

From these syntactic structures, we need to identify the
Kor. noun senswu 'player' & Eng. noun player as the
P1's, the Kor. verb tta & Eng. verb win as the
processes, the other nouns in the VP Kor. meytal & Eng.
medal as the P2's within the Definer. We need to keep
track of the syntactic pathways for these examples in order
to identify the nodes of the complex P1. The pathways of
the syntactic outputs for the examples above can be
described as follows:

KOREAN
(for the main clause)
Process = S.DS.CV1.VP.VI.Vt
P1 = S.DS.CV1.NP
P2 = S.DS.CV1.VP.VI.NP
(for the embedded clause)
Process = S.DS.CV1.NP.NP2.PAPL-CL.CV1.VP.VI.Vt
P1 = S.DS.CV1.NP (same as the one in the main clause).
P2 = S.DS.CV1.NP.NP2.PAPL-CL.CV1.VP.VI.NP

ENGLISH
(for the main clause)
Process = S.DS.CV1.VP.VI.Vt
P1 = S.DS.CV1.NP
P2 = S.DS.CV1.VP.VI.NP
(for the embedded clause)
Process = S.DS.CV1.NP.NP2.RC.SUBCL.VP.VI.Vt
P1 = S.DS.CV1.NP (same as the one in the main clause).
P2 = S.DS.CV1.NP.NP2.RC.SUBCL.VP.VI.NP

As shown above, we find long pathways S.DS.CV1.NP.NP2.
PAPL-CL.CV1 (Korean) and S.DS.CV1.NP.NP2.RC.SUBCL (English)
in the pathways for the P2 in the embedded clause. These
pathways are used to indicate that the P2 is the one in the
embedded clause. These pathways can be simplified by using
the strategy of substitution, which we have already used for
the condition of the passive (cf. 5.12). We do not have to
specify the whole pathways. Since the definer is embedded
within the P1, the long pathway for identifying the embedded
clause can be substituted with 'P1.' By doing so, the long
syntactic pathways of the embedded clause can be reduced as
shown below: (Notice that these embedded clauses are
represented as 'Definer' at the level of concept.)

Within Definer:

**KOREAN**

PROCESS = P1.VP.VI.Vt
P1 = P1 in the main clause
P2 = P1.VP.VI.NP

**ENGLISH**

PROCESS = P1.VP.VI.Vt
P1 = P1 in the main clause
P2 = P1.VP.VI.NP

Here it is worthwhile to notice, in the examples above, that the P1 within Definer is realized as a relative pronoun in English and as zero in Korean. No matter what the syntactic realizations of the P1 in Definer are, however, the conceptual representations should be able to indicate the P1 of the main clause as the P1 of the embedded clause. To assign the same argument of the P1 within the Definer as the P1 in the main clause, we pass the argument 'P1' of the main clause to the P1 of the Definer. The conceptual representations for both Korean and English are shown below. They are produced by using the commands on the previous page.

**KOREAN**

**<< Conceptual Representation >>**

(S (CL1 (PROCESS PPI)
  (SUBJ (PPART SENSWU)
   (DEF (PROCESS TTA)
     (PART1 SENSWU)
     (PART2 (PPART KUM MEYTAL))
     (LOC (PPART SWUYENG)))
   (PART1 (PPART SENSWU)
     (DEF (PROCESS TTA)
       (PART1 SENSWU)
       (PART2 (PPART KUM MEYTAL)))


ENGLISH

**< Conceptual Representation >**

(S (CL1 (PROCESS DRINK)
   (SUBJ (GV )
      (EPI TALL)
      (PPART PLAYER)
      (DEF (PROCESS WIN)
         (PART1 PLAYER)
         (PART2 (GV )
            (CLASER GOLD)
            (PPART MEDAL))
         (Tense ED))
      (PART1 (GV )
         (EPI TALL)
         (PPART PLAYER)
         (DEF (PROCESS WIN)
            (PART1 PLAYER)
            (PART2 (GV )
               (CLASER GOLD)
               (PPART MEDAL))
            (Tense ED))
      (PART2 (PPART BEER))
      (Tense S)))

Examples of complex and compound sentences are illustrated below:

>> a sungmaeyse un meytalul ttan sonyenun hankwuk chwulsinul cohaahanta.
   'The boy who won the silver medal in the horse race likes people from Korea.'

**< Conceptual Representation >**

(S (CL1 (PROCESS COHAHA)
   (TOPIC (PPART SONYEN)
      (DEF (PROCESS TTA)
         (PART1 SONYEN)
         (PART2 (PPART UN MEYTAL))
         (LOC (PPART SUNGMA)))
      (PART1 (PPART SONYEN)
         (DEF (PROCESS TTA)
(PART1 SONYEN)
(PART2 (PPART UN MEYtal))
(LOC (PPART SUNGma)))
(PART2 (PPART HANKwuk CHWULSIN))
(Tense N)))

>> b. nanun papul mekko yenghinun sakwalul mekessta.
'I ate some rice and Yenghi ate some apples.'

**< Conceptual Representation >**

(S (CL1 (PROCESS MEK)
 (TOPIC (PPART NA))
 (PART1 (PPART NA))
 (PART2 (PPART PAP)))
 (CL2 (PROCESS MEK)
 (TOPIC (PPART YENgHI))
 (PART1 (PPART YENgHI))
 (PART2 (PPART SAKwa))
 (Tense ESS)))

ENGLISH

>> a. The boy who ate lunch went to school.

**< Conceptual Representation >**

(S (CL1 (PROCESS GC)
 (SUBJ (PPART BOY)
 (DEF (PROCESS EAT)
 (PART1 BOY)
 (PART2 (PPART LUNCH))
 (Tense ED)))
 (PART1 (PPART BOY)
 (DEF (PROCESS EAT)
 (PART1 BOY)
 (PART2 (PPART LUNCH))
 (Tense ED))

(Tense ED)
 (DIR (PPART SCHOOL))))
b. I loved Korea and they loved America.

**< Conceptual Representation >**

(S (CL1 (PROCESS LOVE)
 (SUBJ (PPART I))
 (PART1 (PPART I))
 (PART2 (PPART KOREA))
 (Tense D))
 (CL2 (PROCESS LOVE)
 (SUBJ (PPART THEY))
 (PART1 (PPART THEY))
 (PART2 (PPART AMERICA))
 (Tense D)))
5.2. The Sememic Supplement II

The preceding section describes the role relationships with respect to processes in a general way. However, a properly conceived decoder should not accept, at the conceptual stratum decoding, which combines participants with a conceptually incompatible process (%I drink the table. The symbol % marks conceptually ill-formed sentences). The function of the Sememic Supplement II is to reject such decodings. The information used here to check whether provisional conceptual representation is conceptually well-formed or not is based on the theoretical assumption that each process has its own tactic properties (Lamb 1987, Ikegami 1980, Hayes 1978). It has been observed in Linguistics that most verbs are restricted in the range of items that they may take as participants. For example, the verb 'eat' normally requires a P1 which is 'Human' or 'Animal,' and the verb 'flow' normally has a P1 which is 'Liquid'. Within the framework of generative transformational grammar this phenomenon is referred to as 'selectional restrictions' (cf. Bach 1964, Katz and Fodor 1964). If a taxonomy of entities is interconnected with taxonomy of processes, it is necessary to relate the presupposed participants to specific processes and to indicate the mutual relations in the system. The entries of the Sememic Supplement II are the linear representations of these interconnections. As mentioned in section 5.1, the
reason to set up two sememic supplements results from practical need, not from theoretical requirements.

The present section is concerned with the design of a procedure to check provisional interconnection between a process and its participants. One way of defining this procedure is to check information placed in a dictionary by a linguist. The information includes the general semotactic codes of processes and the dominant properties (cf. 5.1.2) of participants. The decoding process uses this information by matching the general semotactic codes which are entailed by a process with the properties of participants associated with the process. For example, for such a clause as nanun mwulul masinta 'I drink water,' the linguist must have defined in advance the general semotactic code for the process masi 'drink.' The process masi presupposes a participant that has the property 'Animal' or 'Human Being' as the P1 and a participant that has the property 'edible things that are liquid' as the P2. The general semotactics of the lexeme masi 'to drink' can be written as

\[(P1.HB\_P2.ED\_LIQ) \text{ or } (P1.ANI\_P2.ED\_LIQ)\].

The dominant properties of the input participants, na and mwul are 'HB' (HUMAN BEING) and 'ED\_LIQ' (Edible-Liquid), respectively. After this information has been specified, the decoding process can check to see if the properties of the provisional participants match the general semotactic requirement of the process masi. Since the semantic properties of the lexemes na 'I' and mwul 'water' meet
one of the general semotactic statements of the process masi 'to drink,' (i.e. (P1.HB_P1.ED-LIQ)), the input is accepted as appropriate. However, in %I drink the table, the semantic property of the lexeme table (NONED) conflicts with that of the required P2 of the process drink (ED-LIQ) and the provisional decoding is therefore marked as anomalous.

The approach of defining such general semotactic properties and of comparing them with the properties of the arguments is useful not only for rejecting ill-formed clauses but also for disambiguating cases of polysemy, which computer scientists such as Charniak (1984) and Simons (1970) call 'word sense ambiguity.' Although Webster's dictionary gives more than 15 meanings for the lexeme have, let us say for the sake of an example that it realizes three sememes with the meanings of 'possess' (S1: I have money), 'eat' (S2: I had lunch) and 'be related as kin' (S3: I have an aunt). Of several possible sememes, the choice of a single sememe is made by referring to properties of P2. For example, if the lexeme have represents S2, the property of the following argument must be 'edible things that are solid'. If it represents S1, the property of the following argument should be 'concrete objects'. If it represents S3, the P2 must be 'HB' (Human Beings). That is, the concept S2-have has the same semotactic properties as that realized by the lexeme eat, and the concept S1-have has the same semotactic properties
as that realized by the lexeme possess. The network in Figure 5.13 shows these relationships:

```
'possess'(S1)  'be related as kin'(S3)  'eat' (S2)
   possess       have       eat
```

Fig. 5.13.

We now need to raise two questions related to the Sememic Supplement II—1) How can we represent semantic properties and 2) What types of information should be contained in the Sememic Supplement II?

5.2.1. How Can We Represent Semantic Properties?
In connection with this question, we discuss two issues—
(1) Dominant semantic properties for participants and General semotactics for processes, and (2) Notation for this semological information.

(1) Semantic properties for nouns and General semotactics for verbs.

As mentioned above, all semantic properties are interconnected in the form of a network. It is impossible, however, to implement such a relational network directly on a computer. Therefore, we convert the network information to a linear representation. We also introduce a simplification by determining which of several properties of a
concept are dominant intensional properties and including only these in the sememic dictionary. Dominant properties are meant, in this study, as essential properties for a specific purpose. Dominant properties can be different for different purposes. For example, the property 'ED-SOL' (Edible-Solid) for the thing 'apple' carries more weight than the others if the purpose is to check whether the participant 'apple' is possible as the P2 of the process 'drink.' However, if the purpose were to compare 'a green apple' and 'a red apple,' the property 'RED' or 'GREEN' would be marked as distinctive to serve the purpose. In the present study, which may be regarded as a pilot project, only a few properties have been selected as dominant. None of the properties considered outside of the context of a particular concept is inherently more central or dominant than any other. For a particular concept under a specific purpose, however, we can identify certain properties as dominant.

What, then, are the criteria for determining the dominant properties in our system? To find an answer, let us consider the semantic network of Figure 5.14. It is observed in the semantic network of Fig. 5.14 that a property of a higher level is in general implied by a property of a lower level in the hierarchy. For instance, the thing 'apple' resides in the semantic network as one member of the class of edible things that are solid, which in turn resides as one member of the class of inanimate
entities. This property 'inanimate entities' is in turn connected to concrete objects.

Fig. 5.14

In the composition of a concept, this study treats, in most cases, a property at a lower level as the dominant property since it is apt for the purpose, i.e. to check provisional interconnection between participants and specific processes. For example, for the lexeme *leg*, we assign the property 'BP-IA' (Body Part-InAlienable), which resides at the lower level in our semantic hierarchy. However, the properties are not mutually exclusive as we can see in the semantic network in fig. 5.14. They are all related to one another. There can be solid things that are edible (apple) or non-edible (table), and there also can be liquid things
that are edible (water) or non-edible (gasoline). Accordingly, we need to specify the semantic path in some cases, and not always just treat a property at a lower level as the dominant one. So, if it is necessary to assign more than one property as dominant, we have to specify the semantic path as needed. For example, we assign the property 'ED-SOL' for the thing 'apple' since it is needed to distinguish it from the property 'NONED-SOL' for the thing 'table,' and we also distinguish the property 'BP-IA' (for a body part such as 'leg') from 'BP-A' (for a body part such as 'kidney').

In this study, as mentioned before, this specification is made in order to check whether or not the properties of the given input are appropriate for the participants of processes such as drink, eat or give. No attempt is made here to consider the numerous additional properties required to make a fully adequate semantic network. The properties listed in Fig. 5.15 are arbitrarily chosen ones for modeling a system to test our decoding process. This information could be easily elaborated by adding new information, with no change required in the computational process (cf. Chapter 6). The semantic properties considered in this study are listed in Figure 5.15.
Fig. 5.15

Since a taxonomy of participants is interconnected with a taxonomy of processes, we define the general semotactics of processes based on the semantic properties of participants. Figure 5.16 shows examples of the general semotactic statements based on the semantic properties above for verbs like masi 'drink', mek 'eat', ka 'go', and ca 'sleep'.

<table>
<thead>
<tr>
<th>Processes</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>masi 'drink'</td>
<td>HUMAN</td>
<td>ED-LIQ</td>
</tr>
<tr>
<td></td>
<td>ANIMAL</td>
<td>ED-LIQ</td>
</tr>
<tr>
<td>mek 'eat'</td>
<td>HUMAN</td>
<td>ED-SOL</td>
</tr>
<tr>
<td></td>
<td>ANIMAL</td>
<td>ED-SOL</td>
</tr>
<tr>
<td>ka 'go'</td>
<td>HUMAN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANIMAL</td>
<td></td>
</tr>
<tr>
<td>ca 'sleep'</td>
<td>HUMAN</td>
<td>ANIMAL</td>
</tr>
</tbody>
</table>

Fig. 5.16

Sometimes, however, we need to specify a blocking property rather than an enabling one. If the process were
give, we could not take the property 'body parts which are
inalienable' as the property associated with P2. In Korean,
it is impossible to alienate some parts of the body from
somebody, e.g. legs, nose, etc. However, we can alienate
entities that have the property 'ANIMAL,' 'HUMAN BEING,'
'ED-SOL,' 'ED-LIQ,' 'Con-Obj,' 'TIME' 'PLACE,' 'BP-A' or
even 'ABSTRACT'.

Ex. I gave my daughter to him. ('HUMAN BEING')
I gave the cat to her. ('ANIMAL')
I gave the apple to my sister. ('ED-SOL')
I gave some water to him. ('ED-LIQ')
I gave the book to you. ('Con-Obj')
I gave almost eight hours to the cause. ('TIME')
I gave Shan (as the name of a village) to my son.
('PLACE')
I gave my kidney to him. ('BP-A')
I gave all my love to the children. ('ABSTRACT')
%I gave my leg to the boy. ('BP-IA')

The P2 of the process give has many properties that we
could list. However, it would be more economical to list
just one or two negative properties instead of attempting
to list all of the positive properties. By specifying
a single blocking property instead of enumerating eight
enabling properties, we can write much simpler semotactics
for the verb give: nine entries are reduced to one.
Therefore, for simplicity and efficiency, we need to
distinguish a blocking property from an enabling one in
designing a system.
For computer implementation and manageability, we need to formalize this semological information within a general semotactics. The general format is shown in Fig. 5.17.

Cat. # ((E/B, P1Pro.) ((E/B, P2Pro.) ((E/B, P3Pro.))))
(E/B stands for enabling or blocking. The category number will be discussed in the following section).

Fig. 5.17.

This can be rewritten as

Category Number (E/B, P1Pro.)
(for processes that require only one participant)

Category Number (E/B, P1Pro.) (E/B, P2Pro.)
(for processes that require two participants)

Category Number (E/B, P1Pro.) (E/B, P2Pro.) (E/B, P3Pro.)
(for processes that require three participants)

(2) Notations for representing semotactic information.

The first issue raised at the beginning of this section involves semological information. In connection with this issue we consider how this information will be represented
in the form of entries. A part of the Master Dictionary is repeated in Figure 5.19 for reference. (The reader is reminded that semological information is included in the Master Dictionary (cf. 3.3)).

**General Semotactics of Processes**

<table>
<thead>
<tr>
<th>Process Categories</th>
<th>Category Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>((E HUMAN) (E ED-SOL))</td>
</tr>
<tr>
<td>(2)</td>
<td>((E ANI.) (E ED-SOL))</td>
</tr>
<tr>
<td>(3)</td>
<td>((E HUMAN) (E ED-LIQ))</td>
</tr>
<tr>
<td>(4)</td>
<td>((E ANI.) (E ED-LIQ))</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>(12)</td>
<td>((E HUMAN) (B AB-IA) (E HUMAN))</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.18

<table>
<thead>
<tr>
<th>Heading</th>
<th>Syntactic Code</th>
<th>Semological Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANGEYMWUL</td>
<td>1 NOUN</td>
<td>8</td>
</tr>
<tr>
<td>CAP</td>
<td>1 Vt</td>
<td>10 11 12</td>
</tr>
<tr>
<td>CHANGIN</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>CHELKAPSEN</td>
<td>1 NOUN</td>
<td>5</td>
</tr>
<tr>
<td>COHAHA</td>
<td>1 Vt</td>
<td>5 6</td>
</tr>
<tr>
<td>CWU</td>
<td>1 Vt</td>
<td>8 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASI</td>
<td>1 Vt</td>
<td>3 4</td>
</tr>
</tbody>
</table>

Fig. 5.19

Following the format of the Master Dictionary as shown in Figure 5.17, we include in Figure 5.18 a list of combinations of semantic properties and in Fig. 5.19 (a) number(s) for the semotactic categories (for the entries of verbs) and for the semantic properties (for the entries
of nouns). The numbers for nominal entries are symbols arbitrarily chosen to index the semantic properties that go with each item. (The numbers and their corresponding properties are listed in Fig. 5.15). The numbers to the right of the verbal heading in Fig. 5.19 match the category numbers in parentheses in Fig. 5.18; They are the cross-reference numbers. Category (1) consists of two pairs, ((E HUMAN) (E ED-LIQ)). The first pair gives information about Participant 1. The second pair gives information about Participant 2. The second member of the first pair means that the property associated with P1 is property 'HUMAN.' The second member of the second pair means that the property associated with P2 is property 'ED-LIQ.' The capital 'E' as the first member of a pair is an abbreviation for 'Enabling.' The capital 'B' as the first member of a pair is an abbreviation for 'Blocking.' Let us say that one of the semotactic codes for the process give is ((E HUMAN) (B BP-IA) (E HUMAN)). Here, the 'E' of the first pair means that the property associated with the P1 is an enabling property. The 'B' of the second pair means that the property associated with the P2 is a blocking property. The 'E' of the third pair means that the property associated with the P3 is an enabling property. The second member of each pair indicates the property associated with P1, P2 and P3, respectively. Each category consists of information about general semotactic properties. For example, in the Korean entries, the VERB masi 'drink'
contains two category numbers--one has property 'HUMAN' associated with P1 and the other has property 'ANIMAL' associated with its P1. Both have the same property 'ED-LIQ' associated with their P2's. When a verb such as masi 'drink' is looked up in the Master Dictionary, the numbers 3 and 4 are found. These numbers point to Category 3 and Category 4, which consist of ((E HUMAN) (E ED-LIQ)) and ((E ANI.) (E ED-LIQ)), respectively, as indicated in Figure 5.20.

\[
\begin{array}{c|c|c|c|c}
\text{P1} & \text{E HUMAN} & \text{P1} & \text{E ANI.} \\
\text{P2} & \text{E ED-LIQ} & \text{P2} & \text{E ED-LIQ} \\
\end{array}
\]

Fig. 5.20.

In the English entries, the VERB have contains seven category numbers--three categories that have property 'HUMAN' associated with their P1's. These three are distinguished by the property associated with P2. The first has a 'Human' P2. The second has a 'Ed-Sol' P2 and the third has a 'Non-Ed' P2. The other three category numbers have property 'ANIMAL' associated with the P1, following the same pattern as the first three in their P2 properties. That is, the properties of the P1 and the P2 are mapped as follows:

\[
\begin{array}{c|c}
\text{P1} & \text{P2} \\
\text{Human} & \text{Human} \\
\text{Animal} & \text{Ed-Sol} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{P1} & \text{P2} \\
\text{Human} & \text{Non-Ed} \\
\end{array}
\]
The entry *have* therefore has six semotactic codes as shown in Figure 5.21.

![Figure 5.21](image)

In this section, it might appear that the strategy of having the process point to a category which defines the properties of the participant(s) required by that process is too complicated. If the inventory of possible properties becomes large enough to deal with specific cultural phenomena, the possibilities of combined properties could become unwieldy. This computational problem can be easily handled by the machine, however. Moreover, as Keenan (1976) and Osgood & Bock (1977:94) have ascertained, the properties 'living' or 'moving' are crucial for the motivational salience of P1. Accordingly, even though there are numerous properties, more than 50% of the P1 properties should be 'Human' or 'Animal.'
5.2.2. What Types of Information Should Be Contained in Sememic Supplement II?

The previous section explains the numbers for nominal entries as arbitrary symbols for indexing the semantic properties that go with each argument and the category numbers for verbal entries comprising the general semotactic information about processes. Here, we might ask why information about semotactics is put in the Master Dictionary, rather than in a Sememic Supplement. The reason is given in the section on the Master Dictionary (3.3). The Master Dictionary, which includes semological information, allows us to have simpler dictionary entries (cf. 3.3). Thus semological information is stored in the Master Dictionary, rather than in Sememic Supplement II.

The next question concerns what type of information needs to be contained in the Sememic Supplement II if semological information is stored in the Master Dictionary. As mentioned at the beginning, the function of the Sememic Supplement II is to check whether or not the properties of participants match those of the arguments that a specific process presupposes. The crucial question here is how this matching process can be completed without mixing linguistic information with computational procedure. Just as we keep track of the pathways in the syntactic tree structure for constructing a representation of the conceptual level, we can keep track of the pathways of the semantic tree structure for the checking of the general semotactics of
a process. The entries of the Sememic Supplement II is
the list of pathways represented in the semantic structure
tree (cf. Appendix B). By keeping track of semantic nodes
appearing in the semantic tree the matching status between
the process of a clause and its participants is checked.
To describe the process, let us take an example. The
semantic tree structure of nanun papul mekessta,
'I ate some rice' can be drawn as shown in Figure 5.22.

![Diagram of semantic tree structure]

Fig. 5.22

The semantic pathways that lead to the Process mek and
the Participants na and pap can be derived from the
tree above as follows:

S.PROCESS = mek  'eat'
S.P1.PPART = na   'I'
S.P2.PPART = pap  'rice'

From such semantic pathways, we want to check to see if the
semantic properties of na and pap match the general
semotactics of the Process mek. In executing this
checking procedure, we retrieve semological information
from the Master Dictionary. Actually, when the verb
mek is activated for syntactic decoding, the category numbers 1 and 2 are found, which provide the information '((E HB) (E ED-SOL))' and '((E ANI) (E ED-SOL))', respectively. This information is noted for use at a later time. Likewise, when the nouns na 'I' and pap 'rice' are looked up in the Master Dictionary, the property numbers 1 and 4 are found, which point to information 'Human Being' and 'ED-SOL,' respectively. This information found in the noun entries is also noted for use at a later time. Figure 5.23 shows how these data are organized (cf. section 6.2).

![Diagram](image)

**Fig. 5.23**

After the semantic tree is produced with the aid of the Sememic Supplement I, the semotactic information noted from the Master Dictionary entries is referred to. Of the two possible pairs of properties (numbers 1 and 2) for the semotactics of the process mek 'eat,' the first

\(((E HUMAN) (E ED-SOL))\) (number 1) matches the properties
of the participants na 'I' and pap 'rice.' The Noun property numbers 1 and 4 are compared to properties of the P1 na 'I' and the P2 pap 'rice' of the clause. Thus the requirement of the processmek 'eat'

\((\text{E HUMAN}) (\text{E ED-SOL})\)

matches the Noun properties

\text{HUMAN} \quad \text{ED-SOL}

Since the properties match the conceptual representation for this input sentence is accepted as conceptually well-formed. On the other hand, a sentence like %I ate the table shows conflict between the properties as shown below and is therefore rejected.

\begin{tabular}{ll}
\textbf{Verb Property} & \textbf{Noun Properties} \\
\% \((\text{E HB}) (\text{E Ed-SOL})\) & HB \quad \text{Non-Ed} \\
\end{tabular}

The strategy for keeping track of the semantic tree structure can also be applied to complex sentences. If the example were one that contained an embedded sentence such as the tall player who won the gold medal drinks beer, the semantic tree structure produced would be shown in Figure 5.24.
Fig. 5.24.

The properties of the process and the participants within the Definer can be derived by the same strategy used for deriving the process and the participants of the main proposition. The strategy is to keep track of the semantic tree as follows:

- \( S.P1.DEL.PROCESS = \text{win} \)
- \( S.P1.DEL.P1.PPART = \text{player} \)
- \( S.P1.DEL.P2.PPART = \text{medal} \)

The matching process, which is to compare the general semotactic code of the Process with the properties of the Participants within Definer, can be written as

\[ S.P1.DEL.PROCESS : S.P1.DEL.P1.PPART \quad S.P1.DEL.P2.PPART \]

Here, the path \( S.P1.DEL.PROCESS \) retrieves category numbers which consist of properties of P1 and P2, which are presupposed by the process within the Definer. The process \( \text{win} \) has several pairs of participant requirements. Of the several possibilities, only the one which consists of \( ((E \text{ HB}) \ (E \text{ CON-OBJ})) \) will match the properties
of the given participants. The path `S.P1.DEF.P1.PPART` retrieves the property of P1 within the Definer, and the path `S.P1.DEF.P2.PPART` retrieves the property of the P2 within the Definer. The (P1.player) has the property 'HUMAN BEING' and the (P2.medal) has the property 'CON-OBJ'. According to the format of the Sememic Supplement II, Category 5 match as the properties of P1 and P2 as follows:

<table>
<thead>
<tr>
<th>Verb Property</th>
<th>Noun Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E HB) (E CON-OBJ)</td>
<td>(HB CON-OBJ)</td>
</tr>
</tbody>
</table>
CHAPTER 6. ADDING NEW INFORMATION

The main role of language as a social activity is to allows people to add new information to their information systems. Let us imagine that a speaker A and a speaker B are communicating with each other, and A tells B that his mother went back to school. B already knew A's mother, but he did not know the fact that she went back to school. What happens as a result of that communication is that B adds a new connection to his system for the new information that A's mother went back to school. As mentioned in the introduction, if we adopt the viewpoint that thought processes consist of operations on nections, i.e. adding new connections, blocking or activating the existing connections, then we conclude that learning is mainly a process of adding new connections to the existing system. Therefore, in order to design a realistic natural language understanding system, we must enable such a system to recognize new information and to store it properly. Without this ability to expand, a system would have to be complete from the start, which is of course impossible.

We may distinguish two methods of incremental learning that can be considered in designing a system.

(1) Asking and Being Told.
(2) Guessing.

The present system operates only by means of the first method. Children can fill gaps in their knowledge by asking and by being told by their older siblings, playmates, parents, or teachers. When they hear a new lexical item,
they ask 'What is X?' Being told, they add the new item to their systems and change the configuration of their systems. By analogy, we design a model that gets help from a user rather than one which carries out the process of adding new information by guessing. Such a model may be called 'a user-guided model.'

System modification consists mainly of (1) adding new lexical items and (2) adding new syntactic statements. In terms of relational networks, these two refinements are actually the same process, adding new sections (cf. Copeland 1982:166). This study deals only with adding new lexical items, the quantitatively more important process. The rest of this chapter is concerned with an interactive procedure for adding new lexical items (6.1) and with describing the computer data structures for adding new lexical items (6.2).

6.1. An Interactive Procedure for Adding new Lexical Items

When the system encounters a new string of graphemes bounded by spaces or punctuation, it first needs to check whether the string contains a simple typing errors. If not, it is a new graphemic word. In order to check this, the following prompt appears on the screen.

The item ___ is not in the dictionary.
Is this a typo? (y/n)

If a user types 'y' (yes), the system asks
Enter the correct item:

If a user types 'n' (no), the system assumes that a new lexical item has been encountered. In order to add the new lexical item into the system, the system needs to obtain

1) Syntactic information, i.e. (a) lexical categori(es) and
2) Semological information, i.e. semantic properties for NOUNs and semantic codes for Verbs.

As a way of eliciting this information, rather than ask questions as a child would, the system proceeds to interact with the user in a mode that has become customary for computers—by means of menus. It begins with the 'Main Menu', shown in Figure 6.1.

```
** MAIN MENU **
1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT
```

Fig. 6.1

The Main Menu leads to various submenus in a tree structure, described in the remainder of this section (6.1.1 - 6.1.4). The choice of 'Display New Information' is for use after new information has been added.

6.1.1. Help

The choice of 'Help' provides a user's guide for adding
new information for the user who is not familiar with the notation of the system. This guide provides information about how to add a new lexical item. If the user is already familiar with the notation used in the system, he does not have to consult the 'Help' screen for reference. If the user calls for Help, Figure 6.2 appears on the screen.

-In order to add a new item, you need to specify the number of the lexical categories and to identify the names of the lexical categories from the Menu for lexical categories.

-You also need to choose the semantic properties that can be associated with the entity, or the semotactic code that can be applicable to the process from the Menu.

-If an appropriate semantic property or a semotactic code is not included in the Menu, you can add one by entering the command '0.' Command '0' calls up a prompt that asks you to identify the name of a new semantic property or a new semotactic code.

Fig. 6.2.

6.1.2 'Add New Information' Option

This section introduces the user to interaction with the system for adding new lexical items. During most of the work for adding new information, the user is in direct communication with the machine.

When the user chooses the option for adding new information from the Main Menu in Fig. 6.1, a menu for lexical categories considered in the system is displayed as shown in Figure 6.3. The new item is also shown under the Lexical Category Menu for the user's convenience.
** Lexical Category Menu **

*<TYPE1 Lexical Categories>*
1. NOUN

*<TYPE2 Lexical Categories>*
2. Vt   3. Vi   4. COPVERB   5. BIVERB

*<TYPE3 Lexical Categories>*
10. ADV   11. RP   12. Def  13. nPPT
14. nPPL  15. nPPAG  16. nPPS  17. nPPIN
18. nPPQ  19. nPDDAT  20. nPPD  21. nPPAB
22. nPPINS 23. pres  24. past  25. fut

THE NEW ITEM =

Fig. 6.3

As a way of asking about the 'class cleavage' phenomenon (Bloomfield 1933:) of the new item, the next prompt appears as:

Lexical Category Numbers for this item >>

The number provided by the user at this prompt gives the system information not only about class cleavage, but also about the proper time to exit from the Lexical Category Menu. If the user enters the number '1' at the above prompt, this means both that 'The new lexical item has a single lexical category,' and that 'the next prompt should be displayed only once before exiting from the Lexical Category Menu.' If he enters the number '2' at the above prompt, this means both that 'The new lexical item has two lexical categories,' and that 'the next prompt should be displayed twice before exiting from the Lexical Category
Menu.' After entering the number of lexical categories into which a new lexical item can fit, the user is asked to enter the names of the lexical categories, e.g. NOUN, Vt, etc.

Enter the number of the lexical category >>

At this point the user has three choices which will be illustrated. The three choices are:

A. To choose NOUN
B. To choose VERB
or
C. To choose NOUN and VERB

A. To Choose Noun.

If a user intends to add the new item 'orange' (as in I ate the orange), he enters the number '1' (NOUN) at the prompt:

Enter the number of the lexical category =

Then, a sub-menu for semantic properties is displayed along with the new lexical item and its lexical category as shown in Figure 6.4.
** <TYPE 1 Semantic Property Menu> **

0. NewCategory
1. HUMAN
2. ANIMAL
3. ED-LIQ
4. ED-SOL
5. CONC-OBJ
6. PLACE
7. BP-IA (Body Part-InAlienable)
8. SPORTS
9. ABSTRACT
10. PLANT

THE NEW ITEM = ORANGE
THE LEX. CAT = NOUN

Fig. 6.4

The next prompt asks the user to enter the number of semantic property that can be associated with the new item 'orange.'

Enter the number of a semantic property >>

At this prompt, the user has two choices--1) to choose a property that can be associated with the new item 'orange' from the menu in Fig. 6.4, or 2) to add a new property to the existing menu and associate that with the new item. Thus the user can either choose the property 'ED-SOL' (4) into which the item 'orange' can fit or add the property 'FRUIT' if he wants to specify a more specific kind of property. In order to add a new property, the user enters the command '0' at the prompt:

Enter the number of a semantic property >>

In this case, the system asks for the name of the new property with the prompt:

Enter the name of the new semantic property >>
The user enters the name of the new property as follows:

Enter the name of a new semantic property >> FRUIT

The addition of the new property FRUIT requires the user to modify the existing Semotactic Code Menu. Since this property was not present before adding it, the user needs to add new semotactic codes for the verb eat. The existing semotactic codes for the verb eat are

((E HB) (E ED-SOL)) and ((E ANI) (E ED-SOL)). To these codes the user adds the semotactic codes ((E HB) (E FRUIT)) and ((E ANI) (E FRUIT)) by entering command '0.' The procedure to add a new semotactic code is explained in what follows.

B. To Choose VERB (Vt, Vi, etc.)

If the user intends to add the new item 'embrace' (as in I will embrace the idea or I will embrace her), he enters the number '2' (Vt) at the prompt

Enter the number of the lexical category >>

Then a sub-menu for semotactic codes is displayed along with the new lexical item and its lexical category as shown in Figure 6.5.
** <Type 2 Semotactic Code Menu> **

*<TYPE2>*
1. HUMAN  2. ANIMAL  3. ED-LIQ  4. ED-SOL  5. CONC-OBJ

*<TYPE2>*
0. New Code
1. E HUMAN E ED-SOL  2. E ANIMAL E ED-SOL
7. E ANIMAL ANY-THING ANY-THING
8. E HUMAN B BP-IA E HUMAN
9. E HUMAN B BP-IA E ANIMAL
10. E ANIMAL E HUMAN  11. E HUMAN E HUMAN
16. E HUMAN E ANIMAL E HUMAN
17. E HUMAN E ED-LIQ E HUMAN
18. E HUMAN E ED-SOL E HUMAN

THE NEW ITEM = EMBRACE
THE LEX. CAT. = Vt

Fig. 6.5

Since each process defines its own participants
(cf. Section 5.2), this menu displays semotactic codes
(i.e. combinations of semantic properties). The menu for
semantic properties is also displayed in case the user
intends to add a new semotactic code. When the user enters
a new semotactic code, he needs to refer to the semantic
properties of arguments which are presupposed by a process.
Since a process can require more than one semotactic code,
the following prompt appears in order to elicit the number
of semotactic codes that are required by the process.

Enter the number of semotactic code(s) applicable to
this item >>

In the case of the item embrace, a user can consider
two cases such as I will embrace her or I will embrace
the theory. So, at the above prompt, the user enters '2.'

Enter the number of semotactic code(s) applicable to this item >> 2

Then, the prompt which asks the number of a semotactic code is displayed twice. (As explained in section 6.1.2, the number '2' at the above prompt gives the system information not only about class cleavage, but also about the proper time to exit from the Lexical Category Menu).

Enter the number of a semotactic code >>
Enter the number of a semotactic code >>

At these prompts, a user can enter two category numbers—one for (P1.HB P2.HB) (number 11) for the input I will embrace her and the other for (P1.HB P2.ABS) for the input I will embrace the theory. The user chooses the semotactic category number '11' which stands for (P1.HB P2.HB) from the Semotactic Code Menu (Fig. 5.5). However, the user needs to add a new semotactic code for (P1.HB P2.ABS) since this code is not included in the Menu (Fig. 5.5). Therefore, the user enters the number '11' (for the semotactic code (P1.HB P2.HB)) and '0' (for adding the new semotactic code (P1.HB P2.ABS)).

Enter the number of a semotactic code >> 11
Enter the number of a semotactic code >> 0

Command '0' calls up the following prompt:

Enter a new semotactic code >>

In order to enter a new somotactic code, the user is required to follow the notations used in this system. (See section 4.2 for detailed information). The new somotactics for the process embrace (as in I will embrace the theory)
requires the property 'Human Being' for P1 and the property 'Abstract' for P2. These properties are both enabling ones. Therefore, the user enters the semotactic code
(E HB E ABS) at the following prompt:

Enter the new semotactic code >> E HB E ABS

C. Choose NOUN and VERB

If a user wants to add the new lexical item plant which can be either Vt or NOUN (as in I planted the tree or I like the plant), he enters the number '2' at the prompt:

Enter the number of lexical categori(es) applicable to this item >>

Next the following prompt is displayed.

Enter the number of the lexical category =

After the user responds, the same prompt will appear again because the user has specified 2 lexical categories. He then is asked to identify the second category. In this case the user chooses the number for NOUN ('1') and the number for Vt ('2').

Enter the number of the lexical category >> 1

Enter the number of the lexical category >> 2

He then exits from the Lexical Category Menu. If the lexical category 'NOUN' is chosen, the menu for semantic properties in Fig. 6.4 is displayed. The user then follows the prompts as described in section 6.2.1 (i.e. for NOUNs). In relation to the lexical category 'Vt,' the menu
for semotactic codes in Fig. 6.5 is displayed. The user then follows the prompts as described in section 6.2.1 (i.e. for Verbs). System then returns to Main Menu.

6.1.3 Display New Information

After providing new information, the user will generally want to save the information in the dictionary. Before saving the information, however, he may want to check the information since he might have made a mistake. In order to confirm the information, the user calls up the option for displaying new information. The new information is then displayed. In the case of the item orange, the following information is shown:

** <Information about a new item> **

THE LEXICAL ITEM = ORANGE
Lexical Category No. for this item = 1
THE LEX. CAT. = NOUN PROP. = FRUIT

In the case of the item embrace, the following information is displayed:

** <Information about a new item> **

THE LEXICAL ITEM = EMBRACE
Lexical Category No. for this item = 1
THE LEX. CAT = Vt Semotactic Code = E HB E HB, E HB E ABS

After this new information is displayed, system returns to Main Menu.
6.1.4 Exit

After looking at the new information, the user chooses Exit option from the Main Menu (Fig. 5.1). If he is satisfied with the information as shown on the screen, he enters command 'y' (yes) at the following prompt:

Do you want to save this item? (y/n)

The new item is then stored in the Master Dictionary, and a parse tree and a conceptual representation is displayed for the sentence in which the new word was present. For instance, after adding the item *orange* in order to parse the input *I ate the orange*, the lexical item *orange* is stored in the Master Dictionary, and a parse tree and a conceptual representation for the input are displayed (see the output on the next page). The same procedure as described in Chapters 3, 4 and 5 is applied in comprehending the given sentence.

On the other hand, if the user made a mistake while he was adding new information (e.g. typos, wrong properties, wrong semotactic codes, etc), he can type 'n' (no) at the above prompt—the information is disregarded and he enters the new information again.

As a way of summarizing what we have described so far, we illustrate the process for adding new lexical items *orange* and *embrace*. 
ORANGE

>> I ate the orange.

ERROR> The item ORANGE is not in the dictionary.

Is this misspelled?(y/n) n

Do you want to add the item?(y/n) y

***< MAIN MENU >***

1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT

COMMAND >> 2

***<Lexical Category Menu>***

*<TYPE1 Lexical Categories>*
1. NOUN
*<TYPE2 Lexical Categories>*
2. Vt  3. Vi       4. COPVERB  5. BIVERB
*<TYPE3 Lexical Categories>*
6. InDef 7. CONJ  8. pl  9. IPVt
10. ADV 11. RP  12. Def 13. nPPT
14. nPPL 15. nPPAG 16. nPPS 17. nPPIN
18. nPPQ 19. nPPDAT 20. nPPD 21. nPPAB
22. nPPINS 23. pres 24. past 25. fut

THE NEW ITEM = ORANGE
Enter the number of lexical categories applicable to this item : 1
Enter the number(s) of lexical categori(es):
LEX. CAT. No. >> 1

**<TYPE1 Semantic Property Menu>**

0. NewCategory
1. HUMAN
2. ANIMAL
3. ED-LIQ
4. ED-SOL
5. CONC-OBJ
6. PLACE
7. BP-IA
8. SPORTS
9. ABSTRACT
10. PLANT
   THE NEW ITEM = ORANGE
   THE LEX CATEGORY = NOUN
Enter the Prop. No. >> 4

***< MAIN MENU >***

1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT

COMMAND >> 3

**<Information about a new item>**
The new item = ORANGE
Lexical Category N. for this item : 1
LEX. CAT. = NOUN PROP. NO. = 4

***< MAIN MENU >***

1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT

COMMAND >> 4

Do you want to add the new vocabulary?(y/n) y
**<The parsed list>**

(S (DS (Cl (NP (NP2 (Noun I))))
   (VP (Tns (past ED)))
   (VI (Vt EAT))
   (NP (DET (Def THE))
      EMPTY))
   (NP2 (Noun ORANGE)))))))

**< Conceptual Representation >**

(S (Cl1 (Process EAT)
   (Subj (Ppart I))
   (Part1 (Ppart I))
   (Part2 (Gv)
      (Ppart ORANGE))
   (Tense ED)))

EMBRACE

>> I will embrace her.

ERROR> The item EMBRACE is not in the dictionary.

Is this misspelled?(y/n) n

Do you want to add the item?(y/n) y

***< MAIN MENU >***

1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT

COMMAND >> 2

THE NEW ITEM = EMBRACE
Enter the number of lexical categories applicable to this item : 1
Enter the number(s) of lexical category(ies):
LEX. CAT. No. >> 2
**< TYPE2 Semotactic Code Menu >**

*<TYPE1>*
1. HUMAN  2. ANIMAL  3. ED-LIQ  4. ED-SOL  5. CONC-OBJ
*<TYPE2>*
0. New Category
1. E HUMAN  E ED-SOL
2. E ANIMAL  E ED-SOL
3. E HUMAN  E ED-LIQ
4. E ANIMAL  E ED-LIQ
5. E HUMAN  E CONC-OBJ
6. E HUMAN  ANY-THING  ANY-THING
7. E ANIMAL  ANY-THING  ANY-THING
8. E HUMAN  B BP-IA  E HUMAN
9. E HUMAN  B BP-IA  E ANIMAL
10. E ANIMAL  E HUMAN
11. E HUMAN  E HUMAN
12. E HUMAN
13. E ANIMAL
14. E PLANT
15. E HUMAN  E CONC-OBJ  E HUMAN
16. E HUMAN  E ANIMAL  E HUMAN
17. E HUMAN  E ED-LIQ  E HUMAN
18. E HUMAN  E ED-SOL  E HUMAN
19. E HUMAN  B SPORTS
20. E PLACE  E PLANT
THE NEW ITEM = EMBRACE
THE LEX. CAT. = Vt

Enter the number of semotactic codes applicable to this item : 2
SEMITACTIC CAT. NO. >> 11
SEMITACTIC CAT. NO. >> 0
Enter the new semotactic code : e human e abstract

**< MAIN MENU >**

1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT

COMMAND >> 3

**<The information about adding a new item>**
The new item = EMBRACE
Lexical Category No. for this item : 1
LEX. CAT. = Vt
SEMITACTIC CAT. NO. = 11 22
***< MAIN MENU >***

1. HELP
2. ADD NEW INFORMATION
3. DISPLAY NEW INFORMATION
4. EXIT

COMMAND >> 4

Do you want to add the new vocabulary?(y/n) y

**<The parsed list>**

(S (DS (C1 (NP (NP2 (NOUN I))))
   (VP (Tns (fut WILL))
    (VI (Vt EMBRACE))
    (NP (NP2 (NOUN HER)))))

**< Conceptual Representation >**

(S (CL1 (PROCESS EMBRACE)
   (SUBJ (PPART I))
   (PART1 (PPART I))
   (PART2 (PPART HER))
   (Tense WILL)))

After the new item embrace is added to the linguistic system along with the necessary information about conceptual properties this item can be used in other input strings.

For example:

>> I embraced her.

**<The parsed list>**

(S (DS (C1 (NP (NP2 (NOUN I))))
   (VP (Tns (past D))
    (VI (Vt EMBRACE))
    (NP (NP2 (NOUN HER)))))

**< Conceptual Representation >**

(S (CL1 (PROCESS EMBRACE)
   (SUBJ (PPART I))
   (PART1 (PPART I))
   (PART2 (PPART HER))
   (Tense D))

_
6.2. Computer Data Structures for Adding NewLexical Items

This section is devoted to describing a data structure for adding new lexical items. In regard to this topic, the data structure of the Master Dictionary needs to be discussed since a new lexical item will be added to this existing structure.

When we 'load' a dictionary into main memory, a data structure for the dictionary is constructed in main memory. Therefore, to 'load' the Master Dictionary includes to organize a data structure for storing the information in the Master Dictionary. The data structure employed in this system is shown in Fig. 6.6.

![Diagram](image)

When loading the Master Dictionary, memory space which can temporarily hold 300 lexical items is set up in the form of arrays. (We need to keep in mind that each morpheme is
referenced by numbers. cf. 3.3). Each term of the array is a record which consists of several fields for information about lexical categories and semantic properties. If the Master Dictionary includes 150 lexical items, the 150 (of the 300 potential) positions are filled with the items. The pointer indicated by the downward arrow at the top of Fig 6.6 indicates about how many arrays are filled at the current state. Each position points to a record which consists of three fields--(1) associated lexical category ('sym'), (2) semantic property ('prop') and (3) 'next' for another possible lexical category. The field 'prop' points to another record which consists of two fields--(1) 'no' for semological information for a lexical category. (If 'sym' = NOUN, then 'no' = the number of a semantic property, and if 'sym' = VERB, then 'no' = the number of a semotactic code), (2) 'next' for other possible semological information.

Let us consider an example. If the Master Dictionary entry for the item plant is as follows,

\[
\begin{array}{ll}
\text{plant} & \text{NOUN 5} \\
\text{VERB} & 18
\end{array}
\]

and the item resides fifth position in the Master Dictionary, this entry is stored in main memory as shown in Figure 6.7 in accordance with the data structure shown in Fig. 6.6.
The number '5' (which the field 'prop' under the NOUN points to) represents the number in a property array, which is shown in Figure 6.8. The pointer indicates the present number of semantic properties.

The number '18' (which the field 'prop' under the Vt points to) represents the code number in a semotactic array, each of the numbers consists of properti(es) of presupposed argument(s). The semotactic array is shown in Figure 6.9. The pointer indicates the present number of semotactic codes.

Having explained the data structure of the Master Dictionary, we now turn our attention to how the existing
data structure is changed in adding new lexical items. When we add a new lexical item, the pointer which indicates the current number of lexical items in the Master Dictionary moves to the next element. If the Master Dictionary includes 150 lexical items as shown in Fig. 6.6, the data structure of the Master Dictionary is changed as shown in Figure 6.10 after adding the new lexical item orange.

![Diagram](image)

**Fig. 6.10.**

The number '4' represents the semantic property 'ED-SOL' which is associated with the new argument orange. If a user wants to specify a kind of property, and wants to add the property 'FRUIT,' he can add it to the existing data structure of semantic properties in Fig. 6.8. After adding the property 'FRUIT,' the structure in Fig. 6.8 looks like the one shown in Figure 6.11.

![Diagram](image)

**Fig. 6.11.**

If the user wants to add the new item embrace, he needs to add a new semotactic code to the existing data structure.
The new code consists of \((P1.HB \; P2.ABS)\), e.g. I embraced the theory. After the user provides the new semotactic code by entering command '0,' the data structure for semotactic codes in Fig. 6.9 is extended to the one shown in Figure 6.12. Notice that the pointer has moved to the next position and the present number of semotactic codes has increased by one.

\[
\begin{array}{ccccccc}
1 & 2 & 3 & \ldots & 19 & \ldots & 50 \\
\end{array}
\]

\[
\begin{array}{ll}
P1 & HB \\
P2 & ABS \\
\end{array}
\]

Fig. 6.12.
APPENDIX A

Appendix A illustrates the product of each processor (morphological, syntactic and semantic) with the aid of an example from Korean and English.

Korean

```
Conceptual Rep. \{ (Process cohaha) (P1 na) (P2 sakwa) (Tense n) \} : SSII

Semantic Tree : SSI

\{ Syntactic Tree : LSII \}

\{ NOUN nTop NOUN nObj Vt PreT DEC(N) : MD \}

\{ na nun sakwa lul cohaha n ta : MSI \}

\{ nanun sakwalul cohahanta : G \}

Graphemic Rep. nanun sakwalul cohahanta : Input 'I like apples'
```
*1 <Syntactic Tree>

```
S
  /\   |
DS SE  |
  |   C1
  |   |
NP   VP
  |   |
NP2 nM VI Tns
  |   |
NOUN NP
  |   |
NP2 nM PreT DEC(N)
  |   |
NOUN n n n n
  |   |
na nun sakwa lul cohaha n ta
```

*2 <Semantic Tree>

```
S
  /\   |
PROCESS P1 = TOPIC P2 TENSE
  |   |
cohaha na sakwa n
```
English

Conceptual Rep.  
(P1 Gv.boy) (P2 New.IBM PC) : SSII
(Tense s) (Aspect have en)

Semantic Tree : SSI

Syntactic Tree*1 : LSII

Def NOUN PreT Perf Vt InDef NOUN : LSI
Def NOUN PreT Aux-Have Papl Vt InDef NOUN NOUN : MD

Morphological Decoder

The boy’s have en buy an IBM PC : MSII
The boy have s buy en an IBM PC : MSI
The boy has bought an IBM PC : G

Graphemic Rep.  
The boy has bought an IBM PC : Input
*1 <Syntactic Tree>

```
S
 /    |
DS   C1
    /   |
   NF   VP
   /    |
  DET NP2 Tns Aspect VI
   /    /
Def NOUN PreT Perf. Vt NP
```

the boy's have en buy IBM PC

*2 <Semantic Tree>

```
S
 /    |
PROCESS P1 = SUBJ P2 TENSE ASPECT
    /    |
   Gv PPART New PPART
```

buy boy IBM PC s have en
APPENDIX B

<Dictionaries for Korean>

Morphemic Supplement I:

$ I H I L I K I X S I U S I P E L I Y S S S T A A N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
Master Dictionary:

# 47 Lexical Categories:
TYPE1 = NOUN
TYPE2 = Vt Vi
TYPE3 = Adv NId-Vb PasT cAdMT tAdv cAdOPR cAdR nAdT
        nAdDat nAdR nAdL nAdAG NUAdj nAdAG HI Id-Vb
        nSubj nTop FutT nDat-Pol Conj PROGRESSIVE DUAdj
        CLUAdj FPM cAdPUR nAdINS nAdD nObj cAdCD cAdST
        PasPM AtTM PreT PDUAdj PrePM PERFECTIVE nCount
        Dec-N cAdIR cAdCS Dec-Pol POSS Adj

# 9 TYPE1 Property Categories:
1  HUMAN
2  ANIMAL
3  ED-LIQ
4  ED-SOL
5  CON-OBJ
6  PLACE
7  BP
8  SPORTS
9  ABSTARCT

# 17 TYPE2 Property Categories:
1  1 1 1 4
2  1 2 1 4
3  1 1 1 3
4  1 2 1 3
5  1 1 1 5
6  1 1 0 0
7  1 2 0 0
8  1 1 2 7 1 1
9  1 1 2 7 1 2
10 1 1 1 1
11 1 1 1 2
12 1 1 1 7
13 1 2 1 1
14 1 1
15 1 2
16 1 1 1 7
17 1 2 1 7

# Lexical Items:
AN    2 Adv
      Vt
ANI   1 NId-Vb
ASS   1 PasT
CAL   1 Adv
CAMACA 1 cAdMT
CANGEYMWUL 1 NOUN
CAP   1 Vt
CHA   1 Vt
CHANGIN 1 NOUN
CHELPEN 1 NOUN
CHEPHOTANGHA 1 Vi
CHINGCHANPAT 1 Vi
<table>
<thead>
<tr>
<th>Word</th>
<th>POS</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHONG</td>
<td>NOUN</td>
<td>5</td>
</tr>
<tr>
<td>CHOYCHO</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>CHWULSIN</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>CHWUNGMWUKONG</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>CIKUM</td>
<td>tAdv</td>
<td></td>
</tr>
<tr>
<td>CIMAN</td>
<td>cAdOPR</td>
<td></td>
</tr>
<tr>
<td>CIP</td>
<td>NOUN</td>
<td>6</td>
</tr>
<tr>
<td>COHAHA</td>
<td>Vt</td>
<td>5 6</td>
</tr>
<tr>
<td>CU</td>
<td>Vt</td>
<td>8 9</td>
</tr>
<tr>
<td>CUNGMYENG</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>ECEY</td>
<td>tAdv</td>
<td></td>
</tr>
<tr>
<td>EPS</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>ESE</td>
<td>cAdR</td>
<td></td>
</tr>
<tr>
<td>ESS</td>
<td>PasT</td>
<td></td>
</tr>
<tr>
<td>EY</td>
<td>nAdT</td>
<td></td>
</tr>
<tr>
<td>EYKEY</td>
<td>nAdDat</td>
<td></td>
</tr>
<tr>
<td>EYSE</td>
<td>nAdR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nAdL</td>
<td></td>
</tr>
<tr>
<td>HA</td>
<td>Vt</td>
<td></td>
</tr>
<tr>
<td>HALMENI</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>HAN</td>
<td>NUAdj</td>
<td></td>
</tr>
<tr>
<td>HANKWUK</td>
<td>NOUN</td>
<td>6</td>
</tr>
<tr>
<td>HANTHEY</td>
<td>nAdAG</td>
<td></td>
</tr>
<tr>
<td>HI</td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Id-Vb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nSubj</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>ISS</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>KA</td>
<td>Vi</td>
<td>14 15</td>
</tr>
<tr>
<td></td>
<td>nSubj</td>
<td></td>
</tr>
<tr>
<td>KAYIN</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>KEPUKSEN</td>
<td>NOUN</td>
<td>5</td>
</tr>
<tr>
<td>KES</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>KEYSS</td>
<td>FutT</td>
<td></td>
</tr>
<tr>
<td>KI</td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>KIEK</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>KKEY</td>
<td>nDat-Pol</td>
<td></td>
</tr>
<tr>
<td>KO</td>
<td>Conj</td>
<td></td>
</tr>
<tr>
<td>KONGPU</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>KOYANGI</td>
<td>NOUN</td>
<td>2</td>
</tr>
<tr>
<td>KSP</td>
<td>PROGRESSIVE</td>
<td></td>
</tr>
<tr>
<td>KU</td>
<td>DUAdj</td>
<td></td>
</tr>
<tr>
<td>KUM</td>
<td>ClUAdj</td>
<td></td>
</tr>
<tr>
<td>KWENTHWU</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>KWUNGTO</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>KYENGCHAL</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>KYENGKI</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>L</td>
<td>FPM</td>
<td></td>
</tr>
<tr>
<td>LE</td>
<td>cAdPUR</td>
<td></td>
</tr>
<tr>
<td>LI</td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>nAdINS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nAdD</td>
<td></td>
</tr>
<tr>
<td>LUL</td>
<td>nObj</td>
<td></td>
</tr>
<tr>
<td>LYEKO</td>
<td>cAdPUR</td>
<td></td>
</tr>
<tr>
<td>Word</td>
<td>Morphological Form</td>
<td>Tag</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>MAL</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>MANTU</td>
<td>1 Vt</td>
<td>6</td>
</tr>
<tr>
<td>MASI</td>
<td>1 Vt 6</td>
<td>--</td>
</tr>
<tr>
<td>MEK</td>
<td>1 Vt 1</td>
<td>1 2</td>
</tr>
<tr>
<td>MEYTAL</td>
<td>1 NOUN</td>
<td>5</td>
</tr>
<tr>
<td>MOK</td>
<td>1 NOUN</td>
<td>7</td>
</tr>
<tr>
<td>MWUL</td>
<td>2 NOUN Vt</td>
<td>10 11 13</td>
</tr>
<tr>
<td>MYEN</td>
<td>1 cAdCD</td>
<td></td>
</tr>
<tr>
<td>MYENSE</td>
<td>1 cAdST</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>3 PasPM AttM PreT</td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>NALSSI</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>NAMCA</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>NAY</td>
<td>2 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>NAYIL</td>
<td>1 tAdv</td>
<td></td>
</tr>
<tr>
<td>NAYLI</td>
<td>1 Vi</td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>NOLYEK</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>NUN</td>
<td>3 PrePM nTop PreT</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>1 Vi</td>
<td></td>
</tr>
<tr>
<td>OLAY</td>
<td>1 Adv</td>
<td></td>
</tr>
<tr>
<td>PAL</td>
<td>1 NOUN</td>
<td>7</td>
</tr>
<tr>
<td>PANG</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>PAP</td>
<td>1 NOUN</td>
<td>4</td>
</tr>
<tr>
<td>PAT</td>
<td>1 Vt</td>
<td>5</td>
</tr>
<tr>
<td>PAY</td>
<td>1 NOUN</td>
<td>7</td>
</tr>
<tr>
<td>PAYM</td>
<td>1 NOUN</td>
<td>2</td>
</tr>
<tr>
<td>PELI</td>
<td>2 PERFECTIVE Vt</td>
<td>6</td>
</tr>
<tr>
<td>PHAL</td>
<td>1 NOUN</td>
<td>7</td>
</tr>
<tr>
<td>PHOWHAN</td>
<td>1 NOUN</td>
<td>8</td>
</tr>
<tr>
<td>PI</td>
<td>1 NOUN</td>
<td></td>
</tr>
<tr>
<td>PPALLI</td>
<td>1 Adv</td>
<td></td>
</tr>
<tr>
<td>PPI</td>
<td>1 Vt 16 17</td>
<td></td>
</tr>
<tr>
<td>SA</td>
<td>1 Vt 1 3 5 11</td>
<td></td>
</tr>
<tr>
<td>SAKWA</td>
<td>1 NOUN</td>
<td>4</td>
</tr>
<tr>
<td>SALAM</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>SAYNGKAK</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>SE</td>
<td>1 cAdR</td>
<td></td>
</tr>
<tr>
<td>SENSEWU</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>SEOUL</td>
<td>1 NOUN</td>
<td>6</td>
</tr>
<tr>
<td>SEYKYEY</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>SI</td>
<td>1 nCount</td>
<td></td>
</tr>
<tr>
<td>SONYEN</td>
<td>1 NOUN</td>
<td>1</td>
</tr>
<tr>
<td>SS</td>
<td>1 PasT</td>
<td></td>
</tr>
<tr>
<td>SSO</td>
<td>1 Vt</td>
<td></td>
</tr>
<tr>
<td>SU</td>
<td>1 NOUN</td>
<td>9</td>
</tr>
<tr>
<td>SUNGMA</td>
<td>1 NOUN</td>
<td>8</td>
</tr>
<tr>
<td>Word</td>
<td>Type</td>
<td>Number</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>SWUYENG</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>TA</td>
<td>Dec-N</td>
<td></td>
</tr>
<tr>
<td>TAKA</td>
<td>cAdIR</td>
<td></td>
</tr>
<tr>
<td>TANCHE</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>TANGHA</td>
<td>Vt</td>
<td>9</td>
</tr>
<tr>
<td>TANI</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>TELATO</td>
<td>cAdCS</td>
<td></td>
</tr>
<tr>
<td>TENCIKI</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>TENNIS</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>THEYKWENTO</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>TO</td>
<td>cAdCS</td>
<td></td>
</tr>
<tr>
<td>TONG</td>
<td>ClUAdj</td>
<td></td>
</tr>
<tr>
<td>TONGSAYNG</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>TOY</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>TTA</td>
<td>Vt</td>
<td>5</td>
</tr>
<tr>
<td>TULLI</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>UL</td>
<td>nObj</td>
<td></td>
</tr>
<tr>
<td>ULO</td>
<td>nAdINS</td>
<td></td>
</tr>
<tr>
<td>UN</td>
<td>nTop</td>
<td>2</td>
</tr>
<tr>
<td>UPNITA</td>
<td>Dec-Pol</td>
<td></td>
</tr>
<tr>
<td>UY</td>
<td>POSS</td>
<td></td>
</tr>
<tr>
<td>WULI</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>YEL</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>YELSIMHI</td>
<td>Adv</td>
<td></td>
</tr>
<tr>
<td>YENGHI</td>
<td>NOUN</td>
<td>1</td>
</tr>
<tr>
<td>YEPPU</td>
<td>Adj</td>
<td></td>
</tr>
<tr>
<td>YUTO</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>TTALKI</td>
<td>NOUN</td>
<td>4</td>
</tr>
</tbody>
</table>
Lexemic Supplement I

<table>
<thead>
<tr>
<th>#</th>
<th>0</th>
<th>HANKWK</th>
<th>0</th>
<th>CHWULSIN</th>
<th>-1</th>
<th>NOUN</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>KO</td>
<td>0</td>
<td>ISS</td>
<td>-1</td>
<td>PROGRESSIVE</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>KWENTHwu</td>
<td>0</td>
<td>KYUNGKI</td>
<td>-1</td>
<td>NOUN</td>
<td>8</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>PAL</td>
<td>0</td>
<td>MOK</td>
<td>-1</td>
<td>NOUN</td>
<td>7</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>SEYKYEY</td>
<td>0</td>
<td>CHOYCHO</td>
<td>-1</td>
<td>NOUN</td>
<td>9</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>SUNGMA</td>
<td>0</td>
<td>SENSEWU</td>
<td>-1</td>
<td>NOUN</td>
<td>1</td>
</tr>
</tbody>
</table>
Lexemic Supplement II

S    DS  SE
DS   C1  CONJ  C2
DS   C1
DS   MC
C2   NP    VP
MC   C1  ADVM  NP  VP
C1   NP    VP
C1   VP
CONJ  Conj
ADVM  cAdST
ADVM  cAdMT
ADVM  cAdR
ADVM  cAdCS
ADVM  cAdOPR
ADVM  cAdCD
ADVM  cAdPUR
ADVM  cAdIR
NP   NP2  nM
NP2  NOUN
NP2  UADJ  NP2
NP2  Att-Adj  NP2
NP2  PAPL-CL  NOUN
UADJ  NP2  POSS
UADJ  DUAdj
UADJ  NUAdj
UADJ  PDUAdj
UADJ  CLUAdj
Att-Adj  Adj  AttM
PAPL-CL  C1  Pap1M
Pap1M  PasPM
Pap1M  PrePM
Pap1M  FPM
nM   nSubj
nM   nTop
nM   nObj
nM   nCount
nM   EMPTY
VP   ADJUNCT  VI  vV
VI   NP    Vt
VI   Vi
VI   Vt
VI   Vt  PASSIVE
VI   Adj
VI   COMPLEMENT
ADJUNCT  ADV  ADJUNCT
ADJUNCT  EMPTY
COMPLEMENT  NP2  Id-Vb
COMPLEMENT  NP2  NIId-Vb
ADV   tAdv
ADV   Adv
ADV   TADV
| ADV  | LADV  |
| ADV  | DADV  |
| ADV  | AGADV |
| ADV  | DatADV |
| ADV  | INSADV |
| TADV  | NP2  | nAdT |
| LADV  | NP2  | nAdL |
| DADV  | NP2  | nAdD |
| AGADV  | NP2  | nAdAG |
| DatADV  | NP2  | nAdDat |
| INSADV  | NP2  | nAdINS |
| Tns  | Past |
| Tns  | Pret |
| Tns  | FutT |
| Asp  | PERFECTIVE |
| Asp  | PROGRESSIVE |
| PASSIVE  | HI |
| vV  | Tns |
| vV  | Asp  | Tns |
| vV  | EMPTY |
| SE  | Dec-N |
| SE  | Dec-Pol |
Sememic Supplement I

Sentence | S |
#Sentence D CL1 CL2
CL1 = D.DS.C1 |
CL2 = | D.DS.MC |
CL2 = | D.DS.C2 |
Clause | CL1 |
Clause | CL2 |
#Clause A PROCESS SUBJ TOPIC PART1 PART2 PART3 Tense Aspect
   DIR LOC COND MTIME
Tense = A.VP.vV.Tns |
Aspect = A.VP.vV.Asp.PROGRESSIVE |
Aspect = A.VP.vV.Asp.PERFECTIVE |
PROCESS = A.VP.VI.Vi |
PROCESS = A.VP.VI.Vt |
PROCESS = A.VP.VI.Adj |
PROCESS = A.VP.VI.Vdt |
pass-cond = A.VP.VI.PASSIVE |
subj-cond = A.NP.nM.nSubj |
top-cond = A.NP.nM.nTop |
SUBJ = @subj-cond | A.NP.NP2 |
TOPIC = @top-cond | A.NP.NP2 |
PART1 = !pass-cond | A.NP.NP2 |
PART1 = @pass-cond | A.VP.ADJUNCT.ADV.AGADV.NP2 |
PART1 = @pass-cond | A.VP.ADJUNCT.ADJUNCT.ADV.AGADV.NP2 |
PART2 = !pass-cond | A.VP.VI.NP.NP2 |
PART2 = A.VP.VI.COMPLEMENT.NP2 |
PART2 = @pass-cond | A.NP.NP2 |
PART3 = A.VP.ADJUNCT.ADV.DADV.NP2 |
DIR = A.VP.ADJUNCT.ADV.DADV.NP2 |
DIR = A.VP.ADJUNCT.ADJUNCT.ADV.DADV.NP2 |
LOC = A.VP.ADJUNCT.ADV.LADV.NP2 |
COND = @A.ADMC.cAdCD | A.C1 |
MTIME = @A.ADMC.cAdMT | A.C1 |
NounPhrase | SUBJ |
NounPhrase | PART1 |
NounPhrase | PART2 |
NounPhrase | PART3 |
NounPhrase | TOPIC |
NounPhrase | DIR |
NounPhrase | LOC |
UClause | COND |
UClause | MTIME |
#UClause E CL1
CL1 = E |
Clause | CL1 |
#NounPhrase B DES PPART DEF
DES = B.NP2.*.Att-Adj.\^ |
DES = B.NP2.*.UADJ.\^ |
PPART = B.NP2.*.NOUN |
DEF = B.NP2.*.PAPL-CL |
Adjectives | DES |
Clause  |  DEF.C1  |  NIL  |  NIL  |  NIL  |  PPART  |
#Adjectives  F  PDEIC  DEIC  NUM  EPI  CLASER
PDEIC = F.NP2.*.UADJ.PDUAdj |
DEIC = F.NP2.*.UADJ.DUAdj |
NUM = F.NP2.*.UADJ.NUAdj |
EPI = F.NP2.*.Att-Adj |
CLASER = F.NP2.*.UADJ.CluAdj |
Sememc Supplement II

S.CL1.PROCESS : S.CL1.PART1.PPART |
S.CL1.PROCESS : S.CL1.PART1.PPART | S.CL1.PART2.PPART |
S.CL1.PROCESS : S.CL1.PART1.PPART | S.CL1.PART2.PPART |
   S.CL1.PART3.PPART |
S.CL1.PART1.DEF.PROCESS : S.CL1.PART1.DEF.PART1.PPART |
   S.CL1.PART1.DEF.PART2.PPART |
S.CL1.PART1.DEF.PROCESS : S.CL1.PART1.DEF.PART1 |
   S.CL1.PART1.DEF.PART2.PPART |
S.CL1.PART2.DEF.PROCESS : S.CL1.PART2.DEF.PART1.PPART |
   S.CL1.PART2.DEF.PART2.PPART |
S.CL1.PART2.DEF.PROCESS : S.CL1.PART2.DEF.PART1 |
   S.CL1.PART2.DEF.PART2.PPART |
S.CL2.PROCESS : S.CL2.PART1.PPART |
S.CL2.PROCESS : S.CL2.PART1.PPART | S.CL2.PART2.PPART |
S.CL2.PROCESS : S.CL2.PART1.PPART | S.CL2.PART2.PPART |
   S.CL2.PART3.PPART |
S.CL2.PART1.DEF.PROCESS : S.CL2.PART1.DEF.PART1.PPART |
   S.CL2.PART1.DEF.PART2.PPART |
S.CL2.PART1.DEF.PROCESS : S.CL2.PART1.DEF.PART1 |
   S.CL2.PART1.DEF.PART2.PPART |
S.CL2.PART2.DEF.PROCESS : S.CL2.PART2.DEF.PART1.PPART |
   S.CL2.PART2.DEF.PART2.PPART |
S.CL2.PART2.DEF.PROCESS : S.CL2.PART2.DEF.PART1 |
   S.CL2.PART2.DEF.PART2.PPART |
<Dictionaries for English>

Morphemic Supplement I

$ D S
   ES ED
   ING
$
EATEN EAT EN
BOUGHT BUY EN
GONE GO EN
ATE EAT ED
WENT GO ED
WAS BE 3rd. Sg. Past
WERE BE 1st/2nd. Past
WON WIN ED
SOLD SELL ED
GAVE GIVE ED
HAS HAVE S
HAD HAVE ED
IS BE S
AM BE 1st. Sg. Pres
ARE BE 2nd. Sg. Pres
BE 1/2/3 Pl

$
0 ( ) STEM ( ) 1
0 ( ) STEM ( I Y ) 1

$ NOUN s NOUN Pl
NOUN es NOUN Pl
Vt s Vt 3rd. Sg. Pres
Vt es Vt 3rd. Sg. Pres
Vi s Vi 3rd. Sg. Pres
Vi es Vi 3rd. Sg. Pres
Morphemic Supplement II

Vt pres \ pres Vt
Vt past \ past Vt
Vi papl \ papl Vi
Vt papl \ papl Vt
Vt prog \ prog Vt
Vi pres \ pres Vi
Vi past \ past Vi
Vi prog \ prog Vi
COPVERB pres \ pres COPVERB
COPVERB past \ past COPVERB
Aux-Be pres \ pres Aux-Be
Aux-Be past \ past Aux-Be
Aux-Have pres \ pres Aux-Have
Aux-Have past \ past Aux-Have
BIVERB pres \ pres BIVERB
BIVERB past \ past BIVERB
BIVERB papl \ papl BIVERB
NOUN pl \ pl NOUN
Master Dictionary

# 32 Lexical Categories:
TYPE1 = NOUN
TYPE2 = Vt Vi COPVERB BIVERB
TYPE3 = InDef CONJ pl IPVT ADV RP Def nPPT nPPPL
        nPPAG nPFS nPPIN nPPQ nPPD nPPAB nPPINS pres
        past fut papl prog Aux-Be Aux-Have Adj-C Adj-D
        Adj-Cl Adj-N

# 10 TYPE1 Property Categories:
  1 HUMAN
  2 ANIMAL
  3 ED-LIQ
  4 ED-SOL
  5 CONC-OBJ
  6 PLACE
  7 BP-IA
  8 SPORTS
  9 ABSTRACT
 10 PLANT

# 19 TYPE2 Property Categories:
  1 1 1 1 4
  2 1 2 1 4
  3 1 1 1 3
  4 1 2 1 3
  5 1 1 1 5
  6 1 1 0 0
  7 1 2 0 0
  8 1 1 2 7 1 1
  9 1 1 2 7 1 2
 10 1 2 1 1
 11 1 1 1 1
 12 1 1
 13 1 2
 14 1 10
 15 1 1 1 5 1 1
 16 1 1 1 2 1 1
 17 1 1 1 3 1 1
 18 1 1 1 4 1 1
 19 1 1 2 8

# Lexical Items:
  A  1 InDef
  AMERICA  1 NOUN  6
  AN  1 InDef
  AND  1 CONJ
  ANIMAL  1 NOUN  2
  APPLE  1 NOUN  4
  ARM  1 NOUN  7
  ARREST  1 Vt  11
  ARROW  1 NOUN  5
  AT  2 nPPT
      nPPPL
  BAD  1 Adj-C
THEM 1 NOUN 1
THERE 1 ADV
THEY 1 NOUN 1
THOSE 1 Adj-D
THOUGH 1 Adj-N
TIME 2 NOUN Adj-C1
TO 1 nPPD
TROUBLE 1 NOUN 9
TWO 1 Adj-N
UNIVERSITY 1 NOUN 6
UP 1 nPPAB
WATERMELON 1 NOUN 4
WE 1 NOUN 1
WHEN 1 Adj-N
WHICH 1 RP
WHO 1 RP
WHOM 1 RP
WILL 1 fut
WIN 1 Vt 5
WITH 1 nPPINS
XXIVTH 1 Adj-N
Lexemic Supplement I

<table>
<thead>
<tr>
<th>#</th>
<th>0</th>
<th>CALL</th>
<th>2</th>
<th>UP</th>
<th>-1</th>
<th>Vt</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>COHEN</td>
<td>0</td>
<td>HOUSE</td>
<td>-1</td>
<td>NOUN</td>
<td>6</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>GET</td>
<td>1</td>
<td>INTO</td>
<td>0</td>
<td>TROUBLE</td>
<td>-1</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>IBM</td>
<td>0</td>
<td>PC</td>
<td>-1</td>
<td>NOUN</td>
<td>5</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>RICE</td>
<td>0</td>
<td>UNIVERSITY</td>
<td>-1</td>
<td>NOUN</td>
<td>6</td>
</tr>
</tbody>
</table>
S   DS
DS   C1
DS   C1 CONJ C2
C1   NP   VP
C2   C1
NP   DET NP2
NP   NP2
DET   InDef EMPTY
DET   Def EMPTY
NP2   NOUN
NP2   pl NOUN
NP2   ADJ NP2
NP2   NP2 PP
NP2   NOUN RC
ADJ   Adj-D
ADJ   Adj-N
ADJ   Adj-C
ADJ   Adj-C1
PP   LPP
PP   TPP
PP   DPP
PP   INSPP
PP   AGPP
LPP   nPPL NP
TPP   nPPT NP
DPP   nPPD NP
INSPP   nPPINS NP
AGPP   nPPAG NP
RC   RP SUBCL
SUBCL   NP   VP
SUBCL   VP
VP   Tns VI
VP   Tns Asp VI
VI   Vi
VI   PASSIVE Vt
VI   Vt   NP
VI   IPVt NP COMPLEMENT
VI   COPVERB COMPLEMENT
VI   BIVERB NP3 NP4
VI   BIVERB NP
VI   VI PP
COMPLEMENT   ADJ
COMPLEMENT   NP
NP3   NP
NP4   NP
Tns   pres
Tns   past
Tns   fut
Tns   EMPTY
Asp   PERFECTIVE
Asp   PROGRESSIVE
PASSIVE   Aux-Be pap1
PERFECTIVE   Aux-Have pap1
PROGRESSIVE  Aux-Be prog
Semantic Supplement I

Sentence | S |
#Sentence D CL1 CL2
CL1 = D.DS.C1 |
CL2 = D.DS.C2.C1 |
Clause | CL1 |
Clause | CL2 |

#Clause A PROCESS SUBJ PART1 PART2 PART3 Tense Aspect
DIR LOC INS TIME
Pass-cond = A.VP.VI.*.PASSIVE |
Biverb-cond = A.VP.VI.*.BIVERB |
PROCESS = A.VP.VI.*.Vi |
PROCESS = A.VP.VI.*.Vt |
PROCESS = Biverb-cond |
PROCESS = A.VP.VI.*.IPVT |
PROCESS = A.VP.VI.*.COMPLEMENT.ADJ |
SUBJ = A.NP |
PART1 = @Pass-cond | A.VP.VI.*.PP.AGPP.NP |
PART1 = !Pass-cond | A.NP |
PART2 = @Pass-cond | A.NP |
PART2 = A.VP.VI.*.NP4.NP |
PART2 = !Pass-cond | A.VP.VI.*.NP |
PART3 = A.VP.VI.*.PP.DATPP.NP |
PART3 = A.VP.VI.*.NP3.NP |
Tense = A.VP.Tns |
Aspect = A.VP.Asp.PROGRESSIVE |
Aspect = A.VP.Asp.PERFECTIVE |
DIR = A.VP.VI.*.PP.DPP.NP |
LOC = A.VP.VI.*.PP.LPP.NP |
TIME = A.VP.VI.*.PP.TPP.NP |
INS = A.VP.VI.*.PP.INSPP.NP |

NounPhrase | SUBJ |
NounPhrase | PART1 |
NounPhrase | PART2 |
NounPhrase | PART3 |
NounPhrase | DIR |
NounPhrase | LOC |
NounPhrase | TIME |
NounPhrase | INS |

#NounPhrase B GV NEW DEIC NUM EPI CLASER PLURAL PPART DEF
GV = @B.DET.Def | B.DET.EMPTY |
NEW = @B.DET.InDef | B.DET.EMPTY |
DEIC = B.NP2.*.ADJ.Adj-D |
NUM = B.NP2.*.ADJ.Adj-N |
EPI = B.NP2.*.ADJ.Adj-C |
CLASER = B.NP2.*.ADJ.Adj-C1 |
PPART = B.NP2.*.NOUN |
PLURAL = B.NP2.*.p1 |
DEF = B.NP2.*.RC.SUBCL |
Clause | DEF | NIL | NIL | PPART |
Sememc Supplement II

S.CL1.PROCESS : S.CL1.PART1.PPART | S.CL1.PART2.PPART |
S.CL1.PART3.PPART |
S.CL1.PROCESS : S.CL1.PART1.PPART | S.CL1.PART2.PPART |
S.CL1.PROCESS : S.CL1.PART1.PPART |
S.CL1.PART1.DEF.PROCESS : S.CL1.PART1.DEF.PART1.PPART |
S.CL1.PART1.DEF.PROCESS : S.CL1.PART1.DEF.PART1 |
S.CL1.PART1.DEF.PROCESS : S.CL1.PART1.DEF.PART2.PPART |
S.CL1.PART2.DEF.PROCESS : S.CL1.PART2.DEF.PART1.PPART |
S.CL1.PART2.DEF.PROCESS : S.CL1.PART2.DEF.PART2.PPART |
S.CL1.PART2.DEF.PROCESS : S.CL1.PART2.DEF.PART1 |
S.CL1.PART2.DEF.PROCESS : S.CL1.PART2.DEF.PART2.PPART |
S.CL2.PROCESS : S.CL2.PART1.PPART | S.CL2.PART2.PPART |
S.CL2.PROCESS : S.CL2.PART3.PPART |
S.CL2.PROCESS : S.CL2.PART1.PPART |
S.CL2.PROCESS : S.CL2.PART1.PPART |
S.CL2.PART1.DEF.PROCESS : S.CL2.PART1.DEF.PART1.PPART |
S.CL2.PART1.DEF.PROCESS : S.CL2.PART1.DEF.PART1 |
S.CL2.PART1.DEF.PROCESS : S.CL2.PART1.DEF.PART2.PPART |
S.CL2.PART2.DEF.PROCESS : S.CL2.PART2.DEF.PART1.PPART |
S.CL2.PART2.DEF.PROCESS : S.CL2.PART2.DEF.PART2.PPART |
S.CL2.PART2.DEF.PROCESS : S.CL2.PART2.DEF.PART1 |
S.CL2.PART2.DEF.PROCESS : S.CL2.PART2.DEF.PART2.PPART |
CONST maxnosym = 150;
maxLexprod = 100;
maxnoedge = 1200;
maxnoword = 250;
maxbuffsize = 20;
maxprefix = 50;
maxirrwd = 40;
maxmorstat = 10;
maxAmorii = 5;
maxmoristat = 40;
maxAlsup = 8;
maxIlsupstat = 10;
maxAlex = 5;
maxnofcn = 5;
maxAsarg = 20;
maxpathsize = 15;
maxAsenstat = 15; { correct }
maxsemstat = 100;
maxpathisize = 10;
maxAsiistat = 7;
maxsiistat = 50;

TYPE
{ The General Purpose Types }
str80 = STRING[80];
str15 = STRING[15];
str9 = STRING[9];
BufferType = RECORD
  lsbit: BOOLEAN;
dno: INTEGER;
END;
ProcessType = ARRAY [0..6] OF BYTE;
PropertyPtr = ^PropertyRecord;
PropertyRecord = RECORD
  no: BYTE;
  next: PropertyPtr;
END;
PropArray = ARRAY [1..5] OF PropertyPtr;
{ Master Dictionary }
LexPropPtr = ^LexPropRecord;
LexPropRecord = RECORD
  sym : INTEGER;
  prop : PropertyPtr;
  next : LexPropPtr;
END;
MasterDicRecord = RECORD
  vocab: Str15;
  num : BYTE;
  next: LexPropPtr;
END;
MasDicPtr = ^MasterDicRecord;
{ Morphicon Supplement I }
OneStatMor = RECORD
  psmo: ARRAY [1..2] OF BYTE;
  mprf,msuf: ARRAY [1..2] OF STRING[15];
pbh: ARRAY [1..2] OF BOOLEAN;
END:

{ Morphicon Supplement II }
OneStatMorII = RECORD
  len: BYTE;
  fir,sec,seq: ARRAY [1..maxAmorii] OF BYTE;
END:

{ Lexical Supplement I }
LexSupType = RECORD
  dwd: ARRAY [0..maxAlsup] OF INTEGER;
  next: LexPropPtr;
END:

{ Lexicon Supplement II }
OneGrammar = RECORD
  sym: ARRAY [0..maxAlex] OF BYTE;
  no: BYTE
END;

GrammarType = ARRAY [1..maxLexprod] OF OneGrammar;
QueuePtr = 'QueueElement;
QueueElement = RECORD
  no: INTEGER;
  next: QueuePtr
END;

QueueType = RECORD
  front, rear: QueuePtr
END;
OneEdge = RECORD
  gr, pos, stvx, endvx, lb: BYTE;
  propose, status: BOOLEAN;
  q: QueueType
END;

EdgesType = ARRAY [1..maxnoedge] OF OneEdge;
sequence = RECORD
  st, fin: BYTE
END;

NodePtr = 'NodeType;
NodeType = RECORD
  child: ARRAY[1..maxAlex] OF NodePtr;
  bptr: NodePtr;
  no, backno: BYTE;
  par: INTEGER;
END;

{ Semicon Supplement I }
ParameterType = RECORD
  no: BYTE;
  t: NodePtr;
END;
ItemOfFcmTable = RECORD
  tag: ARRAY [1..maxAsarg] OF BOOLEAN;
  dirty: ARRAY [1..maxAsarg] OF BOOLEAN;
  start,ending,vno: BYTE;
  initial: ParameterType;
  arg: ARRAY [0..maxAsarg] OF ParameterType;
END;
ItemOfStatement = RECORD
  no1,no2: BYTE;
END;

PathType = ARRAY [0..maxpathsize] OF BYTE;
ConceptPtr = 'ConceptType;
ConceptType = RECORD
  cd: ARRAY[1..maxAsarg] OF ConceptPtr;
  no,conc: BYTE;
  pnode: NodePtr;
END;

ItemType = RECORD
  status,pathno: BYTE;
END;

ItemOfPropBuff = RECORD
  sym: BYTE;
  prop: PropertyPtr;
END;

VAR
  General Purpose Declaration
  vocab,MastorDic,NewVoc: Str15;
  out: TEXT;
  ProcTab : ARRAY [1..100] OF ProcessType;
  PropCount: ARRAY [1..30] OF BYTE;
  PropBuff : ARRAY [1..30,1..2] OF ItemOfPropBuff;
  scrbuff,inbuff: ARRAY [1..maxbuffsize] OF Str15;
  LMPtr, BPMtr, PPtr, DNPtr: INTEGER;
  MPtr, MVPtr, MBPtr: BYTE;
  maxterm,emptyno,startno,maxsym: BYTE;
  gsym: ARRAY [1..maxsym] OF Str15;
  SyntaxError, ErrorFound, FirstParse, found, fastload, quit: BOOLEAN;
  ch: CHAR;
  AssPtr,MaxPathPtr,MsepPtr: BYTE;
  buffer: ARRAY [1..maxbuffsize] OF BufferType;
  ExistMS2: BOOLEAN;
  TypePtr,Tptr: ARRAY [1..3] OF BYTE;
  propsym: ARRAY [1..30] OF str9;
  maxprop,maxproc: BYTE;

  Master Dictionary
  Dictionary: ARRAY[1..maxword] OF MasDicPtr;
  ReHeap: MasDicPtr;

  Morphicon Supplement I
  prefix,suffix: ARRAY [0..maxpsfix] OF BYTE;
  MorStat: ARRAY [0..maxmorstat] OF OneStatMor;
  irregular: ARRAY [1..maxirrwd] OF str15;
  irrarr: ARRAY [0..maxirrwd,0..3] OF BYTE;

  Morphicon Supplement II
  MorIIStat: ARRAY [0..maxmoriistat] OF OneStatMorII;
  moriibuff: ARRAY [1..maxbuffsize] OF BOOLEAN;

  Lexical Supplement I
  LexSup: ARRAY [0..maxlsupstat] OF LexSupType;
{ Lexicon Supplement II }
gram: GrammarType;
grseq : ARRAY [0..maxLexprod] OF sequence;
edges : EdgesType;
NewEdge: OneEdge;
active: BOOLEAN;
ParsePtr: QueuePtr;
ParseTree, PartialParseTree: QueueType;
PTreePtr: NodePtr;
{ Semicont Supplement I }
FcmTable: ARRAY [1..maxnofcn] OF ItemOfFcmTable;
StatTable: ARRAY [1..maxsemstat,0..maxAsemstat] OF ItemOfStatement;
PathTable: ARRAY [1..maxsemstat] OF PathType;
FcmNameTab: ARRAY [0..maxnofcn] OF BYTE;
MaxSemIITabNo, SemIITab, NoOffCn, NoOfPath: BYTE;
ConceptTree: ConceptPtr;
{ Semicont Supplement II }
SemIITab: ARRAY [1..maxsiistat] OF PathIIType;
StatIIITab: ARRAY [1..maxsiistat] OF StatIIType;
AssSym: ARRAY [1..maxsiistat] OF BYTE;
checkbit: ARRAY [0..maxLexprod] OF BYTE;

PROCEDURE GetAWord(VAR f: TEXT);
|
This is the code for extracting a word from the given sentence.
|
VAR buff: STRING[15];
i: BYTE;
BEGIN
  IF NOT EOLN(f) THEN
    BEGIN
      buff:= ' ';
      REPEAT
        READ(f, ch)
        UNTIL NOT (ch IN [' ','.']); {Skip all initial blanks.}
      buff[i]:= ch;
      i:= i + 1;
      IF ch IN ['#','|','!', '0','*','.'', '=''] THEN
        vocab:= ch
      ELSE
        BEGIN
          WHILE NOT (ch IN [' ','.'']) AND (NOT EOLN(f)) DO
            BEGIN
              READ(f, ch);
              i:= i + 1;
              buff[i]:= ch
            END;
          IF EOLN(f) THEN i:= i + 1;
          DELETE(buff, i, 16-i);
        vocab:= buff;
      END;
    END;
  END;
END;
PROCEDURE GetSymNo(str: str15; VAR sno: BYTE);

VAR found: BOOLEAN;
BEGIN
  sno := 0;
  found := FALSE;
  WHILE (NOT found) AND (sno < maxsym) DO
    BEGIN sno := sno + 1;
      found := grsym[sno] = str;
    END;
  IF NOT found THEN
    BEGIN maxsym := maxsym + 1;
      sno := maxsym;
      grsym[sno] := str;
    END;
END;

PROCEDURE MatchWordWithDic(str: str15; VAR i: INTEGER; VAR match: BOOLEAN);
BEGIN
  i := 0; match := FALSE;
  WHILE (i < DMPtr) AND (NOT match) DO
    BEGIN i := i + 1;
      match := str = Dictionary[i]’.vocab;
    END;
END;

FUNCTION StUpCase(st: str15): str15;
VAR i: BYTE;
BEGIN
  FOR i := 1 TO LENGTH(st) DO
    st[i] := UpCase(st[i]);
  StUpCase := st;
END;

PROCEDURE WriteStringInBuffer (i: BYTE; writeiern: BOOLEAN);
VAR dno,j,k: INTEGER;
BEGIN
  if i > MPPtr then writeln(‘buffer[i].dno -> i ’, i);
  dno := buffer[i].dno;
  IF buffer[i].lsbit THEN
    BEGIN
      j := 0;
      WHILE (j < LexSup[dno].dw[0]) DO
        BEGIN j := j + 2;
          k := LexSup[dno].dw[j];
          IF (k < 1) OR (k > DMPtr) THEN
            BEGIN Writeln(‘ERROR>> Print Error’);
              EXIT;
            END;
        END;
    END;
END;
ELSE IF writeintern THEN
    WRITE(', Dictionary[k].vocab)
ELSE WRITE(out, ', Dictionary[k].vocab);
END;
END
ELSE
BEGIN
    IF (dno < 1) OR (dno > DMPtr) THEN
        BEGIN WRITELN('ERROR>> Print Error');
END
ELSE IF writeintern THEN
    WRITE(', Dictionary[dno].vocab)
ELSE WRITE(out, ', Dictionary[dno].vocab);
END
END;

PROCEDURE PrintMorpheme(pnode: NodePtr);

VAR i: INTEGER;
BEGIN
    IF pnode <> NIL THEN
        BEGIN
            IF pnode^.par < 0 THEN
                BEGIN
                    i:= -pnode^.par;
                    WriteStringInBuffer (i, TRUE);
                    WriteStringInBuffer (i, FALSE);
                END
ELSE IF pnode^.par <> emptyno THEN
                BEGIN
                    FOR i:= 1 TO pnode^.no DO
                        PrintMorpheme(pnode^.child[i]);
                END
        END;
END;
PROCEDURE ReadLexCat(VAR dic: TEXT);

VAR i: INTEGER;
BEGIN
  READLN(dic, maxterm);
  maxsym:= maxterm;
  GetAWord(dic); GetAWord(dic);
  i:= 0; TypePtr[1]:= 1;
  WHILE vocab <> '§' DO
    BEGIN IF EOLN(dic) THEN READLN(dic);
      GetAWord(dic);
      IF vocab = 'TYPE2' THEN
        BEGIN TypePtr[2]:= i + 1;
          GetAWord(dic);
        END
      ELSE IF vocab = 'TYPE3' THEN
        BEGIN TypePtr[3]:= i + 1;
          GetAWord(dic);
        END
      ELSE IF vocab <> '§' THEN
        BEGIN i:= i + 1;
          type[i]:= vocab;
        END;
    END;
  END;
END;

PROCEDURE ReadPropCat(VAR dic: TEXT);

VAR i,j: BYTE;
BEGIN
  READLN(dic, maxprop);
  FOR i:= 1 TO maxprop DO
    BEGIN READ(dic, j); GetAWord(dic);
      READLN(dic);
      propsym[i]:= vocab;
    END;
END;

PROCEDURE ReadProcCat(VAR dic: TEXT);

VAR i,j,k: BYTE;
BEGIN
  READLN(dic, maxproc);
  FOR i:= 1 TO maxproc DO
BEGIN
READ(dic, j); k := 0;
WHILE NOT REQLN(dic) DO
BEGIN
READ(dic, j);
k := k + 1;
ProcTab[i][k] := j;
END;
REQLN(dic);
ProcTab[i][0] := k;
END;

PROCEDURE ReadProperty(VAR dic: TEXT; VAR q: PropertyPtr);

VAR i: BYTE;
s, r: PropertyPtr;
BEGIN
NEW(r); s := r;
WHILE NOT REQLN(dic) DO
BEGIN
READ(dic, i);
NEW(s^.next);
s^.next^.no := i;
s := s^.next;
END;
IF r = s THEN
BEGIN
q := NIL;
DISPOSE(r);
END
ELSE
BEGIN
s^.next := NIL;
q := r^.next;
END;
END;

PROCEDURE GetTermNo(str: str15; VAR tno: BYTE);

VAR found: BOOLEAN;
BEGIN
 tno := 0; found := FALSE;
WHILE (NOT found) AND (tno < maxterm) DO
BEGIN
 tno := tno + 1;
found := graym[tno] = str;
END;
END;

PROCEDURE ReadVocab(VAR dic: TEXT);

VAR p,q: LexPropPtr;
i: INTEGER;
dic, tno: BYTE;
BEGIN
DMPtr := 0;
WHILE NOT EOF(dic) DO
  BEGIN
    DMPtr := DMPtr + 1;
    GetAWord(dic);
    NEW(Dictionary[DMPtr]);
    Dictionary[DMPtr]^.vocab := vocab;
    NEW(p);
    Dictionary[DMPtr]^.next := p;
    READ(dic, num);
    Dictionary[DMPtr]^.num := num;
    FOR i := 1 TO num DO
      BEGIN NEW(q);
        p^.next := q;
        GetAWord(dic);
        GetTermNo(vocab, tno);
        q^.tno := tno;
        IF tno < TypePtr[3] THEN
          ReadProperty(dic, q^.prop)
        ELSE q^.prop := NIL;
        q^.next := NIL;
        p := q;
      READLN(dic);
      END;
    END;
    FOR i := 1 TO DMPtr DO
      Dictionary[i]^.next := Dictionary[i]^.next^.next;
  END;
END;

PROCEDURE ReadMasterDic(VAR dic: TEXT);
{ This is the code for loading words from the dictionary file to main memory. }
BEGIN
  GetAWord(dic);
  ReadExtCat(dic);
  maxsym := maxterm;
  ReadPropCat(dic);
  GetAWord(dic);
  ReadProcCat(dic);
  READLN(dic);
  ReadVocab(dic);
  CLOSE(dic);
END;

PROCEDURE PrepareInput;
{ This is the code for reading an input sentence, extracting words, preparing an array for the words, and sorting the array, inbuff. }
VAR i,j,pos,counter,SentenceSize: BYTE;
  thuff: STRING[136];
  EndOfSentence,flag: BOOLEAN;

BEGIN  // IF ORDER TO RECOGNIZE THE ORDER OF CONSTITUENTS AFTER SORTING,
          // INITIALLY ACTUAL ADDRESSES ARE GIVEN BY NEGATIVE INTEGERS. //
FOR i:= 1 TO maxbuffsize DO
  inbuff[i]:=' '; 
EndOfSentence:= FALSE;
  counter:= 1;  // READ WORDS AND PUT THEM INTO inbuff. //
WHILE NOT EndOfSentence DO
  BEGIN
    FOR i:= 1 TO 136 DO
      thuff[i]:=' ';
    READLN(thuff);
    WRITELN(out, thuff);
    pos:= 1;
    WHILE thuff[pos] = ' ' DO
      pos:= pos + 1;
    SentenceSize:= LENGTH(thuff);
    j:= 0;  flag:= TRUE;
    WHILE (pos <= SentenceSize) AND (NOT EndOfSentence) DO
      BEGIN
        IF (thuff[pos] IN ['.', '?']) THEN
          EndOfSentence:= TRUE
        ELSE IF thuff[pos] <> ' ' THEN
          BEGIN j:= j + 1;
            flag:= TRUE;
            inbuff[counter][j]:= UpCase(thuff[pos])
          IF pos = SentenceSize THEN
            BEGIN
              DELETE(inbuff[counter], j+1, 15-j);
              counter:= counter + 1
            END
          END;
          ELSE IF flag THEN
            BEGIN DELETE(inbuff[counter], j+1, 15-j);
              i:= j; j:= 0;
              counter:= counter + 1;
              flag:= FALSE
            END;
          END;
        END;
        pos:= pos + 1
      END;
    END;
    IF j = 0 THEN
      BEGIN j:= i;
        counter:= counter - 1
      END;
    DELETE(inbuff[counter], j+1, 15-j);
  MEPtr:= counter;
END;
PROCEDURE ReadPreSuffix(VAR g: TEXT);

VAR j: BYTE;
dno: INTEGER;
match: BOOLEAN;
BEGIN
GetAWord(g);
j := 0;
WHILE vocab <> '$' DO
  BEGIN IF BOLN(g) THEN READLN(g)
    ELSE
      BEGIN
        j := j + 1;
        MatchWordWithDic(vocab, dno, match);
        prefix[j] := dno;
        GetAWord(g);
      END;
    END;
  prefix[0] := j;
  READLN(g);
  GetAWord(g);
  j := 0;
  WHILE vocab <> '$' DO
    BEGIN IF BOLN(g) THEN READLN(g)
      ELSE
        BEGIN
          j := j + 1;
          MatchWordWithDic(vocab, dno, match);
          suffix[j] := dno;
          GetAWord(g);
        END;
      END;
    suffix[0] := j;
  READLN(g);
END;

PROCEDURE ReadIrregularWord(VAR g: TEXT);

VAR i,j: BYTE;
dno: INTEGER;
match: BOOLEAN;
BEGIN
GetAWord(g);
i := 0;
WHILE vocab <> '$' DO
  BEGIN
    i := i + 1;
    irregular[i] := vocab;
    j := 0;
    WHILE NOT BOLN(g) DO
      BEGIN
        j := j + 1;
        GetAWord(g);
        MatchWordWithDic(vocab, dno, match);
        irrarr[i,j] := dno;
      END;
READLN(g);
GetAWord(g);
irarr[i,0] := j;
END;
READLN(g);
irarr[0,0] := i;
END;

PROCEDURE ReadMorphicon(VAR mor: TEXT);

VAR i: BYTE;
BEGIN
ReadPresuffix(mor);
ReadIrregularWord(mor);
i := 0;
WHILE NOT EOF(mor) DO
BEGIN
i := i + 1;
WITH MorStat[i] DO
BEGIN
READ(mor, pseno[1]);
GetAWord(mor); GetAWord(mor);
IF vocab = ')' THEN
pbit[1] := FALSE
ELSE
BEGIN
mpref[1] := vocab;
GetAWord(mor);
GetAWord(mor);
pbit[1] := TRUE;
END;
GetAWord(mor); GetAWord(mor); GetAWord(mor);
IF vocab = ')' THEN
pbit[2] := FALSE
ELSE
BEGIN
msuf[1] := vocab;
GetAWord(mor);
GetAWord(mor);
END;
READ(mor, pseno[2]);
END;
END;
CLOSE(mor);
MorStat[0].psno[1] := i;
END;

PROCEDURE MatchPrefix (pref, str: str15; VAR match: BOOLEAN);

VAR len: BYTE;
BEGIN
len:= LENGTH(pref);
str:= COPY(str, 1, len);
match:= str = pref;
END;

PROCEDURE MatchSuffix (suff, str: str15; VAR match: BOOLEAN);

VAR slen, len: BYTE;
BEGIN
  len:= LENGTH(str);
  slen:= LENGTH(suff);
  DELETE(str, 1, len-slen);
  match:= str = suff;
END;

PROCEDURE ExpandBuffer(statno, bno, eno: BYTE);

VAR i, j: BYTE;
BEGIN
  FOR i:= MBPtr DOWNTO bno+1 DO
    inbuff[i-eno+1]:= inbuff[i];
  j:= MorStat[statno].pem[1];
  FOR i:= 1 TO j DO
    inbuff[bno+i-1]:= scrbuff[i];
    inbuff[bno+j]:= scrbuff[eno];
  FOR i:= 1 TO MorStat[statno].pem[2] DO
    inbuff[bno+eno-i]:= scrbuff[j+i];
  MBPtr:= MBPtr + eno - 1;
END;

PROCEDURE MatchStemWithDic(statno: BYTE; str: str15;
    VAR bno: BYTE; VAR match: BOOLEAN);

VAR tstr: str15;
  plen, slen, len: BYTE;
  dno: INTEGER;
BEGIN
  WITH MorStat[statno] DO
    BEGIN
      match:= TRUE;
      IF pbit[1] THEN
        BEGIN
          plen:= LENGTH(mprf[1]);
          tstr:= COPY(str, 1, plen);
          match:= mprf[1] = tstr;
          IF match THEN
            BEGIN
              DELETE(str, 1, plen);
              str:= CONCAT(mprf[2], str);
            END;
        END;
      IF pbit[2] AND match THEN
        BEGIN
          len:= LENGTH(str);
        END;
    END;
END;
slen := LENGTH(msuf[1]);
tstr := COPY(str, len-slen+1, slen);
match := msuf[1] = tstr;
IF match THEN
  BEGIN
    DELETE(str, len-slen+1, slen);
    str := CONCAT(str, msuf[2]);
  END;
END;
IF match THEN
  MatchWordWithDic(str, dno, match);
IF match THEN
  BEGIN
    bno := bno + 1;
    scrbuff[bno] := str;
  END;
END;

PROCEDURE ParseMor(pno, sno, statno: BYTE; str: str15;
  VAR bno: BYTE; VAR match: BOOLEAN);

VAR i, len, tlen: BYTE;
  regstr, tstr: str15;
BEGIN
  IF (pno = 0) AND (sno = 0) THEN
    MatchStemWithDic(statno, str, bno, match)
  ELSE IF pno > 0 THEN
    BEGIN
      pno := pno - 1;
      bno := bno + 1;
      i := 0;
      match := FALSE;
      regstr := str;
      WHILE (i < prefix[0]) AND (NOT match) DO
        BEGIN
          i := i + 1;
          tstr := Dictionary[prefix[i]].vocab;
          len := LENGTH(tstr);
          tlen := LENGTH(str);
          IF tlen < len THEN
            BEGIN
              MatchPrefix(tstr, str, match);
              IF match THEN
                BEGIN
                  DELETE(str, 1, tlen);
                  scrbuff[bno] := tstr;
                  ParseMor(pno, sno, statno, str, bno, match);
                END;
              IF NOT match THEN
                str := regstr;
            END;
          END;
        END;
      END;
    ELSE
      BEGIN
        sno := sno - 1;
        bno := bno + 1;
        i := 0;
        match := FALSE;
        regstr := str;
      END;
    END;
WHILE (i < suffix[0]) AND (NOT match) DO
BEGIN i:= i + 1;
tstr:= Dictionary[suffix[i]].vocab;
tlen:= LENGTH(tstr);
len:= LENGTH(str);
IF tlen < len THEN
BEGIN
  MatchSuffix(tstr, str, match);
  IF match THEN
  BEGIN
    DELETE(str, len-tlen+1, tlen);
    srbuff[bno]:= tstr;
    ParseMor(pno, sno, statno, str, bno, match);
  END;
  IF NOT match THEN
  str:= regstr;
END;
END;

PROCEDURE FindIrregularWord(str: str15; bno: BYTE; VAR match: BOOLEAN);

VAR i,j,k: BYTE;
BEGIN
  i:= 0; match:= FALSE;
  WHILE (i < irarr[0,0]) AND (NOT match) DO
  BEGIN i:= i + 1;
    match:= irarrar[i] = str;
  END;
  IF match THEN
  BEGIN
    k:= irarr[i,0];
    FOR j:= MBPtr DOWNTO bno+1 DO
      inbuff[j+k-1]:= inbuff[j];
    FOR j:= 1 TO k DO
      inbuff[bno+j-1]:= Dictionary[irarr[i,j]].vocab;
    MBPtr:= MBPtr + k - 1;
  END;
END;

PROCEDURE ParsingOfMorphicon(str: str15; bno: BYTE; VAR match: BOOLEAN);

VAR i,j: BYTE;
BEGIN
  FindIrregularWord(str, bno, match);
i:= 0;
  WHILE (i < MorStat[0].psno[1]) AND (NOT match) DO
  BEGIN i:= i + 1;
    j:= 0;
    ParseMor(MorStat[i].psno[1], MorStat[i].psno[2], i, str, j, match);
    IF match THEN ExpandBuffer(i, bno, j);
PROCEDURE WriteStr15(str: str15);

VAR m, k: BYTE;
BEGIN
  k := 15 - LENGTH(str);
  WRITE(str);
  WRITE(' ':k);
  WRITE(out, str);
  WRITE(out, ' ':k);
END;

PROCEDURE WriteStr9(str: str9);

VAR m, k: BYTE;
BEGIN
  k := 9 - LENGTH(str);
  WRITE(str);
  WRITE(' ':k);
  WRITE(out, str);
  WRITE(out, ' ':k);
END;

PROCEDURE UpStr9(VAR str: str9);
VAR i: BYTE;
BEGIN
  FOR i := 1 TO LENGTH(str) DO
    str[i] := UpCase(str[i]);
END;

PROCEDURE DisplayMenu1;

VAR i: BYTE;
BEGIN
  ClrScr;
  WRITELN(' **<TYPE1 Semantic Property Menu>**'); WRITELN;
  WRITELN(' 0. NewCategory');
  WRITELN(out, ' **<TYPE1 Semantic Property Menu>**'); WRITELN;
  WRITELN(out, ' 0. NewCategory');
  FOR i := 1 TO maxprop DO
    BEGIN
      WRITELN(' ', i, ' ', propsym[i]);
      WRITELN(out, ' ', i, ' ', propsym[i]);
    END;
END;

PROCEDURE ReadMenu1(sym: BYTE; VAR p: PropertyPtr);

VAR i: BYTE;
  q: Propertyptr;
BEGIN
  WRITELN(' THE NEW ITEM = ', NewVoc);
  WRITELN(' THE LEX CATEGORY = ', grsyms[grsym]);
  WRITE('Enter the Prop. No. >> '); READLN(i);
  WRITELN(out, ' THE NEW ITEM = ', NewVoc);
  WRITELN(out, ' THE LEX CATEGORY = ', grsyms[grsym]);
  WRITELN(out, 'Enter the Prop. No. >> ', i);
  IF i = 0 THEN
    BEGIN maxprop := maxprop + 1;
      WRITE('Enter the name of category : ');
      READLN(propsym[maxprop]);
      UppStr9(propsym[maxprop]);
      WRITELN(out, 'Enter the name of category : ', propsym[maxprop]);
      i := maxprop;
    END;
  NEW(q); q^.no := i; q^.next := NIL;
  p := q;
END;

PROCEDURE DisplayMenu2;

VAR i,j,m,n,k: BYTE;
  control: ARRAY [1..2] OF CHAR;
  first: BOOLEAN;
BEGIN
  ClrScr;
  control[1] := 'E';
  WRITELN(' ***< TYPE2 Semotactic Code Menu >**');
  WRITELN(' *(TYPE1)*'); n := 0;
  WRITELN(out, ' ***< TYPE2 Semotactic Code Menu >**');
  WRITELN(out, ' *(TYPE1)*'); n := 0;
  FOR i := 1 TO maxprop DO
    BEGIN
      n := n + 1;
      n := n MOD 7;
      IF n = 0 THEN
        BEGIN n := 1;
          WRITELN;
          WRITELN(out);
        END;
      IF i < 10 THEN
        BEGIN WRITE(' ', i, ' . ');
          WRITE(out, ' ', i, ' . ');
        END
      ELSE
        BEGIN WRITE(i, ' . ');
          WRITE(out, i, ' . ');
        END;
      WriteStr9(propsym[i]);
    END;
  WRITELN;
  WRITELN(' *(TYPE2)*');
WRITE(' 0. New Category');
WRITELN(out);
WRITELN(out, ' *<TYPE2>*');
WRITE(out, ' 0. New Category');
FOR i := 1 to maxproc DO
BEGIN
  n := i MOD 2;
  IF n = 1 THEN
    BEGIN WRITELN; WRITELN(out);
      first := TRUE;
    END
  ELSE first := FALSE;
  IF i < 10 THEN
    BEGIN WRITE(' ', i, ' . ');
      WRITE(out, ' ', i, ' . ');
    END
  ELSE
    BEGIN WRITE(i, ' . ');
      WRITE(out, i, ' . ');
    END;
  FOR m := 1 TO ProcTab[i][0] DO
BEGIN
  k := m MOD 2;
  IF ProcTab[i][m] = 0 THEN
    BEGIN IF k = 0 THEN
      WRITE('ANY-THING ');
      WRITE(out, 'ANY-THING ');
    END
  ELSE IF k = 0 THEN
    WriteStr9(propsym[ProcTab[i][m]])
  ELSE
    BEGIN
      WRITE(control[ProcTab[i][m]], ' ');
      WRITE(out, control[ProcTab[i][m]], ' ');
    END;
  END;
  IF first THEN
  BEGIN
    k := (6 - ProcTab[i][0]) DIV 2;
    m := 11*k + 1;
    WRITE(' :m');
    WRITE(out, ' :m');
  END;
  END;
  WRITELN;
  WRITELN(out);
END;

PROCEDURE ReadMenu2(sym: BYTE; VAR ptr: PropertyPtr);

VAR i,j,m,n,num,k: BYTE;
p,q: PropertyPtr;
exloop: BOOLEAN;
PROCEDURE RecogInput(VAR j: BYTE);
VAR temp: STRING[40];
first: BOOLEAN;
i,dbuf: BYTE;
BEGIN
READLN(temp);
WRITELN(out, temp);
temp := temp + ' ';
dbuf := LENGTH(temp);
FOR i := 1 TO 6 DO
  rbuf[i] := '';
i := 1;
WHILE (temp[i] = ' ') AND (i < dbuf) DO
  i := i + 1;
j := 1; first := TRUE;
WHILE (i <= dbuf) AND (j <= 6) DO
  BEGIN IF temp[i] <> '' THEN
    rbuf[j] := temp[j] + temp[i]
  ELSE IF temp[i-1] <> '' THEN
    BEGIN UpStr9(rbuf[j]);
      ji := j + 1;
    END;
  END;
i := i + 1;
END;
j := j - 1;
END;
BEGIN
WRITELN(' THE NEW ITEM = ', NewVoc);
WRITELN(' THE LEX. CAT. = ', grsym[sym]);
WRITE('Enter the number of semotactic codes applicable to this item: ');
READLN(num);
WRITELN(out, ' THE NEW ITEM = ', NewVoc);
WRITELN(out, ' THE LEX. CAT. = ', grsym[sym]);
WRITELN(out, 'Enter the number of semotactic codes applicable to this item: ', num);
NEW(p); i := 1;
ptr := p;
FOR m := 1 TO num DO
  BEGIN WRITE(' SEMOTACTIC CAT. NO. >> '); READLN(i);
    IF i > maxproc THEN
      BEGIN WRITELN;
        WRITELN('ERROR>> No Such Semotactic Code : ', i);
        WRITE('Do you want to exit?(y/n) ');
        REPEAT
          ch := ReadKey;
          ch := UpCase(ch);
        UNTIL ch IN ['Y', 'N'];
        WRITELN;
        IF ch = 'Y' THEN EXIT
      ELSE
      END;
END;
BEGIN
    WRITE(' SEMOTACTIC CAT. NO. : '); READLN(i);
END;

END;
WRITELN(out, ' SEMOTACTIC CAT. NO. >> ', i);
IF i = 0 THEN
BEGIN
    WRITE(out, ' Enter the new semotactic code : ');
    RecogInput(kk);
    maxproc:= maxproc + 1;
    FOR j:= 1 TO kk DO
    BEGIN
        k:= j MOD 2;
        IF k = 0 THEN
        BEGIN
            found:= FALSE;
            n:= 0;
            WHILE (NOT found) AND (n < maxprop) DO
            BEGIN
                n:= n + 1;
                found:= rbuf[j] = propsys[n];
            END;
            IF NOT found THEN
            BEGIN
                ptr:= NIL;
                WRITELN('ERROR>> No Such Semantic Property : ', rbuf[j]);
                WRITE(' Press the RETURN to continue');
            ch:= ReadKey; WRITELN;
                EXIT;
            END;
        END ELSE
        BEGIN
            IF rbuf[j] = 'E' THEN
            n:= 1
            ELSE IF rbuf[j] = 'B' THEN
            n:= 2
            ELSE
            BEGIN
                ptr:= NIL;
                WRITELN('ERROR>> No Such Condition : ', rbuf[j]);
                WRITE(' Press the RETURN to continue');
            ch:= ReadKey; WRITELN;
                EXIT;
            END;
        END;
    ProcTab[maxproc][j]:= n;
END;
    ProcTab[maxproc][0]:= kk;
i:= maxproc;
END;
NEW(q); q^.no:= i;
p^.next:= q;
p:= q;
END;
p'.next:= NIL;
ptr:= ptr'.next;
END;

procedure HelpMenu;
  VAR ch: CHAR;
  BEGIN
    ClrScr;
    writeln('This is a help menu.');
    writeln;
    write('Press the RETURN to continue');
    writeln(out, 'This is a help menu.');
    writeln(out);
    ch:= ReadKey; writeln;
  END;

procedure AddingNewWord(VAR ptr: LexPropPtr; VAR num: BYTE);

VAR i,j: INTEGER;
p,q: LexPropPtr;
more: BOOLEAN;
ch: CHAR;

BEGIN
  ClrScr;
  writeln('**Lexical Category Menu**');
  writeln(out, '**Lexical Category Menu**');
  j:= 0;
  FOR i:= 1 TO maxterm DO
    BEGIN
      j:= j + 1;
      IF i = TypePtr[1] THEN
        BEGIN
          writeln('**Lexical Categories**');
          writeln(out, '**Lexical Categories**');
        END
      ELSE IF i = TypePtr[2] THEN
        BEGIN
          j:= 1; writeln;
          writeln('**Lexical Categories**');
          writeln(out);
          writeln(out, '**Lexical Categories**');
        END
      ELSE IF i = TypePtr[3] THEN
        BEGIN
          j:= 1; writeln; writeln(out);
          writeln('**Lexical Categories**');
          writeln(out, '**Lexical Categories**');
        END;
      j:= j MOD 5;
      IF j = 0 THEN
        BEGIN
          j:= 1
          writeln;
          writeln(out);
        END;
      writeln;
    END;
  END;
IF i < 10 THEN
BEGIN WRITE(', i, '.');
WRITE(out, ', i, '.');
END
ELSE
BEGIN WRITE(i, '.');
WRITE(out, i, '.');
END;
WriteStr15(grsym[i]);
END;
WRITELN(\' THE NEW ITEM = \', NewVoc);
WRITE(\'Enter the number of lexical categories applicable to \');
WRITE('this item : ');
READLN(num);
WRITELN(out); WRITELN(out, \' THE NEW ITEM = \', NewVoc);
WRITE(out, \'Enter the number of lexical categories applicable to \');
WRITE(out, \'this item : \', num);
WRITELN(\'Enter the number(s) of lexical category(es):\');
WRITELN(out, \'Enter the number(s) of lexical category(es):\');
i:= 1; NEW(p);
ptr:= p;
FOR j:= 1 TO num DO
BEGIN WRITE(\' LEX. CAT. No. >> \'); READLN(i);
IF i > maxterm THEN
BEGIN WRITELN;
WRITELN(\'ERROR\'> No Such Lexical Category : \', i);
WRITE(\'Do you want to exit?(y/n) \');
REPEAT
ch:= ReadKey;
ch:= UpCase(ch);
UNTIL ch IN ['Y','N']; WRITELN;
IF ch = 'Y' THEN EXIT
ELSE
BEGIN
WRITE(\' LEX. NO. >> \'); READLN(i);
END;
END;
WRITELN(out, \' LEX. CAT. No. >> \', i);
NEW(q); q\'.sym:= i; q\'.prop:= NIL;
q\'.next:= NIL; p\'.next:= q;
p:= q;
END;
ptr:= ptr\'.next;
p:= ptr;
WHILE p <> NIL DO
BEGIN
IF p\'.sym < TypePtr[2] THEN
BEGIN DisplayMenu1;
BEGIN ReadMenu(p\'.sym, p\'.prop)
END
ELSE IF p\'.sym < TypePtr[3] THEN
BEGIN DisplayMenu2;
BEGIN ReadMenu2(p\'.sym, p\'.prop);
END;
PROCEDURE DisplayNewVoc(p: LexPropPtr; n: BYTE);

VAR q: PropertyPtr;
    ch: CHAR;
BEGIN
  ClrScr;
  WRITELN; WRITELN;
  WRITELN(' **<Information about adding a new item>**'); WRITELN;
  WRITELN(out); WRITELN(out);
  WRITELN(out, ' **<Information about adding a new item>**'); WRITELN(out);
  IF p = NIL THEN
    BEGIN WRITELN('No information was added.');
      WRITELN(out, 'No information was added.');
    END
  ELSE
    BEGIN
      WRITELN(' The new item = ', NewVoc); WRITELN;
      WRITELN(' Lexical Category No. for this item : ', n); WRITELN;
      WRITELN(out, ' The new item = ', NewVoc); WRITELN(out);
      WRITELN(out, ' Lexical category No. for this item : ', n);
      WRITELN(out);
      WHILE p <> NIL DO
        BEGIN WRITE('  LEX. CAT. = ');
          WRITE(out, '  LEX. CAT. = ');
          WriteStr15(grsym[p^.sym]);
          IF p^.sym < TypePtr[2] THEN
            BEGIN WRITE('  PROP. NO. = ');
              WRITE(out, '  PROP. NO. = ');
            END
          ELSE IF p^.sym < TypePtr[3] THEN
            BEGIN WRITE('  SEMOTACTIC CAT. NO. = ');
              WRITE(out, '  SEMOTACTIC CAT. NO. = ');
            END;
          IF p^.sym < TypePtr[3] THEN
            BEGIN
              q := p^.prop;
              WHILE q <> NIL DO
                BEGIN WRITE(q^.no, ' ');
                  WRITE(out, q^.no, ' ');
                  q := q^.next;
                END;
              WRITELN;
              WRITELN(out);
              WRITELN(out);
              p := p^.next;
            END;
        END;
    END;
  WRITELN;
  WRITELN(out);
WRITE(' Press the RETURN to continue');
    ch:= ReadKey; WRITELN;
END;

PROCEDURE AddSemProp;
BEGIN
    DisplayMenu1;
    WRITELN; WRITELN;
    WRITELN(out); WRITELN(out);
    maxprop:= maxprop + 1;
    WRITE('THE NAME OF SEMANTIC PROPERTY >> ');
    READLN(propsym[maxprop]);
    Upstr9(propsym[maxprop]);
    WRITELN(out, 'THE NAME OF SEMANTIC PROPERTY >> ', propsym[maxprop]);
END;

PROCEDURE MorphMenu;

VAR i,n: BYTE;
    ch: CHAR;
    p: LexPropPtr;
    savemp,savemp: BYTE;
    first: BOOLEAN;
BEGIN
    p:= NIL;
    savemp:= maxprop;
    savemp:= maxprop;
    first:= TRUE;
    REPEAT
        ClrScr;
        WRITELN; WRITELN;
        WRITELN('      *** MAIN MENU ***'); WRITELN; WRITELN;
        WRITELN('  1. HELP'); WRITELN;
        WRITELN('  2. ADD NEW INFORMATION'); WRITELN;
        WRITELN('  3. DISPLAY NEW INFORMATION'); WRITELN;
        WRITELN('  4. DISPLAY SEMANTIC PROPERTY MENU'); WRITELN;
        WRITELN('  5. ADD NEW SEMANTIC PROPERTY'); WRITELN;
        WRITELN('  6. DISPLAY SEMOTACTIC CODE MENU'); WRITELN;
        WRITELN('  7. EXIT'); WRITELN; WRITELN;
        WRITE(' Command >> ');
        READLN(i);
        WRITELN(out); WRITELN(out);
        WRITELN(out, ' *** MAIN MENU ***'); WRITELN(out); WRITELN(out);
        WRITELN(out, ' 1. HELP'); WRITELN(out);
        WRITELN(out, ' 2. ADD NEW INFORMATION'); WRITELN(out);
        WRITELN(out, ' 3. DISPLAY NEW INFORMATION'); WRITELN(out);
        WRITELN(out, ' 4. DISPLAY SEMANTIC PROPERTY MENU'); WRITELN(out);
        WRITELN(out, ' 5. ADD NEW SEMANTIC PROPERTY'); WRITELN(out);
        WRITELN(out, ' 6. DISPLAY SEMOTACTIC CODE MENU'); WRITELN(out);
        WRITELN(out, ' 7. EXIT'); WRITELN(out); WRITELN(out);
        WRITELN(out, ' Command >> ', i);
        IF i = 1 THEN HelpMenu
        ELSE IF i = 2 THEN
            BEGIN


maxproc := savenc;
AddingNewVoc(p,n);
END
ELSE IF i = 3 THEN DisplayNewVoc(p,n)
ELSE IF i = 4 THEN
BEGIN DisplayMenu1;
WRITELN;
WRITE('Press the RETURN to continue');
ch := ReadKey; WRITELN;
END
ELSE IF i = 5 THEN AddSemProp
ELSE IF i = 6 THEN
BEGIN DisplayMenu2;
WRITELN;
WRITE('Press the RETURN to continue');
ch := ReadKey; WRITELN;
END
ELSE IF i = 7 THEN
BEGIN
REPEAT
WRITELN;
WRITE('Do you want to add the new vocabulary?(y/n) ');
ch := ReadKey; WRITELN;
WRITELN(out, 'Do you want to add the new vocabulary?(y/n) ', ch);
ch := UPCASE(ch);
IF ch = 'Y' THEN
BEGIN DWPtr := DWPtr + 1;
NEW(Dictionary[DWPtr]);
Dictionary[DWPtr]^next := p;
Dictionary[DWPtr]^num := n;
END
ELSE IF ch = 'N' THEN
BEGIN maxprop := savemp;
maxproc := savenc;
END;
UNTIL (ch = 'N') OR (ch = 'Y');
END;
UNTIL (i = 7);
END;

PROCEDURE Morphotactics;

VAR i: INTEGER;
j: BYTE;
ch: CHAR;
match: BOOLEAN;
str: STRING[15];
BEGIN
j := 1;
ErrFound := FALSE;
WHILE (j <= MPtr) AND (NOT ErrFound) DO
BEGIN


MatchWordWithDic(inbuff[j], i, match);
IF match THEN
    BEGIN buffer[j].dno:= i;
        j:= j + 1;
    END
ELSE
    BEGIN
        str:= inbuff[j];
        ParsingORmorphicon(str, j, match);
        IF NOT match THEN
            BEGIN
                CLRScr;
                WRITELN;
                WRITELN('ERROR> The item ', str,
                    ' is not in the dictionary.'); WRITELN;
                WRITELN(out);
                WRITELN(out, 'ERROR> The item ', str,
                    ' is not in the dictionary.'); WRITELN(out);
                NewVoc:= str;
                WRITE('Is this misspelled? (y/n) ');
                ch:= ReadKey; WRITELN;
                WRITELN(out, 'Is this misspelled? (y/n) ', ch);
                IF (ch = 'n') OR (ch = 'N') THEN
                    BEGIN
                        WRITELN;
                        WRITE('Do you want to add the item? (y/n) ');
                        ch:= ReadKey; WRITELN;
                        WRITELN(out);
                        WRITELN(out, 'Do you want to add the item? (y/n) ', ch);
                        IF (ch <> 'n') AND (ch <> 'N') THEN
                            MorphMenu
                        ELSE
                            ErrorFound:= TRUE;
                    END
                ELSE
                    BEGIN
                        WRITELN;
                        WRITE('Enter the correct item >> ');
                        READLN(inbuff[j]);
                        inbuff[j]:= StUpCase(inbuff[j]);
                        WRITELN(out);
                        WRITELN(out, 'Enter the correct item >> ', inbuff[j]);
                        NewVoc:= inbuff[j];
                    END
            END
        END;
    END;
END;
PROCEDURE FixSequence;

VAR i,j,k: BYTE;
    found: BOOLEAN;
BEGIN
    FOR i:= 1 TO MorIIStat[0].len DO
        BEGIN
            FOR j:= 1 TO MorIIStat[i].len DO
                BEGIN
                    k:= 0; found:= FALSE;
                    WHILE (k < MorIIStat[i].len) AND (NOT found) DO
                        BEGIN
                            k:= k + 1;
                            found:= (MorIIStat[i].sec[j] = MorIIStat[i].fir[k]);
                        END;
                    MorIIStat[i].seq[j]:= k;
                END;
        END;
    END;
END;

PROCEDURE ReadMorII(VAR morii: TEXT);

VAR nostat,no,k: BYTE;
    fitem: BOOLEAN;
BEGIN
    nostat:= 0;
    WHILE NOT EOF(morii) DO
        BEGIN
            nostat:= nostat + 1;
            no:= 0; fitem:= TRUE;
            WHILE NOT EOFN(morii) DO
                BEGIN
                    GetAWord(morii);
                    IF vocab <> '\' THEN
                        BEGIN
                            k:= 0; found:= FALSE;
                            WHILE (k < maxterm) AND (NOT found) DO
                                BEGIN
                                    k:= k + 1;
                                    found:= grsym[k] = vocab;
                                END;
                            no:= no + 1;
                            IF fitem THEN
                                MorIIStat[nostat].fir[no]:= k
                            ELSE
                                MorIIStat[nostat].sec[no]:= k;
                        END;
                    ELSE
                        BEGIN
                            fitem:= FALSE;
                            MorIIStat[nostat].len:= no;
                            no:= 0;
                        END;
                    END;
                READLN(morii);
            END;
        END;
    MorIIStat[0].len:= nostat;
PROCEDURE CorrectSequence(statno, buffno: BYTE);

VAR i: BYTE;
impurity: BOOLEAN;
tbuff: ARRAY[1..5] OF BufferType;
BEGIN
  WITH MorIIStat[statno] DO
    BEGIN
      impurity := FALSE;
      FOR i := 0 TO len-1 DO
        BEGIN
          IF moriibuff[buffno+i] THEN
            impurity := TRUE;
        END;
      IF NOT impurity THEN
        BEGIN
          FOR i := 1 TO len DO
            tbuff[i] := buffer[seq[i]+buffno-1];
          FOR i := 1 TO len DO
            BEGIN
              buffer[i+buffno-1] := tbuff[i];
              moriibuff[i+buffno-1] := TRUE;
            END;
        END;
    END;
END;

PROCEDURE SearchMorII (statno:BYTE);

VAR n, tmax, start, ending, inc: BYTE;
p: LexPropPtr;
BEGIN
  tmax := MBPtr - MorIIStat[statno].len + 1;
  FOR n := 1 TO tmax DO
    BEGIN
      found := TRUE;
      start := n; ending := n + MorIIStat[statno].len - 1;
      inc := 0;
      WHILE (start <= ending) AND found DO
        BEGIN
          found := FALSE;
          inc := inc + 1;
          p := Dictionary[buffer[start].dnol].next;
          WHILE (p <> NIL) AND (NOT found) DO
            BEGIN
              found := p^.sym = MorIIStat[statno].fir[inc];
              p := p^.next;
            END;
          start := start + 1;
        END;
      IF found THEN CorrectSequence(statno,n);
PROCEDURE ExecuteMorII;

VAR statno, buffno, i: BYTE;
    found: BOOLEAN;
BEGIN
    FOR i:= 1 TO MBPtr DO
        moriibuf[i]:= FALSE;
    FOR statno:= 1 TO MorIIStat[0].len DO
        BEGIN
            SearchMorII(statno);
        END;
END;
PROCEDURE ReadLexSupp(VAR lsp: TEXT);

VAR i,j,sno: BYTE;
    k,dno: INTEGER;
    match: BOOLEAN;
    p,q: LexPropPtr;
BEGIN
  i:= 0;
  WHILE NOT EOF(lsp) DO
    BEGIN
      GetAWord(lsp);
      IF vocab = '#' THEN
        BEGIN
          i:= i + 1;
          READ(lsp, k);
          j:= 0;
          WHILE k >= 0 DO
            BEGIN
              j:= j + 1;
              LexSup[i].dw[j]: = k;
              GetAWord(lsp);
              j:= j + 1;
              MatchWordWithDic(vocab, dno, match);
              LexSup[i].dw[j]: = dno;
              READ(lsp, k);
              NEW(p); LexSup[i].next: = p;
            END;
          LexSup[i].dw[0]: = j;
        END
        ELSE
          BEGIN
            NEW(q);
            p^.next: = q;
            k:= 0; match:= FALSE;
            WHILE (k < maxterm) AND (NOT match) DO
              BEGIN
                k:= k + 1;
                match:= grsym[k] = vocab;
              END;
            IF (NOT match) THEN WRITELN('ERROR
')
            ELSE q^.sym:= k;
            ReadProperty(lsp, q^.prop);
            READLN(lsp);
            q^.next: = NIL;
            p:= q;
          END;
    END;
  CLOSE(lsp);
  LexSup[0].dw[0]: = i;
  FOR j:= 1 TO i DO
    LexSup[j].next: = LexSup[j].next^.next;
END;

PROCEDURE MatchComplexWord(i,j: INTEGER; VAR match: BOOLEAN);

VAR k,l,m,bpos,dist,dno: INTEGER;
    matched: ARRAY [1..5] OF INTEGER;
BEGIN
bpos := i; m := 0;
k := 2; match := buffer[bpos].dno = LexSup[j].dwd[2];
WHILE (k < LexSup[j].dwd[0]) AND match DO
BEGIN bpos := bpos + 1;
k := k + 1; dist := LexSup[j].dwd[k];
k := k + 1; dno := LexSup[j].dwd[k];
l := 0; match := buffer[bpos].dno = dno;
WHILE (l < dist) AND (bpos < MBPtr) AND (NOT match) DO
BEGIN l := l + 1;
bpos := bpos + 1;
match := buffer[bpos].dno = dno;
END;
m := m + 1;
matched[m] := bpos;
END;
IF match THEN
BEGIN buffer[i].dno := j;
buffer[i].lsbit := TRUE;
FOR k := 1 TO m DO
FOR l := matched[k] - k + 1 TO MBPtr - k DO
buffer[l].dno := buffer[l+1].dno;
MBPtr := MBPtr - m;
END;
END;

PROCEDURE ExecuteLexSupplement;

VAR i, j: INTEGER;
match: BOOLEAN;
BEGIN
FOR i := 1 TO MBPtr DO
buffer[i].lsbit := FALSE;
i := 0;
WHILE (i < MBPtr) DO
BEGIN
i := i + 1;
j := 0; match := FALSE;
WHILE (j < LexSup[0].dwd[0]) AND (NOT match) DO
BEGIN j := j + 1;
MatchComplexWord(i, j, match);
END;
END;
END;
PROCEDURE ReadLexicon(VAR g: TEXT);
{
This is the code for reading the grammar from the grammar file and
transforming the symbol to the corresponding number.
}

VAR temp: STRING[15];
  j: INTEGER;
maxl,n,itsym,i,k: BYTE;
  first,found: BOOLEAN;
  ans: CHAR;

BEGIN
emptyno:= maxterm + 1;  { ATTACH THE EMPTY AND STARTING SYMBOLS. }
gsym[emptyno]:= 'EMPTY';
startno:= emptyno + 1;
gsym[startno]:= 'S';
mmaxsym:= startno;
i:= 0;  itsym:= 1;  { READ THE GRAMMAR SYMBOLS IFF THOSE ARE NEW, }
FOR j:= 0 TO startno DO
  checkbit[j]:= 1;
FOR j:= startno+1 TO maxlexprod DO
  checkbit[j]:= 0;
WHILE NOT EOF(g) DO  { INSERT THEM INTO GRAMMAR SYMBOLS AND FILL THE }
  BEGIN j:= -1;  { CORR. NUMBERS INTO THE GRAMMAR TABLE. }
i:= i + 1;
  WHILE NOT EOLN(g) DO
    BEGIN j:= j + 1;
      GetAWord(g);  { CHECK IF THE WORD IS IN THE }
      GetSymNo(vocab, k);
      gsym[j].sym[0]:= k;
    END;
  END;
gsym[j].no:= j;
EOLN(g);  { THE SAME SYMBOL IN THE LHS OF THE GRAMMAR WAS GROUPED. }
IF itsym <> gsym[i].sym[0] THEN  { IT SPEED UP PARSING. }
  BEGIN grseq[itsym].fin:= i - 1;
    itsym:= gsym[i].sym[0];
    grseq[itsym].st:= i;
  END;
END;
grseq[itsym].fin:= i;
CLOSE(g);
MPtr:= i
END;

(*---------------------------------------------------------------------------------------------------------*)

(* THE PARSING PHASE *)

(*---------------------------------------------------------------------------------------------------------*)

PROCEDURE MergeQueue(VAR mq: QueueType; sq: QueuePtr);
VAR fq,p,q: QueuePtr;
BEGIN
  fq:= mq.front;
  NEW(p);  mq.front:= p;
p'.no := fq'.no; fq := fq'.next;
WHILE fq <> NIL DO
  BEGIN New(q);
    q'.no := fq'.no;
    p'.next := q;
    p := q;
    fq := fq'.next;
  END;
  p'.next := sq; sq'.next := NIL;
  mq.rear := sq;
END;

FUNCTION MatchWithVertex( lblr: BYTE ): BOOLEAN;
VAR p: QueuePtr;
  k: BYTE;
  found: BOOLEAN;
BEGIN
  found := FALSE;
  p := edges[NewEdge.endvx].q.front;
  WHILE (p <> NIL) AND NOT found DO
    BEGIN k := p'.no;
      found := (lblr = k);
      p := p'.next
    END;
  MatchWithVertex := found;
END;

PROCEDURE ExpandAEdge( evx: BYTE; eno: INTEGER);
VAR ptr: QueuePtr;
BEGIN
  MEPtr := MEPtr + 1;
  edges[MEPtr].next := NewEdge;
  WITH edges[MEPtr] DO
    BEGIN pos := pos + 1;
      endvx := evx;
      IF pos > gram[gr].no THEN
        BEGIN status := TRUE;
          lb := gram[gr].sym[0];
        END
      ELSE lb := gram[gr].sym[pos];
      propose := FALSE;
      New(ptr);
      ptr'.no := eno; ptr'.next := NIL;
      MergeQueue(q, ptr);
    END;
END;

PROCEDURE CombineCompleteEdge;
/
This is the code for finding the edges which can go over the grammar
symbols based on the present complete edge and adding that to edge queue.
For example,
A \rightarrow B C
1 5
the present complete edge:
G \rightarrow C D
C \rightarrow F H
3 5 5 9

*(The result of this procedure)*
Add the following two edges into edge:
G \rightarrow C D
A \rightarrow B C
3 9 1 9

Note: The order is important.

VAR i: INTEGER;
p: = QueuePtr;
BEGIN
active:= FALSE;
FROM PRESENT TO EDGES AFTER INITIALIZATION.
FOR i:= PPtr-1 DOWNTO MVPtr+1 DO
IF (NOT edges[i].status) AND (NewEdge.lb = edges[i].lb) AND (NewEdge.stvx = edges[i].endvx) THEN
BEGIN
MVPtr:= MVPtr + 1;
edges[MVPtr]: = edges[i];
WITH edges[MVPtr] DO
BEGIN pos:= pos + 1;
endvx:= NewEdge.endvx;
IF pos > gram[gr].no THEN
BEGIN status:= TRUE;
lb:= gram[gr].sym[0]
END
ELSE lb:= gram[gr].sym[pos];
propose:= FALSE;
NEW(p);
ptr->no:= PPtr; ptr->next:= NIL;
MergeQueue(q, ptr);
END
END
PROCEDURE CombineActiveEdge;
{This is the code for finding any complete edge which can make the present
go over the symbol and adding this combined present edge into edge queue.
For example,
the present edge: A \rightarrow B C
3 7
Try to find the complete edge in the following type:
C \rightarrow alpha (a sequence of nonterminals)
7 a (any number)
*(The result)*
Add the edge: A \rightarrow B C
3 a
}
VAR i: INTEGER;
BEGIN
  i := MVPtr + 1;
  IF NewEdge.lb = emptyno THEN  { TREATMENT FOR THE EMPTY SYMBOL }
    BEGIN active := FALSE;
      MVPtr := MVPtr + 1;
      edges[MVPtr] := NewEdge;
      WITH edges[MVPtr] DO
        BEGIN pos := pos + 1;
          lb := gram[gr].sym[0];
          status := TRUE
        END
    END;
  { TREATMENT FOR TERMINAL SYMBOLS }
  IF (NewEdge.lb <= maxterm) AND active THEN
    BEGIN active := FALSE;
      IF MatchWithVertex(NewEdge.lb) THEN
        ExpandABedge(NewEdge.endvx+1, -NewEdge.endvx)
      END;
    END;
  WHILE active AND (i < PPtr) DO  { TREATMENT FOR NONTERMINAL SYMBOLS }
    BEGIN
      i := i + 1;
      IF edges[i].status AND (NewEdge.endvx = edges[i].stvx)
        AND (NewEdge.lb = edges[i].lb) THEN
        BEGIN active := FALSE;
          ExpandABedge(edges[i].endvx, i);
        END;
    END;
END;

PROCEDURE ProposeABedge;
{ This is the code for proposing the present nonterminal symbol.
  For Example,
  <GRAMMAR>
  A -> B C
  A -> D E
  A -> F G
  the present edge: Q -> R A
  6
  7
  *(The result)*
  Add three new edges in the edge queue:
  A -> B C  A -> D E  A -> F G
  7 7 7 7 7 7
}

VAR i,j,k: INTEGER;
  found, ok, CheckTerm: BOOLEAN;
  p: QueuePtr;
BEGIN
  i := MVPtr; found := FALSE;
  WHILE (NOT found) AND (i < PPtr-1) DO
BEGIN  { CHECK IF THAT EDGE WAS PREVIOUSLY PROPOSED. }
i:= i + 1;
  IF edges[i].propose AND (Newedge.endvx = edges[i].endvx)
    AND (NOT edges[i].status) AND (NewEdge.lb = edges[i].lb) THEN
    found:= TRUE
END;
IF NOT found THEN
BEGIN
  FOR i:= grseq[NewEdge.lb].st TO grseq[NewEdge.lb].fin DO
    BEGIN
      ok:= TRUE;
      { IN CASE THAT 1ST SYMBOL IS TERMINAL IN GRAMMAR }
      IF gram[i].sym[1] <= maxterm THEN
        ok:= MatchWithVertex(gram[i].sym[1]);
      IF ok THEN
        BEGIN MEPtr:= MEPtr + 1;
          edges[PPtr].propose:= TRUE;
          WITH edges[MEPtr] DO
            BEGIN pos:= 1; stvx:= NewEdge.endvx;
              endvx:= stvx; status:= FALSE;
              gr:= i; lb:= gram[i].sym[1];
              propose:= FALSE;
              NEW(q.front); q.rear:= q.front;
              q.front^.no:= i; q.front^.next:= NIL
            END;
          END { IF ok }
        END { for }
    END { IF NOT found }
  END;
END;

PROCEDURE Recombination(iq: QueuePtr; VAR rq: QueueType);
VAR tptr: QueueType;
BEGIN
  NEW(rq.front);
  rq.rear:= rq.front; rq.rear^.no:= iq^.no;
  iq:= iq^.next;
  WHILE iq <> NIL DO
    BEGIN IF iq^.no > 0 THEN
      Recombination(edges[iq^.no].q.front, tptr)
    ELSE
      BEGIN NEW(tptr.front);
        tptr.rear:= tptr.front; tptr.rear^.no:= iq^.no;
      END;
      rq.rear^.next:= tptr.front;
      rq.rear:= tptr.rear;
      iq:= iq^.next;
    END;
  END;
  rq.rear^.next:= NIL;
END;

PROCEDURE ExpandQueue(fq: QueuePtr; VAR mq: QueuePtr);
VAR tmq, p, q: QueuePtr;
ch: char;
BEGIN
tmq:= mq;
NEW(mq); p:= mq;
p->no:= fq->no;
fq:= fq->next;
WHILE fq <> NIL DO
  BEGIN NEW(q);
    q->no:= fq->no;
p->next:= q;
p:= q;
fq:= fq->next;
  END;
p->next:= tmq;
END;

PROCEDURE Recombination;
VAR Exqueue, ptr: QueuePtr;
  eno: INTEGER;
  ch: char;
BEGIN
  Exqueue:= NIL;
  ExpandQueue(NewEdge.q.front, Exqueue);
  eno:= Exqueue->no;
  Exqueue:= Exqueue->next;
  NEW(ParseTree.front); ParseTree.rear:= ParseTree.front;
  ParseTree.rear->no:= eno;
  WHILE Exqueue <> NIL DO
    BEGIN
      eno:= Exqueue->no;
      Exqueue:= Exqueue->next;
      IF eno > 0 THEN
        BEGIN
          ExpandQueue(edges[eno].q.front, Exqueue);
          eno:= Exqueue->no;
          Exqueue:= Exqueue->next;
        END;
        NEW(ptr); ptr->no:= eno;
        ParseTree.rear->next:= ptr;
        ParseTree.rear:= ptr;
      END;
      ParseTree.rear->next:= NIL;
    END;
  END;

PROCEDURE Parser(VAR found: BOOLEAN);
BEGIN
  found:= FALSE;
  WHILE (NOT found) AND (PPtr < MEPtr) DO
    BEGIN IF MEPtr > maxnoedge THEN
          BEGIN WRITELN("ERROR>> Edge size overflow : ");
            EXIT;
        END;
        PPtr:= PPtr + 1;
        NewEdge:= edges[PPtr];
        WITH NewEdge DO

IF (lb = startno) AND (stvx = 1) AND (endvx = MBPtr+1) THEN
BEGIN
  SyntaxError := FALSE;
  found := TRUE;
  Recombination;
  ParsePtr := ParseTree.front;
END
ELSE IF lb <= maxsym THEN
BEGIN
  active := TRUE;
  IF NewEdge.status THEN
    CombineCompleteEdge
  ELSE CombineActiveEdge;
  IF active THEN
    ProposeEdge
END
END;
IF SyntaxError THEN
BEGIN
  WRITELN;
  WRITELN('ERROR: Syntax Error');
  ErrorFound := TRUE
END
END;

PROCEDURE PopFromParseTree(VAR n:INTEGER);
BEGIN
  n := ParsePtr^.no;
  ParsePtr := ParsePtr^.next;
END;

PROCEDURE ConstructTree(upnode: NodePtr; tno: BYTE;
                          upno: INTEGER; VAR pnode: NodePtr);
VAR i,j,l,k: INTEGER;
BEGIN
  IF ParsePtr <> NIL THEN
  BEGIN
    PopFromParseTree(i);
    NEW(pnode);
    IF i > 0 THEN
      BEGIN
        pnode^.no := gram[i].no;
        pnode^.par := gram[i].sym[0];
        pnode^.bptr := upnode;
        pnode^.backno := upno;
        FOR j := 1 TO gram[i].no DO
          pnode^.child[j] := NIL;
        FOR j := 1 TO gram[i].no DO
          BEGIN IF gram[i].sym[j] = emptyno THEN
            BEGIN
              NEW(pnode^.child[j]);
              pnode^.child[j]^.no := 0;
              pnode^.child[j]^.par := emptyno;
              pnode^.child[j]^.bptr := pnode;
              pnode^.child[j]^.backno := j;
            END
          END;
        END;
      END;
  END;
END;
PROCEDURE PrintParseList(pnode: NodePtr; w: INTEGER);

VAR i, j: INTEGER;
BEGIN
  IF pnode^.par = emptyno THEN
    BEGIN
      WRITE(' ', grsym[emptyno]);
      WRITE(out, ' ', grsym[emptyno]);
    END
  ELSE IF pnode^.par >= 0 THEN
    BEGIN
      WRITE('(', grsym[pnode^.par], ', ');
      WRITE(out, '(', grsym[pnode^.par], ', ');
      j := LENGTH(grsym[pnode^.par]) + w + 2;
      FOR i := 1 TO pnode^.no DO
        BEGIN
          IF i <> 1 THEN
            BEGIN
              WRITELN; WRITELN(out);
              WRITE(' :j); WRITE(out, ' :j);
            END;
            PrintParseList(pnode^.child[i], j);
          IF (pnode^.child[i].par <> emptyno) AND (pnode^.child[i].par >= 0) THEN
            WRITE(''); WRITE(out, '');
        END;
    END;
  ELSE
    BEGIN
      i := -pnode^.par;
      if (i < 0) OR (i > 255) then writeln(' i ', i);
      WriteStringInBuffer (i, TRUE);
      WriteStringInBuffer (i, FALSE);
    END;
END;
PROCEDURE GetFxnNO(str: str15; VAR fno: BYTE);

VAR found: BOOLEAN; sno: BYTE;
BEGIN
  GetSymNo(str, sno);
  fno := 0; found := FALSE;
  WHILE (NOT found) AND (fno < NoOfFcn) DO
  BEGIN fno := fno + 1;
  found := FcnNameTab[fno] = sno;
  END;
  IF NOT found THEN
  BEGIN NoOfFcn := NoOfFcn + 1;
  fno := NoOfFcn;
  FcnNameTab[fno] := sno;
  END;
END;

PROCEDURE GetArgNo(str: str15; fcnno: BYTE; VAR retno, varno: BYTE);

VAR sno: BYTE;
  found: BOOLEAN;
BEGIN
  GetSymNo(str, sno);
  retno := 0; found := FALSE;
  WHILE (NOT found) AND (retno < varno) DO
  BEGIN retno := retno + 1;
  found := FcnTable[fcnno].arg[retno].no = sno;
  END;
  IF NOT found THEN
  BEGIN varno := varno + 1;
  retno := varno;
  FcnTable[fcnno].arg[retno].no := sno;
  END;
END;

PROCEDURE ReadyFcnTable(VAR h: TEXT; VAR fcnno: BYTE);

VAR sno, k: BYTE;
  found: BOOLEAN;
BEGIN
  GetAWord(h); GetSymNo(vocab, sno);
  fcnno := 0; found := FALSE;
  WHILE (fcnno < NoOfFcn) AND (NOT found) DO
  BEGIN fcnno := fcnno + 1;
  found := sno = FcnNameTab[fcnno];
  END;
  IF NOT found THEN
  BEGIN NoOfFcn := NoOfFcn + 1;
  fcnno := NoOfFcn;
  FcnNameTab[fcnno] := sno;
  END;
  GetAWord(h); GetSymNo(vocab, sno);
FUNCTION ComparePaths(path1, path2: PathType): BOOLEAN;

VAR k: BYTE; match: BOOLEAN;
BEGIN
  IF path1[0] <> path2[0] THEN
    ComparePaths := FALSE
  ELSE
    BEGIN
      k := 0; match := TRUE;
      WHILE (k < path1[0]) AND (match) DO
        BEGIN
          k := k + 1;
          match := path1[k] = path2[k];
        END;
        ComparePaths := match;
      END;
  END;
END;

PROCEDURE GetPathNo(path: PathType; VAR pno: BYTE);

VAR tpath: PathType; found: BOOLEAN;
BEGIN
  pno := 0; found := FALSE;
  WHILE (NOT found) AND (pno < NoOfPath) DO
    BEGIN
      pno := pno + 1;
      tpath := PathTable[pno];
      found := ComparePaths(path, tpath);
    END;
  IF NOT found THEN
    BEGIN
      NoOfPath := NoOfPath + 1;
      pno := NoOfPath;
      PathTable[pno] := path;
    END;
END;

PROCEDURE GetStatement(VAR h: TEXT; fcnno, statno: BYTE);

VAR path: PathType;
  str: str15;
  ra, np, pno, smo: BYTE;
BEGIN
na := 2; np := 0;
StatTable[statno, 2].no1 := 2;
WHILE NOT EOLN(h) DO
    BEGIN
        GetAWord(h); str := vocab;
        IF str = '|' THEN
            BEGIN
                path[0] := np;
                GetPathNo(path, pno);
                StatTable[statno, na].no2 := pno;
                na := na + 1;
                StatTable[statno, na].no1 := 2;
                np := 0;
            END;
        ELSE IF str = 'O' THEN
            StatTable[statno, na].no1 := 3
        ELSE IF str = '!' THEN
            StatTable[statno, na].no1 := 4
        ELSE
            BEGIN
                np := np + 1;
                GetSymNo(str, sno);
                path[np] := sno;
                IF str = '*' THEN
                    BEGIN sno := path[np-1];
                        path[np-1] := path[np];
                        path[np] := sno;
                    END;
                END;
            END;
        END;
    END;
    StatTable[statno, 0].no1 := na - 1;
END;

PROCEDURE ReadSemicolon(VAR h: TEXT);

VAR j, k: BYTE;
str1, str2: str15;
fcno, statno, varno, retno: BYTE;
BEGIN
    GetAWord(h); GetSymNo(vocab, fcno);
    GetAWORD(h);
    FcnNameTab[1] := fcno;
    NocOfPath := 0;
    GetStatement(h, 1, 1);
    NocOfFcm := 1; fcno := 1; statno := 1;
    READLN(h);
    WHILE NOT EOLN(h) DO
        BEGIN
            WHILE NOT EOLN(h) DO
                BEGIN
                    GetAWord(h); str1 := vocab;
                    IF str1 = '#' THEN
                        BEGIN
                        END;
                    END;
                END;
        END;
    END;
END;
PROCEDURE FindArgument(FcnItem: ItemOfFcnTable; sm: BYTE; VAR ano: BYTE);

VAR found: BOOLEAN;
BEGIN
  ano := 0; found := FALSE;
  WHILE (NOT found) AND (ano < FcnItem.vmo) DO
    BEGIN
      ano := ano + 1;
      found := FcnItem.arg[ano].no = sm;
    END;
    IF NOT found THEN ano := 0;
  END;
END;

PROCEDURE FindChild(ptr: NodePtr; sm: BYTE; VAR i: BYTE; VAR found: BOOLEAN);
BEGIN
  i := 0; found := FALSE;
  WHILE (i < ptr^.no) AND (NOT found) DO
    BEGIN
      i := i + 1;
      found := ptr^.child[i].par = sm;
    END;
  END;
END;
PROCEDURE ComputeAPath(FcnItem: ItemOfFcnTable; path: PathType;
        VAR rptr: NodePtr; VAR match: BOOLEAN);

LABEL 10;
VAR i,j,ano,resi: BYTE;
p,resp,r: NodePtr;
found, triedstar: BOOLEAN;
ch : char;
BEGIN
  FindArgument(FcnItem, path[1], ano);
  IF ano = 0 THEN
    rptr:= FcnItem.initial.t
  ELSE
    rptr:= FcnItem.arg[ano].t;
    i:= 1; match:= rptr <> NIL;
    triedstar:= FALSE;
  WHILE i< path[0]) AND match DO
    BEGIN
      i:= i + 1;
      IF grsym[path[i]] = '*' THEN
        BEGIN
          p:= rptr;
          FindChild(rptr, path[i+2], j, found);
          IF found THEN
            BEGIN resi:= i;
              resp:= rptr;
              triedstar:= TRUE;
              i:= i + 2;
              rptr:= rptr'.child[j];
            END
          ELSE
            BEGIN
              FindChild(rptr, path[i+1], j, found);
              IF found THEN
                BEGIN i:= i - 1;
                  rptr:= rptr'.child[j];
                END
              ELSE match:= FALSE
            END
        END
      ELSE IF grsym[path[i]] = ' ' THEN
        rptr:= p
      ELSE
        BEGIN
          FindChild(rptr, path[i], j, found);
          IF found THEN
            BEGIN p:= rptr;
              rptr:= rptr'.child[j];
            END
          ELSE match:= FALSE
        END
    END
END;
IF triedstar AND (NOT match) AND (resi < path[0]) THEN
BEGIN
p := resptr;
rptr := p;
i := resi;
triedstar := FALSE;
match := TRUE;
GOTO 10;
END;
END;

PROCEDURE EvaluatePaths(FcnItem: ItemOfFcnTable; PC: BYTE;
VAR rptr: NodePtr);

VAR i: BYTE;
macth: BOOLEAN;
path: PathType;
BEGIN
i := 1;
macth := TRUE;
WHILE (i < StatTable[PC, 0].no1) AND macth DO
BEGIN
i := i + 1;
path := PathTable[StatTable[PC, i].no2];
ComputeAPath(FcnItem, path, rptr, match);
IF StatTable[PC, i].no1 = 4 THEN
macth := NOT macth;
END;
IF NOT match THEN rptr := NIL;
END;

PROCEDURE Rearrangement(FcnItem: ItemOfFcnTable;
VAR cptr: ConceptPtr);

VAR i, j: BYTE;
p: NodePtr;
BEGIN
FOR i := 1 TO FcnItem.arg[0].no DO
BEGIN
NEW(cptr`.cd[i]);
cptr`.cd[i]`.pnode := FcnItem.arg[i].t;
cptr`.cd[i]`.conc := FcnItem.arg[i].no;
cptr`.cd[i]`.no := 0;
FOR j := 1 TO 20 DO
cptr`.cd[i]`.cd[j] := NIL;
p := FcnItem.arg[i].t;
END;
END;

PROCEDURE PrepareFcnCall(IFcnItem: ItemOfFcnTable; PC: BYTE;
VAR match, same: BOOLEAN;
VAR ano, preano: BYTE; VAR RFcnItem: ItemOfFcnTable);
VAR i,j,k: BYTE;
    first: BOOLEAN;
    path: PathType;
    rptr: NodePtr;
BEGIN
    RFcnItem:= FcnTable[StatTable[PC,1].no2];
    i:= 1; match:= TRUE; first:= TRUE;
    WHILE (i < StatTable[PC,0].no1) AND match DO
        BEGIN
            i:= i + 1;
            path:= PathTable[StatTable[PC,i].no2];
            IF grsym[path[1]] = 'NIL' THEN
                j:= j + 1
            ELSE
                BEGIN
                    ComputePath(IFcnItem, path, rptr, match);
                    IF StatTable[PC,i].no1 = 4 THEN
                        match:= NOT match
                    ELSE IF StatTable[PC,i].no1 = 2 THEN
                        BEGIN
                            IF first THEN
                                BEGIN
                                    first:= FALSE;
                                    k:= StatTable[PC,i].no2;
                                    FindArgument(IFcnItem, PathTable[k][1], ano);
                                    preano:= 0; same:= FALSE;
                                    WHILE (preano < IFcnItem.vno AND (NOT same)) DO
                                        BEGIN
                                            preano:= preano + 1;
                                            same:= IFcnItem.dirty[preano] AND
                                                (IFcnItem.arg[preano].t = IFcnItem.arg[ano].t)
                                                AND (preano <> ano);
                                        END;
                                    IF IFcnItem.tag[ano] OR same THEN
                                        match:= FALSE
                                    ELSE
                                        BEGIN
                                            RFcnItem.initial.t:= rptr;
                                            j:= 0;
                                        END;
                                END;
                            ELSE
                                BEGIN
                                    j:= j + 1;
                                    RFcnItem.tag[j]:= TRUE;
                                    RFcnItem.arg[j].t:= rptr;
                                END;
                            END;
                        END;
        END;
END;

PROCEDURE ParsingOfSemicon(FcnItem: ItemOfFcnTable; VAR cpotr: ConceptPtr);
VAR PC, sno, ano, preano: BYTE;
rptr: NodePtr;
first, CallOK, same: BOOLEAN;
FFcnItem: ItemOfFcnTable;
BEGIN
  IF cptr^.node <> NIL THEN
    BEGIN
      cptr^.no := FcnItem.arg[0].no;
      first := TRUE;
      FOR PC := FcnItem.start TO FcnItem.ending DO
        BEGIN
          IF StatTable[PC, 1].noi = 1 THEN
            BEGIN EvaluatePaths(FcnItem, PC, rptr);
              IF rptr <> NIL THEN
                BEGIN
                  ano := StatTable[PC, 1].no2;
                  IF FcnItem.arg[ano].t <> rptr THEN
                    BEGIN
                      FcnItem.tag[ano] := FALSE;
                      FcnItem.arg[ano].t := rptr;
                    END;
                END;
            END ELSE
              BEGIN IF first THEN
                BEGIN
                  first := FALSE;
                  Rearrangement(FcnItem, cptr);
                END;
                PrepareFcnCall(FcnItem, PC, CallOK, same,
                  ano, preano, RFcnItem);
                IF CallOK THEN
                  BEGIN
                    FcnItem.dirty[ano] := TRUE;
                    ParsingOfSemicom(RFcnItem, cptr^.cd[ano]);
                  END ELSE IF same THEN
                    BEGIN
                      NEW(cptr^.cd[ano]);
                      cptr^.cd[ano]^:conc := FcnItem.arg[ano].no;
                    END;
                END;
              END;
            END;
          END;
        END;
      END;
    END;
  END;
PROCEDURE ConstructConTree;

VAR i, j: BYTE;
  FcnItem: ItemOfFcnTable;
  path: PathType;
rptr: NodePtr;
macth: BOOLEAN;
BEGIN
NEW(ConceptTree);
FOR j := 1 TO NodeOfFcm DO
  FOR i := 1 TO FcnTable[j].vno DO
    BEGIN FcnTable[j].arg[i].t := NIL;
    FcnTable[j].tag[i] := FALSE;
    FcnTable[j].dirty[i] := FALSE;
    END;
  FcnItem := FcnTable[1];
  path := PathTable[1];
  FcnItem.initial.t := PTreePtr;
  ComputeAPath(FcnItem, path, rptr, match);
  FcnItem.initial.t := rptr;
  ConceptTree^.pnode := rptr;
  ConceptTree^.conc := path[1];
  FOR i := 1 To 20 DO
    ConceptTree^.cd[i] := NIL;
  ParsingOfSemiCon(FcnItem, ConceptTree);
END;

PROCEDURE WriteConceptTree(cptr: ConceptPtr; w: BYTE);

VAR i, j, k: BYTE;
  found, first: BOOLEAN;
BEGIN
  IF (cptr <> NIL) AND (cptr^.pnode <> NIL) THEN
    BEGIN WRITE('(', grsyn[cptr^.conc], ', ');
    WRITE(out, '(' , grsyn[cptr^.conc], ', ');
    j := LENGTH(grsyn[cptr^.conc]) + w + 2;
    k := 0; found := FALSE;
    WHILE (k < cptr^.no) AND (NOT found) DO
      BEGIN k := k + 1;
      found := cptr^.cd[k] <> NIL;
      END;
    IF NOT found THEN
      PrintMorpheme(cptr^.pnode)
    ELSE
      BEGIN
      first := TRUE;
      FOR i := k TO cptr^.no DO
        BEGIN
        IF (cptr^.cd[i] <> NIL) AND (cptr^.cd[i]^.pnode <> NIL)
          THEN BEGIN IF NOT first THEN
          BEGIN
            WRITEIN; WRITEIN(out);
            WRITE(' :j'); WRITE(out, ' :j');
          END;
          first := FALSE;
          WriteConceptTree(cptr^.cd[i].j);
          END
        END;
      WRITE(')'); WRITE(out, ')');
      END;
    END;
END
END;
END;

PROCEDURE PrintConceptTree;
BEGIN
  WRITELN; WRITELN;
  WRITELN(' **< Conceptual Representation >**');
  WRITELN; WRITELN(out);
  WRITELN(out, ' **< Conceptual Representation >**');
  WRITELN(out); WRITELN(out);
  WriteConceptTree(ConceptTree, 0);
  WRITELN(''); WRITELN(out, '');
END;
PROCEDURE GetSemItem(VAR sup: TEXT; VAR added: BOOLEAN);

VAR i, sno: BYTE;
BEGIN
  SemINo := SemINo + 1;
  GetAWord(sup); i := 0;
  WHILE (vocab <> '|') AND (vocab <> ')' = ') DO
    BEGIN i := i + 1;
      GetSymNo(vocab, sno);
      SemITab[SemINo][i] := sno;
      GetAWord(sup);
    END;
  IF i > 0 THEN
    BEGIN added := TRUE;
      SemITab[SemINo][0] := i;
    END ELSE
    added := FALSE;
END;

PROCEDURE ReadSemConII(VAR sup: TEXT);

VAR i, j: BYTE;
  added: BOOLEAN;
BEGIN
  i := 0; SemINo := 0;
  WHILE NOT EOF(sup) DO
    BEGIN
      i := i + 1; j := 0;
      WHILE NOT EOLN(sup) DO
        BEGIN
          GetSemItem(sup, added);
          IF added THEN
            BEGIN j := j + 1;
              StatITab[i][j] := SemINo;
            END;
          READLN(sup);
          StatITab[i][0] := j;
        END;
      CLOSE(sup);
    MaxSemINStatNo := i;
  END;
END;

PROCEDURE FindPathPtr(q: ConceptPtr; VAR sii: PropertyPtr);

VAR i, j, k, l: BYTE;
  p: NodePtr;
  found: BOOLEAN;
BEGIN
  p := q^.pnode;
  k := p^.par;
  i := 0; found := FALSE;
WHILE (i < p^.no) AND (NOT found) DO
  BEGIN
    i:= i + 1;
    found:= p^.child[i] <> NIL;
  END;
IF NOT found THEN WRITELN('ERROR>> Error in the semicolon supplement');
1:= -p^.child[i]^.par;
j:= 0; found:= FALSE;
WHILE (j < PropCount[1]) AND (NOT found) DO
  BEGIN
    j:= j + 1;
    found:= PropBuff[1, j].sym = k;
  END;
  siiptr:= PropBuff[1, j].prop;
END;

PROCEDURE EvaluatePathII(pathii: PathIType; VAR siiptr: PropertyPtr);
VAR i, j: BYTE;
  q: ConceptPtr;
  match, found: BOOLEAN;
BEGIN
  q:= ConceptTree;
i:= 1; match:= q^.conc = pathii[1];
WHILE (i < pathii[0]) AND match DO
  BEGIN
    i:= i + 1;
    j:= 0; found:= FALSE;
    WHILE (j < q^.no) AND (NOT found) DO
      BEGIN
        j:= j + 1;
        found:= q^.cd[j].conc = pathii[i];
      END;
    IF found THEN
      q:= q^.cd[j]
    ELSE match:= FALSE;
  END;
IF match THEN
  BEGIN
    FindPathPtr(q, siiptr);
    IF siiptr = NIL THEN
      WRITELN('ERROR>> No Property exists.');</n  END
ELSE
  siiptr:= NIL;
END;

PROCEDURE ComparePathII(mprocii, procii: ProcessType; VAR match: BOOLEAN);
VAR i, j, no: BYTE;
BEGIN
  no:= procii[0] DIV 2;
i:= 0; match:= no = mprocii[0];
WHILE (i < no) AND match DO
  BEGIN
PROCEDURE FindRightCat(siiptr: PropArray; no: BYTE;
            VAR catno: BYTE; VAR match: BOOLEAN);

VAR mprocii,procii: ProcessType;
    ptr: PropertyPtr;
    i: BYTE;
BEGIN
    mprocii[0] := no - 1;
    FOR i := 1 TO no-1 DO
        mprocii[i] := siiptr[i+1]^.`no;
        ptr := siiptr[i];
        match := FALSE;
        WHILE (ptr <> NIL) AND (NOT match) DO
            BEGIN catno := ptr^.`no;
                procii := ProcTab[catno];
                ComparePathII(mprocii, procii, match);
                ptr := ptr^.`next;
            END;
        END;
END;

PROCEDURE ParsingOfSemionII;

VAR i,j,catno: BYTE;
    stat: StatIIIType;
    pathii: PathIIType;
    siiptr: PropArray;
    PropError,exist,found: BOOLEAN;
    resptr: PropertyPtr;
    EverTouched,succes{WasReserved: BOOLEAN;
    answer: CHAR;
BEGIN
    i := 0;
    WasReserved := FALSE;
    PropError := FALSE; succes{W := FALSE;
    resptr := NIL; EverTouched := FALSE;
    WHILE (i < MaxSemIIStatNo) AND (NOT PropError) DO
        BEGIN
            i := i + 1;
            stat := StatIIITab[i];
            j := 0; exist := TRUE;
            WHILE (j < stat[0]) AND exist DO
            ...
BEGIN
  j := j + 1;
  siiptr[j] := NIL;
  pathii := SemIIYab[stat[j]];
  EvaluatePathII(pathii, siiptr[j]);
  IF siiptr[j] = NIL THEN
    exist := FALSE;
  END;
  IF exist THEN
  BEGIN
    EverTouched := TRUE;
    IF WasReserved AND (NOT Success) AND (resptr <> siiptr[1]) THEN
      PropError := TRUE
    ELSE
      BEGIN
        IF resptr <> siiptr[1] THEN
          BEGIN
            WasReserved := FALSE;
            Success := FALSE;
          END;
        resptr := siiptr[1];
        FindRightCat(siiptr, stat[0], catno, found);
        IF found THEN
          BEGIN
            WasReserved := FALSE;
            success := TRUE;
          END
        ELSE IF NOT success THEN
          WasReserved := TRUE;
      END;
  END;
  IF PropError OR (NOT EverTouched) OR WasReserved THEN
  BEGIN
    WRITELN; WRITELN; WRITELN(out); WRITELN(out);
    WRITELN('ERROR> Unmatched Properties');
    WRITELN(out, 'ERROR> Unmatched Properties');
    WRITE(' Do you want to see the conceptual tree?(y/n) ');
    answer := ReadKey; WRITELN;
    IF (answer = 'y') OR (answer = 'Y') THEN
      PrintConceptTree;
  END
  ELSE
    PrintConceptTree;
END;
PROGRAM SyntacticParser(INPUT, OUTPUT);
{SR+}

Uses Crt;

{SI DECL.PAS }
{SI MASTER.PAS }
{SI MORI.PAS }
{SI MORII.PAS }
{SI LSI.PAS }
{SI LSII.PAS }
{SI SSL.PAS }
{SI SSII.PAS }

PROCEDURE PutStartingEdge;
  BEGIN
    MVPtr:= MBPtr;
    MEPtr:= MVPtr + 1;
    WITH edges[MEPtr] DO
    BEGIN gr:= 1; pos:= 1;
        stv:= 1; endv:= 1;
        status:= FALSE;
        propose:= FALSE;
        lb:= gram[l].sym[l];
        NEW(q.front); q.rear:= q.front;
        q.rear'.no:= 1; q.rear'.next:= NIL
    END;
  END;

PROCEDURE SetupBuffer;
{ }
  This is the code for 1) recognizing the lexical categories, 2) encoding the
  addresses of Korean words and 3) initializing the edge.
{ }

VAR p: LexPropPtr;
    ptr: QueuePtr;
    NoProp: BYTE;
    i,j: BYTE;
    k: BYTE;
    first,match: BOOLEAN;
BEGIN
  Morphotactics;
  IF ErrorFound THEN EXIT;
  IF Exist2S THEN ExecuteMorII;
  ExecuteLexSupplement;
  FOR j:= 1 TO MBPtr DO
  BEGIN
    IF buffer[j].lsbit THEN
      p:= LexSup[buffer[j].dno].next
    ELSE
      p:= Dictionary[buffer[j].dno]'.next;
    fi:= TRUE;
NoProp := 0;
WHILE p <> NIL DO
  BEGIN
    k := p^.sym;
    IF p^.prop <> NIL THEN
      BEGIN
        NoProp := NoProp + 1;
        PropBuff[j,NoProp].sym := k;
        PropBuff[j,NoProp].prop := p^.prop;
      END;
    WITH edges[j] DO
      IF first THEN [ ENCODE AND PUT THAT NUMBER IN edge. ]
        BEGIN
          q.front := q.front;
          q.rear^.no := k;
          first := FALSE
        END;
      ELSE [ IN THE CASE OF MANY LEXICAL CATEGORIES ]
        BEGIN
          q := q^.front;
          q.rear^.next := q;
        END;
    END;
  END;
  PropCount[j] := NoProp;
  edges[j].q.rear^.next := NIL;
END;
PutStartingEdge;
END;

PROCEDURE PrintBanner;
BEGIN
  ClrScr;
  WRITELN; WRITELN; WRITELN;
  WRITELN('**************************************************************************');
  WRITELN('*');
  WRITELN('WELCOME TO A DYNAMIC DECODER *');
  WRITELN('*');
  WRITELN('**************************************************************************');
  WRITELN; WRITELN;
  WRITELN(' Please follow the prompt information to start up.');
  WRITELN;
END;

PROCEDURE ReadFileAndOpen (VAR gg: TEXT; VAR tword: str15;
                           fsuf: str15; message: str80; VAR bypass: BOOLEAN);

VAR answer: CHAR;
  ok: BOOLEAN;
len, i: BYTE;
ttstr: str15;
BEGIN
  bypass:= FALSE;
  IF fastload THEN len:= 0
  ELSE
    BEGIN
      WRITE(message); READLN(vocab);
      len:= LENGTH(vocab);
      END;
    IF len = 0 THEN
      BEGIN i:= POS('.', tword);
        IF i = 0 THEN
          vocab:= tword + fsuf
        ELSE
          BEGIN
            vocab:= COPY(tword, 1, i-1);
            vocab:= vocab + fsuf;
          END;
        END;
    END
  ELSE
    BEGIN i:= POS(' ', vocab);
      IF i = 0 THEN
        vocab:= vocab + fsuf
      ELSE IF (len-i+1) >= 4 THEN
        BEGIN ttstr:= COPY(vocab, i, 4);
          ttstr:= StUpCase(ttstr);
          IF ttstr <> fsuf THEN
            BEGIN
              WRITE('WARNING> The mismatch of file type');
              WRITE('Do you want to put the name again? (y/n) ');
              READLN(answer);
              IF (answer = 'Y') OR (answer = 'y') THEN
                BEGIN WRITE(message);
                  END;
            END;
      END;
    END;
  END;
REPEAT
  {SI-} ASSIGN(gg, vocab); RESET(gg); {SI+}
ok:= (IOresult = 0);
IF NOT ok THEN
  BEGIN
    WRITE('WARNING> The file ', vocab, ' does not exist. ');
    WRITE('Do you want to continue the execution? ');
    WRITE(' without this dictionary? (y/n) ');
    answer:= ReadKey; WRITELN;
    IF (answer = 'Y') OR (answer = 'y') THEN
      bypass:= TRUE
    ELSE
      BEGIN WRITE(message);
    END;
  ELSE
    ...
READLN(vocab);
i := POS('.', vocab);
IF i = 0 THEN
    vocab := vocab + fsuf;
END;
END;
UNTIL (ok OR bypass);
len := LENGTH(vocab);
i := POS('.', vocab);
tword := COPY(vocab, 1, i-1);
WRITEln(out, message, vocab);
END;

PROCEDURE InitializeParser;

VAR bypass, found: BOOLEAN;
dicn, passw : STRING;
ch: CHAR;
i: BYTE;
gy: TEXT;

BEGIN
    found := FALSE;
    WRITEln; WRITEln; i := 0;
    WHILE (NOT found) AND (i < 3) DO
        BEGIN WRITE(' PassWord: ');
            i := i + 1;
            passw := '';
        REPEAT
            ch := ReadKey;
            IF ch <> $13 THEN passw := passw + ch;
            UNTIL ch = $13; WRITEln;
        IF (passw = 'CHANGIN') OR (passw = 'changin') THEN
            found := TRUE
        ELSE
            WRITEln('Please try again.'); WRITEln
        END;
    IF i > 2 THEN HALT;
    IF ParamCount > 0 THEN
        BEGIN fastload := TRUE;
            dicn := ParamStr(1) + '.OUT';
            WRITEln; WRITEln;
            WRITEln(' Dictionaries are being loaded... ');
        END
    ELSE
        BEGIN fastload := FALSE;
            WRITE('<< Enter the output file name >> : '); READLN(dicn);
            IF dicn = '' THEN dicn := 'NONAME.OUT';
        END;
    ASSIGN(out, dicn); REWRITE(out);
    WRITEln(out, '<< Enter the output file name >> : ', dicn);
    ReadFileAndOpen(gy, dicn, '.MAS',
        'Enter the name of the master dictionary [.MAS]: ', bypass);
    MasterDic := vocab;
ReadMasterDic(gg);
ReadFileAndOpen(gg, dicn, '.MS1',
  'Enter the name of the MS 1 [.MS1]: ',bypass);
ReadMorphicon(gg);
ReadFileAndOpen(gg, dicn, '.MS2',
  'Enter the name of the MS 2 [.MS2]: ',bypass);
ExistMS2:= NOT bypass;
IF NOT bypass THEN
  ReadMrII(gg);
  ReadFileAndOpen(gg, dicn, '.LS1',
    'Enter the name of the LS 1 [.LS1]: ',bypass);
  ReadLexSupp(gg);
  ReadFileAndOpen(gg, dicn, '.LS2',
    'Enter the name of the LS 2 [.LS2]: ',bypass);
  ReadLexicon(gg);
  ReadFileAndOpen(gg, dicn, '.SS1',
    'Enter the name of the SS 1 [.SS1]: ',bypass);
  ReadSemonconI(gg);
  ReadFileAndOpen(gg, dicn, '.SS2',
    'Enter the name of the SS 2 [.SS2]: ',bypass);
  ReadSemonconII(gg);
END;

PROCEDURE ExecuteLogout;

VAR g: TEXT;
  i,j,k,m: INTEGER;
  p: LexPropPtr;
  q: PropertyPtr;
BEGIN
  vocab:= MasterDic;
  i:= FOS(' ', vocab);
  DELETE(vocab, i, 1);
  vocab:= vocab + ' .BAC';
  {$I-} ASSIGN(g, vocab); RESET(g); CLOSE(g); {$I+}
  IF IOResult = 0 THEN ERASE(g);
  ASSIGN(g, MasterDic);
  RENAME(g, vocab);
  ASSIGN(g, MasterDic);
  REWRITE(g);
  WRITELN(g, '# ', maxterm, ' Lexical Categories: ');
  WRITE(g, ' TYPE1 = ');
  FOR i:= TypePtr[1] TO TypePtr[2]-1 DO
    WRITE(g, ' ', gsym[i]);
  WRITELN(g);
  WRITE(g, ' TYPE2 = ');
  FOR i:= TypePtr[2] TO TypePtr[3]-1 DO
    WRITE(g, ' ', gsym[i]);
  WRITELN(g);
  WRITE(g, ' TYPE3 = ');
  j:= 0;
  FOR i:= TypePtr[3] TO maxterm DO
    BEGIN j:= j + 1;
WRITE(g, ' ', grsym[i]);
IF (j > 8) AND (i < maxterm) THEN
BEGIN j:= 0;
  WRITELN(g);
  WRITE(g, ' ');
END;
END;
WRITELN(g);
WRITELN(g, '# ', maxprop, ' TYPE1 Property Categories:');
FOR i:= 1 TO maxprop DO
  WRITELN(g, ' ', i, ' ', propsym[i]);
WRITELN(g, '# ', maxproc, ' TYPE2 Property Categories:');
FOR i:= 1 TO maxproc DO
  BEGIN WRITE(g, ' ', i, ' ');
    FOR j:= 1 TO ProcTab[i][0] DO
      WRITE(g, ' ', ProcTab[i][j]);
    WRITELN(g);
  END;
WRITELN(g, '# Vocabularies:');
FOR i:= 1 TO DNPtr DO
  BEGIN WRITE(g, ' ', Dictionary[i]`.vocab);
    k:= 15 - LENGTH(Dictionary[i]`.vocab);
    FOR m:= 1 TO k DO
      WRITE(g, ' ');
    WRITE(g, ' ', Dictionary[i]`.num,' ');
    p:= Dictionary[i]`.next;
    FOR j:= 1 TO Dictionary[i]`.num DO
      BEGIN k:= 15 - LENGTH(grsym[p`.sym]);
        WRITE(g, grsym[p`.sym]);
        IF p`.prop <> NIL THEN
          BEGIN m:= 1 TO k DO WRITE(g, ' ');
            WRITE(g, ' ');
          END;
        q:= p`.prop;
        WHILE q <> NIL DO
          BEGIN WRITE(g, ' ', q`.no);
            q:= q`.next;
          END;
      WRITELN(g);
      p:= p`.next;
      IF p <> NIL THEN WRITE(g, ' ');
    END;
  END;
CLOSE(g);
END;

BEGIN {Main Program}
  PrintBanner;
  InitializeParser;
  quit:= FALSE;
  REPEAT
    CLRScr;
  ...
ErrorFound := FALSE;
SyntaxError := TRUE;
FirstParse := TRUE;
WRITE;
WRITE('>> ');
WRITE(out, '>> ');
PrepareInput;
SetupBuffer;
FPtr := MVPtr;
REPEAT
  found := FALSE;
  IF NOT ErrorFound THEN
    Parser(found);
    WRITE(line, WRITE(out);
  IF found THEN
    BEGIN
      ConstructTree(NIL, startno, 0, PTreePtr);
      WRITE(out);
      WRITE(out);
      WRITE(' **<The parsed list>**');
      WRITE(out);
      WRITE(out, ' **<The parsed list>**');
      WRITE(out);
      PrintParseList(PTreePtr, 0);
      WRITE('');
      WRITE(out, ' ');
      ConstructConFunTree;
      ParsingOfSemConII;
      WRITE(out);
      WRITE(out);
      WRITE(out);
      WRITE(out);
      FOR
      BEGIN
        WRITE('Please hit the RET key to continue. ');
        ch := ReadKey;
        WRITE(out);
      END;
      UNTIL (FPtr >= MSPtr) OR ErrorFound;
      WRITE('Do you want to continue(y/n)? ');
      ch := ReadKey;
      quit := (ch = 'n') OR (ch = 'N');
      UNTIL quit;
      CLOSE(out);
      CLRScr;
      REPEAT
        WRITE('Do you want to save all your work(y/n)? ');
        ch := ReadKey;
        ch := UPCASE(ch);
        IF ch = 'Y' THEN ExecuteLogout;
        UNTIL ch IN ['N', 'Y'];
      END.
FOOTNOTES

1. During the last decade, the ATN (Augmented Transition Network) formalism, which involves context sensitivity, has gained a great deal of attraction and has appealed to many researchers, e.g. Woods (1970), Kaplan (1973), Winograd (1983). Its appeal lies in its ability to deal with various complicated syntactic phenomena. Still, it has trouble with problems that have been difficult for syntactic formalisms generally, such as conjunctions, comparatives and ellipsis. ATN grammars also represent the function of a syntactic constituent (ex. Subject, Object, etc.). An ATN grammar apparently has some advantages. There is no requirement for building a separate 'syntactic tree structure' for a sentence, and semantic organization can be embedded directly in the grammar. However, people working on ATN grammar do not stratify the linguistic structure, which is recognized by linguists.

2. The plural marker _-tul_ is enclosed in parentheses when it is optional.

3. Earlier studies (Kim 1974, Song 1981 and Y.J. Kim 1985 among others) have observed semantic conditions from which we can predict the choice of nominalizer _m_ or _ki_. Kim (1974) and Song have suggested the factor 'factivity' as a criterion in choosing _m_ over _ki_; _m_ is then said to be a factive and _ki_ a nonfactive nominalizer. Factivity is meant that 'truth' of covert
proposition is presupposed by the speaker. Consider the following example borrowed from Y.J. Kim (1985:168).

(1) Na-nun ku-ka oykwukin-i- m-ul musiha-n-ta.  
    *ki-lul
    I nTop he-nSubj foreign-be ignore-PreT-DEC(N)
    'I ignore his being a foreigner.'

Since the covert proposition that 'He is a foreigner' is presupposed by the speaker, m is an appropriate form in sentence (1). Y.J. Kim (1985:168-69), however, has suggested some counter examples to Kim's (1974) and Song's proposal, and she has added another semantic factor called 'influence' in addition to 'factivity.' She proposes that if an event exerts either a positive or negative influence on the realization of the covert proposition, ki rather than m is used. She gives the example (pp.170).

(2) John-un te mek- *um-ul kecelha-yess-ta.  
    ki-lul
    John-nTop more eat-ing refuse-PasT-DEC(N)
    'John refused to eat more.'

The sentence 'John refused to eat more' suggests that John has done something to block (influence negatively) the realization of John's eating more.

4. Comrie (1976:25) calls aspects that are not perfective, collectively, 'imperfective'. However, the term 'imperfective' used in this study is synonymous with 'progressive.' Comrie in his introduction to Aspect (1976) and Time (1985) states that 'although both aspect and tense are concerned with time, they are concerned with time in different ways.'
characterizes tense as the time of an action in a sentence with reference to the 'Speech Time'--he characterizes the narrated event in relation to the speech event. On the other hand, aspect is not concerned with relating the time of the situation to any other time, but rather with the internal temporal constituency of one situation. Comrie states the difference as one between "situation-internal time (aspect)" and "situation-external time (tense)."

5. In Korean, some clauses are 'final' (the verb at the end completes a sentence) and some are nonfinal (the verb form does not complete a sentence).

6. In addition to the morphological and syntactic devices, the speaker uses honorific nouns, verbs, or participles to show his respect to the person spoken about or to.

For example:

<table>
<thead>
<tr>
<th>Neutral</th>
<th>Honorific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nouns</td>
<td></td>
</tr>
<tr>
<td>apeci</td>
<td>apenim</td>
</tr>
<tr>
<td>mal</td>
<td>malssum</td>
</tr>
<tr>
<td>nai</td>
<td>yensey</td>
</tr>
<tr>
<td>swul</td>
<td>yakcwu</td>
</tr>
<tr>
<td>pyeng</td>
<td>pyengwhan</td>
</tr>
<tr>
<td>Verbs</td>
<td></td>
</tr>
<tr>
<td>ca-</td>
<td>cwumwusi-</td>
</tr>
<tr>
<td>iss-</td>
<td>keysi-</td>
</tr>
<tr>
<td>mek-</td>
<td>capswusi-</td>
</tr>
<tr>
<td>cwu-</td>
<td>tul-</td>
</tr>
<tr>
<td>Markers</td>
<td></td>
</tr>
<tr>
<td>eykey</td>
<td>kkey</td>
</tr>
<tr>
<td>ka</td>
<td>kkeyse</td>
</tr>
</tbody>
</table>

7. Smith (1978) approaches temporal interpretation with a three-way distinction--Speech Time (SpeechT), Reference
Time (ReferenceT), and Event Time (EventT). Roughly speaking, we can define these temporal notions as follows: SpeechT is the moment of utterance, ReferenceT is the time indicated by a sentence, and EventT refers to the moment at which the relevant event or state occurs.

8. Some verbs are derived from nouns, e.g. panucil 'sewing,' wuntong 'exercise,' kongpwo 'study,' etc., by attaching the denominal verbalizer -ha.

9. Words in linguistics are not always identical with the layman's concept of words, which are defined in terms of potential pauses or spelling. Words are on the average larger than single morphemes and smaller than utterances. Some linguists such as Bloomfield (1933) and Hockett (1958) define a word as the 'minimal free form.' Even if twenty-eighth looks like two words, it is only one minimum free form, i.e. one of its ICs (-th) is bound. According to them, even the nine morpheme sequence in the example (Winograd 1983:68) the man you met in Ankara last year's is a single minimum free form, since the bound form--'s is one of its IC's. The term 'graphemic word' used in this study, however, is delimited by spaces.

10. Even though passive is a lexeme, it is treated in the Lexemic Supplement II in the present system. This analysis is chosen because the passive is expanded from a verb phrase rule.
11. Traditional study has not been concerned with complex lexemes. Because of this the dictionary for complex lexemes is neither morphological nor syntactic information in traditional sense of morphology and syntax. The present study includes the dictionary in syntactic decoding since the dictionaries for morphological information deal with suffixes.

12. The use of number for recognizing a discontinuous constituent is a temporary expedient, not a valid solution.

13. We can classify parsing strategies using three dimensions:

(cf. Winograd 1983: Chap 3)

(1) Top-Down vs. Bottom-Up
(2) Backtracking vs. Parallel
(3) Left-to-Right vs. Right-to-Left

14. Some approaches attempt to analyze the meaning of natural language sentences. Schank (1972) attempts to compute meaning by going directly from graphemic to conceptual representations. He suggests that syntactic constructions are redundant and unrelated to decoding the meaning of an input sentence. His graph system, which has influenced psychologists like Abelson among others, uses structures like those of dependency grammar (Hays 1964). The terms in the graph are of four types, or conceptual categories. They are symbolized as PP, ACT, PA and AA, and they correspond closely to the syntactic categories, Noun, Verb, Adj.
and Adverb, respectively. He represents conceptual structure as follows for the sentence like He ate lunch:

\[ \text{He} \rightarrow \text{eat} \rightarrow \text{lunch} \]

Here, 'P' indicates past and is the dependency symbol linking a PP ('He') to the ACT ('eat'), which is the nub of the conceptualization, as with Simons (1970). The 'O' indicates the objective case, making the dependence of the object PP on the central ACT. Schank uses 11 ACTs as primitives for representing various groups of events. His primitives such as ATRANS (for the conceptual representation of 'possession-changing-actions') or PROPEL (for the conceptual representation of 'application of force') represent groups of events such as trade, buy, exchange, give and hit suck, punch, respectively. Other of Schank's primitives, such as MTRANS (for the conceptual representation of Mental Transfer), involve indexes for the state or condition of an actor, e.g. HEALTH, ANGER (cf. Schank 1984:90-109). Wilks' (1973) system is also considered as a uniform representation, in that information that might be considered as syntactic, semantic and inferential is all expressed within a single type of structure. The fundamental unit in the construction of this meaning representation is the 'template.' Templates are rigid format networks
of more basic building blocks called 'formulae', which correspond to senses of individual words. Each word has its own dictionary entry which consists of word senses. The dictionary entry can be represented in the form of a tree structure of semantic primitives, and is to be interpreted formally by means of dependency relations. The main element in any formula is the rightmost, called its head, and that represents the fundamental category to which the formula belongs. Let us take as an example the tree structure for the action of drinking:

```
(ANI SUBJ) (OBJ) (SELF IN) (TO) (MOVE CAUSE)

(FLOW STUFF) (THIS)

(ANI)

(THRU PART)
```

The sense of drink is expressed as a causing to move a liquid object (FLOW STUFF) by an animate agent, into that same agent (containment case indicated by IN, and formula syntax identifies SELF with the agent) and via an aperture (THRU PART) of the agent. Schank and Wilks argue that it is not necessary to preserve a distinction between syntax and semantics in an understanding
system. They consider that there is no need to separate them and consequently they integrate syntactic and semantic routines.

15. The earlier treatment used in dealing with participant roles of a language is to decide on some small number of participant roles for a whole language, e.g. Fillmore (1969), Lockwood (1973). Although Halliday (1976, 1985) has recognized some additional roles such as 'processor' or 'phenomena' for the verbs see, hear or smell. Lamb (1987) suggests that further analysis indicates that each process defines its own participants. There can thus be indefinitely many different participants roles for a given conceptual system. Following Lamb, in this study, we use the simple labels participant-1, participant-2, etc. instead of inventing new terms for the roles for each process; or for short, 'P1', 'P2', etc.

16. To 'cut' a parsing tree means to connect a semantic tree to an appropriate syntactic pathway. That is, to 'cut' a parsing tree for P1, P2, Process, etc. means to make the semotactic functions point to their corresponding syntactic nodes in a parsing tree. Let us take an example with a syntactic and semantic tree. Figure 1 is a semantic tree and Figure 2 is a syntactic tree for the English input sentence The salesman sold the IBM PC.
17. The subject is usually conceived of 'purely' grammatical—that is not realizing any semantic features. According to Halliday (1983), however, the subject is semantically motivated. The reason we are interested in the grammatical function of subject, rather than in the meaning of the category is because the fundamental concept, like that underlying subject, is ineffable.

18. Since the present system implements dominant semantic properties instead of semantic paths, the properties 'Human Being' and 'Animal' are separately listed (even though human beings are automatically animals).

19. When a class is defined by common properties, they are
called 'intensional properties.' When a class is defined by listing its subsets or members, they are called its extension. Any concept in a conceptual network has both intension and extension.

20. The present study deals with a new string which is either a NOUN or a VERB and is also morphologically simple.
REFERENCES


Choi, H.B. 1937. Wulimalpon (Our Grammar), Ynehicenmwun Chwulphanpwu (Yensei University Press).


Gazdar, G. 1985. NLs, CFLs and CF-PSGs. Automatic Natural Language Processing, ed. by K. Jones and Y. Wilks. 81-93.


