COMPUTER BASED STRUCTURAL ANALYSIS OF DEVERBAL WORDS IN TURKISH

A MASTER'S THESIS
in
Electrical and Electronics Engineering
Middle East Technical Universitiy

By

Serpil AYDIN

January, 1991

Approval of the Graduate School of the Natural and Applied Sciences

Prof. Dr. Alpay ANKARA

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.

Prof. Dr. Erol KOCAOĞLAN

Chairman of the Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.

Asst. Prof. Dr. M. Mete BULUT

Supervisor

Examining Committee in Charge:

Assoc. Prof. Dr. Güney GÖNENÇ (Chairman)

Assoc. Prof. Dr. Melek YÜCEL

Assoc. Prof. Dr. Semih BİLGEN

Asst. Prof. Dr. Işık AYBAY

Asst. Prof. Dr. M. Mete BULUT

ABSTRACT

COMPUTER BASED STRUCTURAL
ANALYSIS
OF
DEVERBAL
WORDS IN TURKISH

AYDIN, Serpil
M.S. in Electrical and Electronics Engineering
Supervisor: Asst. Prof. Dr. M. Mete Bulut
January 1991, 118 pages

In this thesis work, a software for the structural analysis of deverbal words for Turkish language is developed.

The program has four main units: Root identification, suffix recognition, meaning extraction, and learning. For the analysis of words the first three units are used. The function of the fourth unit, learning, is to teach the unknown roots to the system.

The steps for the analysis of a given word are as follows: First, identify the root of the word using depth first search method.

For this step, a data bank called Root Bank is used. If it is not possible to identify the root of a given word the learning unit is activated. As the second step, the suffixes affixed to the root are found. For this operation, another data bank, which is called SuffixBank, is used.

Once the root and the suffixes which constitute the given word are found, the meaning of the word is determined using a third data bank, which is called Meaning Bank. To do so the meaning changes induced by each suffix is extracted from the Meaning Bank, and concatenated in a proper manner to find the overall description of the given word.

For the developed software, the source code occupies 97 KBytes. For the data banks 18 KBytes are used. The analysis time of a given word with two suffixes, which is the most common usage in Turkish, is 1.50 seconds for an IBM-AT compatible computer.

Key Words: Deverbal word, structural analysis, suffix, suffix deriving verbs from verbal roots, suffix deriving nouns from verbal roots.

Science Code: 619.02.05.

ÖZET

TÜKÇE' DEKİ FİİL SOYLU SÖZCÜKLERİN
BİLGİSAYARA
DAYALI
YAPISAL ÇÖZÜMLENMESİ

AYDIN, Serpil
Yüksek Lisans Tezi, Elektrik ve Elektronik Mühendisliği
Tez Yöneticisi: Yrd. Doç. Dr. M. Mete Bulut
Ocak 1991, 118 sayfa

Bu tez çalışmasında, Tükçe'deki fiil soylu sözcüklerin çözümlenmesi için bir yazılım geliştirilmiştir.

Program kök belirleme, sonek tanıması, anlam çıkarma ve öğrenme olmak üzere dört esas üniteden oluşmaktadır. Kelimelerin çözümlenmesi için ilk üç ünite kullanılmaktadır. Dördüncü ünitenin işlevi bilinmeyen kökleri sisteme öğretmektir.

Verilen bir kelimenin çözümlenmesi iki basamaktan oluşur: İlk basamakta kelimenin kökü derinliğine tarama metodu kullanılarak bulunur. Bunu yapabilmek için Kök Bankası adı verilen

bir veri bankası kullanılır. Verilen kelimenin kökünün belirlenmesi mümkün değilse, öğrenme ünitesi faaliyete geçer. İkinci basamakta köke gelen sonekler bulunur. Bu işlemi yapabilmek için Sonek Bankası adı verilen başka bir veri bankası kullanılır.

Kelimeyi oluşturan kök ve sonekler bulunduğunda kelimenin tanımlaması Anlam Bankası adı verilen üçüncü bir veri bankası kullanılarak belirlenir. Bunun için her sonekin meydana getirdiği anlam kaymaları Anlam Bankası'ndan çıkarılır ve bunlar kelimenin tam tanımlamasını vermek üzere uygun bir şekilde birleştirilir.

Geliştirilen yazılım 97 KByte yer tutmaktadır. Veri bankaları için toplam 18 KByte kullanılmaktadır. Türkçe'de en çok kullanılan bir biçim olan iki sonekli bir sözcüğün IBM-AT uyumlu bir bilgisayarca tanınması 1.50 saniye tutmaktadır.

Anahtar Kelimeler: Fiil soylu sözcük, yapısal çözümleme, sonek, fiil soylu kökten fiil türeten sonek, fiil soylu kökten isim türeten sonek.

Bilim Dalı Sayısal Kodu: 619.02.05.

ACKNOWLEDGMENTS

I am grateful to my supervisor Asst. Prof. M. Mete Bulut for his valuable advice and criticism as well as his strong encouragement, guidance, and the enthusiasm throughout the planning and development of this thesis.

I owe special thanks to Assoc. Prof. Dr. Güney Gönenç and Assoc. Prof. Dr. Semih Bilgen for their helpful comments and suggestions.

I would like to express my particular thanks to my family and especially to Asst. Erkan Tın, Department of Computer Engineering and Information Sciences, Bilkent University for their infinite moral support and motivation, particularly in times of despair and flurry.

TABLE OF CONTENTS

Page

ABSTRACT iii

ÖZET v

ACKNOWLEDGEMENTS vii

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ABBREVIATIONS xiii

1 INTRODUCTION **1**

1.1 Scope of the Thesis 2

2 GENERAL CHARACTERISTICS OF THE TURKISH LANGUAGE **4**

2.1 Sounds in the Turkish Language 5

2.1.1 Vowels and Vowel Harmony 5

2.1.2 Consonants, Consonant Harmony and Changes in Consonants 6

2.2 Possessive Suffixes 7

Page

2.3 Classification of Turkish Words 8

2.4 Verbal Roots and Their Suffixes 8

2.5 Possible Structures of the Deverbal Words 9

2.6 Conjugation of the Verbs 10

2.7 Verb Structure 11

2.8 Basic Tenses 12

2.8.1 Informative Tenses 13

2.8.2 Optative Tenses

2 .9 Compound Tenses 15

2.10 Compound Verbs 15

2.11 Personal Endings 16

3 DEVERBAL WORD RECOGNITION BY USING KNOWLEDGE BANKS 17

3.1 Root Bank 18

3.1.1 Structure of the Root Bank 18

3.1.2 Deformation Positions for a Verbal Root 21

3.2 Suffix Bank 23

3.2.1 Structure of the Suffix Bank 24

3.2.2 Rules for the Suffixes 26

3.3 Meaning Bank 30

Page

3.4 Data Structure Used 31

3.5 How to Set Turkish Letters and Their Ordering 33

4 SOFTWARE 36

4.1 Introduction 36

4.2 Determining the Root of a Given Word 39

4.3 Extracting the Suffixes of a Given Word 46

4.4 The Meaning of a Given Word 57

4.5 How to Learn a New Root 65

5 CONCLUSION 73

LIST OF REFERENCES 77

APPENDICES 79

A USER'S MANUAL FOR DEVERBAL WORD RECOGNITION PACKAGE **79**

B ROOT BANK **90**

C SUFFIX BANK **94**

D RETURN NUMBERS AND THE MEANING BANK **102**

D.1 Meaning Bank 102

D.2 Description of the Return Numbers 106

E GLOSSARY **117**

LIST OF TABLES

Table		Page
3.1	Form change, root structure, and return numbers for some roots	21
3.2	Examples for suffix knowledge	23
3.3	Forward Conversion	34
3.4	Backward Conversion	34

LIST OF FIGURES

Figure		Page
3.1	Block diagram for analysis of a word	17
3.2	Doubly linked circular list	32
4.1	Flowchart for analysis of a word	37
4.2	Software structure of the main program	39
4.3	Flowchart for determining the root of a given word	42
4.4	Software structure of determining the root of a given word	44
4.5	Flowchart for the extraction of the suffixes	50
4.6	Software structure of extracting the suffixes	53
4.7	Software structure of the meaning concatenation	63
4.8	Flowchart for learning a new root	66
4.9	Software structure of the learning	69

LIST OF ABBREVIATIONS

RTN	Return Number
SDVV	Suffix Deriving Verbs from Verbal roots
SDNV	Suffix Deriving Nouns from Verbal roots
TS	Tense suffix
PE	Personal Ending

CHAPTER 1

INTRODUCTION

There are some studies for translation to/from Turkish [12], frequency count of Turkish words [13], grammatical structure of Turkish [1-5], [7-9], and origin of the spoken Turkish [6].

Our study is the first one for the computer based recognition of the Turkish language.

In order to understand the meaning of a sentence, all types of the words and combinations must be recognized. Since the recognition of all the words of a language is an involved study, the scope of this thesis work is restricted by the recognition of the deverbal words of Turkish.

Although computer aided recognition of Turkish words seems to be difficult, it is not impossible. Turkish is characterized by a great regularity of patterns and also it has a few irregular words taken from other language families.

Turkish is an agglutinative type of language. There exist a root and a number of suffixes for each word. For this reason, our study uses systematic search for the decomposition of the deverbal words.

In order to accomplish the decomposition, all the relevant rules of the Turkish language are used.

In order to analyse a word structurally, the root and the meaning modifications caused by all the suffixes attached to the root must be recognized. The first step of the analysis process is to identify the root of a given word, and next to determine both inflection and derivation which are induced by the suffixes. Therefore, the root and the suffixes of a deverbal word are found by using knowledge about verbal roots, suffixes which can be attached, and grammatical rules. Knowledge banks are constructed to store verbal roots, their suffixes and meanings. The root and the suffixes of a given word are determined using the Root Bank and the Suffix Bank, respectively. Once the suffixes are determined the meaning changes induced by them are extracted from the Meaning Bank. As the final step of the analysis, the description of the given word is found by the concatenation of the root and suffix inductions in a proper way.

1 .1 SCOPE OF THE THESIS

This thesis describes the implementation aspects of recognizing and finding the meanings of deverbal words in Turkish language using the knowledge banks. For this purpose, a program has been developed. Given words are analysed by using knowledge about verbal roots and their suffixes.

In chapter 2, general characteristics of Turkish language have been used to explain the word-formation of Turkish words. Sound properties of Turkish (i.e., vowel harmony and consonant harmony), verbal roots and their suffixes, possible structure of the deverbal

words, conjugation of the verbs, verb structure, basic and compound tenses and compound verbs are explained in this chapter.

Chapter 3 introduces a detailed insight into the developed program for the recognition of a deverbal word. The program uses three knowledge banks (Root Bank, Suffix Bank, and Meaning Bank) and their structures are proposed in this chapter. Moreover, deformation positions for a verbal root, the general rules and the specific rules for the suffixes are introduced.

Searching method for extracting the root and the suffixes of a given word, concatenating the meaning of the suffixes, and the function of each module are explained in Chapter 4. The flowchart and the software structure of the main procedure and its subprocedures are also given.

Finally, Chapter 5 contains the concluding remarks together with some comments for future research.

In Appendix A, a user's manual is given. The messages appearing on the screen are shown in the user's manual and they are written in Turkish as seen on the screen.

Appendix B and C shows the Root Bank (verbal roots and their identity codes), Suffix Bank (suffixes and their identity codes).

The Meaning Bank (meaning of the suffixes) and how to combine these meanings to a root are shown in Appendix D.

A glossary is provided in Appendix E.

CHAPTER 2

GENERAL CHARACTERISTICS OF THE TURKISH LANGUAGE

Turkish is an agglutinative type of language that belongs to Ural-Altaic language family. In agglutinative languages, there is a root which is the simplest form of a word, and there is a number of suffixes. Each of these suffixes has a different function. One or more suffixes are affixed to a root in order to extend the meaning of the word or create other classes of words [3].

Turkish is characterized by great regularity of patterns, and also it has a few irregular words. Both inflection (conjugation of verbs and declension of nouns), and derivation are indicated by suffixes which are affixed to the roots. Therefore, affixes can be classified into two groups as derivational affixes and inflectional affixes [7].

By using derivational affixes, verbs can be derived from nouns or adjectives, nouns from verbs or adjectives, and also adjectives from nouns or verbs. In other words, derivational affixes change the meaning of the words to which they are added. Let us examine the following examples : "bil" is a verb, and its meaning is "know", but "bilmece" is a noun which means "riddle,puzzle"; "taş" is a noun that means "stone", but "taşlaş" is a verb, and it stands for "petrify, turn to stone".

As it is seen in these examples, new words are produced by means of derivational affixes. However, inflectional affixes do not have any function in word-formation, that is, there is no change in the semantic meaning or class of the word. For example, although "ev" is a nominative form and "evi" is an accusative form, declension of noun can not cause any difference on word-formation [1], [7].

Another important feature of Turkish is to have no distinction of gender (masculine, feminine, neuter).

Before going further, a brief knowledge about vowels, consonants, class of roots, their suffixes, and possible word structures will be examined.

2 .1 SOUNDS IN THE TURKISH LANGUAGE

Turkish alphabet consists of 29 letters, and these letters can be divided into two groups according to their sounds: Vowels, Consonants [6].

2 .1 .1 Vowels and Vowel Harmony

There are eight vowels in Turkish: a, e, ı, i, o, ö, u, ü. These vowels are classified into six groups according to their properties. This classification is as follows [15]:

back vowels	:	a, ı, o, u
front vowels	:	e, i, ö, ü
unrounded vowels	:	a, ı, e, i
rounded vowels	:	o, ö, u, ü
closed vowels	:	ı, i, u, ü
wide vowels	:	a, e, o, ö

There is also vowel harmony in Turkish, and there are some rules for this [1].

i) If the first syllable of a word has a back vowel, vowels of the subsequent syllables are also back ones. If the vowel of the first syllable is a front vowel, the vowels of the other syllables are also front vowels. *For example:* sıcak /hot; atmak /to throw.

ii) Unrounded vowels are followed by unrounded vowels; rounded vowels are followed by either closed-rounded (u,ü) or wide-unrounded ones (a,e). *For example:* gözlük/ glasses; okumak /to read.

iii) The vowels of the suffixes are subject to the vowel harmony, that is, they change in accordance with the vowel of the last syllable of the word. *For example:* -a > sınıf-a, gel-e; -tır > bak-tır, koş-tur.

If the suffix follows another suffix,its vowel changes according to the vowel of the preceding suffix (*e.g.*, ev-de-dir-ler).

A few Turkish words (*e.g.*, inanmak /to believe; avuç/palm of the hand), and foreign words do not obey these rules. There are also six invariable suffixes: -yor, -ken, -daş, -leyin, -ki, -(ı)mtırak.

2.1.2 Consonants, Consonant Harmony and Changes in Consonants

Consonants in Turkish are divided into two groups as harsh and soft consonants [3].

Unvoiced consonants:	ç, f, h, k, p, s, ş, t.
Voiced consonants:	b, c, d, g, ğ, j, l, m, n, r, v, y, z.

The Turkish words mostly end in a harsh consonant; especially, the soft consonants b,c,d and g are rarely found at the ends of

Turkish words. If there is one of these consonants at the end of a foreign word, it changes to the corresponding harsh sound of p,ç,t or k, respectively (*e.g.*, kitab - kitap).

When a suffix beginning with a vowel is attached to a word ending with p,ç,t or k, the end of a word is mostly softened (i.e. it changes to b,c,d or ğ). This rule especially works when the harsh consonant falls between two vowels. However, it rarely works in verb-stems, such as git-, güt-, tat-.

If a suffix beginning with c,d or g is added to a word ending with harsh consonant, the initial consonant of the suffix becomes harsh (ç,t or k respectively). *For example,* if the suffix "di" is affixed to the root "git", the word "git-ti" is obtained.

After attaching a suffix beginning with a vowel to some words of two syllables, the vowel of the second syllable drops. *For example,* ağız > ağzım.

Some foreign words which end in a thin consonant, and whose last syllable has an a,o or u take suffixes with front vowels. *For example,* hal> halde.

2 .2 POSSESSIVE SUFFIXES

If a word ends with a consonant, the following possessive suffixes are used [3], [15]:

-im	(-ım,- um, -üm)	(benim /my)
-in	(-ın, -un, -ün)	(senin /your)
-i	(-ı, -u, -ü)	(onun /his,her,its)
-imiz	(-ımız, -umuz, -ümüz)	(bizim /our)

-iniz (-ınız, -unuz, -ünüz) ..... (sizin /your)

-i (-ı,- u, -ü) ..... (onların /their)

However, if a word ends with a vowel these suffixes change. For example, first singular possessive suffix changes from -im (-ım, -um, -üm) to -m. Third singular and plural possessive suffix also becomes -sı (-si, -su, -sü).

The plural suffix in Turkish is -ler (-lar), and is added directly to the word before any possessive ending. *For example:* Ler-im/ ler-in/ ler-i/ ler-iniz/. -Ler is used when the last syllable of the word has a front vowel, -lar is used if the vowel of the last syllable is a back one.(*e.g.*, ev - evler, kapı - kapılar)

2 .3 CLASSIFICATION OF TURKISH WORDS

Turkish words are divided into four groups according to word bases, that is the root or stem of words [7].

i) Denominal Nouns,

ii) Denominal Verbs,

iii) Deverbal Nouns,

iv) Deverbal Verbs.

Since this thesis studies the use of deverbal words in Turkish with the aim of recognizing and finding the meanings of them, only the verbal roots and their suffixes will be explained in the next section.

2 .4 VERBAL ROOTS AND THEIR SUFFIXES

Classification of the suffixes which are affixed to the deverbal roots is as follows [5]:

i) Suffixes for derivation:

- Suffixes which derive nouns or adjectives from verbal roots,

e.g. - ak > yat-ak /bed

- Suffixes which derive verbs from verbal roots,

e.g. - ele > silk-ele /shake off

- Suffixes which derive verbs, nouns or adjectives from verbal roots,

e.g. - t > taş-ı-t

The word *taşıt* has two meanings. The first one is a verb and its meaning is *to make someone carry something.* The second one is a noun and it means *vehicle.*

ii) Suffixes for conjugation:

- Tense suffixes
  - suffixes for basic tenses

    e.g. - ecek > gel-ecek /He(she,it) will come.
  - suffixes for compound tenses

    e.g. - ti > gel-miş-ti-m /I had come.
- Personal endings

  e.g. - m > yaz-dı-m / I wrote.

2 .5 POSSIBLE STRUCTURES OF THE DEVERBAL WORDS

After affixing the suffixes to a root, the root changes to either a deverbal verb or a deverbal noun.

VR + (SDVV) + (TS)* + (PE)* = Deverbal Verb*

Example : kov - ala - dı - m /I pursued

VR + (SDNV) $^{+}$ *= Deverbal Noun*

Example : gül - üş /laugh

Abbreviations used in the structure :

VR : Verbal Root

SDVV : Suffix which derives verbs from verbs

SDNV : Suffix which derives nouns from verbs

TS : Tense Suffix

PE : Personal Endings

PS : Possessive Suffixes

An asterisk (*) is used to indicate zero or more repetitions, and a plus sign (+) shows that one or more repetitions are allowed.

2 .6 CONJUGATION OF THE VERBS

The infinitive forms of the Turkish verbs terminate with -mek/ -mak. The first part of the infinitive to which -mek/-mak is attached is called *root* or *stem* [15].

General Pattern :

(Verb - Stem) + (Tense Suffix) + (Personal Endings)

Example: yürü-r-üm /I walk.

There are some irregular positions for this pattern.

e.g. oku-yor-lar-mış, where personal ending "lar" is between two tense suffixes.

We can make the verb affirmative or negative.

a. Affirmative

(stem) + (tense suffix) + (personal endings)

Example: yaz-dı-m /I wrote.

b. Negative

(stem) + (negative suffix) + (tense suffix) + (personal endings)

Example: yaz-ma-dı-m /I didn't write.

There is an exception for the negative of the simple present. In the first singular and plural persons, personal ending directly follows the negative stem without taking any other suffix (*e.g.*, gel-me-m). In other persons, "z" is used instead of positive tense suffix -(i/ı/u/ü)r or -(e/a)r (*e.g.*, gel-mez-siniz).

To form affirmative or negative question forms of tenses, the interrogative particle "mi" is inserted between the tense suffix and the personal ending . In the third person plural forms of all tenses, and in all persons of the di-past, it comes at the end (*e.g.*, geliyorlar mı, gitmedin mi). In the first person of the negative-question of the simple present, "z" comes after the negative stem (*e.g.*, gelmez misin).

2 .7 VERB STRUCTURE

i) Transitive, Intransitive Verbs

If the action passes over to a direct object, or the verb can take direct object, that verb is called a transitive verb.

e.g. gör_ /see; tut_ /hold anything or anyone

If the action does not pass over to direct object, that verb is an intransitive one.

e.g. gel_ /come; ağla_ /cry

ii) Active and Passive

If the stems:

end with a consonant (except "l"), the suffix "-il" is added,

end in "l", the suffix "-in" is added,

end with vowel, "-n" is added, to obtain passive voice.

e.g. söylemek >> söyle-n-mek

iii) Reflexive Verbs

If the stem ends with a consonant, the suffix "-in" is attached; if ending of the stem is a vowel, "-n" is added, to obtain reflexive forms.

Although the passive and reflexive forms of some verbs have the same structure, they differ in their meanings.

e.g. Reflexive : Çocuk yıkandı. /The children washed by himself.

Passive: Gömlek yıkandı. /The shirt was washed.

iv) Reciprocal Verbs

If the action is done by more than one subject either in cooperation or in opposition or mutually, a reciprocal verb is used.

e.g. döv_ >> döv_üş

v) Causative, Factitive Verbs

If an intransitive verb is made transitive, it is called causative.

e.g. öl_ >> öl-dür-mek

If a causative or originally transitive verb is made causative, this verb is called a factitive verb.

e.g. öl_ >> öl-dür-t-mek

2 .8 BASIC TENSES

Basic Tenses can be classified into two groups: Five informative tenses and four optative tenses [3], [5-6], [15].

2 .8 .1 Informative Tenses

i) Simple Present

It is used to express a habitual, permanent or repeated action.

Suffix: (i/ı/u/ü/a/e)r

e.g. Dünya güneşin çevresinde döner.

/The earth rotates around the sun./

ii) The Witnessed Simple Past

İt is used to state an action completed at a particular time or during some period in the past.

Suffix: -di (-dı/-du/-dü/-tı/-ti/-tu/-tü)

e.g. Dün gece ütü yaptım. /I ironed yesterday night.

iii) The Reported Simple Past

It is used when the speaker has no direct, first hand information, or if he was just informed, or has just become aware of it.

Suffix: -miş (-mış/-muş/-müş)

e.g. O hastaymış. /(Someone reported that) he was ill.

iv) The Progressive Present

It indicates an action which began in the past and it is still continuing at the moment of speaking.

Suffix is -yor

e.g. O geliyor. /He is coming.

v) The Future Tense

It is used to express an action which will take in the future.

Suffix: -acak (-ecek)

e.g. Yarın gelecek. /He will come tomorrow.

2 .8 .2 Optative Tenses:

i) Imperative

It is the form of the verb used in orders, and the second singular personal ending of imperative is the verb-stem.

Suffix: ...

e.g. Aç. /Open.

ii) Optative or Subjunctive

This form is used to express wishes or concepts whose realization is desired, and it does not express actual situations or happenings.

Suffix: -a (-e)

e.g. Geleyim. /Let me come.

iii) Necessitative

This form expresses possibility, a strong likelihood or logical conclusion.

Suffix: -malı (-meli)

e.g. Bu adam doktor olmalı. /This man must be a doctor.

iv) Remote Condition

It is sometimes used to express a misty conditional, and sometimes to indicate a will related with a condition. It may also express wishes, often beginning with "keşke" /if only/.

Suffix: -se (-sa)

e.g. Hava güzel olsa da pikniğe gitsek.

/I wish the weather be nice and we can go to the picnic./

2 .9 COMPOUND TENSES

They are collected into three groups :

i) Compound Imperfect

It is formed by adding the past suffix of "-di" to the tense suffix.

e.g. Onu gördüğüm zaman basketbol seyrediyordu.

/When I saw him he was watching the basketball game./

ii) Narrative Compound Tenses

It implies lack of a first hand or definite knowledge, the suffix "-miş" is added.

e.g. O kitap okuyormuş.

/He has been reading a book./

iii) Conditional

It expresses a conditional state, and its suffix is -se (-sa).

e.g. Önemli birşey yapıyorsanız rahatsız etmeyelim.

/If you are doing something important, let us not disturb you./

2 .10 COMPOUND VERBS

Compound verbs are divided into five groups.

i) Potentiality and Possibility

It express a physical or mental ability for an action.

e.g. yazabilmek /to be able to write/

ii) Acceleration, Quickness

It indicates acceleration or quickness in an action.

e.g. geliverdi /He(she,it) happened to come./

iii) Continuence

It shows continuence in an action.

e.g. Siz yazadurun ben hemen gelirim.
/Continue with writing, I'll come back soon./

iv) Approximation

It indicates approximation to a state or situation.

(almost,nearly...)

e.g. düş-e-yaz-dı-m. /I almost fell.

v) "etmek, olmak" is used to derive verbs from denominal words.

e.g. kabul etmek /to accept
muhtaç olmak /to be in need

2 .11 PERSONAL ENDINGS

Two basic kinds of personal endings are used in conjugation of the verbs:

	- m	- (y)im	(ben/I)
in di-past	- n	- sin	(sen/thou)
and	- ...	- ...	(o/he,she,it)
conditional	- k	- (y)iz	(biz/we)
- se	- niz	- siniz	(siz/you)
	- ler	- ler	(onlar/they)

In the next chapter, structure of the knowledge banks, data structure used and the rules for the suffixes are discussed.

CHAPTER 3

DEVERBAL WORD RECOGNITION BY USING KNOWLEDGE BANKS

Given words are analysed by using the knowledge about verbal roots, their suffixes, meanings, and the related rules. For this reason, a program has been developed in order to find the meaning of a word. It uses three data files as knowledge banks and they are called *Root Bank, Suffix Bank, and Meaning Bank.* Since these banks are dynamically constructed, it is possible to delete and/or to insert knowledge at any time. In other words, the program has the ability of forgetting and learning.

The figure 3.1 shows the block diagram for the analysis of a given word. After finding the root using the Root Bank, the Suffix Bank is used for determination of the suffixes in the given word. Next, the meaning changes due to the suffixes are extracted from the Meaning

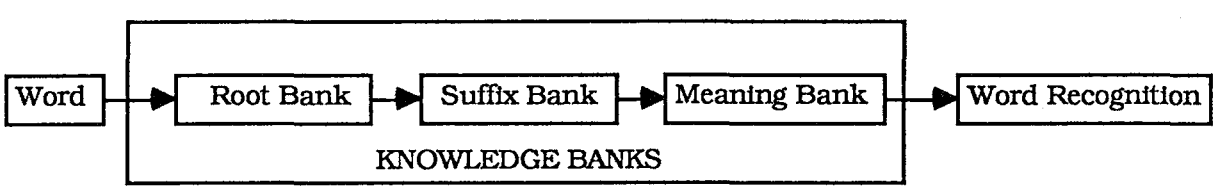


Figure 3.1 Block diagram for analysis of a word

Bank and the root information is combined with this knowledge. Thus, the meaning of the word is obtained at the end of this analysis. However, some problems may appear during the explained process. The structure of the root, suffix, and meaning banks, search method used in the program, learning aspect, deformation positions for a root, general and specific rules that concern suffixes are explained in the following sections.

3 .1 ROOT BANK

In order to analyse a given word, the source of its meaning must be found. In other words, the root of the word has to be determined. For this reason, it is necessary to collect knowledge about the roots. When a suffix, which has more than one meaning is affixed to a root the suitable meaning of the suffix must be found. Moreover, any deformation position for the root and any possible changes on the root must be known. Therefore, a storage bank called *Root Bank* is formed and it contains both the verbal roots and their identity codes.

3 .1 .1 Structure of the Root Bank

There are two ways of storing the verbal root: One of them uses the letter order and the other one uses the number of letters in it. Since it is much easier to take and to arrange the roots according to the letter order, the Root Bank is constructed by indexing letters.

Example:

Alphabetical Order	Number of Letters
al_ / take	al_ / take
avla_ / hunt	bak_ / look
bak_ / look	gel_ / come
gel_ / come	sor_ / ask
sor_ / ask	avla_ / hunt

The identity code of each root in the Root Bank is constructed by using the necessary information about the root. The information required for correct analysis of a word has to cope with the following problems:

i) The root of a given word is not always the simplest root (i.e., the simplest form of a word).

Example: word: yemek / meal
root : ye_

word: yeter / sufficient
root : yet_

As it is seen in this example, the meaning of the first word comes from the root "ye_". However, the second one takes its meaning from "yet_". Since our aim is to find the correct meaning of a given word, finding the correct root is very crucial. In order to express this property, a binary digit (0/1) is used in the first field of the root code. The binary digit "0" means that the root is the simplest root. However, "1" shows any possible change for that root, that is, the root can be replaced by another root which is part of the given word. Therefore, the root "ye_" has "1" in the first field of its code.

ii) The type of the suffix affixed to a root, and its meaning may depend on the root structure. That is, the transitive or intransitive structure of the root may affect the analysis of the given word. For this reason, the second field expresses the root structure. Three numbers are used for this characteristics. "1" indicates a transitive root and "0" shows an intransitive root. However, the structure of some roots may be both transitive and intransitive. The number "2" is used to show the roots having this structure.

iii) If the suffix has more than one meaning and these meanings have the same properties, there is a problem. The problem for that position is how to distinguish these meanings, and which meaning of this suffix is suitable for that root. This problem can be solved by using some identity numbers. These numbers are called *Return Numbers* and they will be explained in section 3.2.1. Thus, other fields of the root code is reserved for return numbers to express the specific meaning of a suffix. However, these numbers are not always necessary. Although one root has the return number of a suffix, the other one may not have that number or may have another return number. This is due to the fact that all the suffixes cannot be affixed to a root. Moreover, this part may be empty for some roots. The following example explains the meaning of the return numbers for a suffix.

Example: Since the suffix "-ak" can cause three different meaning changes on a root, it has three return numbers: 006,007, and 008.

First Meaning: It shows the character of a person,

e.g. word: kork-ak / coward
root: kork_ / be afraid
return number of the first meaning "-ak": 006

Second Meaning: It denotes the place of the action,

e.g. word: yat-ak / bed
root: yat_ / lie down
return number of the second meaning "-ak": 007

Third Meaning: It indicates the tool by which the action is done,

e.g. word: kay-ak / ski
root: kay_ / slide

return number of the third meaning "-ak": 008

Although the return numbers (006, 007, 008) are necessary for the roots "kork_", "yat_", and "kay_", they are not needed for the roots "ye_" and "yet_".

The part of the identity code for the explained examples is shown in Table 3.1. Some commonly used roots and their identity codes showing the necessary knowledge about the verbal roots are given in Appendix B.

Table 3.1 Form change, root structure, and return numbers for some roots

Root	Form Change	Root Structure	Return Number
kay_	0	0	008
kork_	0	0	006
yat_	0	0	007
ye_	1	1	-
yet_	0	0	-

3.1 .2 Deformation Positions for a Verbal Root

The spelling of some roots change when a suffix is affixed to it. That is, the root may be deformed when some suffixes are affixed to it. In order to find the root correctly, we must know what causes this effect. There are some conditions for which causes a deformation occur.

If a given root satisfies the condition part of one of the following rules, the last letter of the root changes.

i) If a root starts with one of the unrounded vowels [a,e,ı,i], ends with one of the wide, unrounded vowels [a,e] and the suffix of the progressive present (-yor) is affixed to the root, then wide vowel becomes unrounded, closed vowel [ı,i].

Example:

Root : başla_ / start

Suffix : -yor

Verb : başlıyor

ii) If a root starts with one of the rounded vowels [o,ö,u,ü], ends with one of the unrounded vowels [a,e,ı,i] and the suffix of the progressive present (-yor) is attached to it, then unrounded vowel changes to rounded, closed vowel [u,ü].

Example:

Root : boya_ / paint

Suffix : -yor

Verb : boyuyor

iii) If a root ends with one of the wide vowels [a,e,o,ö], and the buffer consonant "y_" is added between the root and the suffix, then wide vowel becomes closed, unrounded vowel [ı,i].

Example:

Root : ye_ / eat

Suffix : -ecek

Verb : yiyecek

iv) Unvoiced consonants do not usually change. However, there are some exceptions to this. For example, when a suffix is affixed to "git_" or "güt_", the last letter of the root may or may not change from "t" to "d" as shown below.

Root : git_	Root : git_
Suffix : -er	Suffix : -ti
Verb : gider	Verb : gitti

As it is seen in these examples, change at the last letter of the root "git_" depends on the suffix affixed.

3 .2 SUFFIX BANK

After the determination of the root of a given word, the search operation for finding the suffixes affixed to the root starts. For this search, we have to know how a suffix changes meaning of a root, and which type of suffix causes this effect. Since suffixes may have more than one function, some rules are also necessary to identify these

Table 3.2 Examples for suffix knowledge

Suffix	Suffix Type 1	Meaning 1	Return Num.	Suffix Type 2	Meaning 2	Return Num.
-acak -ecek	derivation suffix (derives noun from verbs)	makes noun for tools aç-acak/opener	003	Tense suffix (future tense)	about future ol-acak/It will be	084
-n	derivation suffix (derives verb from verbs)	makes reciprocal verbs tüt-ü-n/to smoke	065	derivation suffix (derives noun from verbs)	makes noun for objects tüt-ü-n/tobacco	066
-ıcı -ici -ucu -ücü	derivation suffix (derives noun from verbs)	makes nouns for property and carrier uç-ucu/fluid koş-ucu/runner	039	-	-	-

functions. Therefore, the necessary knowledge about the suffixes are collected and this knowledge is shown with some examples in Table 3.2.

In order to store this knowledge, a knowledge bank called *Suffix Bank* is formed. This bank consists of all the suffixes affixed to verbs and their identity codes. This bank can be found in Appendix C.

3 .2 .1 Structure of the Suffix Bank

The suffixes are stored by indexing them according to their first letter. For example, -a, -acak, -aç, -ağan,-ak, ..., -baç, ..., -gan, ...

In order to separate the meaning of a suffix and its type, some identity numbers are produced. These numbers are unique and they are called *Return Numbers*. Therefore, for each meaning of a suffix there exist a Return Number (RTN) and three digits are used for this representation. That is, these numbers can change from "001" to "999." After finding the RTN of a suffix, it is easy to find how the meaning of a root is changed by affixing the suffix.

The identity code of a suffix is constructed as follows:

Fields	1	2	3	4	5	6	7	8	9	10	11	12	13	...
Identity code														

Field 1 is used to represent how many different meanings exist for a suffix. Fields 2,3,4, and 5 correspond to four different types of suffixes. If one of the fields has "0", it means that this suffix cannot be in that type. If the content is "1", this suffix has only one meaning in that type. However, if the number is greater than "1", this

indicates the number of meanings for that type of the suffix. Suffix types on the fields are as follows:

Field 2: Suffixes deriving verbs from verbs (SDVV)

Field 3: Suffixes deriving nouns from verbs (SDNV)

Field 4: Tense suffixes (TS)

Field 5: Personal endings (PE)

Depending on the suffix type, a RTN is obtained from the four successive fields. Field 6 has a digit between 1 and 4, and these numbers correspond to four different types of suffixes as shown below.

1: SDVV 2: SDNV 3: TS 4: PE

When one of these numbers is placed in this field, fields 7,8, and 9 are used to find the return number of that suffix type.

If a suffix has one type and one meaning, these fields will be enough. However, if a suffix has more than one type, the operation explained for the fields from 6 to 9 will be repeated until all the types of that suffix is covered That is, the pointer jumps to each type field and then the RTN will be taken from the successive three fields.

Example:

Suffix: -acak/-ecek

Identity code of this suffix: 3-0210-2-003-2-004-3-084

This suffix has two types. One is SDNV type and the other one is the future tense which is a TS. The former has also two different meanings. The first meaning is an object which is used at the action

(*e.g.*, yak-acak), the second one is an object which makes that action (*e.g.*, koş-acak adam). RTN of these meanings are "003" and "004," respectively. Besides, the future tense is represented by "084".

3 .2 .2 Rules for the Suffixes

Rules can be used to decide which suffix is suitable to join or which meaning of the suffix is appropriate to that place. For this reason, the rules about the suffixes can be divided into two groups: General Rules and Specific Rules.

The *General Rules* determine the types of suffixes which cannot be affixed to some specific suffixes within new formed word. Therefore, these rules are also called negative rules The impossible positions for the suffixes are as follows:

i) SDNV + SDVV
SDNV + SDNV
SDNV + TS
SDNV + PE

If the new formed word becomes a noun, none of the suffixes can be attached to the word. That is because, we only deal with the suffixes which derive words from deverbal words.

ii) TS + SDVV
TS + SDNV

Only personal endings can follow a tense suffix. However, there is a suffix "-ken" which doesn't obey this rule. If the word "geliyorken" is taken as an example, it is seen that "-yor" is a progressive present suffix and the suffix "-ken" is attached to it.

iii) PE + SDVV
PE + SDNV

PE + TS
PE + PE

No suffix can be affixed after the personal endings. However, if the given word has a compound tense, the third plural personal ending "-lar" may come between two tense suffixes. For example, "geliyorlardı" is used more frequently than "geliyordular".

Since a suffix may have more than one type, a rule for this suffix must be established to find its correct type. These rules are called *Specific Rules* and they are explained as follows:

1) The suffix "-a (-e)" has two types: SDNV and TS. If "-a" is the first suffix of a given word and if there is not any suffix following it, the type of this suffix is chosen as SDNV type. At the same time, it indicates an optative. That is, "-a" is a TS. For example, the word "yara" has two meanings. The first meaning is "wound" and it is a noun. The second one is "Let he (she/it) to split." However, if there is at least one suffix after "-a", this suffix shows only a TS (*e.g.*, koşalım).

2) The suffix "-acak (-ecek)" has two types: SDNV and TS. This suffix is the same as the suffix "-a". For example, the word "açacak" has two meanings. The former is a tool used to open something. The latter is a verb and it means that "He (she/it) will open." However, the suffix "-acak" in the word "açacaktım" is definitely a TS.

3) The suffix "-ar (er/-ır/-ir/)" has two types: SDVV and SDNV. If the identity code of a root has the RTN of SDNV type, this suffix changes the word to a noun (*e.g.*, gelir). Otherwise, it forms a verb and it is an SDVV (*e.g.*, batır).

4) The suffix "-im (-ım/-um/-üm)" has two types: SDNV and PE. If the previous suffix is a tense suffix, it is a personal ending. If it is not a tense suffix, "-im" cannot be a personal ending. That is, it is an SDNV type suffix.

5) Although the suffix "-iş (-ış/-uş/-üş)" has SDVV and SDNV types, these types cannot be distinguished. For example, the word "bak-ış" is both a noun and a verb. However, if the number of the suffix is greater than one, it is definitely a verb (*e.g.*, bak-ış-tı-lar) since any suffix cannot come after a noun.

6) The suffix "-k" has two types: SDNV and PE. If there is only one suffix and it is "-k", the type of this suffix is SDNV (*e.g.*, ada-k). If the suffix preceding this suffix is a buffer vowel, then the word becomes a noun (*e.g.*, aç-ı-k). However, if the preceding suffix is a TS, this suffix shows the first plural PE (*e.g.*, gel-di-k).

7) The suffix "-ma (-me)" has two types: SDVV and SDNV. If this suffix is the only suffix of a given word, then it makes a verb and a noun from the verbal root (*e.g.*, kaz-ma). Otherwise, it is an SDVV.

8) The suffix "-mez (-maz)" has two types: SDNV and TS. If there is only one suffix and this suffix is "-mez", it is an SDNV (*e.g.*, tüken-mez). Moreover, it is also a negative of the simple present tense for all suitable conditions (*e.g.*, oku-maz-dı-m).

9) The suffix "-miş (-mış/-muş/-müş)" has two types: SDNV and TS. If there is only one suffix in a given word and if it is "-miş", this suffix both makes a noun and shows a tense. Otherwise, it is a TS (*e.g.*, gel-miş-sin).

10) The suffix "-n" has three types: SDVV, SDNV and PE. If this suffix is the first suffix and there is not any suffix following it, it is an SDVV (*e.g.*, söyle-n). If there is at least one suffix following this suffix, it is again an SDVV (sev-i-n-ç). If the previous suffix is a TS, this suffix is a second PE (*e.g.*, gel-di-n). Otherwise, it changes a word to a noun (*e.g.*, yığ-ı-n).

11) The suffix "-t" has two types: SDVV and SDNV. If the last letter of the root is a vowel, "-l" or "-r," the type of "-t" is SDVV (*e.g.*, kuru-t, çoğal-t, uçur-t). Otherwise, its type is SDNV (*e.g.*, geç-i-t, yak-ı-t, um-u-t).

12) The suffix "-ti (-tı/-tu/-tü)" has two types : SDNV and TS. When the given word has the suffix "-ti" and the last letter of the preceding suffix of "-ti" is "-l", "-n" or "-r," then "-ti" is an SDNV type (*e.g.*, gezinti). Otherwise, it is a TS (*e.g.*, geçti).

These rules are used for deciding which type of the suffix is suitable to that place. However, a suffix may be a part of the other suffix. This position can cause a confusion and wrong analysis of a word. Therefore, there exists some specific rules to prevent this situation. For example;

If the the given word is "bilgi", its root is found as "bil_" and the suffix "-gi" is affixed to it. However, if the given word is "bilgin" and the rule about the suffix "-gi" is not used for this word, the correct solution cannot be reached and the suffixes "gi-" and "n-" are found. Therefore, although the real solution is "bil-gin", the word is analysed as "bil-gi-n". If the rule about the suffix "-gi" is activated, the suffix of word is extracted correctly. The rule for the suffix "gi-" is as

follows: If the suffix "-n" is attached to the suffix "gi-", they must be concatenated and the suffix "gin-" must be obtained.

As it is seen in this example, if a suffix is covered by another suffix and if these suffixes cannot be distinguished, a specific rule has to be established. Moreover, if more than one type of suffix is suitable to the given word, we have to deal with all possible types of the suffix.

There are also some rules for the tense suffixes. If the given word is an imperative, the compound tense suffixes cannot be affixed to the word. Moreover, the narrative compound tense cannot be used after the simple past tense. The conditional compound tense suffix cannot be attached to the remote-condition, optative or necessitative suffixes. Although, there is another conjugation which is called double conditional compound tense, it is only for compound imperfect and narrative tenses. For example, "geliyorduysam" and "geliyormuşsam".

In addition, the suffixes for questions "-mi, -mı, -mu, -mü" come before the personal endings and after the simple present, the progressive present, the reported simple past, the future tense or the necessitative. For example, bilirmiyim, biliyormusun, bilmişmiyiz, bilecekmisiniz. However, this suffix is written separately for the third plural suffix and we do not deal with this situation.

3 .3 MEANING BANK

When all the suffixes of a given word are determined, the return numbers of the suffixes are obtained. The identity codes of the root and the suffixes are used for this task. In order to find the meaning

of the word, we have to concatenate each meaning of these return numbers. For this reason, another bank is formed and it is called *Meaning Bank.*. This bank has all the return numbers of suffixes and their formed meanings. This bank is given in Appendix D.1 in detail. Two examples are given below to show what is meant by the meaning of a suffix.

Example 1: Suffix: -ga (-ge)
RTN: 115
Description: The tool that is used for the action/function mentioned by the root.

Assume that the given word is "süpürge /sweeper". Therefore, its meaning is *"the tool that is used for sweeping (süpürme eyleminde kullanılan alet)."*

Example 2: Suffix: -man (-men)
RTN: 059
Description: The agent who performs the action/function mentioned by the root.

Assume that the given word is "öğretmen /teacher" and its meaning is *"the agent who teaches (öğretme işi yapan kişi)."*

As it is seen in these examples, if we substitute the root of a word with "kök," the meaning is obtained correctly. Moreover, if the given word has more than one suffix, the suffix meanings must be attached. All the meanings are explained in Appendix D.2 in detail.

3 .4 DATA STRUCTURE USED

In order to implement the method, a software for IBM-AT compatible computers was developed. We have used Pascal

programming language and linked representation is used for data movement.

If the roots and the suffixes are stored in an array, the insertion and the deletion of a suffix will be time consuming. However, if linked representation is used, it is possible to store the items anywhere in the memory. That is, dynamic storage is used. To access elements of the list in the correct order, the address or the location of the next element for each element of the list is also saved. Therefore, each data item in a linked representation is a pointer to the next item. However, to find each node which precedes another node, search has to be started from the beginning. The same problem arises when one wishes to insert or delete an arbitrary node. It is clear that singly linked list is one way search and we need a better one called doubly linked list.

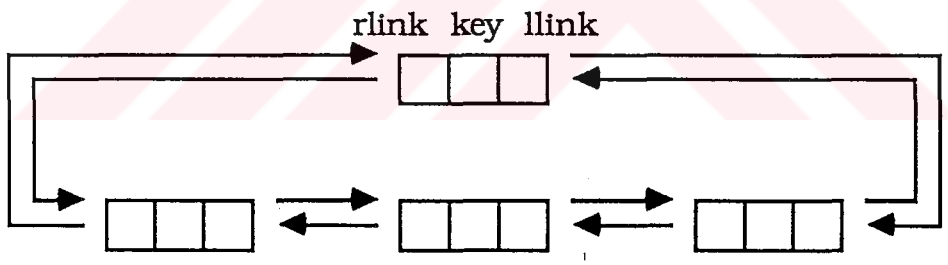


Figure 3.2 Doubly linked circular list

A node has at least three fields in the doubly linked list; data or key, left link (llink) and right link (rlink). The right link is in the forward direction and left link is in the backward direction. Moreover, a doubly linked list may or may not be circular. If it is circular, its last node points back to the first and the complexity of the insertion or the deletion is O(1). That is, these operations take constant time [14]. As it is seen *Doubly Linked Circular List* is more useful than

the other. For this reason, it is used in our program. Figure 3.2 shows a doubly linked circular list.

3 .5 HOW TO SET TURKISH LETTERS AND THEIR ORDERING

The type of the root and the suffix variables used in the program is string type. Any two string values can be compared, because all the string values are compatible. The relational operators "=", "<>", ">", "<", ">=" and "<=" compare strings according to the ordering of the extended ASCII character set.

However, we have to deal with Turkish language and we need Turkish characters (i.e. ı/I, i/İ, ç/Ç, ğ/Ğ, ö/Ö, ş/Ş, ü/Ü). If the original ASCII character set is used, the comparison will be done wrongly and the correct solution cannot be reached. For this reason, another ordering is used for the twenty nine letters by using the letter order of the Turkish language.

After taking the roots and the suffixes from the banks, they are converted into new character set. Moreover, the given word is changed when it is taken from the user. This conversion is called *input conversion* and it is done by the function called InCov. When anything is written on the screen, the backward operation must be done to obtain correct writing. This process is called *output conversion* and the function called OutCov is developed for this purpose. Therefore, the forward and the backward form of the string return in the function identifier. The tables 3.3 and 3.4 show the forward and the backward conversion of the twenty nine letters.

Consider the word "koştu" as an example. After the word is taken from the user, the function InCov is activated and the word becomes

"nrwxy". If the new formed word is sent to the function OutCov, it changes to "koştu".

Table 3.3 Forward Conversion

In	Out
A, a	a
B, b	b
C, c	c
Ç, ç	d
D, d	e
E, e	f
F, f	g
G, g	h
Ğ, ğ	i
H, h	j
I, ı	k
İ, i	l
J, j	m
K, k	n
L, l	o
M, m	p
N, n	q
O, o	r
Ö, ö	s
P, p	t
R, r	u
S, s	v
Ş, ş	w
T, t	x
U, u	y
Ü, ü	z
V, v	{
Y, y	|
Z, z	}

Table 3.4 Backward Conversion

In	Out
a	A, a
b	B, b
c	C, c
d	Ç, ç
e	D, d
f	E, e
g	F, f
h	G, g
i	Ğ, ğ
j	H, h
k	I, ı
l	İ, i
m	J, j
n	K, k
o	L, l
p	M, m
q	N, n
r	O, o
s	Ö, ö
t	P, p
u	R, r
v	S, s
w	Ş, ş
x	T, t
y	U, u
z	Ü, ü
{	V, v
|	Y, y
}	Z, z

The method for identifying root and the suffixes of a given word, and the general structure of the program will be introduced in Chapter 4.

CHAPTER 4

SOFTWARE

4 .1 INTRODUCTION

The program initially loads all the roots and their identity codes from the Root Bank to the doubly linked circular list. If the root of a given word is removed by the method explained in Section 4.1, the complementary part of the word, which is the suffix part, is obtained. If the root is not found and the correctness of the given word is confirmed by the user, the information about the root has to be added to the Root Bank. Then, analysis of the word continues. However, if the given word is not correct, the program goes to start and waits for a new word from the user.

After determining the root, all the suffixes and their identity codes are loaded from the Suffix Bank to another doubly linked circular list. Then, the search operation for the suffixes is started. If it terminates successfully, it is assumed that our goal is achieved. Otherwise, the given word seems to be unsuitable for our program. That is, the suffixes of the given word have to be one of the following types: SDVV, SDNV, TS or PE.

The flowchart for analysis of a word is shown in Figure 4.1.

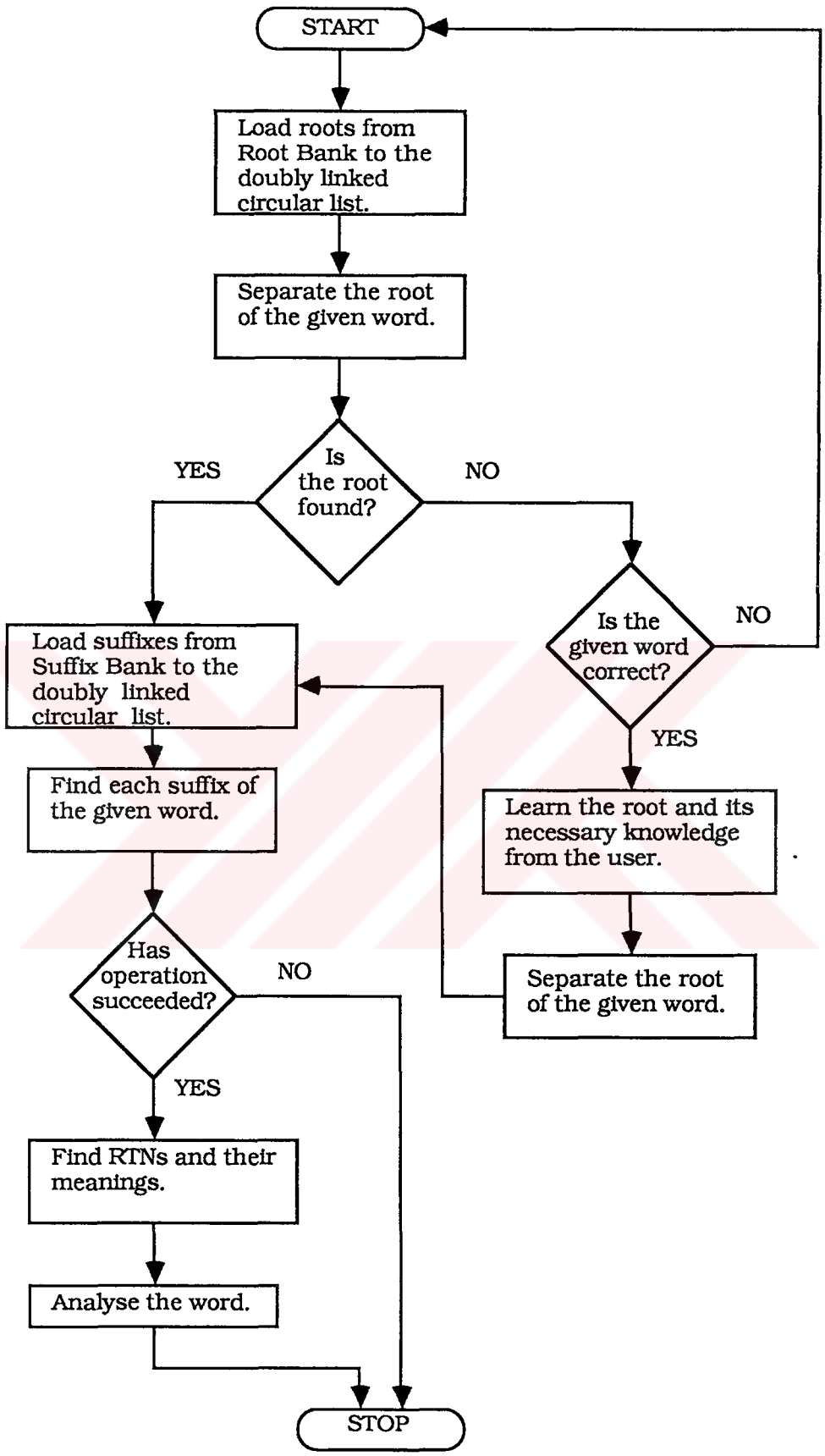


Figure 4.1 Flowchart for analysis of a word

The main program activates four modules called *Determine_Root, Extract_Suffixes, Dispose_Root_Links, Dispose_Suffix_Links.*

Determine_Root procedure :

Function: Determines the root of a deverbal word.

Declaration: Determine_Root

Called By: Main program

Calls: SeparateRoot, NewRoot

Extract_Suffixes procedure :

Function: Extracts the suffixes of a deverbal word.

Declaration: Extract_Suffixes

Called By: Main program

Calls: SeparateSuffix

Dispose_Root_Links procedure :

Function: Disposes the doubly circular linked list constructed for the roots.

Declaration: Dispose_Root_Links

Called By: Main program

Calls: None

Dispose_Suffix_Links procedure :

Function: Disposes the doubly circular linked list constructed for the suffixes.

Declaration: Dispose_Suffix_Links

Called By: Main program

Calls: None

Figure 4.2 shows the software structure of the main program.

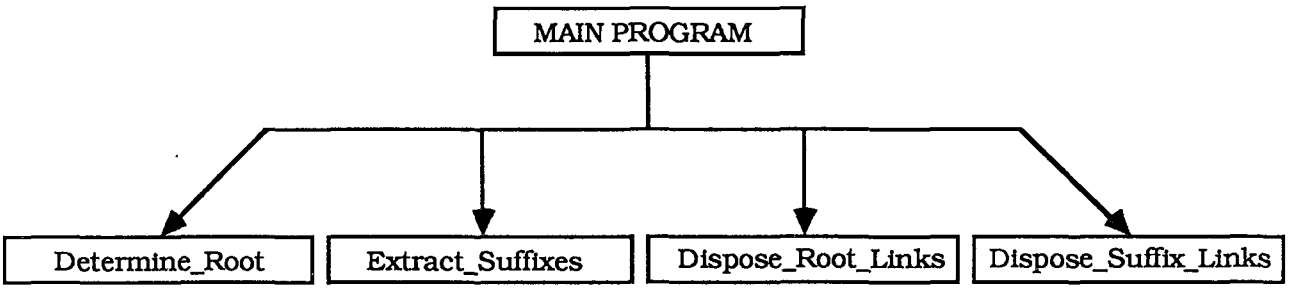


Figure 4.2 Software structure of the main program

4 .2 DETERMINING THE ROOT OF A GIVEN WORD

The search operation to find the root of a given word starts from the left and continues to the right of the word.

The leftmost letter, that is, the first letter of the word is taken and it is assigned to the variable called *temporary root.* When the root search starts, the pointer goes to the head node, that is, the starting of the list and the letters of the root in the node is compared with the letters of the temporary root. If the string value of the temporary root is greater than that of the root in the node, the pointer jumps to the next node and this comparison is repeated until the string value of the temporary root is smaller than that of the root. Then, the following letter of the word is concatenated to the temporary root. This procedure is repeated until either one of the root in the Root Bank is equal to the temporary root or all the letters in the given word is concatenated to the temporary root. Moreover, if the number of letters in the temporary root is greater than or equal to the maximum possible number of letters in a root, the search stops.

There is also a flag (true/false) to show whether the root is found or not found. When the search stops, the flag is checked. If it is equal

to true, the first field of the identity code is taken to see whether the founded root is a simplest root or not. If it has "1" (i.e., the simplest root) in the first field of the code, necessary change for that root is done. After that, the temporary root is equalized to the variable called *root*. However, if it is false at the end of this search, the rules about the roots are activated. If any one of the rules adapts the given word, the necessary changes are done and then the search operation starts again. If the root still cannot be determined, it means either the root does not exist in the bank or there is a spelling error. If the given word is confirmed by the user, the root of the given word is asked to the user and the necessary knowledge about this root is learned. Therefore, the identity code of the root is automatically produced and the root is obtained. Once the root is determined, the remaining part of the given word is extracted. That is, the root is deleted from the word. Now the program is ready to the find the suffixes in the remaining part.

The steps to find the root of a given word are shown in the following example.

Example:

Given word : buldu / He/She/It found

Steps :

1) First letter of the given word: "b"
2) Remaining letters of the given word: "uldu"
3) Search "b" in the Root Bank
4) "b" is not a root
5) Concatenate the first letter of the remaining part to "b": "bu"
6) Remaining letters of the given word : "ldu"

7) Search "bu" in the Root Bank

8) "bu" is not a root

9) Concatenate the first letter of the remaining part to "bu": "bul"

10) Remaining letters of the given word : "du"

11) Search "bul" in the Root Bank

12) "bul" is a root

Root is "**bul_** " and remaining part which has the suffixes is "**-du**"

In this example the root is found at the end of the third trial by using the doubly linked circular list. Therefore, it is understood that searching a root requires breaking up the root into its constituent characters. The following variables are used in the flowchart of the root extraction shown in Figure 4.3.

i : It is used to show the number of the letters in the temporary root which is the searched part of the given word.

rsize : It is the number of the letters in a given word.

rnews : It is the temporary root constructed by using i letter of the given word.

maxrsize : It is the maximum possible number of letters in a root.

rootkod : It is the identity code of the root found.

köktypdf : It shows that the root either changes its form (1) or does not (0).

fçatısı : It indicates the root structure (transitive, intransitive or both).

rootpointer : It points the node whose key field is equal to the root found.

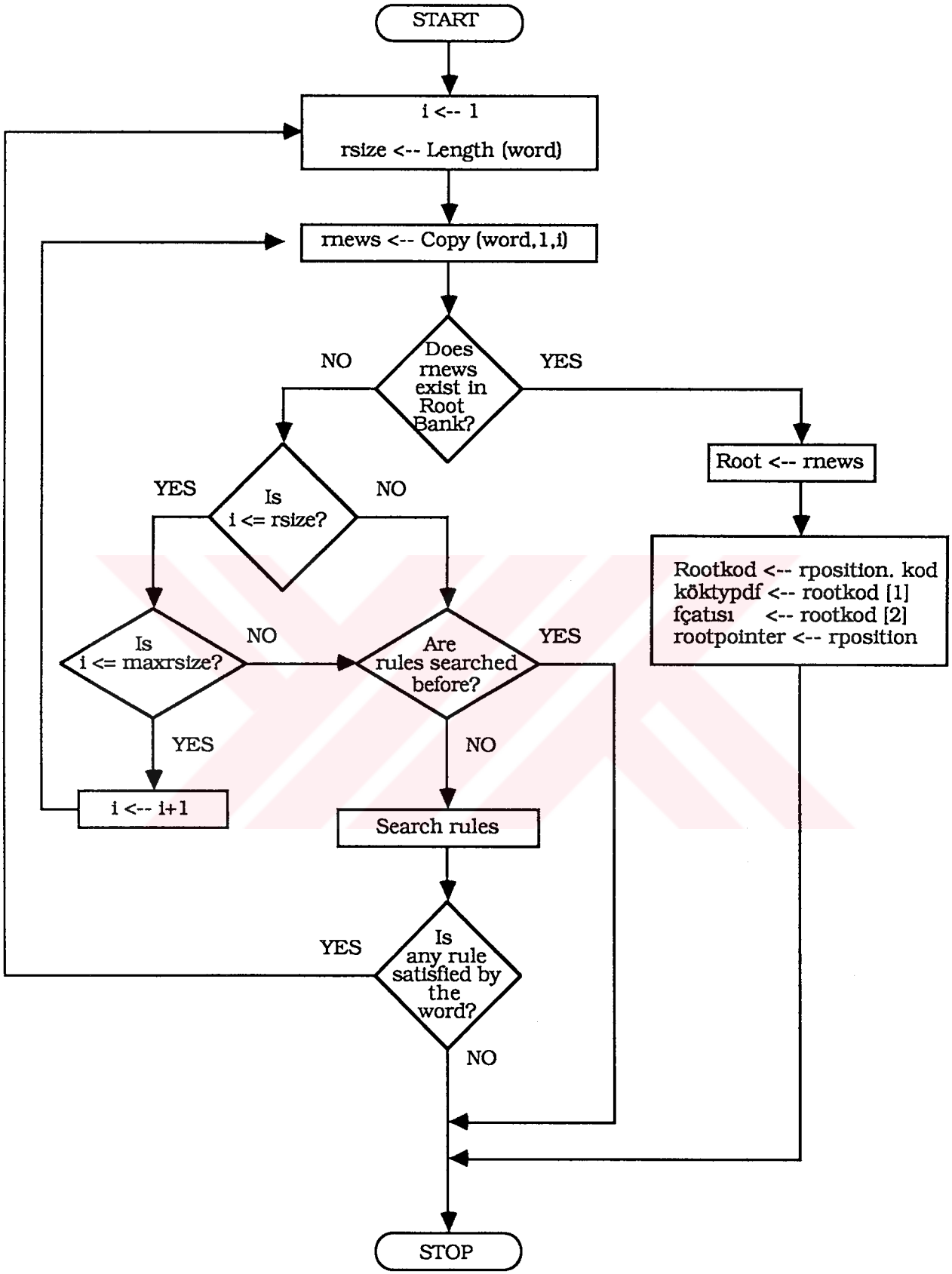


Figure 4.3 Flowchart for determining the root of a given word

The module *Determine_Root* uses two procedures called *SeparateRoot* and *NewRoot.*

SeparateRoot procedure :

Function: Determines the root of the given word, if it exists.

Declaration: SeparateRoot

Called By: Determine_Root

Calls: ReadDataFile, GetWord, SeekRoot, RuleSearch, StrRootKnw

NewRoot procedure :

Function: Learn a new root.

Declaration: NewRoot

Called By: Determine_Root

This module and its sub-modules are explained in Section 4.4.

ReadDataFile procedure :

Function: Reads the roots from the Root Bank and loads them in the doubly circular linked list.

Declaration: ReadDataFile

Called By: SeparateRoot

Calls: InsertDataFile, InCov

InsertDataFile procedure :

Function: Creates a node, places the root in the node, and sets its links.

Declaration: InsertDataFile (key:strroot; inum:strrkod; var cposition:rpointer)

Called By: ReadDataFile

Calls: None

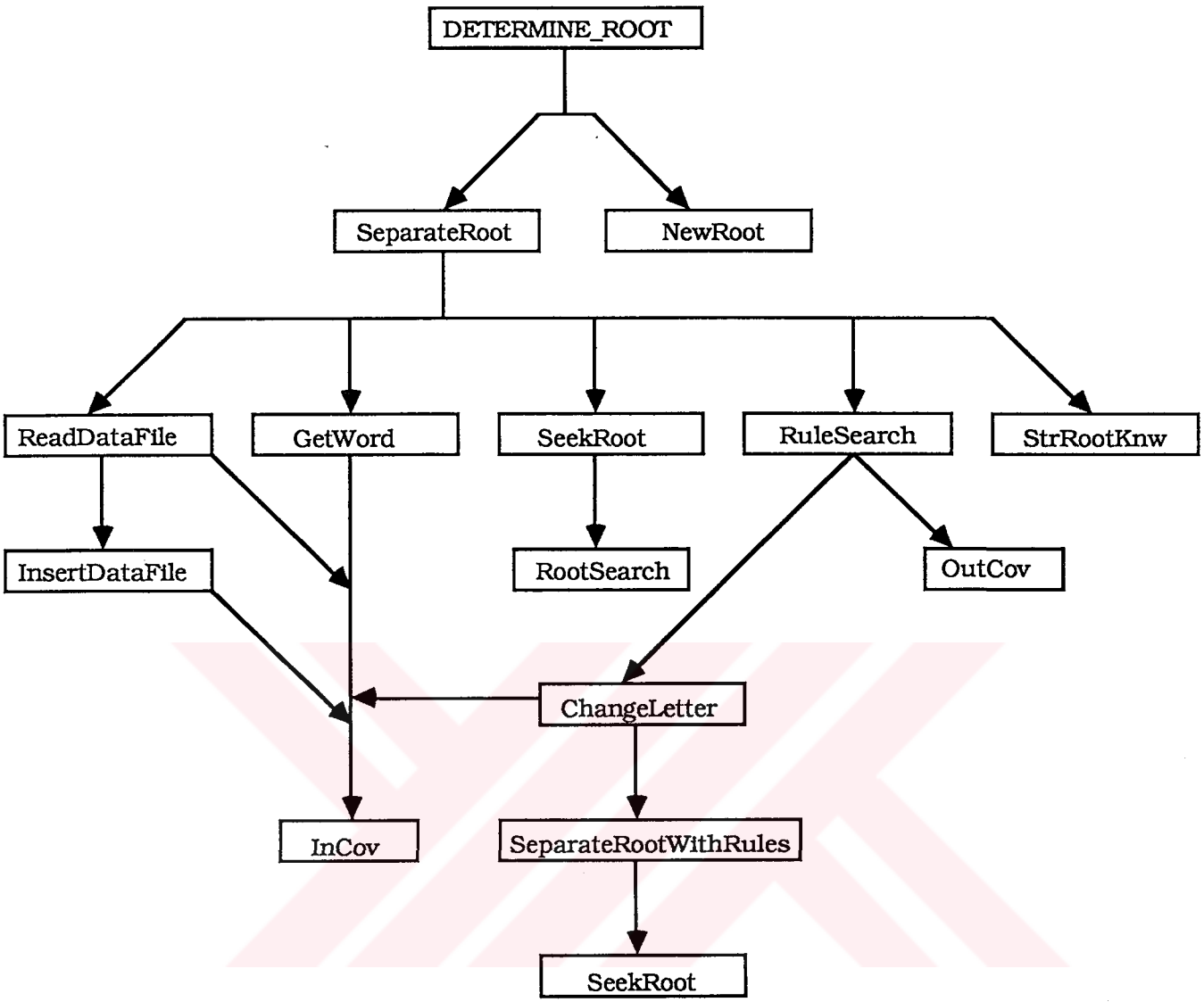


Figure 4.4 Software Structure of determining the root of a given word

GetWord function :

Function: Takes the word from the user and finds the length of the word.

Declaration: GetWord (var rsize: integer)

Called By: SeparateRoot

Calls: InCov

SeekRoot procedure :

Function: Assigns the flag according to the return value of the function called RootSearch which searches the value of the variable newstr.

Declaration: SeekRoot (newstr:strroot; var flag:boolean)

Called By: SeparateRoot, SeparateRootWithRules

Calls: RootSearch

RootSearch function :

Function: Searches the key value in the doubly circular linked list by starting from the node pointed by cposition.

Declaration: RootSearch (key:strroot; var cposition:rpointer)

Called By: SeekRoot

Calls: None

RuleSearch procedure :

Function: Searches the rules, activates the rule if its condition part is satisfied by the word and then seeks the root, sets the boolean variable.

Declaration: RuleSearch (var found:boolean)

Called By: SeparateRoot

Calls: ChangeLetter

ChangeLetter procedure :

Function: Makes necessary letter changes on the word.

Declaration: ChangeLetter (ch:string; chplace:integer;
var flag:boolean)

Called By: RuleSearch

Calls: SeparateRootWithRules

SeparateRootWithRules procedure :

Function: Searches the word whose some letters changed in the procedure called ChangeLetter and sets the variable found.

Declaration: SeparateRootWithRules (var found:boolean)

Called By: Changeletter

Calls: SeekRoot

StrRootKnw procedure :

Function: Assigns the value of koktypdf (i.e., change form) and fçatısı (i.e., root structure) by using the root identity code.

Declaration: StrRootknw

Called By: SeparateRoot

Calls: None

InCov function :

Function: Makes the input conversion.

Declaration: InCov (st:strverb)

Called By: ReadDataFile, GetWord, ChangeLetter

Calls: None

OutCov function :

Function: Makes the output conversion.

Declaration: OutCov (st:strverb)

Called By: RuleSearch

Calls: None

4 .3 EXTRACTING THE SUFFIXES OF A GIVEN WORD

After the root is extracted from the given word, the remaining part of the word is obtained and then the search starts from the

leftmost letter, continues up to the rightmost letter of the remaining part. The first letter of the remaining part is taken to determine whether it is a suffix or not. If the search fails, the letter which is the first of the new remaining part is concatenated to the searched letter. This operation continues until either the suffix is found or the last letter of the given word is reached. Moreover, when the suffix length exceeds the maximum possible suffix size, separation process stops there and if possible, it jumps to the upper level, adds a new letter, continues from there. Besides, a buffer consonant and vowel must be taken into consideration by using the rules. When the suffix is found, it is separated from the remaining part and the procedures repeat again. The following variables are used in the suffix extraction algorithm.

i : It is used to show the number of the letters in the temporary suffix.

remaining part : It is obtained after the root is extracted from the given word.

remainchild : It is the the number of letters in the remaining part.

news : It is the temporary suffix constructed by using i letter of the remaining part.

sufsize : It is the number of the letters in the temporary suffix.

The suffix extraction algorithm may be outlined as follows:

Step 1 : SP <-- 0, i <-- 0.

Step 2 : remainchild <-- Length (remaining).

Step 3 : i <-- i+1.

Step 4 : news <-- Copy (remaining,1,i),

sufsize <-- Length(news),

remainchild <-- remainchild-1.

Step 5 : Search news from the doubly linked circular list whose data come from the Suffix Bank.

Step 6 : If news does not exist in the linked list, go to Step 9. Otherwise, Push(news, remainchild, remaining, SP).

Step 7 : Store identity code of the founded suffix, place their possible types in an array.

Step 8 : Delete(remaining, 1, sufsize).

Step 9 : If "i" is equal to eksize, go to Step 11.

Step 10 : If the remainchild is not equal to "0", go to Step 2.

Step 11 : If flag is true, then go to Step 13.

Step 12 : Pop(news, remainchild, newremainig, SP),

i <-- Length(news), go to Step 3.

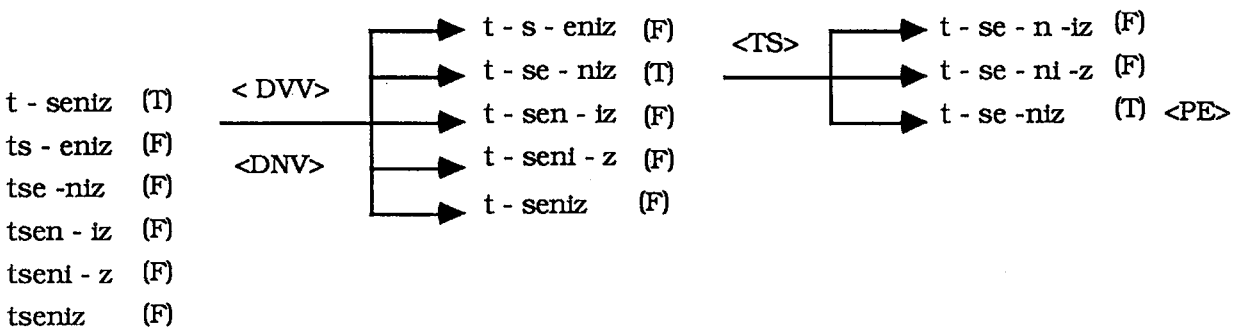
Step 13 : Stop searching the suffixes.

The following example is given to show the extraction of all the suffixes of a given word.

Given word: gevşetseniz

Root of the given word: gevşe /loose

Remaining part: tseniz



Result: **gevşe - t - se - niz**

(Root) (SDVV) (TS) (PE)

After finding all the suffixes in the given word, the possible types of each suffix are checked by using general rules which are affirmative rules and specific rules. If the type of the suffix does not satisfy any general rule and it satisfies any of its specific rules, this suffix type is stored as a solution for this suffix. Then the other types of the suffix is checked and all the founded types of this suffix are placed into the solution set. Next, succeeding suffix is checked. This procedure is repeated until all the suffixes are examined. After that, if exists, the second solution for the analysis of the given word is found. If it exists, the explained steps are repeated. However, if one of the suffixes in the word satisfies any general rule and it does not satisfy any of its specific rule, other suffix combination for the word is searched to see whether it exists or not. If another way is possible, the program goes to the rules. The flow chart of this algorithm is shown in Figure 4.5.

The module *Extract_Suffixes* calls the procedure *SeparateSuffix* and then its sub-modules are executed as shown in Figure 4.6.

SeparateSuffix procedure :

Function: Extracts the suffixes of the given word.

Declaration: SeparateSuffix

Called By: Extract_Suffixes

Calls: ReadDataFile, FindSfx

ReadDataFile procedure :

Function: Reads the suffixes from the Suffix Bank and loads them in doubly circular linked list.

Declaration: ReadDataFile

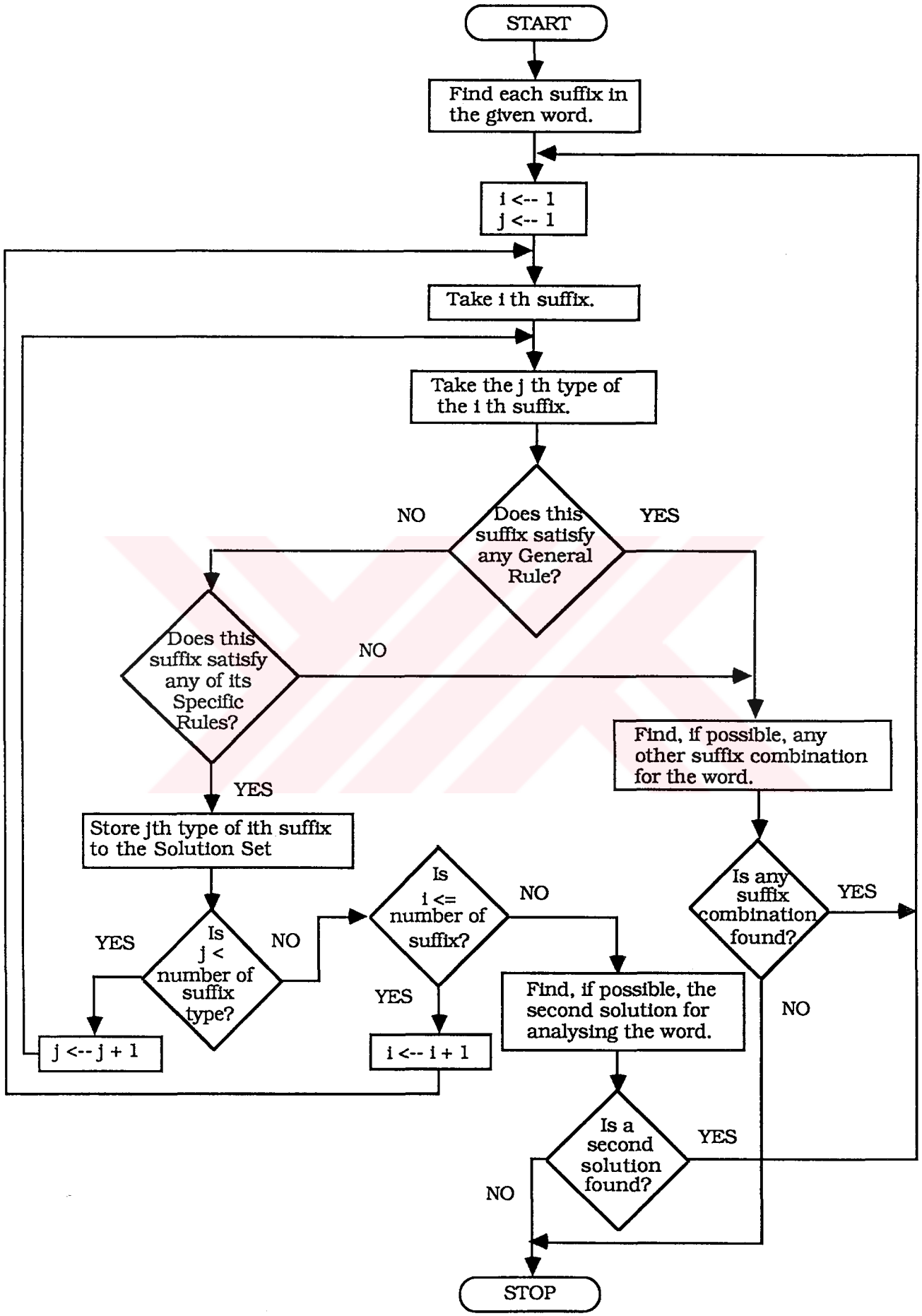


Figure 4.5 Flowchart for the extraction of the suffixes

Called By: SeparateSuffix

Calls: InsertDataFile

InsertDataFile procedure :

Function: Creates a node, places the suffix in the node and sets its links.

Declaration: InsertDataFile (key:strsfx; inum:strkod;
var cposition: spointer)

Called By: ReadDataFile

Calls: None

FindSfx procedure :

Function: Extracts the suffixes of the remaining part and then checks the solution.

Declaration: FindSfx (var newremaining:strremain)

Called By: SeparateSuffix

Calls: FindNewSfx, CheckHarshCnt, CheckFoundSfx, Initialize, WriteSoln, PrepareSecondSolution, OpenMngFile, SearchRoot.

FindNewSfx procedure :

Function: Extracts the suffixes of the remaining part and prepares the knowledge arrays by using identity codes of the suffixes.

Declaration: FindNewSfx (var newremaining:strremain;
found: boolean; i:integer)

Called By: FindSfx

Calls: StoreSfx, StoreSfxKod, PutTypesInArray, CheckCompoundVerbs

SeekSuffix procedure :

Function: Extracts the suffixes of the remaining part, if possible.

Declaration: SeekSuffix (newstr:strsfx; var flag:boolean)

Called By: FindNewSfx

Called By: Search

Search function :

Function: Extracts the suffixes of the remaining part, if possible.

Declaration: Search (key:strsfx; var cposition:spointer)

Called By: SeekSuffix

Calls: None

StoreSfxKod procedure :

Function: Loads the identity code of the determined suffix into the h^{th} place of the suffix code array.

Declaration: StoreSfxKod (h:integer)

Called By: FindNewSfx

Calls: None

PutTypesInArray procedure :

Function: Loads the possible types of the determined suffix into the suffix type array.

Declaration: PutTypesInArray (h:integer)

Called By: FindNewSfx

Calls: None

CheckCompoundVerbs procedure :

Function: Checks the remaining part to see whether there exists a compound verb.

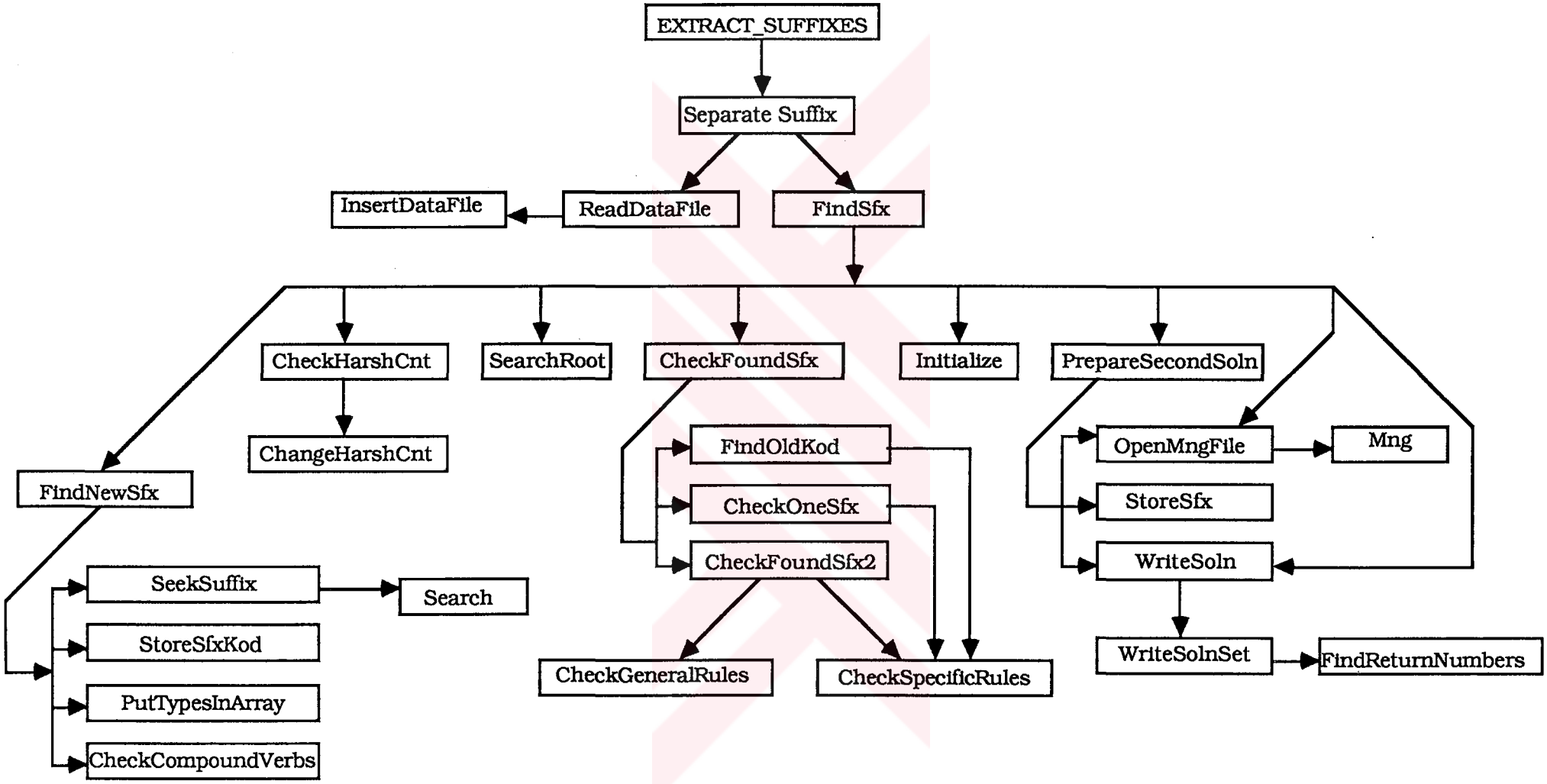


Figure 4.6 Software structure of extracting the suffixes

Declaration: CheckCompoundVerbs (newremaining:strremain;
var i:integer)

Called By: FindNewSfx

Calls: None

CheckHarshCnt function :

Function: Checks the letters of the remaining part to see whether any letter is softened (i.e., it changes to b, c, d, or ğ).

Declaration: CheckHarshCnt (onesoln:integer)

Called By: FindSfx

Calls: ChangeHarshCnt

ChangeHarshCnt procedure :

Function: Changes the soft consonant letter v1 into the harsh consonant v2, and then sets the variable ccs to true.

Declaration: ChangeHarshCnt (v1,v2:char;
var toldremain:strremain; css:boolean)

Called By: CheckHarshCnt

Calls: None

SearchRoot procedure :

Function: Searches another root for the given word by starting with the pointer called rootpointer.

Declaration: SearchRoot

Called By: FindSfx

Calls: RootSearch

CheckFoundSfx procedure :

Function: Checks the found suffixes and finds their possible types.

Declaration: CheckFoundSfx

Called By: FindSfx

Calls: FindOldKod, CheckOneSfx, CheckFoundSfx2

FindOldKod function:

Function:Finds the possible types of the first suffix.

Declaration: FindOldKod

Called By: CheckFoundSfx

Calls: CheckSpecificRules

CheckOneSfx procedure :

Function: Finds the possible types of the suffix, checks the change form field of the root identity code and then changes the root or stores the suffix.

Declaration: CheckOneSfx

Called By: CheckFoundSfx

Calls: CheckSpecificRules

CheckFoundSfx2 function :

Function: Checks the rules and then stores the possible suffix types.

Declaration: CheckFoundSfx2

Called By: CheckFoundSfx

Calls: CheckGeneralRules, CheckSpecificRules

CheckGeneralRules function :

Function: Compares the possible types of the previous suffix (okod) with the types of the searched suffix (nkod).

Declaration: CheckGeneralRules (okod, nkod:strkod)

Called By: CheckFoundSfx2

Calls: None

CheckSpecificRules function :

Function: Checks the specific rules of the searched suffix (ssuf).

Declaration: CheckSpecificRules (ssuf:strsfx; wsfx:integer)

Called By: CheckFoundSfx2

Calls: Specific rule functions for necessary suffixes

Initialize procedure :

Function: Sets the initial values of the arrays, if the found last suffix is not found correctly.

Declaration:Initialize

Called By: FindSfx

Calls: None

PrepareSecondSolution procedure :

Function: Prepares the arrays for the second solution.

Declaration:PrepareSecondSoln (var check:boolean)I

Called By: FindSfx

Calls: OpenMngFile, StoreSfx, WriteSoln

OpenMngFile procedure :

Function: Finds the meanings of the found suffixes, and concatenates them with the root meaning.

Declaration:OpenMngFile

Called By: FindSfx, PrepareSecondSoln

Calls: Mng

StoreSfx procedure :

Function: Loads the found suffixes into the suffix array.

Declaration: StoreSfx

Called By: PrepareSecondSoln

Calls: None

WriteSoln procedure :

Function: Writes the root and the suffixes of the word on to the screen.

Declaration: WriteSoln

Called By: FindSfx, PrepareSecondsoln

Calls: WriteSolnSet

WriteSolnSet procedure :

Function: Writes the possible solutions (root and suffix combinations) for the given word on the screen.

Declaration: WriteSolnSet

Called By: WriteSoln

Calls: FindReturnNumbers

FindReturnNumbers procedure :

Function: Finds the return numbers of the suffixes.

Declaration: FindReturnNumbers

Called By: WriteSolnSet

Calls: None

The functions *InCov* and *OutCov* are called for input and output conversion by some modules.

4 .4 THE MEANING OF A GIVEN WORD

After determining the suffixes of a given word and finding their return numbers, the meaning of each return number can be taken from the Meaning Bank. However, the meaning of each suffix must be written in the correct place. For this reason, a unit file called *Meaning.tpu* is developed to combine the meanings of the suffixes.

Besides finding the meanings of all the suffixes of the given word, we may need some extra words between these meanings. Their places are adjusted by the program depending on the suffix types.

If the given word is a verb, the personal ending is explained at the beginning of the meaning sentence and then the root is added. However, if the word is a noun, the root is placed at the beginning of the meaning sentence. The meaning of the suffixes of the word are combined to the root by adding necessary extra words. In order to combine these meanings, all the meanings are placed in the array *x*. Therefore, the meaning of a suffix or an extra word is in x[i] where *i* shows the place of the suffix meaning or the word in the explanation sentence. Then, the meaning of the word is concatenated by the following statement *Concat (x[1], x[2], x[3], ...)*. The examples shown below explain the meanings of some words.

Examples:

1) Word = Root + SDNV --> NOUN

Word : süzgeç /filter

Root : süz_ /to filter

SDNV : -geç

RTN of SDNV : 31

RTN 31 is used to indicate the equipment/device/tool which can be used for the action/job/task that mentioned by the root.

The meaning of the word: *The equipment/device/tool which can be used for filtering. (süzme eylemine yarayan araç.)*

2) Word = Root + TS + PE --> VERB

Word : geldi /He(she,it) came.

Root : gel_ /to come

Although this word has only one suffix, there are two RTNs. Because, the PE is hidden.

TS : -di

RTN of TS : 80

RTN 80 is used to indicate the witnessed simple past tense form of the root.

PE : -

RTN of PE: 94

RTN 94 is used to indicate the third singular personal ending.

The meaning of the word: The action "to come" was done by the third singular personal ending. (*3. tekil kişinin gelme eylemini görülen geçmiş zamanda yaptığı bildiriliyor.)*

3) Word = Root + TS + TS + PE --> VERB

Word : koşuyordum /I was running.

Root : koş_ /to run

TS: -yor

RTN of SDVV : 83

RTN 83 is used to indicate the progressive present tense form of the root

TS: -du

RTN of TS: 89

RTN 89 is used to indicate the compound imperfect tense form of the root.

PE: -m

RTN of PE: 92

RTN 92 is used to indicate the first singular personal ending.

The meaning of the word: The action "to run" was being performed by the first singular personal ending. (*1. tekil kişinin koşma eylemini yapmakta olduğu hikaye ediliyor.)*

4) Word = Root + SDVV + TS + TS + PE --> VERB

Word : kovalayacaksa / If he(she, it) will pursue.

Root : kov_ /to expel

SDVV: -ala

RTN of SDVV : 11

RTN 11 is used to indicate repetition of the root action.

TS: -acak

RTN of SDNV: 84

RTN 84 is used to indicate the future tense form of the root.

TS: -sa

RTN of SDNV: 91

RTN 91 is used to indicate that the root is used in a conditional clause.

PE: -

RTN of PE: 94

RTN 94 is used to indicate third singular personal ending.

The meaning of the word: *If the third singular person will pursue. (3. tekil kişinin kovma işini belli bir süreyle tekraralanmasını yapacağı gelecek zamanda koşul olarak bildiriliyor.)*

5) Word = Root + Negative suffix + TS + PE --> VERB

Word: okumamalısın /You must not read.

Root: oku_ /to read

SDVV: -ma

RTN of SDVV : 52

RTN 52 is used to indicate negation form of the root.

TS: -malı

RTN of SDNV: 87

RTN 87 is used to indicate necessity.

PE: -sın

RTN of SDNV: 93

RTN 93 is used to indicate the second singular personal ending.

The meaning of the word: *The action "to read" must not be done by the second singular personal ending. (2. tekil kişinin okuma eylemini yapmamasının gerekliliği bildiriliyor.)*

6) The word "okumuş" has two solutions. Because, two types of the suffix "-muş" is suitable for this word.

Root: oku_ /to read

TS: -muş

RTN of TS: 81

RTN 81 is used to indicate the reported simple past tense form of the root.

SDNV: -muş

RTN of SDNV: 62

RTN 62 is used to indicate a person who performed the root action.

The suffix "muş-" is both TS and SDNV.

The first meaning of the word indicates the action "to read" *3. tekil kişinin okuma eylemini duyulan geçmiş zamanda yaptığı bildiriliyor.)*

The second meaning of the word indicates *a person who performed the action "to read". (Okuma eylemini yapmış olma özelliği taşıyan kişi.)*

The procedure *Mng* combining the meanings of the suffixes is called by the main program and then this module calls the procedures *InitAry, AccessKnw, FindMeaning.*

InitAry procedure :

Function: Sets the initial vulue of the arrays.

Declaration: InitAry

Called By: Mng

Calls: None

AccessKnw procedure :

Function: Takes the necessary knowledge from the data file called StoreKnw.

Declaration: AccessKnw

Called By: Mng

Calls: None

FindMeaning procedure :

Function: Finds the meaning of a given word and then writes the meaning.

Declaration: FindMeaning

Called By: Mng

Calls: InitMng, Typ, Type1, Type2, Type3

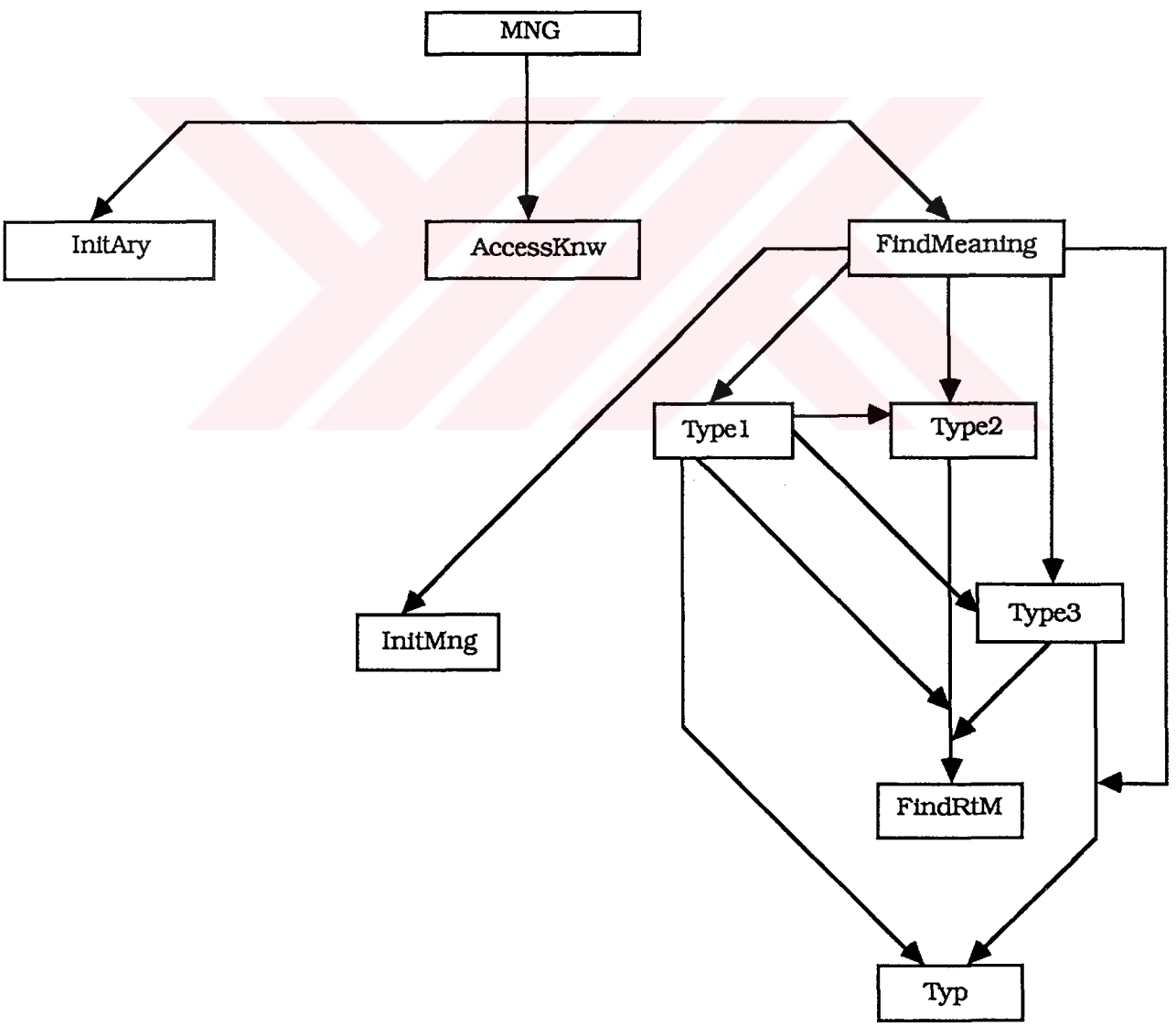


Figure 4.7 Software structure of the meaning concatenation

InitMng procedure :

Function: Sets the initial value of the meaning array.

Declaration: InitMng

Called By: FindMeaning

Calls: None

Typ function :

Function: Assigns the suffix type to the function identifier.

Declaration: Typ (var i1,i2: integer)

Called By: FindMeaning, Type1, Type3

Calls: None

Type1 procedure :

Function: Concatenates the meaning of the SDVV to the root and then continues to find other suffix meanings.

Declaration: Type1 (var i1, i2 : integer)

Called By: FindMeaning

Calls: FindRtM, Typ, Type2, Type3

Type2 procedure :

Function: Finds the meaning of the SDNV and concatenates the previous suffix meanings.

Declaration: Type2 (var i1, i2 : integer)

Called By: Type1, FindMeaning

Calls: FindRtM

Type3 procedure :

Function: Combines the meaning of the TS and then continues the operation.

Declaration: Type3 (var i1, i2 : integer)

Called By: FindMeaning, Type1

Calls: FindRtM, Typ.

FindRtM procedure :

Function: Searches the meaning of the return number from the Meaning Bank.

Declaration: FindRtM (var i1, i2: integer)

Called By: Type1, Type2, Type3

Calls: None

The variables i1 and i2 show the $i1^{th}$ solution type of the $i2^{th}$ suffix of the given word.

The figure 4.7 shows the software structure of the meaning concatenation.

4 .5 HOW TO LEARN A NEW ROOT

If the root of a given word is not found, the unknown root can be learned from the user to update the Root Bank. For this reason, a unit file is constructed. When the program encounters an unknown root, the TPU file called Learn is activated.

The first step of the learn operation is the confirmation of the given word by the user. If the user says that the spelling of the word is incorrect, the learning immediately stops and returns to the main program, waits for another word. If the word is correct and if the user wants to add the new root to the Root Bank, the program starts the learning steps.

The new root is taken from the user and then it is searched in the Root Bank. If this root already exists in the bank, a message

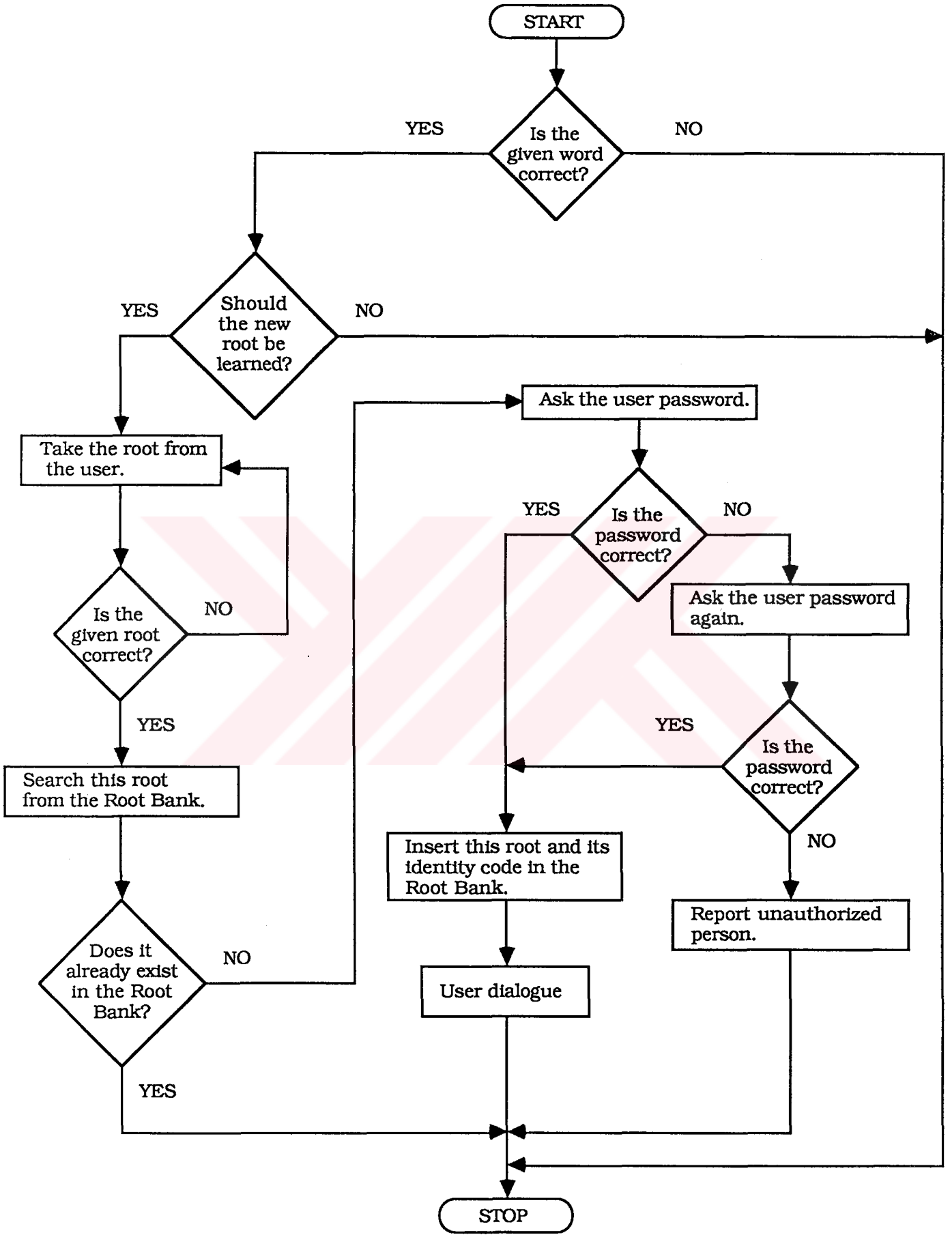


Figure 4.8 Flowchart for learning a new root

appears on the screen to warn the user and then learning stops. If the new root does not exist in the bank, some necessary questions about the suffixes which have more than one type are asked to the user to produce the identity code of the new root. Moreover, the root form, and the root structure is learned from the user. After that, the password is asked to the user. Since the root and its identity code is very important for recognition of a word, the key word is used for entering a new knowledge to the bank. Therefore, unnecessary or incorrect knowledge is prevented. If the user cannot enter the correct password at the first time, he has a second chance. If he still does not know, the program decides that the user is an unauthorized person and learning stops. However, if the correct password is entered, the root and its code is inserted to the bank and the learn operation is achieved. The flowchart of this algorithm is shown in Figure 4.8.

The learning is started by calling the module *NewRoot* from the main program. This module has four sub-modules: *ReadDataFile, Root, LearnSmpForm, Dispose_Root_Links*. The software structure of the learning is shown in Figure 4.9.

ReadDataFile procedure:

Function: Reads the roots from the Root Bank and then loads them to the doubly circular linked list.

Declaration: ReadDataFile.

Called By: NewRoot.

Calls: InsertDataFile

Root procedure :

Function: Checks the correctness of the given word and asks to the user whether he wants to add the new root to the bank,

Declaration: Root.

Called By: NewRoot.

Calls: TakeRootKnow.

LearnSmpForm procedure :

Function: Learns the simple form of the new root, if exists.

Declaration: LearnSmpForm.

Called By: NewRoot.

Calls: Help1, TakeSmpRKnow, SearchKey.

Dispose_Root_Links procedure:

Function: Disposes the links of the doubly linked circular list.

Declarations: Dispose_Root_Links.

Called By: NewRoot.

Calls: None.

The following modules are the sub-modules of the explained modules.

InsertDataFile procedure :

Function: Loads the roots in the Root Bank to the doubly linked circular list in sorted form.

Declaration: InsertDataFile (key: strroot; inum: strrkod; var cposition: rpointer).

Called By: ReadDataFile.

Calls: None.

TakeRootKnow procedure :

Function: Gets the root from the user and then confirms the correctness of the root by asking to the user.

Declaration: TakeRootKnow.

Called By: Root.

Calls: GetRootInput, SearchKey.

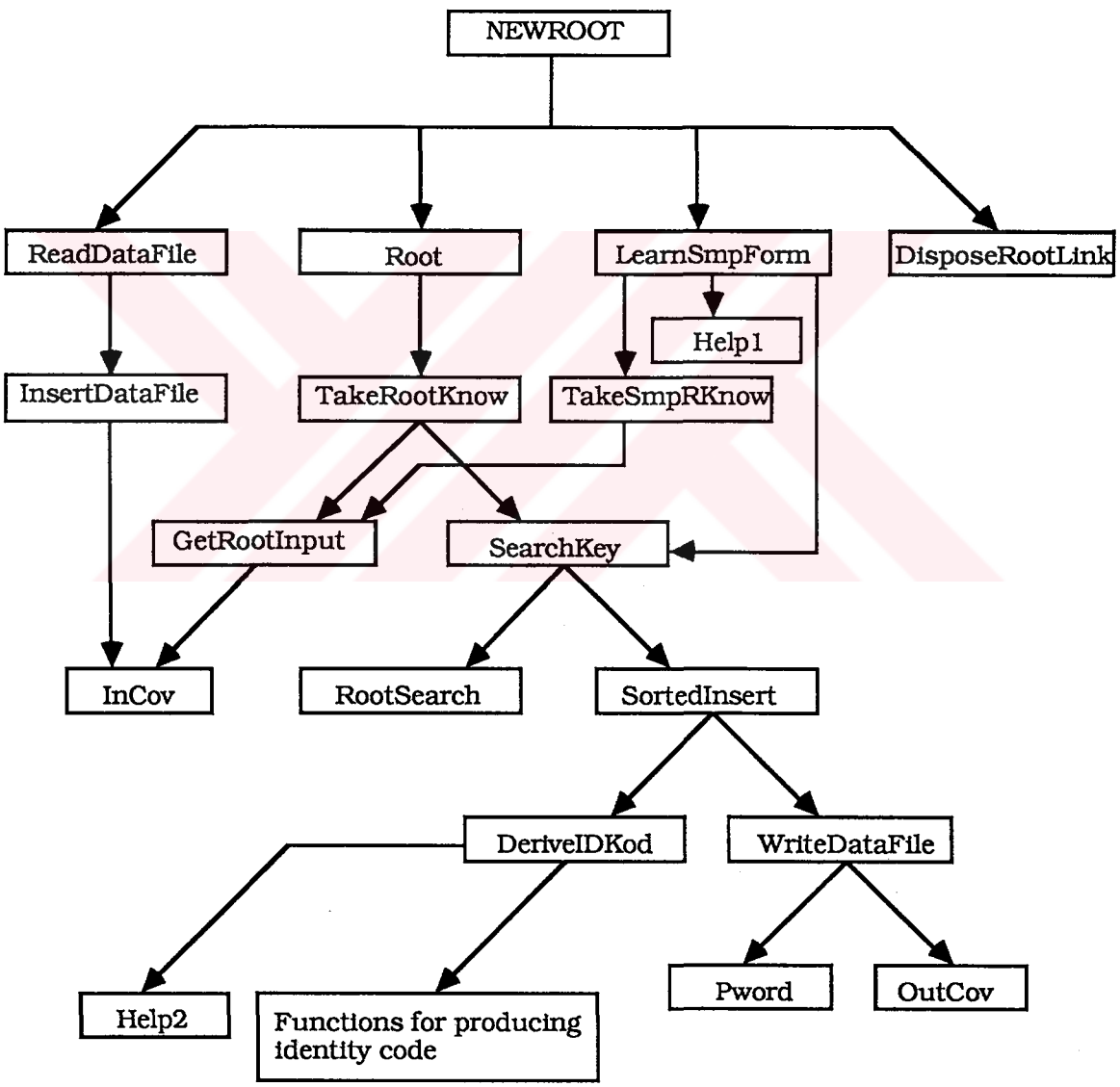


Figure 4.9 Software structure of the learning

GetRootInput procedure :

Function: Takes the root from the user.

Declaration: GetrootInput (message:string;var temproot: strroot).

Called By: TakeRootKnow, TakeSmpRKnow.

Calls: InCov.

SearchKey procedure :

Function: Either calls or not call the procedure SortedInsert depending on the return value of the function RootSearch. If the root is an already existing one, a message appears on the screen and the learning is aborted.

Declaration: SearchKey (key: strroot).

Called By: TakeRootKnow, LearnSmpForm.

Calls: RootSearch, SortedInsert.

RootSearch function:

Function: Searches the new root in the Root Bank and then the flag showing the root existence in the bank returns in the function identifier.

Declaration: RootSearch(key: strroot; var cposition: rpointer).

Called By: SearchKey.

Calls: None.

SortedInsert procedure:

Function: Creates a new node, sets the links of the node by using the pointer cposition and places the value of the variable key in the data field.

Declaration: SortedInsert(var key:strroot; cposition: rpointer).

Called By: SearchKey.

Calls: DeriveIDKod, WriteDataFile.

DeriveIDKod procedure :

Function: Produces the root identity code by asking some necessary questions about the root and the suffixes which have more than one type.

Declaration: DeriveIDKod.

Called By: SortedInsert.

Calls: Suffix functions for producing identity code.

WriteDataFile procedure :

Function: Loads the new doubly circular linked list to the Root Bank, if the return value of the function Pword is true.

Declaration: WriteDataFile.

Called By: SortedInsert.

Calls: Pword, OutCov.

TakeSmpRKnow procedure :

Function: Reads the simple root and requires the confirmation of the root by the user.

Declaration: TakeSmpRKnow.

Called By: LearnSmpForm.

Calls: GetRootInput.

Pword function:

Function: Takes the password from the user. If the password is correct, true is set to the function identifier.

Declaration: Pword.

Called By: WriteDataFile.

Calls: None.

InCov function:

Function: Makes the input conversion of a root.

Declaration: InCov(st:strverb)

Called By: ReadDataFile, GetRootInput.

Calls: None.

OutCov function:

Function: Converts a root to the output form for the user.

Declaration: OutCov(st:strverb)

Called By: WriteDataFile.

Calls: None.

There are also two help procedures to explain the form change and the structure of a root to the user.

General comments on the algorithm and the package are presented in the following chapter.

CHAPTER 5

CONCLUSION

In this thesis a method, which uses knowledge banks, for the analysis and finding the meaning of the deverbal words in Turkish language is proposed. If the given word has more than one meaning, all the possible solutions are found by the program. Moreover, the root or the suffixes of a word can be extracted in different forms for each solution.

In order to implement the method, a software for the IBM-AT compatible computers is developed. The processor of the computer is 80286 and runs at 8 MHz. The program is implemented in Turbo Pascal programming language (version 5). Two units are prepared to separate the program into modules. The main program is used to distinguish the root and the suffixes of a deverbal word by the depth first search method. The training for a new root and combining the meanings of the suffixes to the root are done by the units. Therefore, several code files are managed simultaneously. Totally 97 KBytes are used for the source. The learn unit is about 20 KBytes and the meaning unit is 12 KBytes. Each root occupies 40 bytes, and each suffix occupies 50 bytes in the memory.

There are approximately 1000 verbal roots in Turkish. Since each root occupies 40 bytes, for the Root Bank 40 KBytes of a storage area are used. Including the source and the other banks, approximately 126 KBytes of the storage area are occupied. If all of the deverbal words of Turkish are stored directly, approximately 2.2 MBytes of the storage area are used. This number does not include the meaning definitions.

On the average, there are three letters for each root and two letters for each suffix. Then, for five basic tenses and six personal endings approximately 200 KBytes are used. Similarly, for the imperative 120 KBytes of the storage area are used. For the compound tenses and the verb structure 600 KBytes and 500 KBytes are used, respectively. For the other suffixes deriving verbs or nouns from verbal roots at least 800 KBytes of the storage area are used.

The execution time for the recognition of a given word can be approximated by

$$T_R (R_c) + \Sigma T_{S_i} (S_{i_c})$$

where T_R is the time for the identification of the root, R_c is the number of characters in the root, T_{S_i} is the time for the identification of the suffixes, and S_{i_c} is the number of characters in the i^{th} suffix. T_R and T_{S_i} are the functions of R_c and S_{i_c} , respectively.

The recognition of a root which has three characters takes 0.66 seconds. However, the execution time for the identification of the suffixes depends on the number of suffixes in a given word. If there is only one SDVV type suffix in the word, it is found in 1.50 seconds. If the given word has an SDNV type of suffix, the execution time for the

identification of this suffix is 1.11 seconds. The execution times for the identification of TS and PE are 0.44 and 0.11 seconds, respectively.

It must be pointed out that the execution time does not linearly increase. When the given word has more than one suffix, the expected running time cannot be equal to the real running time. In other words, the execution time cannot be found by adding the execution time of each suffix in a word. The real execution time may be less than the expected time. Because, when the number of suffixes increases, the possibilities for the type of a suffix decrease. For example, the word "geldi" has one suffix and its execution time is 2.20 seconds. However, the word "geldim" has two suffixes and its execution time takes 1.21 seconds. Since the suffix "di" has two types and two meanings, the execution time of the second word is less than the first one. Besides, there is a hidden personal ending in the word "geldi".

In Turkish, most commonly used deverbal words have roots of three characters and two suffixes. This corresponds to about 1.50 seconds for the recognition.

The worst case complexity for searching a root and a suffix is O(n), n is the number of roots or suffixes in the banks.

The program which is developed during this thesis work extracts the meaning of a given deverbal word by combining the suffix induced meaning modifications and the root. We only examine how the suffixes modify the meaning of a word. However, the meanings of the roots are transparent to the program. Some roots may have two or more meanings and each meaning can be examined separately. But it

is impossible to distinguish the meanings without context information. The meanings of the roots can also be added to the Root Bank as suffix meanings stored in the Suffix Bank. However, due to the conceptual dependencies, which are learned via sense organs, it is very difficult, if not impossible, to break the definition loops.

The method introduced in Chapter 3 is also applicable to the denominal words. However, the program based on this method is formed by taking into account the characteristics of the deverbal words. For this reason, the knowledge banks and the rules must be revised to analyse both the deverbal words and the denominal words.

After the recognition of the meanings of all the words in Turkish, the sentence meanings can be recognized by using word knowledge. However, it is not an easy task. The same word in different sentences may have different meanings depending on the context. In order to distinguish the meanings of a word, the grammatical rules and the other words in the sentence must be considered. For this task, the grammatical rules have also to be stored; and the return numbers have to be updated accordingly. After that, the meaning modifications due to the suffixes can be concatenated to the root meaning and the complete meaning of a word in a sentence can be recognized.

LIST OF REFERENCES

[1] BANGUOĞLU, T., "Türkçenin Grameri", Türk Dil Kurumu Yayınları: 528, Ankara, 1986.

[2] ÖZEL, S., "Türkiye Türkçesinde Sözcük Türetme ve Bileştirme", Türk Dil Kurumu Yayınları: 438, Ankara, 1977.

[3] DEMİRAY, K., "Temel Dilbilgisi", İnkılap ve Aka Basımevi, İstanbul,1983.

[4] DEMİRCAN, Ö., "Türkiye Türkçesinde Kök-Ek Bileşmeleri", Türk Dil Kurumu Yayınları: 436, Ankara, 1977.

[5] ATABAY, N., KUTLUK, İ., ÖZEL, S., "Sözcük Türleri", Türk Dil Kurumu Yayınları: 421, Ankara, 1983.

[6] ERGİN, M., "Türk Dil Bilgisi", Bogaziçi Yayınları: 11, İstanbul, 1985.

[7] TOSUN, C., "A Contrastive Study of Word-Formation By Affixation in Turkish and English", Ph.D. Thesis, Hacettepe University, 1977.

[8] KÖKSAL, A., "Automatic Morphological Analysis of Turkish", Ph.D. Thesis, Hacettepe University, 1975.

[9] SAGAY, Z., "Sözcük Çekimi", Bilişim '78, Ankara, 1978

[10] "Türkçe Sözlük", Türk Dil Kurumu Yayınları: 549, Cilt.1-2, Ankara, 1988.

[11] AYDIN, S., BULUT, M. M., "Computer Aided Recognition of Deverbal Words in Turkish", Proceedings of the Fifth International Symposium on Computer and Information Sciences, Vol. 2, Kapadokya, Nevşehir, pp. 1213-1222, 1990.

[12] ÖZENCİ, M. A., "An Experiment in English-Turkish Translation Using Digital Computers", M.S. Thesis, Middle East Technical University, Ankara, 1978.

[13] PIERCE, J., "A Frequency Count of Turkish Words", Milli Eğitim Bakanlığı, 1960.

[14] HOROWITZ, E., SAHNI, S., "Fundamentals of Data Structures in Pascal", Computer Science Press Inc., USA, 1984.

[15] CAN, K., "Yabancılar İçin Türkçe-İngilizce Açıklamalı Türkçe Dersleri", ODTÜ, Fen ve Edebiyat Fakültesi: 42, Ankara, 1987.

APPENDICES

APPENDIX A

USER'S MANUAL

FOR

DEVERBAL WORD RECOGNITION PACKAGE

At some stages, the program interacts with the user to obtain the word, comments, and specifications on related fields. These are input, learning, and output stages.

Input Stage :

At this stage, after the message "Enter the word:", the user enters a deverbal word to be worked on. The user must be careful about the type of the word. If the given word is not a deverbal word, the analysis cannot succeed.

Learning Stage :

If it is not possible to analyse the given word, this means that the root does not exist in the Root Bank. At the learning stage, the program interacts with the user to obtain the new root and its identity code. After entering a new root, the questions about the root and the suffixes which can be affixed to this root are answered by the user through typing the letter of the selected choice. Therefore, the user is provided with the flexibility of deriving the code.

First, the user is required to confirm the given word. Note that if the user disconfirms, the program returns to the start and waits for a new word.

"Verilen kelime: ..."

"Verilen kelime doğru mu ? (E/H) :"

Next, it is asked to the user whether the unknown root is to be learned or not.

"Yeni bir kök öğrenmemi istiyor musunuz ? (E/H) :"

If the root is not required to be learned, the program again returns to the start. Otherwise, the new root is taken from the user and then it is confirmed. The root of the given word is repeatedly asked until the user confirms its correctness.

"Verilen kelimenin kökünü giriniz :"

"Bu kök doğru mu ? (E/H) :"

If the given word is found in the Root Bank, the following message appears on the screen.

"Bu kök biliniyor ..."

If the new root does not exist, the root structure is learned by the question below and also a comment helps to the user for answering.

"Bu kök:

A) Geçişli

B) Geçişsiz

C) Geçişli ve geçişsiz

Açıklama: Eğer kök "neyi,kimi" sorusunu cevaplıyorsa, kök geçişlidir.

Örnek : geçişli kök : yaz_

geçişsiz kök: koş_

geçişli ve geçişsiz kök : bul_

After selecting one of these choices, questions about the suffixes affixed to the root follow.

Question 1:

"-acak/-ecek" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemde kullanılan nesne (örn., yak-acak).

H) Hiçbiri

Seçiminizi Giriniz:

Question 2:

"-aç/ eç" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemi yapan veya yapmak için kullanılan nesne (örn., tık-aç).

B) Kökte bildirilen eylemin özelliğini taşıyan (örn., gül-eç).

H) Hiçbiri

Seçiminizi Giriniz:

Question 3:

"-ak/-ek" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemi genellikle yapan kişi (örn., kork-ak).

B) Kökte bildirilen eylemin yapıldığı yer (örn., yat-ak).

C) Kökte bildirilen eylemi yapmaya yarayan alet (örn., kay-ak).

H) Hiçbiri

Seçiminizi Giriniz:

Question 4:

"-ca/-ce/-ça/-çe" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemi sağlayan veya eylemin sonucu (örn., eğlen-ce).

B) Kökte bildirilen eylemin nedeni (örn., güven-ce).

H) Hiçbiri

Seçiminizi Giriniz:

Question 5:

"-ga/-ge" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemi yapan (örn., süpür-ge).

B) Kökte bildirilen eylemin sonucu (örn.,öner-ge).

C) Kökte bildirilen eyleme maruz kalan yer, nesne, şey (örn., sömür-ge).

H) Hiçbiri

Seçiminizi Giriniz:

Question 6:

"-gaç/-geç/-kaç/-keç" eki köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eyleme yarayan araç (örn., kıs-kaç).

B) Kökte bildirilen eylem duygusu olan, taşıyan, doğuran; eyleme neden olan veya sonucu olan (örn., utan-gaç).

H) Hiçbiri

Seçiminizi Giriniz:

Question 7:

"-gı/-gi/-gu/-gü/-kı/-ki/-ku/-kü" eklerinden biri köke eklenirse, kelimenin anlamı:

A) Kökte bildirilen eylemi yapmaya yarayan alet (örn., bur-gu).

B) Kökte bildirilen eylemin sonucu (örn., çiz-gi).

C) Kökte bildirilen eylemle ilgili nesne. şey (örn., duy-gu).

H) Hiçbiri

Seçiminizi Giriniz:

Question 8:

"-gıç/-giç/ guç/-güç" eklerinden biri köke eklenirse, kelimenin anlamı:

A) Kökte bildirilen eylemi yapmaya yarayan alet (örn., patlan-gıç).

B) Kökte bildirilen eylemle uğraşan kişi (örn., dal-gıç).

C) Kökte bildirilen eylemin durumu (örn., başlan-gıç).

H) Hiçbiri

Seçiminizi Giriniz:

Question 9:

"-gın/-gin/-gun/-gün" eklerinden biri köke eklenirse, kelimenin anlamı:

A) Kökte bildirilen eylemi yaparak veya maruz kalarak edinilen nitelik veya kişi (örn., gez-gin).

B) Kökte bildirilen eylemin yapılması olayı (örn., yan-gın).

H) Hiçbiri

Seçiminizi Giriniz:

Question 10:

"-ı/-i/-u/-ü" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylem sonucu oluşan yapı veya şekil (örn., yaz-ı)

B) Kökte bildirilen eylem için gerekli nesne (örn., kok-u).

C) Kökte bildirilen eylemi tanımlayan ad (örn., koş-u).

H) Hiçbiri

Seçiminizi Giriniz:

Question 11:

"-ım/-im" eklerinden bri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemin birden, bir defada veya aralıklı tekrarlanmasını tanımlayan ad (örn., doğ-um).

B) Kökte bildirilen eylemi yapmak için gerekli nesne (örn., ak-ım).

H) Hiçbiri

Seçiminizi Giriniz:

Question 12:

"-k" eki köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemin sonucunda oluşan (örn., aç-ı-k).

B) Kökte bildirilen eylemin sonucunda oluşan nesne (örn., tükür-ü-k).

C) Kökte bildirilen eyleme maruz kalan şey (örn., ada-k).

H) Hiçbiri

Seçiminizi Giriniz:

Question 13:

"-ma/-me" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemle elde edilen nesne (örn., dol-ma)

B) Kökte bildirilen eylem için gereken araç (örn., kaz-ma)

H) Hiçbiri

Seçiminizi Giriniz:

Question 14:

"-maca/-mece" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemi yapılarak oynanan oyun, oyalanma, sürdürülen iş (örn.,bil-mece).

B) Kökte bildirilen eyleme maruz kalabilen araç, nesne (örn., çek-mece).

C) Kökte bildirilen eylemi kendi kendine yaptıran hayvan (örn., at-maca)

H) Hiçbiri

Seçiminizi Giriniz:

Question 15:

"-maç/-meç" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemle ilgili nesne, şekil (örn., yırt-maç).

B) İşi kökte bildirilen eylem olan (örn., çığırt-maç).

C) Kökte bildirilen eylem için gerekebilen şey (örn., de-meç).

H) Hiçbiri

Seçiminizi Giriniz:

Question 16:

"-mak/-mek" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemi yaparak çalışan alet (örn., ye-mek).

B) Kökte bildirilen eylemi yapmak için gerekli nesne (örn., çak-mak).

H) Hiçbiri

Seçiminizi Giriniz:

Question 17:

"-man/-men" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylem sonucu oluşan (örn., şiş-man).

B) Kökte bildirilen işi yapan kişi, nesne (örn., öğret-men).

H) Hiçbiri

Seçiminizi Giriniz:

Question 18:

"-mış/-miş/-muş/-müş" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Özel kalıplaşmış bir anlam (örn., dol-muş).

H) Hiçbiri

Seçiminizi Giriniz:

Question 19:

"-n" eki köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemin yapılması sonucu oluşan nesne (örn., yığ-ı-n).

B) Kökte bildirilen eyleminin yapılması sürecinin adı (örn., ak-ı-n).

C) Kökte bildirilen eylemin yapılmasında kullanılabilen nesne (örn., tüt-ü-n).

H) Hiçbiri

Seçiminizi Giriniz:

Question 20:

"-sı/-si/-su/-sü" eklerinden biri köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemin yapılma zamanı (örn., yat-sı).

B) Kökte bildirilen eyleme yatkın (örn., sin-si).

C) Kökte bildirilen eylemini yapabilen nesne (örn., tüt-sü).

H) Hiçbiri

Seçiminizi Giriniz:

Question 21:

"-t" eki köke eklenirse,

kelimenin anlamı:

A) Kökte bildirilen eylemin yapılmasına yarayan yer, nesne, araç (örn., taşı-t).

B) Kökte bildirilen eylem i yapabilen nesne (örn., um-u-t).

C) Kökte bildirilen eylemin sonucunda çıkan nesne (örn., yazı-t).

D) Özel kalıplaşmış bir anlam (örn., yoğur-t).

H) Hiçbiri

Seçiminizi Giriniz:

The answers of the above questions are used to generate the identity code of the new root. In order to store the root and its identity code, the user is asked if she/he is an authorized person or not.

"Yaptığınız değişikliklerin geçerli olması için lütfen anahtar-kelimeyi giriniz"

If the given password is incorrect, then the following message appears.

"Girilen anahtar-kelime yanlış! Lütfen bir kez daha deneyiniz ..."

The user is given a second chance to enter the password. If the user fails again, the learn operation stops and the message below appears on the screen.

"Siz değişiklik yapmaya yetkili değilsiniz ..."

If the given password is correct, it is asked if the given root covers a simpler one.

"Bu kökün basit bir şekli var mı ? (E/H)"

Açıklama: Bazı kelimelerin basit bir kökü olduğu halde, kelimenin anlamını o kök vermez.

Example: kelime : azdı > kök: az_

kelime : azaldı > kök: azal_

"azal_" kökünün basit şekli "az_" köküdür.

If the given root has a simple form, then it is asked to the user and confirmed. If it is disconfirmed, the user is repeatedly asked the simple form.

"Verilen kökün basit şeklini giriniz: ...

"Yazılan kök doğru mu ? (E/H)"

When the correct simple form is entered, all of the steps of learning are repeated for the simple form

Output Stage :

At the output stage, the root and the suffixes of the given word, a table for the solution and the meanings of the given word appear on the screen.

After the solution is displayed, any key can be pressed to see the second solution, if it exists. If the second solution does not exist, it is asked to the user whether there exists another root for the given word. If there exists a new root, then it is learned.

Appendix B

ROOT BANK

Some commonly used verbal roots and their identity codes are shown below.

The first field of the root code is used to express the simple root. If the root is a simple root, this field is "1". Otherwise, it is "0".

The second field expresses the root structure. "1" indicates a transitive rot, and "0" shows an intransitive root. The number "2" is used to show the roots having both transitive and intransitive structure.

Other fields of the root code is reserved for return numbers. Each return number has three digits.

If the root *"aç"* is taken as an example: This root is not a simple root and its structure is transitive. Besides, the return numbers of the suffixes *"-acak, -ı, -k, -maca, -mış"* are necessary for the identity code of this root. Therefore, the identity code of this root is *"01003121044054062 "*.

Root	**Identity Code**
aç	01003121044054062
ak	10040125
aksa	01044

Root	Identity Code
al	11041044
aldat	01053
as	01003044050
aş	11053
at	11002006034053
atla	01053
bak	0000204040043
barış	00053
bas	11035040124044050
basıl	00053
başvur	00122
bil	111150350370320 41053
bile	11062116
bileş	00041045
bul	21111035053063
bula	00053127
bulaş	00040044
cevapla	01053
coş	00035032053
çal	110341230530
çiz	01036040045053
dal	00116032037053
daya	00008044053
de	11053
dene	01040046053
dinle	01053
dök	01040044053
döşe	01045053
dur	00007032
duy	01035123
düş	11032044060
düşün	01023
ek	11126
engelle	01053
eski	00023

Root	Identity Code
ez	01044053
geç	01125072
gel	10125
geliş	00040
gez	00038123
gir	00032
giy	01003041
göç	10044060
göster	01116030040
gözle	01050053
gül	00111053
güven	00019
hazırla	01125
ısır	01045
ısıt	00002
iç	11003
içer	11
içerle	00
işle	01044050
it	01053
kaç	00006053
kalk	00044
kay	00008035044053
kaz	110061210400440 51053
kazı	01044
kes	010030340320400 44
kıs	11030044
kıskan	01
kok	00122
kork	00006
koş	00123041053
kov	01053
oku	01
otur	00007
öğret	01060
öl	00032121040

Root	Identity Code
öner	01122
sakla	01040113
say	0100203503212204005306O
seç	01032040060
ser	01036
sev	01035
sık	01032121
soy	01003038
sür	11111034038122040
sürü	01125
taşı	01125
tut	01003002035032
usan	00031
utan	00031
uyu	00
üz	01032
ver	01035122
vur	01038
yak	01072
yan	00038
yap	00040121
yar	11035
yara	00040060
yat	00007032068
yaz	01035122074
ye	1100305763
yet	00035
yönet	01060
yüz	01030

Appendix C

SUFFIX BANK

All the suffixes affixed to verbs and their identity codes are shown below. The first field of the suffix code is used to represent how many different meanings exist for a suffix. Field 2,3,4, and 5 correspond to four different types of suffixes (SDVV, SDNV, TS, PE). Depending on the suffix type, a RTN is obtained from the four successive fields. Field 6,10,... has a digit between 1 to 4 which correspond to the types of the suffixes.

Let's take the suffix *"-a"* as an example. This suffix has two types and two meanings. The first type of this suffix is a SDNV type and the RTN is *"001"*. The second type is a TS type and the RTN is *"086"*. Therefore, the identity code of this suffix is *"2011020013086"*.

Suffix	**Identity Code**
a	2011020013086
abil	110001101
acak	302102003200430 84
aç	2020020022111
adur	110001102
agel	110001102
ağan	101002005
ak	303002006200720 08
akal	110001103
akla	110001010

Suffix	Identity Code
akoy	110001103
ala	10001011
alak	101002012
alı	101002078
am	101002013
amaç	101002014
amak	2020020092015
amaz	100103098
an	101002016
anak	101002017
ar	2110010792018
arak	101002107
arı	01002031
ası	101002020
ay	101002026
ayaz	110001105
baç	101002021
beç	101002021
ca	2020020192023
ce	2020020192023
cama	101002025
ceme	101002025
ç	101002024
ça	2020020192023
çe	1010020192023
dı	2002030803089
dık	101002028
dıkça	101002109
dır	110001029
di	2002030803089
dik	101002028
dikçe	101002109
dir	110001029
du	2002030803089
duk	101002028
dukça	101002109

Suffix	Identity Code
dur	110001029
dü	2002030803089
dük	101002028
dükçe	101002109
dür	110001029
e	2011020013086
ebil	110001101
ecek	302102003200430 84
eç	2020020022111
edur	110001102
egel	110001102
eğen	101002005
ek	303002006200720 08
ekal	110001103
ekle	110001010
ekoy	110001103
ele	110001011
elek	101002012
eli	101002078
em	101002013
emeç	101002014
emek	2020020092015
emez	100103098
en	101002016
enek	101002017
er	2110010792018
erek	101002107
eri	101002031
esi	101002020
ey	101002026
eyaz	110001105
ga	303002115211621 17
gaç	2020020242030
gan	101002033
ge	303002115211621 17
geç	2020020242030

Suffix	Identity Code
gen	101002033
gı	303002034203520 36
gıç	303002118211921 20
gın	2020020322038
gi	303002034203520 36
giç	303002118211921 20
gin	2020020322038
gu	303002034203520 36
guç	303002118211921 20
gun	2020020322038
gü	303002034203520 36
güç	303002118211921 20
gün	2020020322038
ğ	190000000
ı	203002121212221 23
ıcı	101002039
ıkla	110001010
ım	202012040204140 92
ımsa	110001106
ın	100014096
ınca	101002077
ınız	100014096
ıp	101002027
ır	2110010792018
ış	321001047104820 43
ıver	110001104
ız	100014095
i	303002121212221 23
ici	101002039
ikle	110001010
im	202012040204140 92
imse	110001106
in	100014096
ince	101002077
iniz	100014096
ip	101002027

Suffix	Identity Code
ir	2110010792018
iş	32100104710482043
iver	110001104
iz	100014095
k	3030120442045204640 95
kaç	2020020242030
kan	2020020332108
keç	2020020242030
ken	2020020332108
kı	303002034203520 36
kın	2020020322038
ki	303002034203520 36
kin	2020020322038
ku	303002034203520 36
kun	2020020322038
kü	303002034203520 36
kün	2020020322038
l	110001042
lar	100014097
ler	100014097
lım	100014095
lim	100014095
lum	100014095
lüm	100014095
m	200014092
ma	4130010522049205020 51
maca	303002053205420 55
maç	303002127212821 29
madan	101002056
mak	303002049205720 58
malı	100103087
man	2020020592060
maz	2011020373098
me	4130010522049205020 51
mece	303002053205420 55
meç	303002127212821 29

Suffix	Identity Code
meden'	101002056
mek	303002049205720 58
meli	100103087
men	2020020592060
mez	2011020373098
mı	2100110524110
mık	101002061
mış	4022020622063308130 90
mi	2100110524110
mik	101002061
miş	4022020622063308130 90
mu	2100110524110
muk	101002061
mur	101002064
muş	4022020622063308130 90
mü	2100110524110
mük	101002061
mür	101002064
müş	4022020622063308130 90
n	513011112212421252126 4093
nız	100014096
niz	100014096
ntı	101002076
nti	101002076
ntu	101002076
ntü	101002076
nuz	100014096
nüz	100014096
p	110001065
pak	101002066
pek	101002066
r	100103082
sa	2002030853091
sak	101002063
sal	101002067
se	2002030853091

Suffix	Identity Code
sek	101002063
sel	101002067
sı	303002068206920 70
sın	2000240934094
sınız	100014096
sınlar	100014097
si	303002068206920 70
sin	2000240934094
siniz	100014096
sinler	100014097
su	303002068206920 70
sun	2000240934094
sunlar	100014097
sunuz	100014096
sü	303002068206920 70
sün	2000240934094
sünler	100014097
sünüz	100014096
ştır	110001022
ştir	110001022
t	514001071206320722073 2074
tı	302202075308030 89
tır	110001029
ti	302202075308030 89
tir	110001029
tu	302202075308030 89
tur	110001029
tü	302202075308030 89
tür	110001029
u	303002121212221 23
ucu	101002039
um	202012040204140 92
umsa	110001106
un	100014096
unca	101002077
unuz	100014096

Suffix	Identity Code
up	101002027
uş	32100104710482043
uver	110001104
uz	100014095
ü	303002121221222123
ücü	101002039
üm	202012040204140 92
ümse	110001106
ün	100014096
ünce	101002077
ünüz	100014096
üp	101002027
üş	32100104710482043
üver	110001104
üz	100014095
v	101002113
van	201002061
ven	201002061
y	101002026
yor	100103083

Appendix D

RETURN NUMBERS AND THE MEANING BANK

All the return numbers of the suffixes and their formed meanings are shown in Appendix D.1. This knowledge is used for concatenating meanings of the suffixes to the root meaning. Some extra words are also added by the program between these meanings.

In Appendix D.2, a detailed explanation about the meaning of the RTNs is given with examples.

D.1 MEANING BANK

RTN	Description
001	ulaşılan sonuç, nesne,yer
002	yapan veya yapmak için kullanılan nesne
003	kullanılan nesne
004	yapacak olan nesne
005	alışkanlık haline getiren, aşırı veya devamlı yapan veya olan
006	genellikle yapan kişi
007	yapıldığı yer
008	yapmaya yarayan alet
009	türemiş yalın isim
010	olası aralıklarla belli bir süreyle yapması veya devam ettirmesi
011	belli bir süreyle tekrarlaması
012	yapmak zorunda olan veya alışkanlık haline getiren nesne
013	sonunda elde edilen (nesne)
014	yapıldığı yer veya araç
015	yapıldığı yer veya araç

RTN	Description
016	yapan
017	sürekliliğinin veya bulunmasının sonucu
018	yapan (kişi,alet,nesne)
019	nedeni
020	olması veya maruz kalması istenilen
021	gerektiren yer veya nesne
022	sınıflandırmaya dayalı olarak yapması
023	sağlayan veya eylemin sonucu
024	duygusu olan,taşıyan,doğuran; eyleme neden olan veya sonucu olan
025	sürekliliği
026	süreci veya sonucu
027	tamamlayıp
028	maruz kalmış
029	yaptırması
030	yarayan araç
031	özelliğini mecazi olarak taşıyan
032	yaparak veya maruz kalarak edinilen nitelik veya kişi
033	niteliği taşıyan veya alışkanlık haline getiren
034	yapmaya yarayan alet
035	ilgili nesne,şey
036	sonucu
037	olmaması özelliğini taşıyan
038	yapılması olayı
039	yapan, yapabilen
040	birden, bir defada veya aralıklı tekrarlanmasını tanımlayan ad
041	yapmak için gerekli nesne
042	maruz kalması
043	adı
044	sonucunda oluşan
045	sonucunda oluşan nesne
046	maruz kalan şey
047	karşılıklı yapılması veya birlikte yapılması
049	adı

RTN	Description
050	elde edilen nesne
051	için gereken araç
052	olumsuzluk
053	yapılarak oynanan oyun, oylanma, sürdürülen iş
054	maruz kalabilen araç,nesne
055	kendi kendine yaptıran hayvan
056	yapmaksızın (yapmadan)
057	yapmak için gerekli nesne
058	yaparak çalışan nesne
059	sonucu oluşan
060	yapan kişi,nesne
061	oluşan veya bu eylem olmuşcasına bir özellik taşıyan nesne
062	yapmış olma özelliği taşıyan
063	özel kalıplaşmış anlam türemiş
064	yapan nesne
065	yumuşakça veya kesintili yapması
066	alışkanlık haline getiren
067	yatkın, yapan
068	yapılma zamanı
069	yatkın
070	yapabilen nesne
071	gerçekleştirmesi veya neden olması
072	yapılmasına yarayan yer, nesne, araç
073	yapılabilen nesne
074	sonucunda çıkan nesne
075	veya duygusu verme durumunda olma
076	durumunda olma
077	yapıldığında
078	bitirilişinden beri/sonraki zaman
079	bir nesneye uygulaması
080	görülen geçmiş zamanda
081	duyulan geçmiş zamanda
082	geniş zamanda
083	yapmakta
084	gelecek zamanda
085	bir koşul veya dilek olarak

RTN	Description
086	isteniyor
087	gerekliliği
088	emrediliyor
089	hikaye
090	rivayet
091	koşul olarak
092	1.tekil kişinin
093	2.tekil kişinin
094	3.tekil kişinin
095	1.çoğul kişinin
096	2.çoğul kişinin
097	3.çoğul kişinin
098	geniş zamanda
101	yapabilmesi
102	sürdürmesi
103	başka hiçbir eylem olmaksızın sürdürmesi
104	mümkün olduğunca kısa sürede yapması
105	yapma durumuna çok kısa bir süre girip / yaklaşıp yapmaması
106	aşırıya kaçmadan yapması
107	Bir olay veya durum sırasında veya önce yapılmış olması
108	yapıldığı sırada
109	Bir olay veya durumun bağlı olarak tekrarlanması
110	soru
111	özelliği taşıyan
112	maruz kalması veya kendi kendine yapması, eylemden kendinin etkilenmesi
113	oluşmasını sağlayan nesne
115	kullanılan alet
116	sonucu
117	maruz kalan,yer,nesne,şey
118	yapmaya yarayan araç
119	uğraşan kişi
120	durumu
121	sonucu oluşan yapı veya şekil

RTN	Description
122	için gerekli nesne
123	tanımlayan ad
124	yapılması sonucunda oluşan nesne
125	yapılması sürecinin adı
126	yapılmasında kullanılan nesne
127	elde edilen nesne
128	işi olan
129	için gerekebilen şey

D.2 DESCRIPTION OF THE RETURN NUMBERS

RTN

001 -a/-e
Kökün bildirdiği eylemi yaparak ulaşılan sonuç, nesne, yer. (yar-a)

002 -aç/-eç
Kökün bildirdiği eylemi yapan veya yapmak için kullanılan nesne. (tut-aç)

003 -acak/-ecek
Kökün bildirdiği eylemde kullanılan nesne. (yi-y-ecek)

004 -acak/-ecek
Kökün bildirdiği eylemi yapacak olan nesne. (aç-ı-l-acak)

005 -ağan/-eğen
Kökün bildirdiği eylemi alışkanlık haline getiren, aşırı veya devamlı yapan veya olan. (ol-ağan)

006 -ak/-ek
Kökün bildirdiği eylemi genellikle yapan kişi. (kork-ak)

007 -ak/-ek
Kökün bildirdiği eylemin yapıldığı yer. (yat-ak)

008 -ak/-ek
Kökün bildirdiği eylemi yapmaya yarayan alet. (kay-ak)

RTN

009 -amak/-emek
Kökün bildirdiği eylemden türemiş yalın isim. (kaç-amak)

010 -akla/-ikle
Kökün bildirdiği eylemi olası aralıklarla belli bir süreyle yapması veya devam ettirmesi. (dur-akla)

011 -ala/-ele
Kökün bildirdiği eylemi belli bir süreyle tekrarlaması. (kov-ala)

012 -alak/-elek
Kökün bildirdiği eylemi yapmak zorunda olan veya alışkanlık haline getiren nesne. (yat-alak, as-alak)

013 -am/-em
Kökün bildirdiği eylem sonunda elde edilen nesne. (tut-am).

014 -amaç/-emeç
Kökün bildirdiği eyleminin yapıldığı yer veya araç. (dön-emeç).

015 -amak/-emek
Kökün bildirdiği eylemin yapıldığı yer veya araç. (tut-amak)

016 -an/-en
Kökün bildirdiği eylemi yapan. (gül-en)

017 -anak/-enek
Kökün bildirdiği eylemin sürekliliğinin veya bulunmasının sonucu. (sağ-anak, gel-enek)

018 -ar/-er/-ır/-ir/-ur/-ür
Kökün bildirdiği eylemi yapan kişi, alet, nesne. (kes-er)

019 -ca/-ce/-ça/-çe
Kökün bildirdiği eylemin nedeni. (gerek-çe)

020 -ası/-esi
Kökün bildirdiği eylemin olması veya maruz kalması istenilen. (ol-ası)

RTN

021 -baç/-beç
Kökün bildirdiği eylemi gerektiren yer veya nesne. (dolan-baç)

022 -ştır/-ştir
Kökün bildirdiği eylemin sınıflandırmaya dayalı olarak yapılması. (ara-ştır)

023 -ca/-ce/-ça/-çe
Kökün bildirdiği eylemi sağlayan veya eylemin sonucu. (düşün-ce)

024 -ç; -gaç/-geç
Kökün bildirdiği eylem duygusu olan, taşıyan,doğuran; eyleme neden olan veya sonucu olan. (kıskan-ç; utan-gaç)

025 -cama/-ceme
Kökün bildirdiği eylemin sürekliliği. (sürü-n-ceme)

026 -ay/-ey/-y
Kökün bildirdiği eylem süreci veya sonucu. (ol-ay, dene-y)

027 -ıp/-ip/-up/-üp
Kökün bildirdiği eylemi tamamlayıp. (koş-up)

028 -dık/-dik/-duk/-dük
Kökün bildirdiği eyleme maruz kalmış. (tanı-dık)

029 -dır/-dir/-dur/-dür
Kökün bildirdiği eylemi yaptırması. (ye-dir)

030 -gaç/-geç
Kökün bildirdiği eyleme yarayan araç. (süz-geç)

031 -arı/- eri
Kökün bildirdiği eylemin özelliğini mecazi olarak taşıyan. (uç-arı)

032 -gın/-gin/-gun/-gün
Kökün bildirdiği eylemi yaparak veya maruz kalarak edinilen nitelik veya kişi. (gez-gin, bil-gin)

RTN

033 -gan/-gen
Kökün bildirdiği eylemin niteliğini taşıyan veya kök-meyi alışkanlık haline getiren. (alın-gan)

034 -gı/-gi/-gu/-gü
Kökün bildirdiği eylemi yapmaya yarayan alet. (bur-gu)

035 -gı/-gi/-gu/-gü
Kökün bildirdiği eylemle ilgili nesne,şey. (duy-gu,ver-gi)

036 -gı/-gi/-gu/-gü
Kökün bildirdiği eylemin sonucu. (ser-gi,çiz-gi)

037 -maz/-mez
Kökün bildirdiği eylemin olmaması özelliğini taşıyan. (çık-maz)

038 -gın/-gin/-gun/-gün
Kökün bildirdiği eylemin yapılması olayı. (yan-gın)

039 -ıcı/-ici/-ucu/-ücü
Kökün bildirdiği eylemi yapan, yapabilen. (koş-ucu, ak-ıcı)

040 -ım/-im/-um/-üm
Kökün bildirdiği eylemin birden, bir defada veya aralıklı tekrarlanmasını tanımlayan ad. (bak-ım)

041 -ım/-im/-um/-üm
Kökün bildirdiği eylemi yapmak için gerekli nesne. (ak-ım)

042 -l
Öznenin kökün bildirdiği eyleme maruz kalması. (ye-dir-i-l)

043 -iş/-ış/-uş/-üş
Kökün bildirdiği eylemin adı. (bak-ış)

044 -k
Kökün bilrdirdiği eylemin sonucunda oluşan. (aç-ı-k)

045 -k
Kökün bildirdiği eyleminin sonucunda oluşan nesne. (tükür-ük)

RTN

046 -k
Kökün bildirdiği eyleme maruz kalan şey. (ada-k)

047 -iş/-ış/-uş/-üş
Kökün bildirdiği eylemin karşılıklı veya birlikte yapılması. (vur-uş, koş-uş)

049 -ma/-me; -mak/-mek
Kökün bildirdiği eylemin adı. (oku-ma)

050 -ma/-me
Kökün bildirdiği eylemle elde edilen nesne. (dol-ma)

051 -ma/-me
Kökün bildirdiği eylem için gereken araç. (kaz-ma)

052 -ma/-me
Kökün bildirdiği eylemin olumsuzu. (ye-me)

053 -maca/-mece
Kökün bildirdiği eylemi yaparak oynanan oyun, oyalanma, sürdürülen iş. (bil-mece)

054 -maca/-mece
Kökün bildirdiği eyleme maruz kalabilen araç,nesne. (çek-mece)

055 -maca/-mece
Kökün bildirdiği eylemi kendi kendine yaptıran hayvan. (at-maca)

056 -madan/-meden
Kökün bildirdiği eylemi yapmaksızın. (yap-madan)

057 -mak/-mek
Kökün bildirdiği eylemi yapmak için gerekli nesne. (ye-mek)

058 -mak/-mek
Kökün bildirdiği eylemi yaparak çalışan nesne. (çak-mak)

059 -man/-men
Kökün bildirdiği eylem sonucu oluşan. (şiş-man)

RTN

060 -man/-men
Kökün bildirdiği eylemi yapan kişi, nesne. (öğret-men)

061 -mık/-mik/-muk/-mük; -van/-ven
Kökün bildirdiği eylemle oluşan veya bu eylem olmuşcasına bir özellik taşıyan nesne. (kıy-mık; yay-van)

062 -mış/-miş/-muş/-müş
Kökün bildirdiği eylemi yapmış olma özelliği taşıyan. (oku-muş)

063 -mış/-miş/-muş/-müş; -sak/-sek; t
Özel kalıplaşmış anlam. (dol-muş, ye-miş; tut-sak; yoğur-t)

064 -mur/-mür
Kökün bildirdiği eylemi yapan nesne. (yağ-mur)

065 -p
Kökün bildirdiği eylemin yumuşakça veya kesintili yapılması. (ser-p)

066 -pak/-pek
Kökün bildirdiği eylemi alışkanlık haline getiren. (kay-pak)

067 -sal/-sel
Kökün bildirdiği eyleme yatkın, yapan. (uy-sal)

068 -sı/-si/-su/-sü
Kökün bildirdiği eylemin yapılma zamanı. (yat-sı)

069 -sı/-si/-su/-sü
Kökün bildirdiği eyleme yatkın. (sin-si)

070 -sı/-si/-su/-sü
Kökün bildirdiği eylemi yapabilen nesne. (tüt-sü)

071 -t
Kökün bildirdiği eylemi gerçekleştirmesi veya neden olması. (gevşe-t)

RTN

072 -t
Kökün bildirdiği eylemin yapılmasına yarayan yer, nesne, araç. (taşı-t)

073 -t
Kökün bildirdiği eylemi yapılabilen nesne. (um-u-t)

074 -t
Kökün bildirdiği eylemin sonucunda çıkan nesne. (yaz-ı-t)

075 -tı/-ti/-tu/-tü
Kökün bildirdiği eylem (veya duygusu verme) durumunda olma. (ürper-ti)

076 -ntı/-nti/-ntu/-ntü
Kökün bildirdiği eylem durumunda olma. (gez-i-nti)

077 -ince/-ınca/-unca/-ünce
Kökün bildirdiği eylem yapıldığında. (gel-ince)

078 -alı/-eli
Kökün bildirdiği eylemin bitirilişinden beri/sonraki zaman. (bitir-eli)

079 -ır/-ir/-ur/-ür
Kökün bildirdiği eylemin bir nesneye uygulanması. (bat-ır)

080 -di/-dı/-du/-dü/-ti/-tı/-tu/-tü
Kökün bildirdiği eylemin görülen geçmiş zamanda yapıldığı bildiriliyor. (gel-di)

081 -miş/-mış/-muş/-müş
Kökün bildirdiği eylemin duyulan geçmiş zamanda yapıldığı bildiriliyor. (gel-miş)

082 -r
Kökün bildirdiği eylemin geniş zamanda yapıldığı bildiriliyor. (gel-i-r)

083 -yor
Kökün bildirdiği eylemin yapılmakta olduğu bildiriliyor. (gel-i-yor)

RTN

084 -acak/-ecek
Kökün bildirdiği eylemin gelecek zamanda yapılacağı bildiriliyor. (gel-ecek)

085 -sa/-se
Kökün bildirdiği eylemi yapması bir koşul veya dilek olarak bildiriliyor. (gel-se)

086 -a/-e
Kökün bildirdiği eylem isteniyor. (gel-e)

087 -malı/-meli
Kökün bildirdiği eylemi yapmasının gerekliliği bildiriliyor. (gel-meli)

088 -emir
Kökün bildirdiği eylem emrediliyor. (gel)

089 -di/-dı/-du/-dü/-ti/-tı/-tu/-tü
Kökün bildirdiği eylem hikaye ediliyor. (geliyor-du)

090 -miş/-mış/-muş/-müş
Kökün bildirdiği eylem rivayet ediliyor. (geliyor-muş)

091 -sa/-se
Kökün bildirdiği eylem koşul olarak bildiriliyor. (geliyor-sa)

092 Kökün bildirdiği eylemin 1.tekil kişi tarafından yapılması.

093 Kökün bildirdiği eylemin 2.tekil kişi tarafından yapılması.

094 Kökün bildirdiği eylemin 3.tekil kişi tarafından yapılması.

095 Kökün bildirdiği eylemin 1.çoğul kişi tarafından yapılması.

096 Kökün bildirdiği eylemin 2.çoğul kişi tarafından yapılması.

097 Kökün bildirdiği eylemin 3.çoğul kişi tarafından yapılması.

098 -maz/-mez; -amaz,emez
geniş zamanın olumsuzu. (gelemez)

101 (a/-e)bil
Kökün bildirdiği eylemi yapabilmek. (yazabil)

RTN

102 (a/-e)dur;(a/-e)gel
Kökün bildirdiği eylemi sürdürmesi. (bakadur)

103 (a/-e)kal; (a/-e)koy
Kökün bildirdiği eylemini başka hiçbir eylem olmaksızın sürdürmek. (bakakal)

104 (ı/-i/-u/-ü)ver
Kökün bildirdiği eylemini mümkün olduğunca kısa sürede yapmak. (geliver)

105 (a/-e)yaz
Kökün bildirdiği eylemi yapma durumuna çok kısa bir süre girip/-yaklaşıp yapmamak. (koşayaz)

106 -ımsa/-imsa/-umsa/-ümse
Kökün bildirdiği eylemi aşırıya kaçmadan yapmak. (gül-ümse)

107 -arak/-erek
Bir olay veya durum sırasında veya önce kökün bildirdiği eyleminin yapılmış olması. (anla-y-arak)

108 -kan/-ken
Kökün bildirdiği eylem sırasında. (geliyor-ken)

109 -dıkça/-dikçe/-dukça/-dükçe
Bir olay veya durumun kökün bildirdiği eyleme bağlı olarak tekrarlanması. (gel-dikçe)

110 -mi/-mı/-mu/-mü
soru eki

111 -aç/-eç
Kökün bildirdiği eylemin özelliğini taşıyan. (gül-eç)

112 -n
Öznenin kökün bildirdiği eyleme maruz kalması veya kendi kendine yapması, eylemden kendinin etkilenmesi. (yıka-n)

RTN

113 -v
Kökün bildirdiği eylemin oluşmasını sağlayan nesne. (öde-v)

115 -ga/-ge
Kökün bildirdiği eylemde kullanılan alet. (süpür-ge)

116 -ga/-ge
Kökün bildirdiği eylemin sonucu. (öner-ge)

117 -ga/-ge
Kökün bildirdiği eyleme maruz kalan, yer, nesne, şey. (sömür-ge)

118 gıç/-giç/-guç/-güç
Kökün bildirdiği eylemi yapmaya yarayan araç. (patlan-gıç)

119 -gıç/-giç/-guç/-güç
Kökün bildirdiği eylemle uğraşan kişi. (dal-gıç)

120 -gıç/-giç/-guç/-güç
Kökün bildirdiği eylemin durumu. (başlan-gıç)

121 -ı/-i/-u/-ü
Kökün bildirdiği eylem sonucu oluşan yapı veya şekil. (yap-ı, diz-i)

122 -ı/-i/-u/-ü
Kökün bildirdiği eylemi için gerekli nesne. (çak-ı)

123 -ı/-i/-u/-ü
Kökün bildirdiği eylemi tanımlayan ad. (koş-u)

124 -n
Kökün bildirdiği eylemin yapılması sonucunda oluşan nesne. (yığ-ı-n)

125 -n
Kökün bildirdiği eylemin yapılması sürecinin adı. (ak-ı-n)

126 -n
Kökün bildirdiği eylemin yapılmasında kullanılabilen nesne. (tüt-ü-n)

RTN

127 -maç/-meç
Kökün bildirdiği eylemle elde edilen nesne, şekil. (bula-maç)

128 -maç/-meç
İşi kökün bildirdiği eylem olan. (çığırt-maç)

129 -maç/-meç
Kökün bildirdiği eylem için gerekebilen şey. (de-meç)

Appendix E

GLOSSARY

English		Turkish
Acceleration	:	Tezlik
Active Verb	:	Etken Fiil
Agglutinative Languages	:	Bitişken Diller
Approximation	:	Yaklaşma
Causative Verb	:	Oldurgan Fiil
Compound Imperfect Tense	:	Hikaye Bileşik Zaman
Compound Tense	:	Bileşik Zaman
Compound Verb	:	Bileşik Fiil
Conditional	:	Şart
Conjugation of Verbs	:	Fiil Çekimi
Consonant Harmony	:	Sessiz Uyumu
Continuence	:	Sürerlik
Declension of Nouns	:	İsim Çekimi
Denominal Word	:	İsim Soylu Kelime
Derivational Affix	:	Yapım Eki
Deverbal Word	:	Fiil Soylu Kelime
Factitive Verb	:	Ettirgen Fiil
Future Tense	:	Gelecek Zaman
Imperative	:	Emir
Inflectional Affix	:	Çekim Eki

English		Turkish
Intransitive Verb	:	Geçişsiz Fiil
Morphology	:	Biçimbilgisi
Narrative Compound Tense.....	:	Rivayet Bileşik Zaman
Necessitative	:	Gereklilik
Optative	:	İstek
Passive Verb	:	Edilgen Fiil
Personal Ending	:	Şahıs Eki
Possessive Suffix	:	İyelik Eki
Potentiality and Possibility	:	Yeterlik
Progressive Present	:	Şimdiki Zaman
Reciprocal Verb	:	İşteş Fiil
Reflexive Verb	:	Dönüşlü Fiil
Remote Condition	:	Dilek-Şart
Reported Simple Past	:	Duyulan Geçmiş Zaman
Root	:	Kök
Simple Present	:	Geniş Zaman
Stem	:	Gövde
Suffix	:	Sonek
Tense Suffix	:	Zaman Eki
Transitive Verb	:	Geçişli Fiil
Unvoiced Consonant	:	Sert Sessiz
Voiced Consonant	:	Yumuşak Sessiz
Vowel Harmony	:	Sesli Uyumu
Witnessed Simple Past	:	Görülen Geçmiş Zaman