DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MORPHOLOGICAL ANALYSIS IN NATURAL LANGUAGE PROCESSING FOR TURKISH LANGUAGE AND A NEW APPROACH FOR LEXICON DESIGN

by

Emel ALKIM

August, 2006

İZMİR

MORPHOLOGICAL ANALYSIS IN NATURAL LANGUAGE PROCESSING FOR TURKISH LANGUAGE AND A NEW APPROACH FOR LEXICON DESIGN

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program

by
Emel ALKIM

August, 2006
İZMİR

M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MORPHOLOGICAL ANALYSIS IN NATURAL LANGUAGE PROCESSING FOR TURKISH LANGUAGE AND A NEW APPROACH FOR LEXICON DESIGN"** completed by **Emel ALKIM** under supervision of **Prof.Dr. Tatyana YAKHNO** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof.Dr. Tatyana YAKHNO

Supervisor

Yrd. Doç.Dr. Latif SALUM

(Jury Member)

Yrd.Doç.Dr. Adil ALPKOÇAK

(Jury Member)

Prof.Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank Prof.Dr. Tatyana YAKHNO, my supervisor for all her support and patience she gave on me.

I also want to thank to my family, and friends; without their endless support I would never finish this thesis.

Emel ALKIM

MORPHOLOGICAL ANALYSIS IN NATURAL LANGUAGE PROCESSING FOR TURKISH LANGUAGE AND A NEW APPROACH FOR LEXICON DESIGN

ABSTRACT

The main motivation of this thesis is creating a system which can get text as input and build a knowledge structure of it. Thus the system will be able to simulate human beings' knowledge base. Additionally the system will be able to answer questions on the knowledge base.

The system is designed as a shell and has user interfaces for maintaining and extending lexicon and morphological rules. Thus the system is easily extendible and in one point of view, it is language independent. A linguist capable of supplying all information on the language can develop a system specific for that language. Note that; as the design of the system is done on an agglutinative language, some extensions on the software may be needed for the execution of grammar rules on different types of languages.

Keywords: Natural Language Processing, Categorical Lexicon, Morphological Analysis, Agglutinative Language

TÜRKÇE İÇİN DOĞAL DİL İŞLEMEDE BİÇİMBİRİMSEL ÇÖZÜMLEME VE SÖZLÜK TASARIMI İÇİN YENİ BİR YÖNTEM

ÖZ

Bu tezin temel motivasyonu girdi olarak metin alan ve bu metinden bir bilgi yapısı oluşturan bir sistem yaratmaktır. Böylece sistem insanların bilgi tabanını taklit edebilecektir. Ek olarak sistem bu bilgi tabanına dair soruları yanıtlayabilecektir.

Sistem bir kabuk olarak tasarlanmış olup; sözlük ve biçimbirim kurallarını değiştirip geliştirmeye olanak sağlayan arayüzlerle desteklenmiştir. Böylece kolaylıkla geliştirilebilir ve bir bakış açısından dilden bağımsız bir sistem oluşturulmuştur. Yani bir dil hakkındaki bütün bilgiyi sağlayabilen bir dilbilimci o dile özgü sistemi yaratabilir. Belirtmek gerekir ki; sistem tasarımı eklemeli bir dil temel alınarak yapıldığı için, farklı tipteki bir dilin söz dizim kurallarının işlenebilmesi için bazı yazılımda bazı geliştirmeler gerekebilir.

Anahtar Kelimeler : Doğal Dil İşleme, Kategorisel Sözlük, Biçimbirimsel Çözümleme, Eklemeli Dil

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

Natural Language Processing is a promising branch of Artificial Intelligence which is studied on since the first times of artificial intelligence studies. The most interesting property of Natural Language Processing is that it is done very easily by human beings whereas it is very hard to mimic for computers.

Natural language is an important area to study on and can be the key to understanding the intelligence, as the connection to the intelligence is achieved by natural language. Additionally, any single successful parsing of some natural language text would include some kind of knowledge processing. Thus the work that is needed to be done is very complicated and important.

The motivation of this thesis is to achieve a computer program which can retrieve text as input, construct some kind of knowledge base from it and answer questions on that text.

Thus, the creation of the knowledge base would show us the road to mimic human knowledge base. When a human being hears a speech, the natural language input is reflected to the brain by related sensors. Parsing can be named as “Making the connections of the input to the previous knowledge that is located in the brain”. Thus the input gathers meaning that can be useful.

The input of a NLP system is either text or speech, whereas the output must be of a structural representation. The structure of the language should also be specified briefly for constructing the output from the input.

Turkish is the selected natural language to be the design and the development base natural language of this system. Turkish is an agglutinative language, such as Finnish, Hungarian, Quechua and Swahili. An agglutinative language is a morphologically complex language where the words are formed by adding suffixes

to roots. Thus the words can be relatively long and contain so much information in it which may be equivalent to a sentence.

Although there are studies based on statistical approaches which omit morphological analysis phase, most natural language processing studies those intend to work on Turkish should apply some level of morphological analysis to extract the appropriate information from the word as it is an agglutinative language.

1.1. Outline of the Thesis

This thesis documentation consists of seven parts: The second part gives brief introduction to the background of natural language processing, some detailed information on morphological analysis and the related work in addition to the definitions needed for better understanding of the subject.

The third chapter defines the materials and effectors of the system. The methods used for achieving the results are stated in terms of algorithm and the design issues considered.

The fourth chapter defines the system formally, with the formal descriptions of important concepts of the system.

The fifth chapter gives brief information on the characteristics of Turkish language. General information about the morphological characteristics of Turkish is supported in addition to the list of rules that are used by the analyzer.

The sixth chapter discusses the results achieved and evaluates the system performance in levels of the algorithm and data achieved.

The seventh chapter concludes the documentation by stating the roadmap of the study.

There are four appendices of the documentation:

- the user guide of the system,
- the graph illustration of the lexicon with samples of entities (not full list),
- class diagrams of the system,
- entity relationship (ER) diagrams of the database.

The user guide helps the user to use the system by screenshots and explanation of the way that must be followed during design and altering of a system.

The graphical representations of the lexicon are listed just to achieve understanding of the general structure of the lexicon.

The class diagrams of the system are supplied in the form of Unified Modelling Language (UML) Diagrams to achieve better understanding of the software structure.

Entity relationship diagram for the database of the lexicon is illustrated in the last appendix. Also the field lists of all tables are supplied to achieve better understanding of database structure.

CHAPTER TWO
BACKGROUND

Natural language processing is an interdisciplinary working subject which gathers attention from five different domains.

Linguists: want to understand languages.

Psychologists: want to understand the processes of language comprehension and production.

Philosophers: want to know how rational thought relates to language; how words come to mean things.

AI Practitioners: want to develop models of human reasoning, involving language processing.

Computer Scientists: want to develop better applications that involve the processing of natural languages.

2.1. Levels of NLP

Natural language processing is considered to be achieved in five levels which are defined briefly in the following pages.

2.1.1. Phonetics

Phonology is the study of the sound structure of language. Sounds are organized into a system of contrasts, and analyzed in terms of phonemes, distinctive features, or other such phonological units according to the theory used. A phoneme is the minimal unit of the sound system of a language. Some languages have as few as 15; others have as many as 80. No two languages have the same system of phonemes. Distinctive features are used either to define phonemes or as an alternative to the notion of phoneme. Example pairs include +nasal and -nasal, and +voice (voiced) and -voice (voiceless). Nasal sounds are produced when there is complete closure in the mouth and all the air thus escapes through the nose, as in the ‘n’ sound of ‘nasal’.

Voiced sounds are produced while the vocal cords are vibrating, e.g., the ‘b’ sound in ‘bahçe’; voiceless or unvoiced sounds are produced when there is no such vibration, as in the ‘p’ sound of ‘kitap’.

Problems of phonology include the ratio of noise to data, the varying speech rates within and across individuals, and coarticulation. (SFU, n.d.)

2.1.2. Morphology

Morphology is the study of the structure of words. Morphology attempts to find patterns and rules in the way that affixes are used. Using these rules and patterns, a computer can check that words given as input are real words, and have been used correctly, without resorting to storing a huge list of words and each of their uses. (AIDEPOTa, n.d.)

A major morphological problem is ambiguity: the suffix ‘I’, for example, can indicate third singular possessive or accusative of a noun. Another problem is exceptions, for example, the third singular possessive of the noun ‘kitap’ is ‘kitabı’ (not ‘kitapı’).

Morphology level will be described in detail in the following pages.

2.1.3. Syntax

Syntax is the study on the structure of sentences.

Once the building blocks of a sentence have been determined to be correct using morphology, syntax can be used to check they are properly combined. (AIDEPOTb, n.d.)

The most widely used method of checking a sentence is syntactically correct is to attempt to use a *grammar* to build up a *parse tree*.

A grammar is a list of rules that turn symbols into words, and a parse tree is simply a diagramatical way of showing the use of these rules to build up a sentence.

For example the following grammar starts at the symbol **S** and produces very simple sentences.

S ==> V N
N ==> *this*
N ==> *that*
V ==> *do*

These rules show that the symbol **S** should be rewritten, or expanded, to the symbols **V** and **N** (in that order). These symbols can, in turn, be rewritten to the right hand sides of their rules. This process repeats until we have only *terminals* (words) in our sentence.

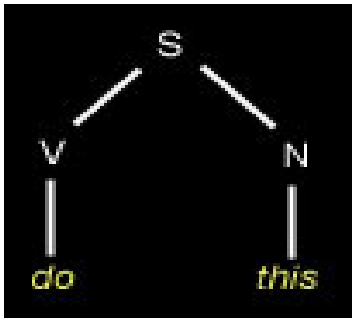

Figure 2.1 Parse tree for 'do this'

A parse tree for the sentence *"do this"* would be as shown in Figure 2.1, wheras the sentence *"stop that"* has no parse tree (according to the grammar we are using) and is therefore syntactically incorrect.

This stage of NLP is often simply termed **parsing** since its main aim is simply to build a parse tree. There are two main classes of parsing; *top-down* and *bottom-up*, each having its advantages and disadvantages.

Top down parsers start with the *start symbol* of the grammar and try different combinations of rules until a sequence of rules is found that generate the sentence being parsed.

Bottom up parsers start with the sentence and looks to find a sequence of rules which could generate the list of words in question.

The problems with parsing become apparent when there is more than one choice of rules to expand in order to create a sentence. This is often termed *ambiguity* and poses a significant nightmare to many NLP applications; given a sentence with many possible meanings, how do you determine the intended one.

2.1.4. Semantics

Semantics is the study of what words, sentences, etc., mean. (Actually, the distinction between syntax and semantics is a source of great controversy in linguistics. Some approaches (e.g., “Generative Semantics”) do not consider them to be distinct. Some considers them to be completely separate, with syntax being primary. Probably most folks today consider them to be separate, but with a blurry dividing line, and with each affecting the other.) The extraction of the semantics can be stated as:

sentence structure + word meaning -> sentence meaning

Once a parse tree has been built using syntactic analysis, the semantics, or meaning, of the sentence needs to be found. This is a very difficult task as it requires *world knowledge*. (AIDEPOTc, n.d.)

The need for semantic information can be illustrated by trying to follow a recipe. Knowing that the recipe is correctly written does not help understand how to carry it out. Knowledge of the relationships between food and it's preparation, the semantics contained with the recipe, is needed.

Another way of putting the difference between syntax and semantics into context is with the following two sentences:

"Colourless green ideas sleep furiously"
"Green sleep colourless furiously green"

The first sentence is syntactically correct in modern English, but after semantic analysis can be seen to be meaningless, whereas the second sentence, although made from the same words, can be dismissed as nonsense by simply regarding it's syntax.

There are no steadfast ways of representing world knowledge or extracting the semantics from a sentence except for very simple domains, although First Order Predicate Calculus seems to be dominating as a representation method.

2.1.5. Pragmatics

Pragmatics is the study about how language is used. The extraction of the pragmatics can be stated as:

sentence meaning + context → more precise meaning

The easiest way to view the task of **pragmatics** is as determining an action given the semantics of a request or sentence. (AIDEPOTd, n.d.)

Since finding the semantics of a sentence is extremely difficult, determining the pragmatics is even harder.

There is as much ambiguity in this area of NLP as any other, if not more. For example, consider the following question posed to a computer:

"Do you have the time?"

A *syntactically* correct answer to this question could be *"Yes"* although this is clearly not *pragmatically* correct as the answer required is the actual time.

To enable a computer to perform NLP "properly" it must be given enough world knowledge (it must know everything about everything), or else it must be given the ability to learn. Much current research is centered around giving computers the ability to learn this world knowledge and therefore help make the semantic and pragmatic stages of NLP more feasible.

2.1.6. Discourse

Discourse is the study of how terms actually occur and co-occur in large corpora; World knowledge: all the general knowledge we have.

Discourse understanding includes perception, analysis (and thus syntactic, semantic and pragmatic interpretation), disambiguation and incorporation. (Russell & Norvig, 1997)

Discourse is about the higher level relations that hold among sequences of sentences in a discourse or a narrative. It merges sometimes with literary theory, but also with pragmatics. (UCSD, n.d.)

One thing to understand is that different sentences do different kinds of "work" in a discourse. We have seen some examples of this already -- noun phrases that refer to new entities, or back to previously introduced ones. Same for whole sentences: Some introduce new events or relations; some used them to introduce something new.

“A car began rolling down the hill”
“It collided with a lamppost.”

One important idea in discourse theory is the idea that much language is performed in the context of some mutual activity. For example two people could be

working on some project together. In this case, they are probably both somewhat aware of the plan that they are both following, and so much of the pragmatic information needed to understand what they are talking about can be thought of in terms of that plan. And sometimes utterances can be understood as if they were steps in the execution of a plan. For example:

“please pass the salt”

This could be thought of as a way to get me the salt, if having salt was part of a plan.

Some people think of sentences like:

“can you pass the salt”

As "indirect speech acts" because they look like questions, but aren't really. One way to think about sentences like this is that the hearer understands that this is probably not a question, but is a conventionalized (and polite) means of asking for the salt.

Another analysis of this sort of sentence is that you are trying to avoid rejection. You do this by considering ways that your plan might fail. So you don't want to have this happen:

“please pass the salt”
“I can't, I'm tied up with ropes.”
“oh, sorry.”

So you ask about potential problems first -- asking about ability. So that if there is a problem, you don't have to ask directly and you won't be rejected. It is sort of like:

"Are you doing anything saturday night?"
"Yes, I'm feeding my goldfish"

So you don't have to be rejected if you actually ask for a date.

2.2. Morphological Analysis in Detail

2.2.1. Models of Morphology

There are three different models of morphology each based on a major of approach to morphology.

2.2.1.1. Morpheme-based Morphology.

Morpheme-based morphology makes use of an Item and Arrangement approach (Hockett, 1954).

In morpheme-based morphology, word forms are analyzed as sequences of morphemes. Lexicon contains a list of all possible stem variants, together with rules which state the distribution of each variant. A morpheme is defined as the minimal meaningful unit of a language. This way of analyzing word forms as if they were made of morphemes put after each other like beads on a string, is called Item-and-Arrangement.

All morphemes are lexical items meaning that just as the root 'çay' (tea) is put in the lexicon, the suffix '–CI' (derives the word 'çaycı' (one who prepares and sells tea) when attached to the root) has to be put in the lexicon.

However, applying such a model rigorously, quickly leads to complications with many forms of allomorphy. Thus, theorists who wish to maintain a strict morpheme-based approach often preserve the idea in cases like these by defining the character changes by morphophonemics.

In a classic Item and Arrangement theory, a word is built up by addition of morphemes, each of which contributes a distinct meaning to the complex word; the relationship between form and meaning is presumed most often to be one-to-one.

2.2.1.2. Lexeme-based morphology.

Lexeme-based morphology normally makes use of an Item-and-Process (Hockett, 1954) approach.

Instead of analyzing a word form as a set of morphemes arranged in sequence, a word form is considered as the result of applying rules that effect changes to word forms and stems. An inflectional rule takes a stem, does some changes to it, and outputs a word-form; a derivational rule takes a stem, and outputs a derived stem; a compounding rule takes word-forms, and outputs a compound stem.

The Item-and-Process approach bypasses the difficulty described above for Item-and-Arrangement approaches. A single underlying form exists with alternating allomorphs. Surface forms are derived from the application of feature changing rules to the underlying form.

Item and Process theorists look at word formation as the operation of processes or rules on base morphemes or words, each rule adding to or changing the form of the base, and concomitantly having some characteristic semantic or morphosyntactic effect; but again, the relationship between process and semantic or morphosyntactic effect is typically one-to-one.

2.2.1.3. Word-based morphology.

Word-based morphology is a (usually) Word-and-Paradigm approach. The word (rather than the stem) is the core unit represented in the lexicon (Mathews, 1972). This kind of theory takes paradigms as a central notion. Instead of stating rules to combine morphemes into word forms, or to generate word-forms from stems, word-

based morphology states generalizations that hold between the forms of inflectional paradigms. The major point behind this approach is that many such generalizations are hard to state with either of the other approaches. The examples are usually drawn from fusional languages, where a given "piece" of a word, which a morpheme-based theory would call an inflectional morpheme, corresponds to a combination of grammatical categories, for example: "third person plural". Morpheme-based theories usually have no problems with this situation, since one just says that a given morpheme has two categories. Item-and-Process theories, on the other hand, often break down in cases like these, because they all too often assume that there will be two separate rules here, one for third person, and the other for plural, but the distinction between them turns out to be artificial.

The Word-and-Paradigm approach does not apply to word-formation. It maps semantic and morphosyntactic properties onto words in a many to one fashion.

2.2.2. Morpheme

Words are formed by combination of one or more free morphemes and zero or more bound morphemes. Morphemes are the smallest meaningful units in the grammar of a language.

Free morphemes are units of meaning which can stand on their own as words. Bound morphemes are also units of meaning; however, can not occur as words on their own: they can only occur in combination with free morphemes. Morphemes may be also classified, on the basis of word formation, characteristics into the following types (Loos, Anderson, Day, Jordan & Wingate, 2004):

- Root: Bound or free, made up of a single free morpheme; a basis for compounding and affixation.

- Stem: Bound or free, made up of one or more morphemes; a basis for affixation.

- Affix: Bound morphemes. There are six different types up to the joining location:
  - Prefix: occurs in the front of a root or stem,
  - Suffix: occurs at the end of a root or stem,
  - Infix: occurs inside of a root or stem,
  - Circumfix: occurs in two parts on both outer edges of a root or stem,
  - Simulfix: replaces one or more phonemes in the root or stem,
  - Suprafix: superimposed on one or more syllables in the root or stem as a suprasegmental.

- Clitic: Phonologically bound but syntactically free morphemes that has syntactic characteristics of a word, but shows evidence of being phonologically bound to another word. There are two types of clitics:
  - Proclitic: occurs at the beginning of a morpheme,
  - Enclitic: occurs at the end of a morpheme.

  (z-endings in Turkish "second plural person suffix" behave like clitics (Good & Yu, 1999)).

2.2.3. Morphological Rules

The constraints on how the morphemes are combined to form the words are defined by the morphological rules.

The linguistic rules for a morphological analyzer, which can be called the morphological rules, achieve the validity of the words. The morphological rules can be grouped in two types considering the characteristics and the application considerations. The rules that apply to the phoneme substitutions are named *Morphophonemics*, whereas the rules that ensures to the validity of the morpheme sequences are named *Morphotactics*.

2.2.3.1. Morphophonemics (Rules of Phoneme Substitution)

The morphophonemics are the morphological rules which check the validity of the word through the phonetic constraints. Morphophonemics can be considered as a sub level between phonology and morphology, and can be used to handle some of the exceptions which are mentioned at the beginning of this section.

They are at the phonology level as they apply phonological constraints. But they are also at the morphology level as they are very important for the morphological analysis process and the morphological structures are the affecters of the phonological constraints.

Thus the morphophonemics can be defined as rules on character substitutions that are affected by the morpheme addition.

A sample morphophonemic for Turkish language is "Final Stop Devoicing - Voicing" (See Consonant Harmony in Chapter Five):

When a voiced consonant at the end of a morpheme is followed by a vowel, the consonant becomes voiceless.

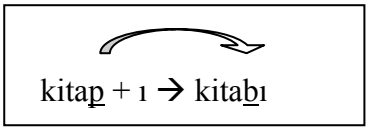

Figure 2.2 Morphophonemic sample on 'kitabı'

Figure 2.2 shows the sample morphophonemic applied to the noun 'kitap' (book); voiceless consonant 'p' at the end of the stem changes to the voiced match 'b' when it is followed by 'ı' (a vowel).

2.2.3.2. Morphotactics (Rules of Morpheme Sequences)

The morphotactics are the patterns describing how morphemes come together to form a valid word. Each morpheme must be a valid entity in the lexicon. The patterns define which entities that are member of specified categories can be combined together. Thus the morphotactics are ordered lists of identifiers of specific categories.

A sample morphotactic for Turkish language is "The Nominal Inflection Pattern" (See page 46).

Table 2.1 Morphotactic sample on 'kitabı'

nominal root	plural suffix (optional)	possessive suffix (optional)	case suffix (optional)	relative suffix (optional)
Kitap	-	I	-	-

Figure 2.2 shows the sample morphotactic applied to the word 'kitabı' (his/her book); the word is parsed and a nominal root and a possessive suffix are matched. The parse is valid as the unmatched components are optional.

2.3. Previous Work

The previous work on related domains will be explained briefly in the following of this chapter.

2.3.1. Morphological Analysis

In general, finite state or statistical methods are being used in the morphological analyzers.

Statistic used for:

- speech recognition
- part-of-speech tagging
- parsing
- machine translation, info retrieval, summarization

Symbolic reasoning used for:

- parsing
- semantic analysis
- generation
- MT, dialog management
- Hybrid approaches

2.3.1.1. Statistical Methods

Statistical natural language processing uses stochastic, probabilistic and statistical methods to resolve some of the natural language processing problems, especially those which arise because longer sentences are highly ambiguous when processed with realistic grammars, yielding thousands or millions of possible analyses. Methods for disambiguation often involve the use of corpora and Markov models. The technology for statistical NLP comes mainly from machine learning and data mining, both of which are fields of artificial intelligence that involve learning from data.

In artificial intelligence **stochastic** programs work by using probabilistic methods to solve problems, as in simulated annealing, neural networks and genetic algorithms. A problem itself may be stochastic as well, as in planning under uncertainty. A deterministic environment is much simpler for an agent to deal with.

In usage-based linguistic theories, where it is argued that competence, or langue, is based on performance, or parole, in the sense that linguistic knowledge is based on frequency of experience, grammar is often said to be probabilistic and variable rather than fixed and absolute. This is so, because one's competence changes in accordance with ones experience with linguistic units. This way, the frequency of usage-events determines one's knowledge of the language in question.

The application of **probability** is fundamental to the building of statistical forms out of data derived from samples. Such samples are chosen by predetermined and arbitrary selection of related variables and arbitrary selection of intervals for

sampling; these establish the degree of freedom. Many courses are given in statistical method. Elementary probability considers only finite sample spaces; advanced probability by use of calculus studies infinite sample spaces. The theory of probability was first developed (c.1654) by Blaise Pascal, and its history since then involves the contributions of many of the world's great mathematicians.

Statistical: We describe our knowledge (and ignorance) mathematically and attempt to learn more from whatever we can observe. This requires us to

1. plan our observations to control their variability (experiment design),
2. summarize a collection of observations to feature their commonality by suppressing details (descriptive statistics), and
3. reach consensus about what the observations tell us about the world we observe (statistical inference).

In some forms of descriptive statistics, notably data mining, the second and third of these steps become so prominent that the first step (planning) appears to become less important. In these disciplines, data often are collected outside the control of the person doing the analysis, and the result of the analysis may be more an operational model than a consensus report about the world.

Wicentowski (2004) introduces a statistical method, which applies four supervised and four unsupervised methods to 32 languages. Turkish is one of them with a lexicon of 25497 entities (with 87 verbs, 29130 verb inflections).

2.3.1.2. Finite State Methods

Finite State Transition Networks (FSTN): A Finite State Transition Network (FSTN) is a simple language model. A FSTN is neutral between recognition (analysis of input) and generation (producing output in the specified language).

Finite State Transducers (FST): Finite State Transducer (FST) is a variant of FSTN with pairs of labels on arcs, defining a mapping between input and output. FSTs can be used for translation.

Recursive Transition Networks (RTN): RTN can be regarded as a specification of a machine, a *pushdown automaton* (*PA*). A pushdown automaton is an FSA that is equipped with an extra memory, a *stack*. For an RTN, network traversal is defined partially in terms of itself. This is the reason for the word 'recursive' in recursive transition network.

Augmented Transition Networks (ATN): ATN consists of an RTN augmented by a set of tests to be satisfied before an arc was traversed and a set of registers that could be used to save intermediate results or global states.

Two-Level Morphology-KIMMO (Koskenniemmi, 1983): An alternate way of implementation to handle the underlying and surface form differentiation.

Two level model has an important place in computational morphology, yet there are some systems such as DECOMP which predates two level model by many years.

MIT's DECOMP module and Hankamer's Keçi system for Turkish can be stated as successive morphological analyzers with a finite state model of morphotactics.

2.3.1.3. Two Level Morphology

Koskenniemi (1983) introduced a model named "two-level morphology" or "KIMMO". Koskenniemi's model is "two-level" in the sense that a word is represented as a direct, letter-for-letter correspondence between its lexical or underlying form and its surface form. For example, the word *kitabım* is given this two-level representation (note that + is a morpheme boundary symbol and *0* is a null character):

Lexical form: k i t a p + I m

Surface form: k i t a b 0 ı m

PC-KIMMO, a language-independent morphological parser shell based on KIMMO, uses finite state machines to apply language rules. PC-KIMMO Version 2 also includes an unification grammar. The rules for Turkish are constructed by Kemal Oflazer and Türk Dil Kurumu (Oflazer, 1994), but the unification grammar is not developed. And also PC-KIMMO is a text-based, command prompted program in which it is really hard to add a new rule or create the grammar.

2.3.1.4. Other Tools for Computational Morphology

There are also works where the morphotactics are context free. One of them is AMPLE an earlier contribution of SIL. AMPLE models morphotactics with a version of categorical morphology. AMPLE has mechanisms that allow long distance dependencies between affixes and thus in principle have greater than finite state power. AMPLE has no direct model of phonological rules and it is therefore necessary to list all the surface forms in which a morpheme might occur.

General trends in the computational morphology literature can be summarized as follows (Öztaner, 1996):

Most morphological analyzers use finite state morphotactics inspite of the existence of some context free systems.

In the majority of systems all morphemes are considered to be dictionary entries. That is they use item and arrangement model.

2.3.2. Work on Turkish

Oflazer, Göçmen & Bozşahin (1994) proposed two interrelated Finite State Machines (FSMs); one for the nominal morphotactics of Turkish and another for the verbal morphotactics of Turkish. The FSMs are based on 22 two-level rules, derived from Turkish morphology. Öztaner (1996) proposed a more complete representation of Turkish morphology with 52 two-level rules.

Two-level rules describe a word with its lexical and surface forms. The lexical representation denotes the structure while the surface representation obeys orthographic rules. The FSMs are designed as parsers, not generators, with the assumption that the user enters legal inputs. The description has been implemented using the PC-KIMMO environment and is based on a root word lexicon of about 23.000 root words.

Oflazer's nominal and verbal FSMs were used in a programming project of the Natural Language Processing course in Boğaziçi University and the morphological part of a B.S. Thesis Project in the Computer Engineering Department of Boğaziçi University in both of which the FSMs were implemented in Prolog.

Another approach to morphological analysis was studied in a Ph.D. Dissertation in Boğaziçi University to implement a spelling checker and corrector. In that work, the morphological structure of Turkish is investigated and defined in terms of morphotactic rules that state the order of the suffixes and morphophonemic rules that state the form of the suffixes. The results of this analysis are used in implementing the Turkish morphological structure by using the Augmented Transition Network (ATN) formalism. A large lexicon of Turkish was constructed and used for the spelling checker part of the implementation. This lexicon is divided into the root lexicon, the suffix lexicon and the proper noun lexicon. The parser of the spelling checker is based on a root-matching algorithm. If the spelling checker could not parse the given input, the corrector part produces the candidate alternatives of the misspelled word, considering the transposition of two letters, one letter missing, extra letter or wrong letter.

The thesis also contains statistical analyses of lexical and morphological elements, and a corpus formed of different topics. The programming language used in implementation was Pascal. The root lexicon used, consisted of 21.727 root words and 9.528 proper nouns. There were 199 suffixes in the suffix lexicon.

Another M.S. Thesis project, in Boğaziçi University uses lexicons constructed by Güngör. The NLP-related part of the thesis implemented a morphological parser in C++. The result of the morphological analysis is used to find the root of the given Turkish word. The aim of the thesis is using this root-finding algorithm in a web-based search to broaden appropriate search results.

Nalbat implemented conjugation of any Turkish verb as an MS-DOS based application. The user inputs the verb, chooses sense, mood, tense and auxiliary tense from the options, and then the conjugated verb is displayed on the screen. This application is a good example for morphological generation. The program can be downloaded from, but it has a CPU constraint, that is, the program may not run on machines faster than 200 MHz.

Hamzaoğlu proposed a lexicon-based approach for machine translation from Turkish to other Turkic languages in his thesis. The Azeri language is chosen as a representative of Turkic languages and problems are defined and solved in this scope. Syntactic structures of sentences are similar for Turkish and the Azeri language; this feature enables employing no syntactic analysis. Only morphological and semantic analyses are carried out. Although the syntaxes of the target and source languages are not apart from each other, translation cannot be viewed as a word for word translation, due to the existence of ambiguous words. The thesis explains possible ways to resolve the ambiguities, and the implementation of a Turkish-Azeri translator using the proposed approach. The lexicon has 6.900 root entries and about 10.000 proper nouns.

In contemporary linguistic research, currently the most popular and best-known approaches to syntactic analysis can be listed as Transformational Grammars (TG), General Phrase Structure Grammars (GPSG) and Head Driven Phrase Structure Grammars (HPSG), which are enhanced variations of traditional Phrase Structure Grammars, Systemic-Functional Grammars (SFG), and Categorical Grammars (CG). Several researches based on these syntactic formalisms or similar derivatives of them have been done for Turkish.

In one of these works, Çiçekli and Korkmaz represented Turkish syntax by using the SFG formalism and implemented the generation of simple Turkish sentences. The application they developed could generate a Turkish sentence from the given semantic representation. Indeed, this semantic representation is not produced computationally; to be able to concentrate on the generation part of the work, Çiçekli and Korkmaz assumed that this description was produced by an application program that was not included in their implementation procedures. In the examples of the paper, these semantic descriptions were given by hand. The generation process had three basic stages. First, the semantic description was input to the unifier and output of the unifier was a rich syntactic description of the sentence. Then, this syntactic description was passed to the linearizer that produced the morphological description of the sentence. The morphological unit, which was based on Oflazer's FSMs, generated the worded text. For the conversion of PS rules into Prolog clauses, Definite Clause Grammars (DCG) notation is used. The DCG formalism was developed by Pereira and Warren, based on Colmerauer's Metamorphosis grammar framework. Most Prolog interpreters will automatically recognize and handle the DCG formalism.

Darcan's M.S. thesis is an important example for Turkish semantics research from a computational viewpoint, because of the approach it uses for man-machine communication. The user inputs a Turkish query, the system transforms this query into an intermediate meaning representation language, and finally this representation is transformed into the target language SQL. The SQL query runs on an imaginary studentcourse- instructor database. The analysis and generation processes include a syntactic parser for analyzing queries, a decision tree working with suffix stripping approach for morphological parsing, and a meaning representation generator for intermediate level transformation of queries. There are two additional modules incorporated in the system, namely a spelling corrector and a history keeper.

Another application, which dealt with semantics, as well as the other levels of NLP, was ALİ which was the first program that could solve primary-school-level

arithmetic problems stated in Turkish. When the user entered the problem text, the morphological part of the program analyzed each word in the input. If there were more than one parses of a word, all possibilities would be passed to the syntactic part. It was the semantic stage of the program which decided the correct meaning within the context. The syntactic part analyzed the sentences with a top-down, left-to-right parser by using a set of phrase structure rules. These rules covered all the target problem set and constituted a subset of basic Turkish sentence types. The semantic and syntactic parts were apart from each other due to computational limitations. Semantic processing started only after the syntactic parsing was completed and results were passed as an input file to the semantic level. The semantic processor used a complex and well-defined form of "templatematching" approach. Semantic templates were prepared considering the types of sentences in the target set of problems. When the appropriate template was matched with the entered problem, an answer generator did the required mathematical calculations and the answers were displayed in a human-readable form. The program also included the commonsense knowledge necessary to find the correct answers for the problems in the target set.

The Turkish Natural Language Processing Initiative (TNLP), funded by the NATO Science for Stability Program III under contract TU-LANGUAGE, was a collaborative research effort for computational analyses of Turkish text and construction of software tools for NLP applications in Turkish. The participants were Bilkent University's Department of Computer Engineering and Information Science, Middle East Technical University's Department of Computer Engineering and Halıcı Computing.

Within this extensive project, many applications were developed. An Online Turkish Morphological Analyzer, which has been developed using the two-level transducer technology of Xerox, is one of these. The user inputs the Turkish word that will be analyzed and the output is given as a list. All possible outputs of words are given if ambiguity exists. It is also possible for the user to enter more than one word by separating them with spaces. If the user enters a misspelled or

ungrammatical word, the program display asterisks as output, although some misspelled words are also (incorrectly) accepted and analyzed by the program.

Another application of this project was a Turkish Spelling Corrector. The user enters a Turkish word and possible correct forms would be seen if the word is misspelled. This corrector will generate all possible Turkish words that are within a small distance of the given incorrect word, where distance is measured by the number of character insertions, deletions, changes and transpositions.

All of the above-mentioned related works did “analysis” on the subject they were focused, but not all of them did “generation”. In any level of NLP, analysis means to examine the given specific constituent of the natural language in terms of rules of the constituents’ related level. In the morphological level, words are examined according to the morphotactic and morphophonemic rules. In the syntactic level, phrasal constituents are examined according to the grammatical relations. In the semantic level, words, phrases or sentences are examined according to their meanings. “Generation”, on the other hand, requires the synthesis of the information that will be given as the input to produce a wellformed sentence expressing that information.

The problem of representing temporal knowledge, which manifests itself in the question-answering task that has to be performed by our conversation program, is considered in many disciplines, mainly computer science, linguistics, philosophy, and psychology. Allen addresses the problems from the perspective of artificial intelligence. The preconditions that a temporal representation system should consider are given as significant imprecision (representing relative knowledge), uncertainty of information (allowing indefinite temporal relations), variation in the grain of reasoning (considering the time grains depending on the knowledge modeled), and persistence (continued validity of the knowledge in the temporal reasoning).

One of the techniques used for modeling time is the state space approach. A state is a description of the world at a time point. Actions are modeled as functions between states. A state S1 is true until an action causes it to be false. The change is

represented by deleting the state S1 and asserting a new state S2. This process provides a notion of persistence, although the other preconditions are not provided.

An alternative temporal representation is the one used by database systems, in which each fact is indexed by a date. A date may be represented as an integer in a simple system or a representation based on calendar dates and times can be chosen for a more precise system. It is the appropriate approach for systems that can assign a date for any event. But it is not possible to represent a relation between events E1 and E2 if the only available knowledge is their not happening at the same time.

Temporal information can also be represented by using before/after chains. In this approach, representing relative knowledge is quite easy. But extensions should be considered in order to represent two events irrelevant to each other. In the situation calculus, knowledge is represented as a series of situations, which are the description of the world at a point of time. Actions and events are functions from one situation to another. These features are similar to the state space approach, but the situation calculus has a reverse notion of persistence, that is, a fact that is true for an instance of time should be explicitly reproven for the succeeding points of time.

Allen himself proposes an interval-based approach. He claims that the events that seem to be instantaneous are indeed decomposable, so representing them with time points, which are not decomposable units, will not be useful. The work only considers times of events, so they are better represented by intervals. There can be the ‘before’, ‘meet’, ‘during’, ‘overlaps’, ‘starts’, ‘finishes’ relations, their inverses, and the ‘=’ relation between the intervals. An indefinite piece of temporal knowledge can be represented by more than one relation at a time. An algorithm and a transitivity table between relations are given to assign the appropriate relations to items of temporal knowledge.

CHAPTER THREE
MATERIALS AND METHODS

This study achieves a system that stores extra information in the lexicon. The information can be syntactical, semantical or any type of information that can be used for the analysis of the input text.

This chapter presents the main motivation of the study and defines the main components of the system briefly. The design and development issues of the study are clarified in detail using examples when necessary.

3.1. Environment

The system is developed using *Java* programming language with JBuilder Development Environment (DE) and *MySQL* database system as the storage environment.

Java programming language is selected as it is an object oriented programming language which achieves environment independency. Java applets enable the enhancement of user friendliness and can be published through web easily. JBuilder DE is used for as the development environment as it is an easy to use development environment which enables the applets to be designed visually.

The Java project developed consists of 3853 lines of code with 50 classes. The classes are grouped and distributed into 10 packages according to their purpose and use. The class diagrams will be supplied as Appendix C.

MySql Database is selected as it is a free and easy to maintain database engine.

The data needed by a root-driven, dictionary using morphological analyzer can be grouped in two types: the lexicon and the morphological rules. The MySql database

developed consists of five tables; responsible for holding lexicon entities, and the morphological rules.

The lexicon is stored using two tables:

- One table of the lexicon *'TNLP_GROUPS',* holds all the entities;
- The second '*TNLP_GROUP_PARENTS_REL*' holds the relations of the entities in the lexicon.

The detailed information about the structure of the lexicon can be found in the following pages.

The storage of the morphological rules is managed by three distinct tables. One main table, *'TNLP_RULES'*, is used for holding the list of rules with name and type information of the specified rule.

The other two tables are used for holding the components of the rules. There are two distinct tables as there are two distinct types of morphological rules (see *Morphological Rules* section in *Background* chapter) and definitely as each rule type needs different types of components. *'TNLP_VARIATION_RULE'* holds the components of the Morphophonemics, whereas *'TNLP_ORDER_RULE'* holds the components of the Morphotactics, Thus, the real information about each rule is stored in the table specific for that type of rule.

The Entity Relationship Diagram and specifications of the table structures for the database *'lexicon'* are supplied as Appendix D.

The retrieval of prime numbers and the mathematical operations on prime numbers are achieved by a package named: *primes*. The package consists of three classes which are written by Langlois (2004).

3.2. Lexicon

The system is designed in the light of morpheme based morphological analysis approach, thus the lexicon is responsible for holding the list of valid morphemes. The entities in the lexicon can be grouped as root words and affixes. Additionally the linguistic information about each entity should be stored in the lexicon for using throughout analysis purposes. Samples of linguistic information about an entity can be its morphological category such as root or suffix, or type of the suffix such as 'third plural person', etc.

The root words are taken from an electronic dictionary developed by Prof.Dr. Kemal Oflazer and his team. Some of the old words are eliminated and the rest is inserted in our lexicon adding lexical information to each entity. The lexical information about roots is represented by the categorization of them in different levels. The categorization groups and distributes the roots in fifty nine categories which are illustrated in Table 1 at Appendix B.

The affixes and the lexical information about them -the categories of affixes- are taken from a Turkish grammar book about affix structure of Turkish (Adalı, 1979). The categorization structure of affixes and the placement of affixes in the lexicon are illustrated as a graph at Appendix B. The affixes are stored in a special format for enabling alternations.

The database is defined as case sensitive and alternate characters are defined with capital letters. For example, possessive suffix *'ım, im, um, üm'* is stored as *'Im'*. In this example, *'I'* is an identifier which can be either *'ı'*, *'i'*, *'u'*, *'ü'*; the selection of the appreciate character is achieved by morphonemics. The list of these identifiers is illustrated in Table 3.1

Table 3.1 List of Alternation Identifier

Identifier	List of Alternations
‘I’	‘ı’, ‘i’, ‘u’, ‘ü’
‘A’	‘a’, ‘e’
‘D’	‘d’, ‘t’
‘C’	‘c’, ‘ç’

At the time, there are 4935 root words and 50 inflection affixes in the lexicon. The distribution of root words through the categories is illustrated in Figure 3.1 respectively. The main structure of the lexicon and all the affixes are specified at Appendix B.

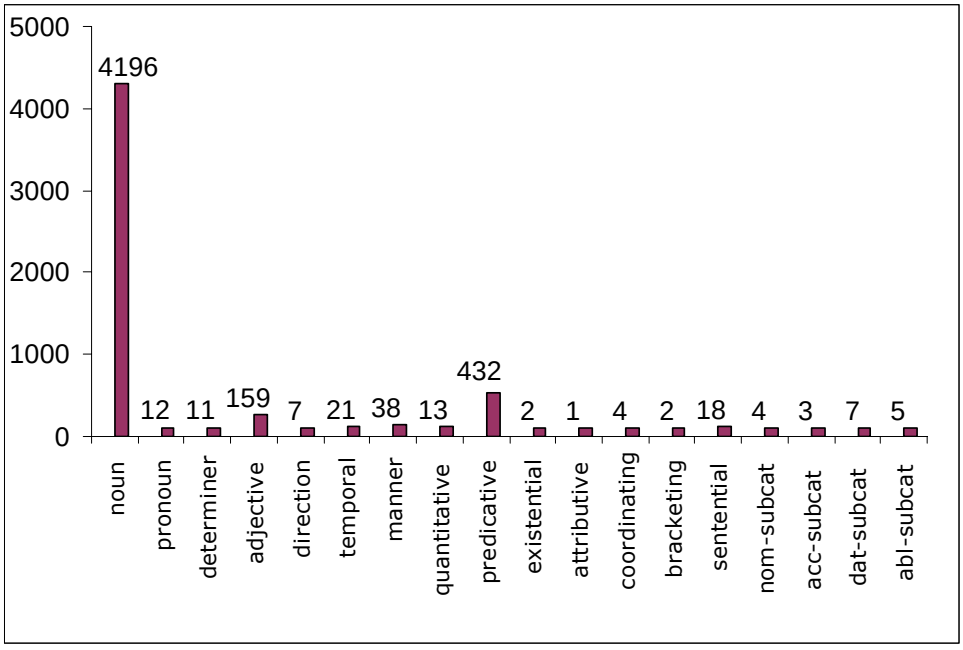

Figure 3.1 Distribution of words through categories

The lexicon is designed as a categorical lexicon, to enhance the information that can be stored in the lexicon. The entities and the categories in the lexicon form a directed graph. The linguistic information about an entity in the lexicon is represented by the categories which are ancestors of that entity; in other words, the categories which are on the path to the root node.

The computation cost for such a system would be extremely high as the lexicon will hold both the entities and the categories, and the retrieval of the linguistic

information about one entity would cause the traversal of the graph each time. Therefore, to eliminate the computation cost, each entity in the lexicon is identified by a prime number and the linguistic information about an entity is stored as the multiplication of the prime identifiers of ancestor categories. Prime numbers are unique and the multiplication process does not destroy their existence or uniqueness. In other words, the multiplication of prime numbers is a data which all the primes can be extracted from it with no information loss. Thus the retrieval and query of linguistic information is just a division process.

3.2.1. Linguistic Information in the Lexicon

The linguistic information about words in the language can be either morphosyntactic: such as stem, inflectional and derivational suffixes, syntactic: such as grammatical category and complement structures, and semantic: such as multiple senses and thematic roles (Yorulmaz, 1997).

The form-based structure of the traditional lexicons which could also be called "one-dimensional" is not always sufficient enough for natural language systems. Therefore this lexicon is designed as a "multi-dimensional" lexicon by storing the linguistic information also in the lexicon forming a graph notation.

Any kind of information that is needed by the Natural Language Processing system can be put in the lexicon if it can be expressed by a category. The categories can be extended by the lexicon user (the linguist) at runtime. Thus, the lexicon is extendible.

For a sample lexicon entity: "third singular person possessive suffix", the lexicon holds the information about:

If the affix can be added to a noun or verb,
If the affix is inflectional or derivational,
If the affix is a possessive,
If the affix shows a person whose count is singular and number is the third person.

Thus the lexicon can answer a question like “What is the person number of ‘third singular person possessive suffix’?” without any assumptions or extra knowledge.

3.3. Morphological Rules

The morphological rules ensure the validity of the word through the morphological constraints. The morphological rules used by the system are constructed in the light of the information on Chapter 5.1. Morphological Characteristics of Turkish Language. New morphological rules can be added to the system and existing morphological rules can be managed at runtime by a linguist using the Rule Management Interface. The issues on how to manage morphological rules to the system is defined briefly at Appendix A. User Guide.

3.4. Algorithm of Morphological Analysis

The algorithm designed for the morphological analyzing process takes input as word and returns all the possible morphological structures of that word.

The algorithm can be summarized as:

- Extract all possible char sequences – token candidates from the input word, place them on a two dimensional table named “Tokens’ Matrix” (as illustrated in Table 3.2).
- Apply all morphophonemics (variation rules) to the each char sequence in the table, generate possible modifications.
- Query the database for each char sequence and all possible modifications (result of morphophonemics); store all database results for the character sequence in “Tokens’ Matrix”.
- Till all token candidates in the table are processed, go to Step 2.
- Apply morphotactics (order rules) through “Tokens’ Matrix”.
- Store results.

The flowchart of the algorithm is illustrated in Figure 3.2.

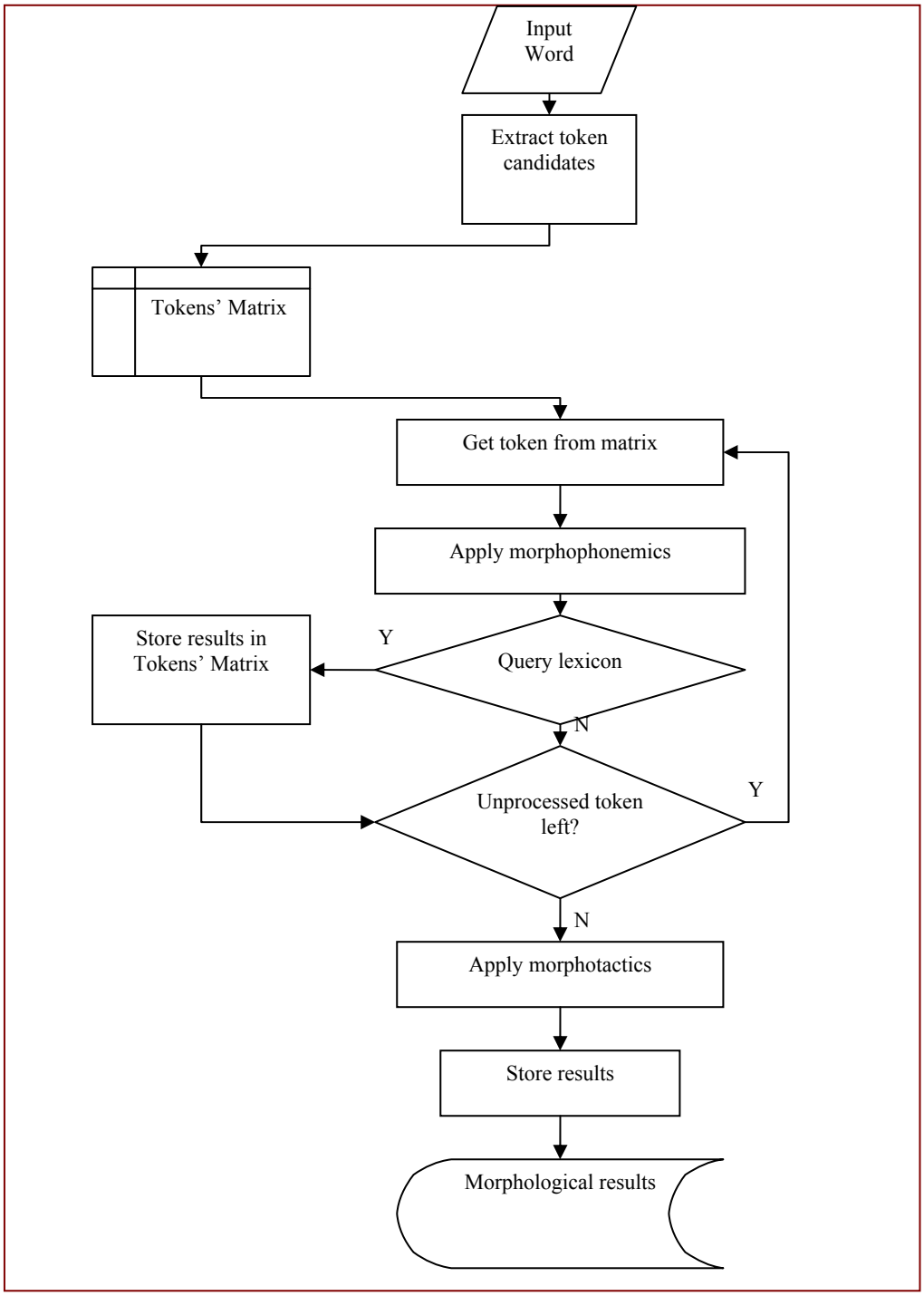

Figure 3.2 Flowchart of the algorithm

The analyzer uses the matrix structure to reduce the complexity, so that each token candidate is processed only once. The use of tokens' matrix is explained briefly with a sample word in the following example.

Example 3.1

"kitabım" is chosen as the sample to illustrate the process flow of the algorithm. The motivation of selecting this sample is the existence of the morphonemics alternations, that we can see the application of the morphonemics by the algorithm.

The matrix structure for the word sample "kitabım" is illustrated on Table 3.2. Each row of the matrix is filled by starting from the character at the number of the row and getting one more character each time till the end of the word. For the third row, the retrieval starts from third character of the word and produces five different substrings till the end of the word.

The tokens' matrix is traversed by getting the row, analyzing each column at the row from left to right. The character sequence at each cell is input to the rules of morphophonemics. The application of the morphonemics produces list of all the possible alternations of that character sequence. Then all the possible character sequences produced by morphophonemics are queried through the lexicon. Lastly the morphotactics are applied to the tokens' matrix to form a valid word structure.

The arrows on Table 3.2 points to the result of morphophonemics and the checks point to the valid lexicon entities (the character sequences that are matched in the lexicon). Note that we assume, the cells except from 'kitab' and 'ım' are not matched by morphophonemics and the character sequences except from 'kitap' and '(I)m' are not matched in the lexicon.

Table 3.2 Tokens' matrix for the word 'kitabım'

				√ kitap		
K	Ki	kit	kita	kitab	kitabı	kitabım
İ	İt	ita	itab	itabı	itabım	
T	Ta	tab	tabı	tabım		
A	Ab	abı	abım			
B	Bı	bım				
I	ım					
M						
	Im √					

The morphotactics should be applied to construct a valid word structure. For better understanding of the importance of the morphotactics, assume that we have two '(I)m' in the lexicon:

'(I)m' : possessive suffix,
'(I)m' :personal suffix.

Also we have just one morphotactic:

"noun + possessive suffix".

The morphotactic will achieve the match of '(I)m' as a possessive suffix. Although there are two entities for character sequence '(I)m' in the lexicon, the personal suffix '(I)m' cannot be a valid result as it cannot be added to the nouns, as there is no morphotactic such as "noun + personal suffix".

Then we have the structure:

'kitap' + '(I)m'
noun + possessive suffix

CHAPTER FOUR
FORMAL MODEL

This chapter states the formal model of the morphological analyzer in terms of data that is used by the system. The structure of the morphological analyzer and the data that is used by the analyzer are defined briefly with formal definitions. The structure of the data that is used by the analyzer is also specified of data structures definitions and sample instances.

4.1. Formal Descriptions

4.1.1. Morphological Analyzer

A morphological analyzer *M* is defined as:

M = (R, G, F, I, S)

where

R is a finite set of **rules** (of different types),

G is a finite set of **groups** (entities of lexicon),

F: (R x G x I) → *S* is a **function** called morphological function,

I is an input **text** to be analyzed, ordered list of the alphabet characters,

S is a finite set of morphological result structures.

A rule *R1* is defined as:

R1 = (V | O)

where

V is a variation rule,

O is an **order rule**.

A morphological result structure is an ordered list of entities' identification numbers, **ordered list of groups**, where each group belongs *G*. Thus, a morphological result structure *S1* is defined as:

S1 = (g1, g2, g3, …)

where

g1, g2, g3, … ϵ G.

4.1.2. Variation Rule

A variation rule *V* is defined as:

V = (CL, AL)

where

CL is a finite set of **conditions** (conditions to be satisfied for the rule to be interpreted),

AL is a finite set of **actions** (actions to be taken when the rule is applied).

A condition *C* is defined as:

C = (L, S, W, G, M, GL)

where

L ϵ *GL* is the group identifier of the **lexical item** to be matched,

S ϵ *GL* is the group identifier of the **surface item** to be matched,

W is either ***first*** or ***last*** (determining the direction of the search),

G ϵ *GL* is the identifier of the group to be searched,

M is either ***previous***, ***this*** or ***next*** (determining which morpheme to search),

GL is a finite set of **groups** (entities of lexicon).

An action *A* is defined as:

A = (L, S, W, G, M, GL)

where

L ϵ *GL* is the group identifier of the **lexical item** to be changed,

S ϵ *GL* is the group identifier of the **surface item** to be changed,

W is either ***first*** or ***last*** (determining the direction of the search),

G ϵ *GL* is the identifier of the group to be searched,

M is either ***previous***, ***this*** or ***next*** (determining which morpheme to search),

GL is a finite set of **groups** (entities of lexicon).

4.1.3. Order Rule

An order rule *O* is defined as:

O = (OG, GL)

where

OG is a finite **ordered list of groups**, where each group ϵ *GL*,

GL is a finite set of **groups** (entities of lexicon).

4.1.4. Group

A group *G* is defined as:

G = (I, S, T)

where

I ϵ *P* is a **unique identifier** for the group, where *P* is the infinite set of all **prime number**s,

S ϵ *L* is a representing **text** for the group, where *L* is the finite set of all **valid strings** accepted by the specific language,

T is a multiplication of identifiers of specific groups.

4.2. Data Structures

4.2.1. Variation Rule

A variation rule *V* is defined as:

V = (CL, AL)

where

CL is a set of CONDITIONS,

AL is a set of ACTIONS.

A condition *C* is defined as:

C = (L, S, W, G, M, GL)

where

L, S, G ϵ **integer** and is prime number,

W ϵ **integer**, and is either ‘0’ or ‘1’ (first = 0; last = 1),

M ϵ **integer**, and is either ‘0’, ‘1’ or ‘2’ (previous = 0; this = 1; next = 2),
GL is set of GROUPS.

An action *A* is defined as:

A = (L, S, W, G, M, GL)

where

L, S, G ϵ **integer** and is prime number,
W ϵ **integer**, and is either ‘0’ or ‘1’ (first = 0; last = 1),
M ϵ **integer**, and is either ‘0’ or ‘1’ (previous = 9; this = 1; next = 2),
GL is set of GROUPS.

4.2.2. Order Rule

An order rule *O* is defined as:

O = (OG, GL)

where

OG is a finite **ordered list of integers** (prime numbers),
GL is a finite set of GROUPS.

4.2.3. Group

A group *G* is defined as:

G = (I, S, T, L, P)

where

I ϵ **integer** and is prime number,
S ϵ **string**,
T ϵ integer.

4.3. Examples

4.3.1. Variation Rule

The variation rule for consonant softening can be determined with one condition and one action. Rule can be stated as:

Condition: If the first character of the next morpheme is a vowel, (Table 4.1);

Action: Change *'p'*, the last character of the current morpheme, to *'b'* (Table 4.2).

Table 4.1 Conditions

Text of Component	Lexical Specification	Surface Specification	Search Function	Group	Location
the first character of the next morpheme is a vowel	Vowel	Vowel	First	Character	Next morpheme

Table 4.2 Actions

Text of Component	Lexical Specification	Surface Specification	Search Function	Group	Location
Change 'p' to 'b'	'p'	'b'	Last	Character	Current morpheme

The consonant softening rule CS is defined as:

CS = (CL, AL)

where

CL = {C1, C2},

AL = {A1};

with a sample lexicon set *GL*:

GL = {(2,character,1),(3,vowel,2),(5,p,2),(7,b,2),(11,ı,6)}.

The conditions *C1* and *C2* are defined as:

C1 = (5, 7, 1, 2, 1, {(2,character,1),(3,vowel,2),(5,p,2),(7,b,2),(11,ı,6)}).

C2 = (3, 3, 0, 2, 2, {(2,character,1), (3,vowel,2), (5,p,2), (7,b,2), (11,ı,6)}).

The action *A1* is defined as:

A1 = (5, 7, 1, 2, 1, {(2,character,1), (3,vowel,2), (5,p,2), (7,b,2), (11,ı,6)}).

4.3.2. Order Rule

The order rule for nominal pattern of Turkish can be determined with eight morpheme categories in the order specified in Table 5.4.

And the morpheme categories can be defined as:

Plural Suffix:	-lAr		
Possessive Suffix:	-(I)m	-(I)mIz	
	-(I)n	-(I)nIz	
	-(I)	-lArI	
Case Suffix:	-(y)I	-(y)lA	
	-(y)A	-(y)DA	
	-DAn	-(n)In	
	-nI	-nA	
	-nDA	-nDAn	
Relative Suffix:	-ki		

Thus nominal pattern rule *VP* is defined as:

NP = (OG, GL)

where

OG = {3, 11, 13, 17, 19},

with a sample lexicon set *GL*:

GL = {(2,root,1),(3,nominal_root,2),(5,affix,1),(7,suffix,5),(11,plural,35),(13,possessive,35),(17,case,35),(19,relative,35),(23,ev,6),(29,kitap,6),(31,lAr,11*35),(37,m,13*35) ,(41,I,17*35),(47,ki,19*35)}.

4.3.3. Group

The group *G1* is defined as:

G1 = (47, ki, 19*35).

A sample lexicon is illustrated at Appendix B.

CHAPTER FIVE
CHARACTERISTICS OF TURKISH

5.1. Morphological Characteristics

Turkish is a morphologically complex, agglutinative language where the words are formed by affixing roots.

The morphological characteristics of Turkish can be expressed by two types of morphological rules: morphophonemics and morphotactics (See Background).

5.1.1. Morphophonemics (Rules of Phoneme Substitution)

There are two important morphophonemics in Turkish. This section defines these rules which can be stated as the vowel and the consonant harmony.

Each valid word in Turkish –except foreign words- must apply these rules. The roots and stems which are retrieved from foreign languages are exceptions to these rules. Then the harmony is achieved within the last syllable of the root or the stem and the suffix.

5.1.1.1. Vowel Harmony

Vowel harmony is defined with examples ‘okul’, ‘ev’ as roots ‘Im’ – possessive suffix as suffix.

Major Vowel Harmony (Palatal assimilation): Front vowels are followed by front vowels whereas back vowels are followed by back vowels:

{e, i, ü, ö} is followed by {e, i, ü, ö},
{a, ı, u, o} is followed by {a, ı, u, o}.

Example 5.1

The purpose of this morphonemic can be defined as the constraints it defines for the choose of the appreciate character for ‘I’:

okul + Im

Constraint : 1. u is followed by {a, ı, u, o}.

ev + Im

Constraint : 1. e is followed by {e, i, ü, ö}.

Minor Vowel harmony (Labial assimilation): Unrounded vowels are followed by unrounded vowels whereas round vowels are followed by either unrounded-open vowels or round-close vowels:

{e, i, a, ı} is followed by {e, i, a, ı},
{ü, ö, u, o} is followed by either {ü, u} or {e, a}.

Example 5.2

The purpose of this morphonemic can be defined as the constraints it defines for the choose of the appreciate character for ‘I’:

okul + Im

Constraint : 2. u is followed by either {ü, u} or {e, a}.

ev + Im

Constraint : 2. e is followed by {e, i, a, ı}.

The appreciate character is chosen by the help of the constraints defined by both harmonies and the alternation character list of ‘I’ (‘ı’, ‘i’, ‘u’, ‘ü’).

okul + Im

Constraint : 1. u is followed by {a, ı, u, o}.

2. u is followed by either {ü, u} or {e, a}.

3. 'I' is either ('ı', 'ı', 'u', 'ü').

Result: I is 'u'.

okul + Im → okulum

ev + Im

Constraint : 1. e is followed by {e, i, ü, ö}.

2. e is followed by {e, i, a, ı}.

3. 'I' is either ('ı', 'i', 'u', 'ü').

Result: I is 'i'.

ev + Im → evim

5.1.1.2. Consonant Harmony

Final Stop Devoicing - Voicing: Voiced consonants at the end of a morpheme change to the appropriate voiceless consonant when followed by vowel:

{p} changes to {b},

{ç} changes to {c} when the root is polysyllable or there are {n, r, l, v} before {ç},

{t} changes to {d},

{k} changes to {ğ} when there is vowel before {k},

{k} changes to {g} when there is {n} before {k},

{g} changes to {ğ} when there is {o} before {g}.

Example 5.3

kitap + Im → kitabım

‘p’ changes to ‘b’ as it is followed by a vowel. Table 5.1 shows the rule occurrences on word ‘kitabım’; which consist samples of Final Stop Devoicing and vowel harmony.

Table 5.1 Rule Occurences on Sample Word ‘kitabım’

Stem	Affix	Word	Rule Occurrence
kitap	(I)m	kitabım	p changes to b, I changes to ı

Suffix - Initial Devoicing - Voicing: Voiced consonants at the end of a morpheme are followed by a voiced consonant, whereas voiceless consonants at the end of a morpheme are followed by a voiceless consonant. {D} and {C} are two allomorphs. The suffixes are stored with allomorphs in the lexicon.

{D} changes to {d} by default, changes to {t} if there is voiceless consonant at the end of the previous morpheme,

{C} changes to {ç} by default, changes to {c} if there is voiced consonant at the end of the previous morpheme.

Example 5.4

kitap + CI → kitapçı

‘C’ changes to ‘ç’ as ‘p’ at the end of the previous morpheme is voiceless. Table 5.2 shows the rule occurrences on word ‘kitapçı’; which consist samples of Suffix Initial Devoicing and vowel harmony.

Table 5.2 Rule Occurences on Sample Word ‘kitapçı’

Stem	Affix	Word	Rule Occurrence
kitap	CI	kitapçı	C changes to ç, I changes to ı

5.1.2. Morphotactics (Rules of Morpheme Sequences)

Table 5.3 shows **the verbal inflectional model** for Turkish, a word inflected from a verb has to apply this rule.

Table 5.3 Verbal Inflection Pattern (Oflazer, Göçmen & Bozşahin, 1994)

verbal root	voice suffixes (optional)	Negation suffix (optional)	compound verb s. (optional)	main tense suffix	question suffix (optional)	second tense s. (optional)	Person suffix

Voice suffixes are: reflexive, reciprocal, causative and passive suffixes.

The shortest verb can be constructed with one verbal root, one time suffix and one person suffix. These three suffixes are obligatory, the others are optional. And in this study compound verb suffixes are omitted.

Table 5.4 shows **the nominal inflectional model** for Turkish, a word inflected from a noun has to apply this rule.

Table 5.4 Nominal Inflection Pattern (Oflazer, Göçmen & Bozşahin, 1994)

nominal root	plural suffix (optional)	possessive suffix (optional)	case suffix (optional)	relative suffix (optional)

For the sample of '*geliyordum'*, the true result of the morphological analyzing would be as stated in Table **5.5** and it is an appropriate structure up the rule above.

Table 5.5 Morphological Structure of 'geliyordum'

Gel	(i)yor	dI	m
Verbal Root	Main Tense S.	Second Tense S.	Person Suffix

5.2. Syntactical Characteristics

Turkish is a verb final language in which the word order can be characterized as subject – object – verb (SOV). Though other orders are grammatically possible, as the case –not the location- of the noun phrase determine its grammatical function in the sentence.

CHAPTER SIX
RESULTS

This chapter states the results of the study in terms of the process flow with consideration of complexity issues.

6.1. Inputs

The inputs of the system are the input word, the lexicon and the list of morphological rules.

The study introduced a system with a specialized categorical lexicon which can store additional information of the lexicon. The lexicon can be extended continually in the runtime and the success of the analysis is increased by the extension of the lexicon.

The morphological rules can also be extended for retrieving more successful analysis results.

6.2. Outputs

The output of a successful analysis is the set of possible morphological structures that can be assigned to the word.

A morphological structure constructed by the analyzer is an ordered list of prime numbers. The results of the morphological analyzer are a set of morphological structures as a word can have more than one legal morphological structure and not one the correct match can be done during the morphological analysis phase as it is affected by syntax level, semantics level, etc. Thus, the results structure is a set of ordered lists of prime numbers and each ordered list shows one legal morphological structure of the input word.

6.3. Evaluation of the Algorithm

The complexity of the algorithm of the morphological analyzer can be formulized as:

$n^2pd + tn^2 \approx n^2$ as p and t are constants, d too small,

where

n is the number of characters in the word,

p is the number of morphophonemics,

d is the complexity of the division process (database lookup),

t is the number of morphotactics.

CHAPTER SEVEN
CONCLUSION

We gather most of the information about the world by natural language. Thus a natural language processing system capable of building knowledge structures and connecting them together to build a knowledge base is a promising approach to Turkish Natural Language Processing as it is a step to mimicking human beings' behaviour.

7.1. Achieved Success

The introduced system promises the storage of the lexicon as a knowledge structure in which all entities are connected in the form of a graph. The system achieves the storage of the relations within the lexicon entities, thus can answer questions about the entity. The storage of information in the lexicon enables building a knowledge base that will be helpful during the analysis of the text in all levels.

The lexicon structure introduced in this thesis is open for extension not only as the amount of information, also as the type of information that can be stored. The system enables the growth of the depth of the lexicon as it does not limit the type of information. This enables the growth of the success of the study at the phase of actual using.

The result of the system is the valid possible morphological structures of the input text. Each morphological structure is the list of the identifiers of lexicon entities, thus morphological structures can also answer questions about the information stored in the lexicon.

7.2. Shortages

The main shortage of this study is the requirement of the linguist to store all the information in the lexicon and all the rules by hand. The success of the system is limited by the amount of information stored.

Other shortages are the boundary of the maximum prime number and the time cost of the system on long input texts.

7.3. Future work

The success of the developed morphological analysis can be extended by:

- Extending the number of morphological rules,
- Extending the size and the depth of the lexicon (the linguistic information about each entity in the lexicon).

New analysis engines can be designed and developed which will use the information stored in the lexicon. The system is designed in the consideration of these extensions. Thus, new rule types and new engines can be developed with less effort and use the results of already developed systems.

The system can also be enhanced by adding some kind of self learning ability to extend the lexicon programmatically. That will overcome the main shortage of the system.

REFERENCES

Adalı, O. (1979). *Türkiye Türkçesinde Biçimbirimler*. Ankara: Türk Dil Kurumu Yayınları.

AIDEPOTa (n.d.) *Natural Language Processing*. Retrieved September 8, 2005, from http://ai-depot.com/NaturalLanguage/Processing-Morphology.html

AIDEPOTb (n.d.) *Natural Language Processing.* Retrieved September 8, 2005, from http://ai-depot.com/NaturalLanguage/Processing-Syntax.html

AIDEPOTc (n.d.) *Natural Language Processing.* Retrieved September 8, 2005, from http://ai-depot.com/NaturalLanguage/Processing-Semantics.html

AIDEPOTd (n.d.) *Natural Language Processing.* Retrieved September 8, 2005, from http://ai-depot.com/NaturalLanguage/Processing-Pragmatics.html

Çetinoğlu, A. (2001). A Prolog Based Natural Language Processing Infrastructure For Turkish. Boğaziçi University.

Good J. & Yu A.C.L. (1999). *Morphosyntax of Two Turkish Subject Pronominal Paradigms.* Pittsburgh: University of Pittsburgh and University of Chicago.

Hockett, C. (1954). *Two models of grammatical description*. Word 10, 210-231.

Koskenniemi K. (1983). Two-level morphology: A general computational model for word-form recognition and generation.

Langlois, O. (2004). *Finding primes & proving primality*. Retrieved February 27, 2004 from http://www.utm.edu/research/primes/prove/prove2_3.html.

Loos, E.E., Anderson, S., Day, D. H., Jordan, P. C. & Wingate, J. D. (2004). *Glossary of Linguistic Terms*. Retrieved June 27, 2005 from http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/contents.htm.

Oflazer, K. (1994) Two_level description of Turkish morphology. *Linguistic and Literary Computing, Vol.9, No:2*.

Oflazer, K., Göçmen E. & Bozşahin C. (1994). *An Outline of Turkish Morphology*. Retreived November 5, 2004 from http://www.lcsl.metu.edu.tr/pubs.html.

Öztaner, S.M. (1996). *A Word Grammar Of Turkish With Morphophonemic Rules*. Ankara: The Middle East Technical University.

Russell S.J. & Norvig P. (1997). *Artificial Intelligence A Modern Approach.* New Jersey: Prentice Hall.

SFU (n.d.) *Natural Language Laboratory at Simon Fraser University – About Natural Language*. Retrieved October 8, 2005, from http://www.cs.sfu.ca/research/groups/NLL/1.html

Wicentowski, R. (2002). Modeling and Learning Multilingual Inflectional Morphology in a Minimally Supervised Framework. Maryland: The Johns Hopkins University.

Wikipedia, (2005). *Morphology (linguistics).* Retrieved June 07, 2005 from http://en.wikipedia.org/wiki/Morphology_(linguistics).

UCSD (n.d.) *Natural Language Processing.* Retrieved September 8, 2005, from http://cogsci.ucsd.edu/%7Ebatali/108b/lectures/natlang.txt

Yorulmaz, A.K. (1997). *Design and implementation of a computational lexicon for Turkish*. Ankara: Department of computer engineering and information science and the institute of engineering and science of Bilkent University.

APPENDIX A
USER MANUAL

This user manual describes how to use the developed morphological analyzer as a stand alone program. Each interface of the program will be illustrated with screenshots and described in detail.

1.1. Lexicon

The entities in the lexicon can be either real lexicon entities such as a stem ‘kitap’ or the category entity such as ‘stem’ itself. Thus, we will name all the values in the lexicon as groups from now on.

The lexicon management interface enables the user to add new groups to the lexicon and edit existing ones.

Figure 1 illustrates the lexicon management interface. Existing groups are listed at the lower part of the screen. The listing can be done for dictionary entities, categories, phonemes or all groups by choosing the appropriate type from the combo box above the list and pressing the “Filter Lexicon Items” button.

The types in the combo box are mapped to the group types in the lexicon –which will be described later in this section. Except that, the “Category” type at the combo box represents unification of “Category” and “Category (Show in result text)” types. The list is always alphabetically sorted.

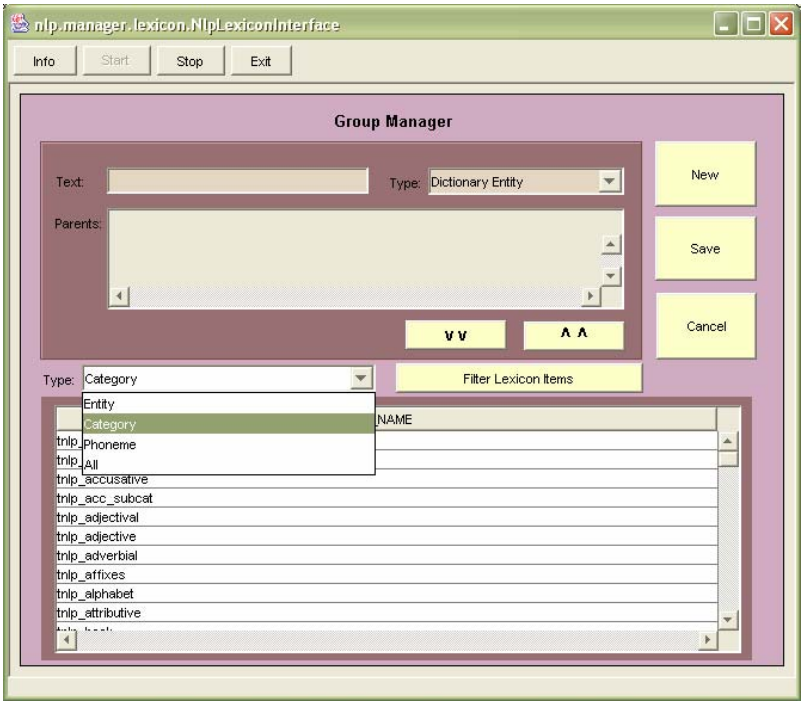

Figure 1 Lexicon Management Interface

The upper part of the screen consist the containers used for showing and editing the information about a group.

Any dictionary entity should be expressed with a string, should have a type and the parents – the linguistic category - of it must be chosen.

The text box with label “Text” contains the textual representation of the group. The combo next to the text box represents the type of the group. There are four different types of groups:

Dictionary Entity: formal dictionary entities; words and affixes.

Category: groups which are stored for expressing linguistic information. Not needed for visual representation of results.

Category (Show in result text): the categories that will be listed in the results structure.

Phoneme: the letters are also stored in the lexicon with this specific type.

The list box with label “Parents” lists the parents of that group. The parents are chosen by locating the specific group at the list below, selecting and added to the parents list by pressing to the button “^^”, and removed by selecting from the parents list and pressing to the button “vv”. The groups that are chosen as parent must be the direct parents of the group.

The buttons on the right are used for controlling the process. “New” button initiates the new group addition process by clearing the related containers. “Save” button finalize the adding or editing process by saving the changes to the database. “Cancel” button finalize the adding or editing process by discarding the changes.

1.1.1. Adding a New Group

The procedure that must be followed for adding a new group to the lexicon is explained using the sample “Third singular possessive suffix – I”.

New group adding procedure is initiated by pressing the “New” button. The related containers are cleared for new group insertion.

The textual representation “I” is typed to the text box with label “Text”. The appropriate type for this group is chosen from the type combo box. The type of this group is “Dictionary Entity” as it is an affix.

Then, the linguistic information about this group must be stored by selecting the appropriate parents. For the sample “I”, the parents chosen must be “tnlp_third” (expressing the number), “tnlp_singular” (expressing the count) and “tnlp_possession” (expressing possession). “tnlp_suffix” (expressing being suffix) must not be chosen as the other parents (“tnlp_third”, “tnlp_singular” and “tnlp_possession”) are children of it. In other words any group having “tnlp_third” as its parent is already a child of “tnlp_suffix”, thus no need to add it again.

Lastly, the adding procedure is completed by pressing “Save” button and the group is stored in the lexicon or by pressing “Cancel” button and nothing is stored in the lexicon.

1.1.2. Editing an Existing Group

When a group is selected, the information about that group is located in the upper part of the screen for editing. Figure 2 illustrates the editing phase of the “Third singular possessive – I”.

The editing procedure is alike adding procedure in so many ways. The only difference is the values are initially set.

The required changes are done by typing the new text in the text box, selecting the appropriate type from the combo box and changing the parents by adding new parents and/or removing existing ones.

Lastly, the editing procedure is completed by pressing “Save” button and the changes are saved or by pressing “Cancel” button and the changes are discarded.

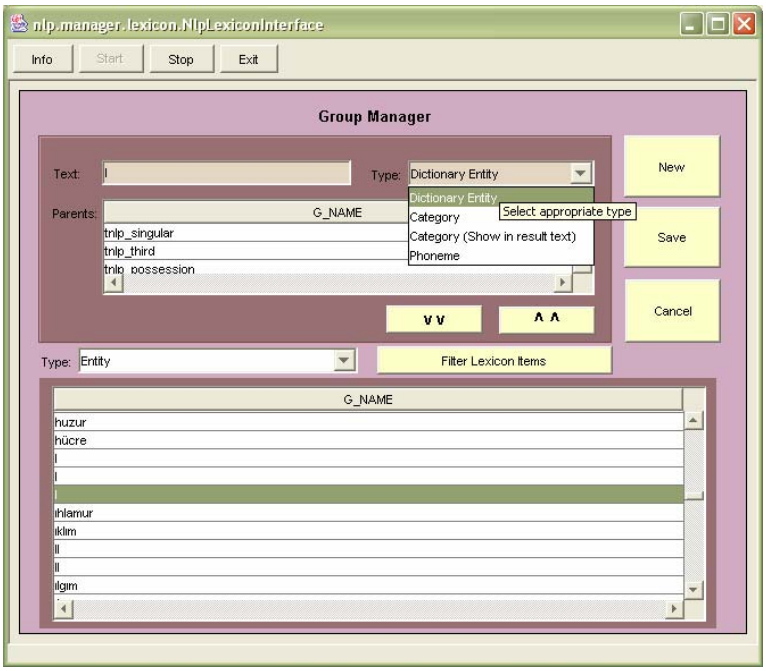

Figure 2 Editing a lexicon group (Sample entry “I” selected for editing)

1.2. Morphological Rules

The morphological rules management interface enables the user to add new rules to the system and edit existing ones.

There are two types of rules: morphophonemics and morphotactics. More information about the rule types can be retrieved from Chapter 1 Background.

The interface illustrated in Figure 3 consists of list of existing rules identified with their names. The list can be filtered using rule types by selecting the appropriate type from the combo box above the list and pressing “Filter Rules” button.

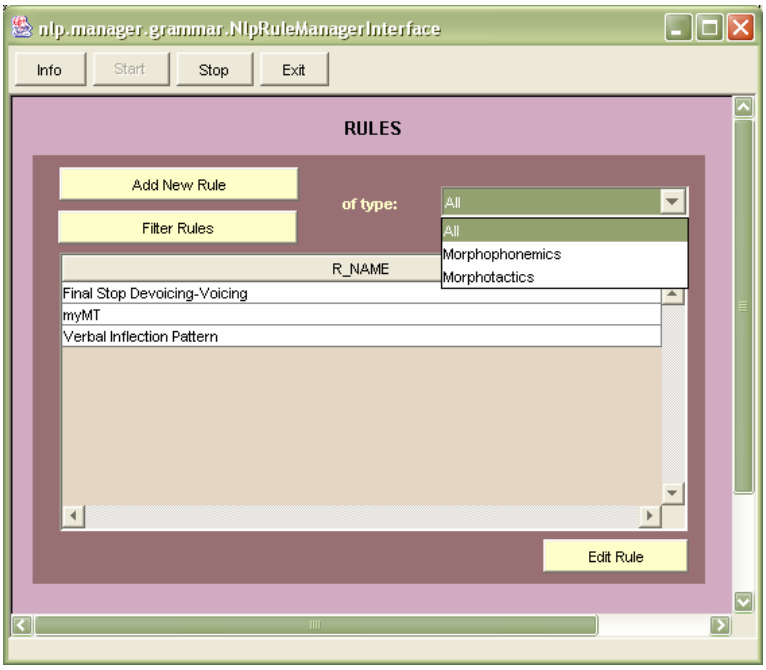

Figure 3 Morphological Rule Management Interface

New rule adding procedure is initiated by selecting the appropriate rule type from the type combo box and pressing “Add New Rule” button. The rule type must be selected initially as each rule type has a different interface for managing the rule.

The editing procedure is initiated by selecting the relevant rule and pressing “Edit Rule” button at the bottom of the list.

The adding and editing procedure for different rule types will be defined briefly for the rest of this chapter.

1.2.1. Morphotactics

The morphotactics are rules about ordering. The interface for adding or editing a morphotactic is illustrated in Figure 4 with sample of ’Verb Inflection Pattern’.

Any morphotactic should be expressed with a name and should have an ordered list of groups.

The name of the rule is typed to the text box with label “Rule Name”. At the left of the interface there is a group listing tool which achieves the same facilities with the group list in the Lexicon Management Tool and works in the same manner. The group list is updated by selecting the appropriate type at the combo box and pressing “Filter” button.

At the right of the interface the current order components are listed. Note that there can be more than one groups specified for one order number. For instance, “Verb Inflection Pattern” has two different alternates for order number ‘4’. Both of them are valid, but they can not be subject to the inflection at the same result structure.

The component adding procedure is achieved by locating the appropriate group at the “Lexicon Categories” list, selecting it and pressing “>>” button. Then the order component editing interface which is illustrated in Figure 5 is displayed.

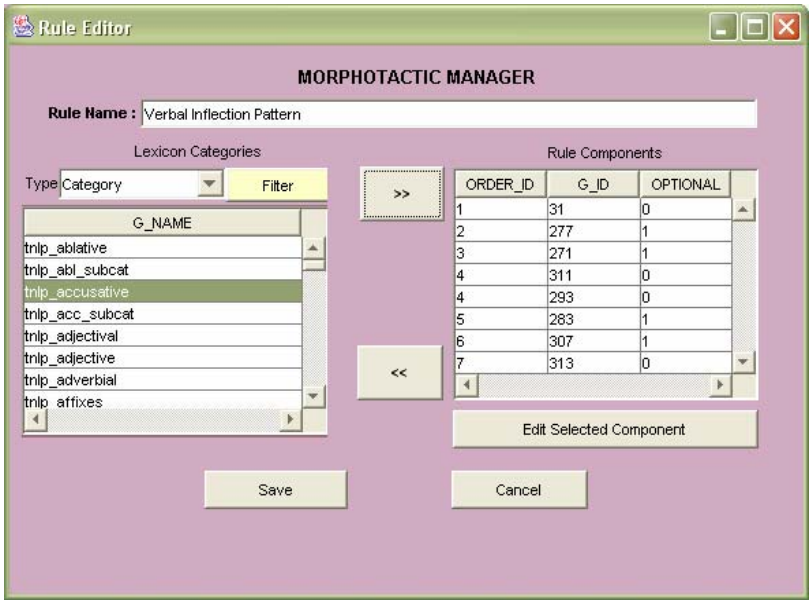

Figure 4 Morphotactic Editing Interface

Figure 5 Morphotactic Order Component Interface

Figure 4 illustrates the interface for adding a new order component to the morphotactic. The number on the left identifies the order number for this component. When a new component is being added this number is set to the last existing components order number plus one, but it can be changed to any. The component

name is retrieved from the group selected for adding. The check box on the right is checked if this order component is optional, is left unchecked if it is required for the morphotactic.

Editing an existing order component is achieved by selecting the specific component from the order components list and pressing “Edit Selected Component” button. Then the order component editing interface is displayed again. The order number, optional or not and component name can be changed.

Deleting a component is achieved by selecting the specific component from the order components list and pressing “<<” button. Note that, the order numbers for the following components are not updated; it must be done manually if necessary.

1.2.2. Morphophonemics

The morphophonemics are the rules about character substitutions. Figure 6 illustrates the interface for adding and editing a new morphophonemic.

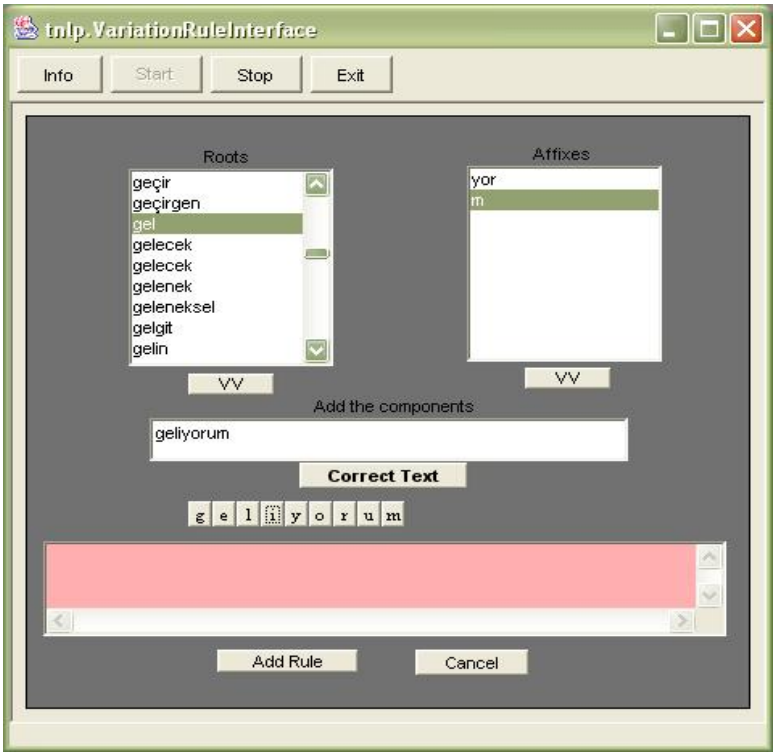

Figure 6 Variation Rule Interface

The list on the left of the window, which is named “roots’ list”, consist all the roots in the lexicon. The list on the right, which is named “affixes’ list”, consist all the affixes in the lexicon. The box in the middle is named “components’ box” and is filled with components to construct a sample word.

A new morphophonemic rule is added to the system by constructing a sample of the rule. As morphophonemics are affected by morphemes the sample is constructed by selecting root and affixes from the appropriate lists.

The sample word is constructed in the components’ box by firstly selecting a sample root from the roots’ list and clicking the button at the bottom of the roots’ list, and then firstly selecting a sample affix from the affixes’ list and clicking the button at the bottom of the affixes’ list. When the sample word is constructed, “Correct Text” button is clicked for listing the characters as small buttons.

The rule is stored by adding the appropriate conditions to be matched and the appropriate actions defining the character substitution.

Each condition or action - substitution component - must be specified by clicking the small character buttons that are subject to the substitution. When the user clicks a button the character substitution component interface which is illustrated in Figure 7 is shown.

Each character substitution component specified using the character substitution component interface is added to the list at the bottom of the window with a representing text.

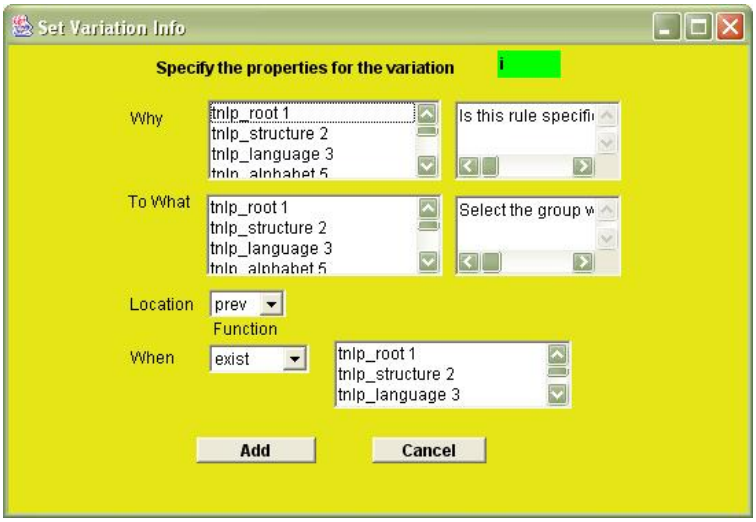

Figure 7 How to add a character substitution

The storage of a character substitution component is done by specifying the character group to be matched, the function that is used for matching, the morpheme to be considered, the condition the character group must satisfy and the substitution that will take place (the result character group).

The first list specifies the condition the character group must satisfy, why this substitution is applied. The character group chosen in this list must contain the character which was clicked on, which is written on the green box at the top of the window. This list can be filled by answering the question: "Is this substitution is specific to this character only? If not, it is specific for which group?"

The second list specifies which character group would be the result of the substitution, if this is an action substitution. If this is a condition substitution, it is same as the first list.

The location, which can be 'previous', 'this' or 'next', specifies which morpheme is subject to the substitution.

The function, which can be 'first' or 'last', specifies where to start the search – beginning or end.

The last list specifies the character group to be matched using the function.

The editing of character substitution component is finalized by storing the component by pressing “Add” button or discarding the component by pressing “Cancel” button.

1.3. Morphological Analyzer

Figure 8 illustrates the morphological analyzer’s parsing interface. The input word is written on the text box at the top of the window and the parsing process is initiated by clicking the “Parse” button. The results of the parsing are listed on the list below.

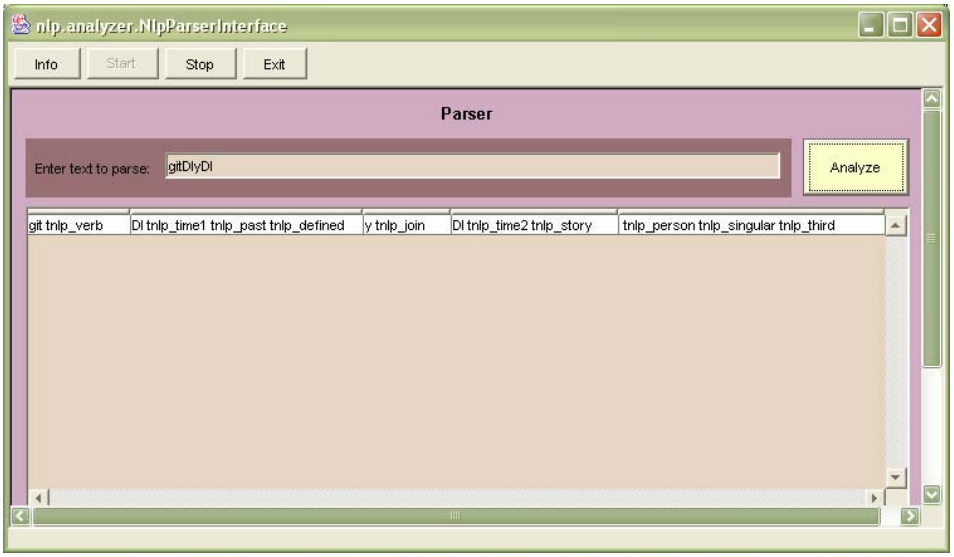

Figure 8 Morphological Parser Interface (Sample parse "gitDIyDI")

Each alternate result is shown as ordered lists. Each component of the list is represented by a text which consist the ancestor categories that are marked with type “Category (Show in result text)”.

Figure 8 illustrates a sample parsing for the word “gitDIyDI”.

APPENDIX B
THE LEXICON

Table 1 Lexicon word categories (Yorulmaz, 1997)

1.level	2.level	3.level	4.level
nominal	noun	common	
	pronoun	proper	
		personal	
		demonstrative	
		reflexive	
		indefinite	
		quantification	
		question	
	sentential	act	infinitive
		fact	participle
adjectival	determiner	article	
		demonstrative	
		quantifier	
	adjective	quantitative	cardinal
			ordinal
			fraction
			distributive
		qualitative	
adverbial	direction		
	temporal	point-of-time	
		time-period	fuzzy
			day-time
			season
	manner	qualitative	
		repetition	
	quantitative	approximation	
		comparative	
		superlative	
		excessiveness	
verb	predicative		
	existential		
	attributive		
conjunction	coordinating		
	bracketing		
	sentential		
post- position	nom-subcat		
	acc-subcat		
	dat-subcat		
	abl-subcat		
	gen-subcat		
	ins-subcat		

Table 1 lists the categories used for stem categorization. The categorization can be viewed as a tree. The levels discriminate the levels of the tree; for example “proper”

in the ‘3. level’ at third row, is child of “pronoun” at the ‘2. level’ -which is a child of “noun” at ‘1.level’. First four levels of the categorization which is introduced at Yorulmaz (1997) is applied to the lexicon.

The structure of the lexicon is illustrated as graphs in the following pages. The lexicon has a structure of a one way graph as any group in the lexicon can have more than one parent. The arrows bind parents to the children. Note that, the list of characters, stems and derivation suffixes are incomplete.

The green circles with doubled boundaries represent the nodes that will be continued in the other figures.

The complexity of the figures are tried to be overcame using different colours for some nodes and the arrows that leave the node.

Note that Ø symbol is used for empty string.

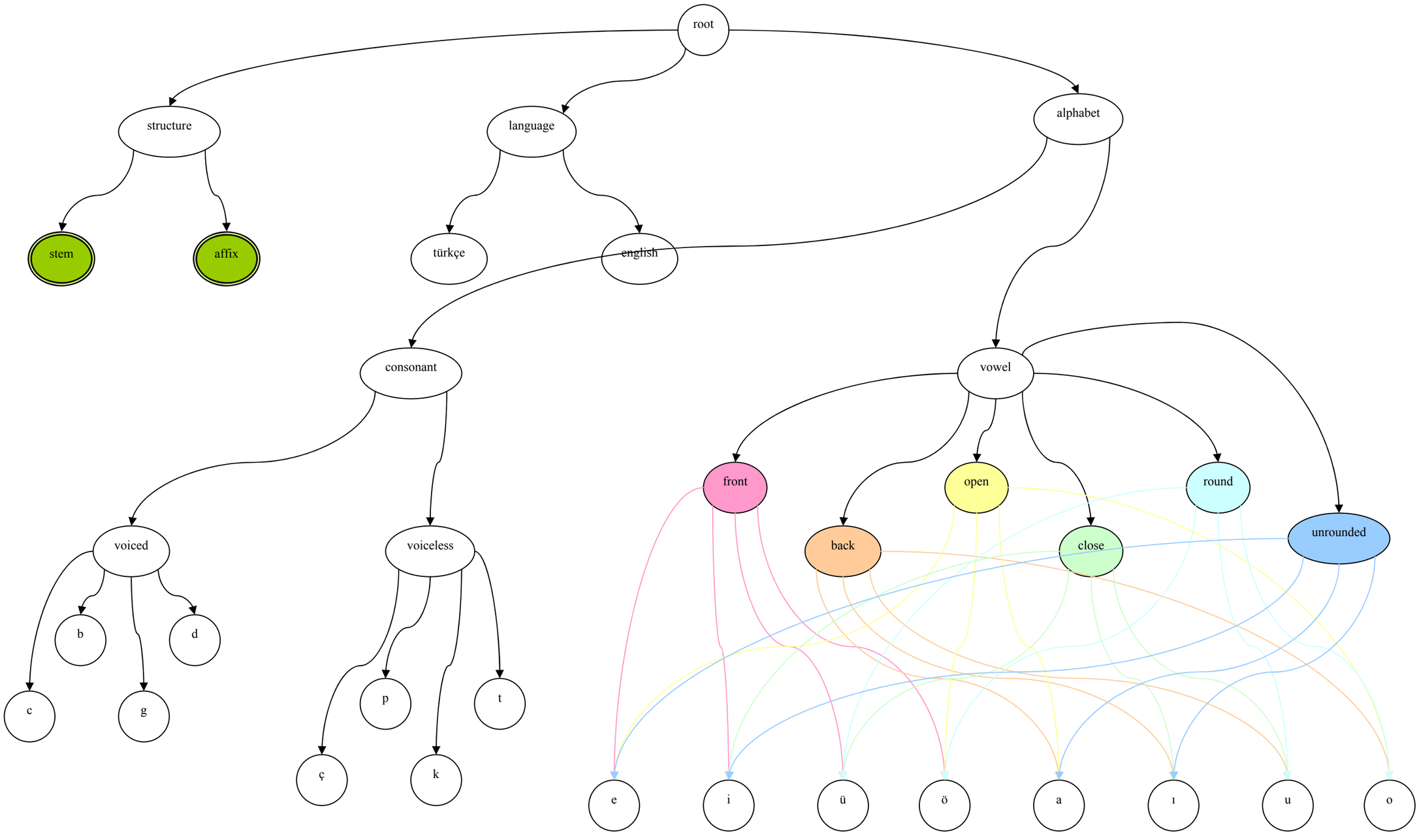

Figure 9 Graphical representation of the lexicon

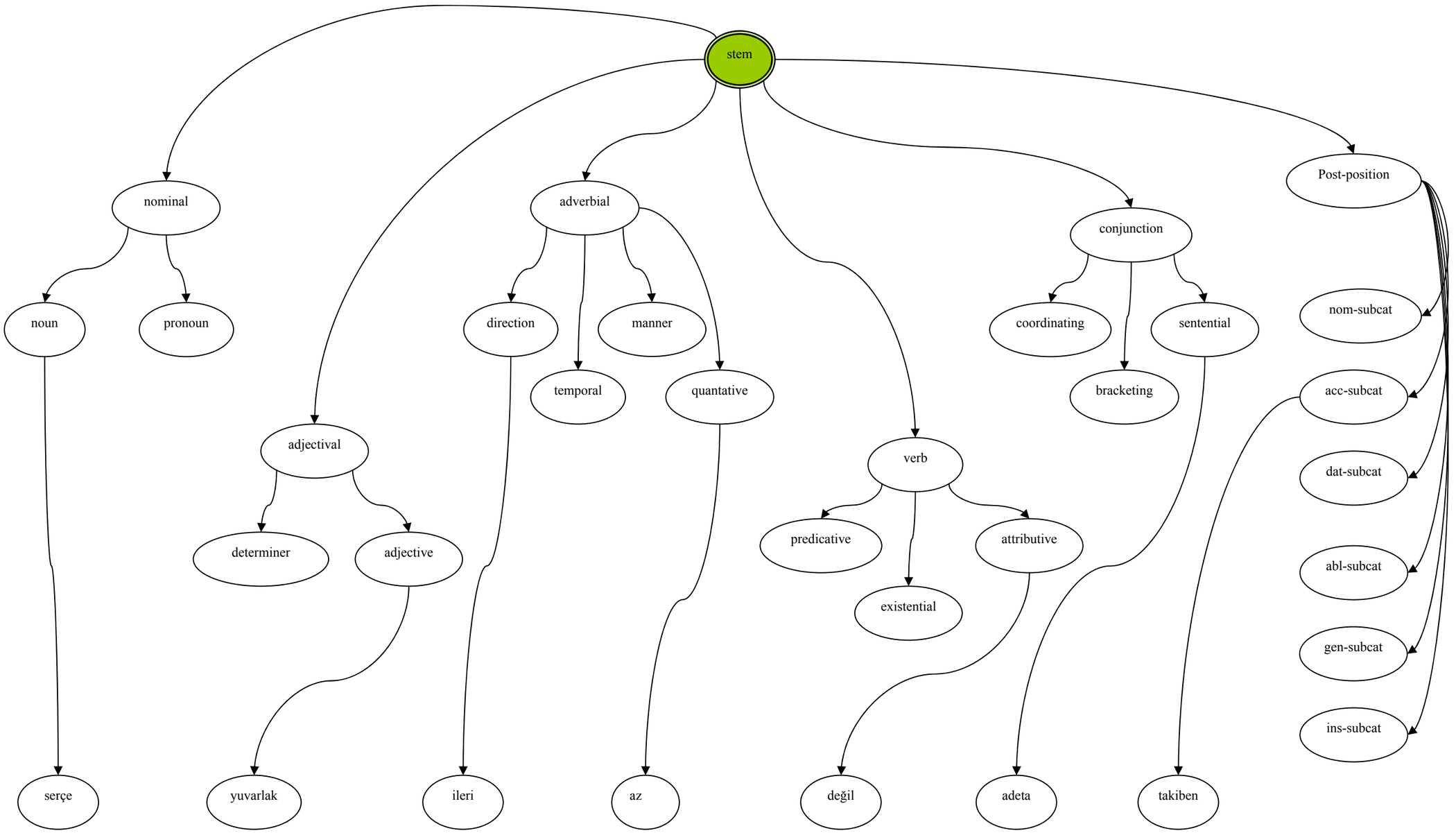

Figure 10 Graphical representation of stems in the lexicon with sample lexicon entities

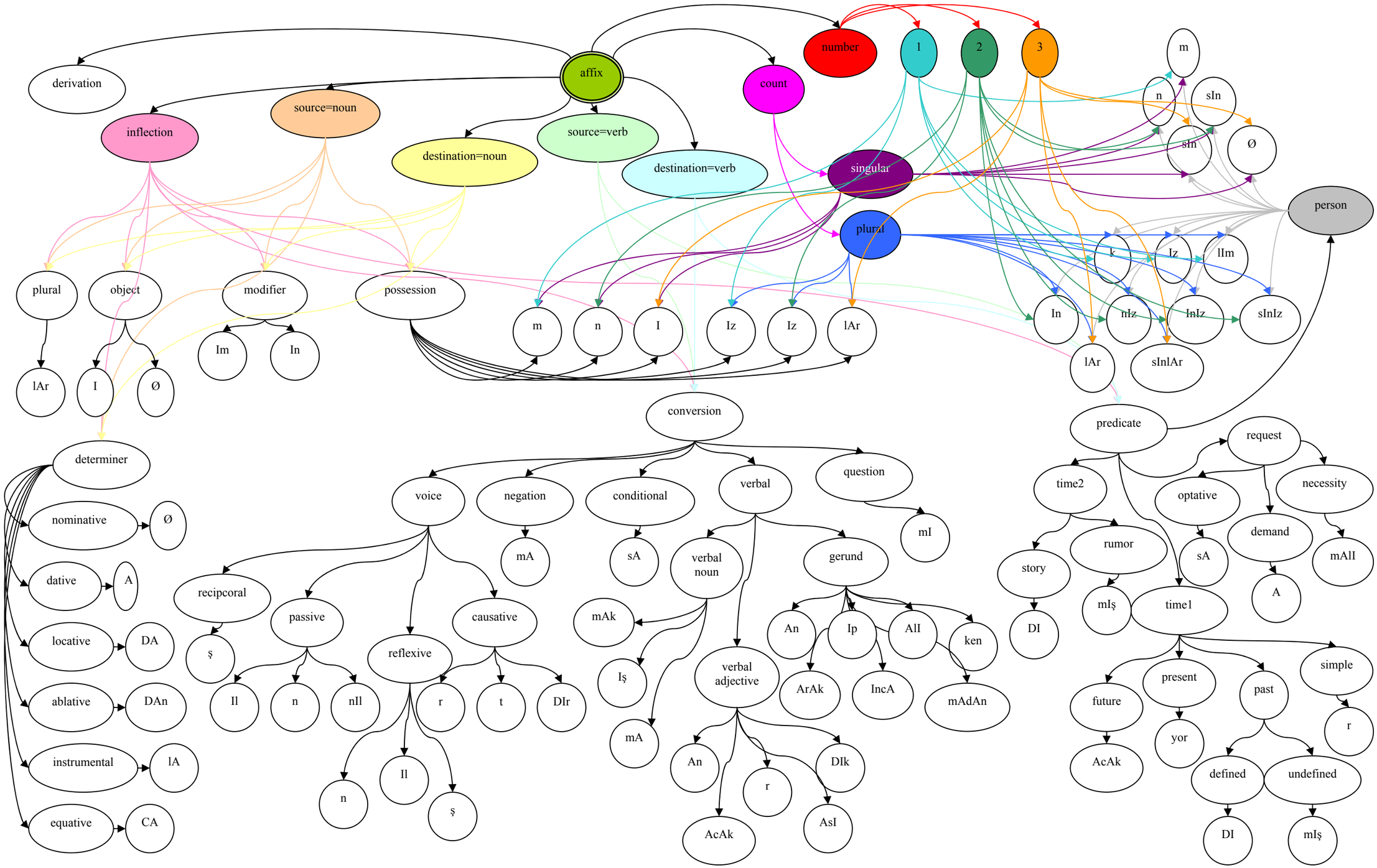

Figure 11 Graphical representation of affixes in the lexicon with full list of lexicon entities

APPENDIX C
UML DIAGRAMS

The Project consist of six main packages:

- analyzer
    - interfaces
    - ma
- connectivity
- manager
    - grammar
    - lexicon
- primes
- rule
- types.

The UML Diagrams of each package is illustrated below.

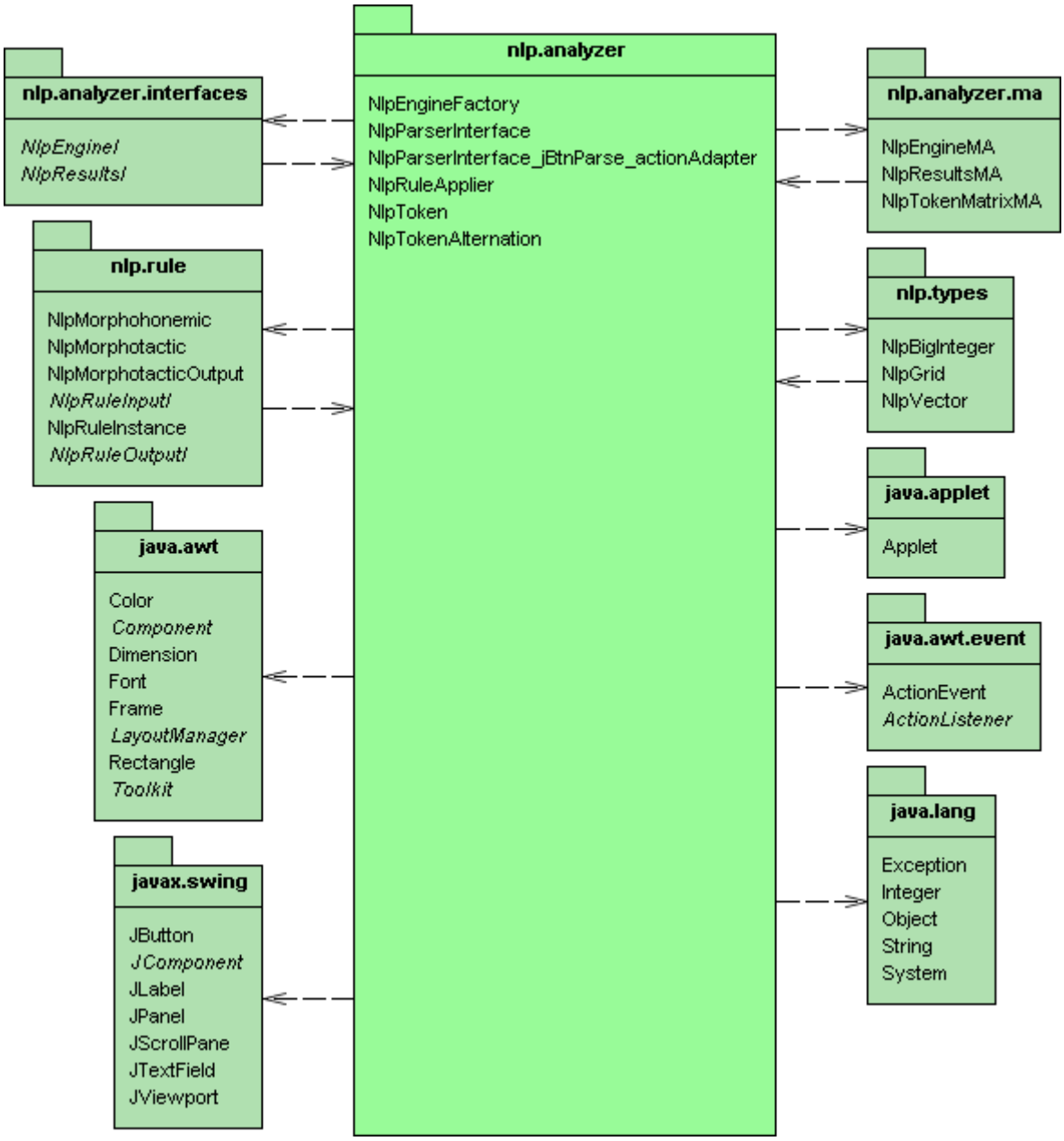

Figure 12 UML Diagram for *nlp.analyzer* package

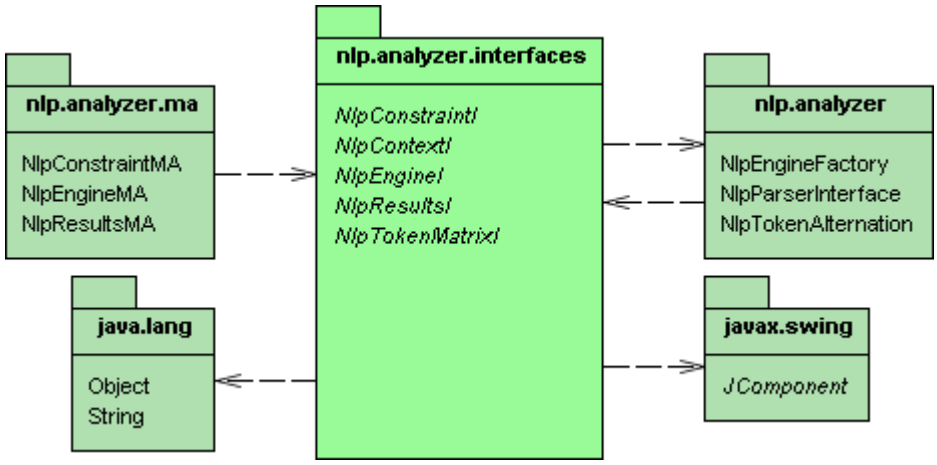

Figure 13 UML Diagram for *nlp.analyzer.interfaces* package

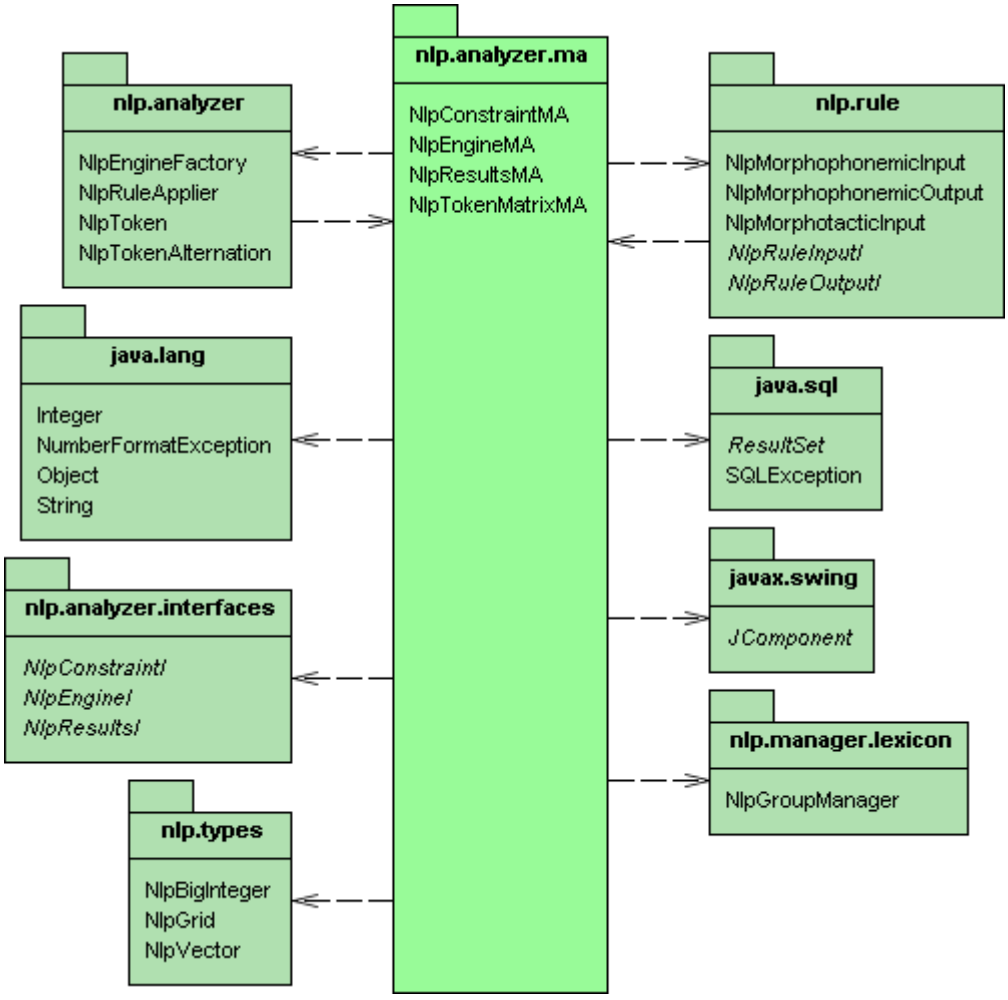

Figure 14 UML Diagram for *nlp.analyzer.ma* package

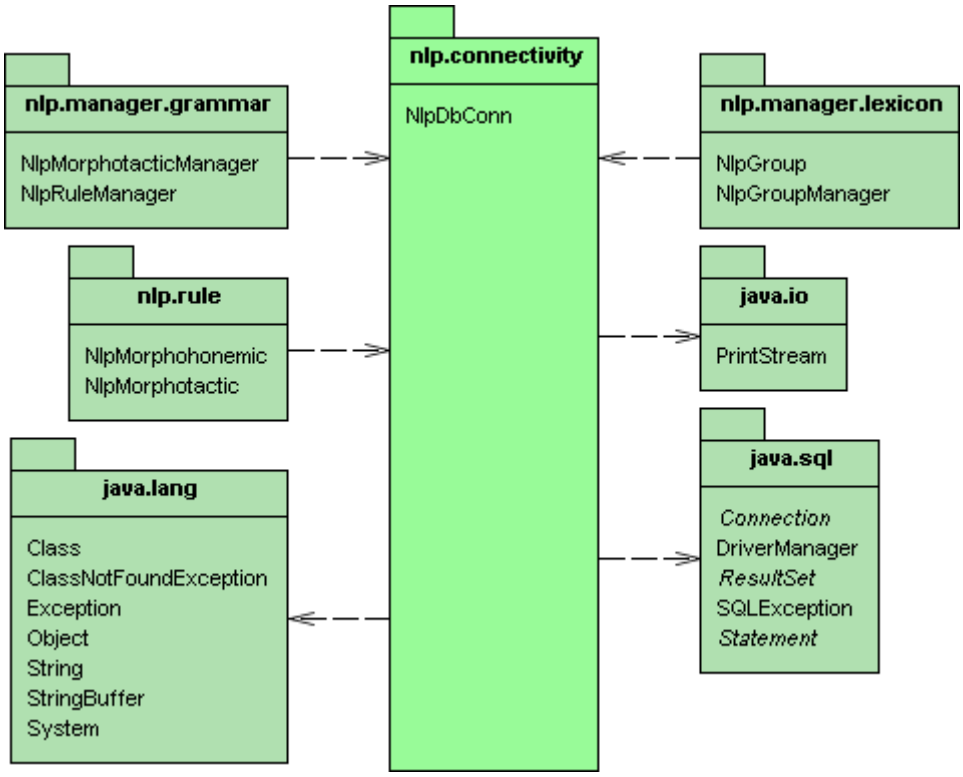

Figure 15 UML Diagram for *nlp.connectivity* package

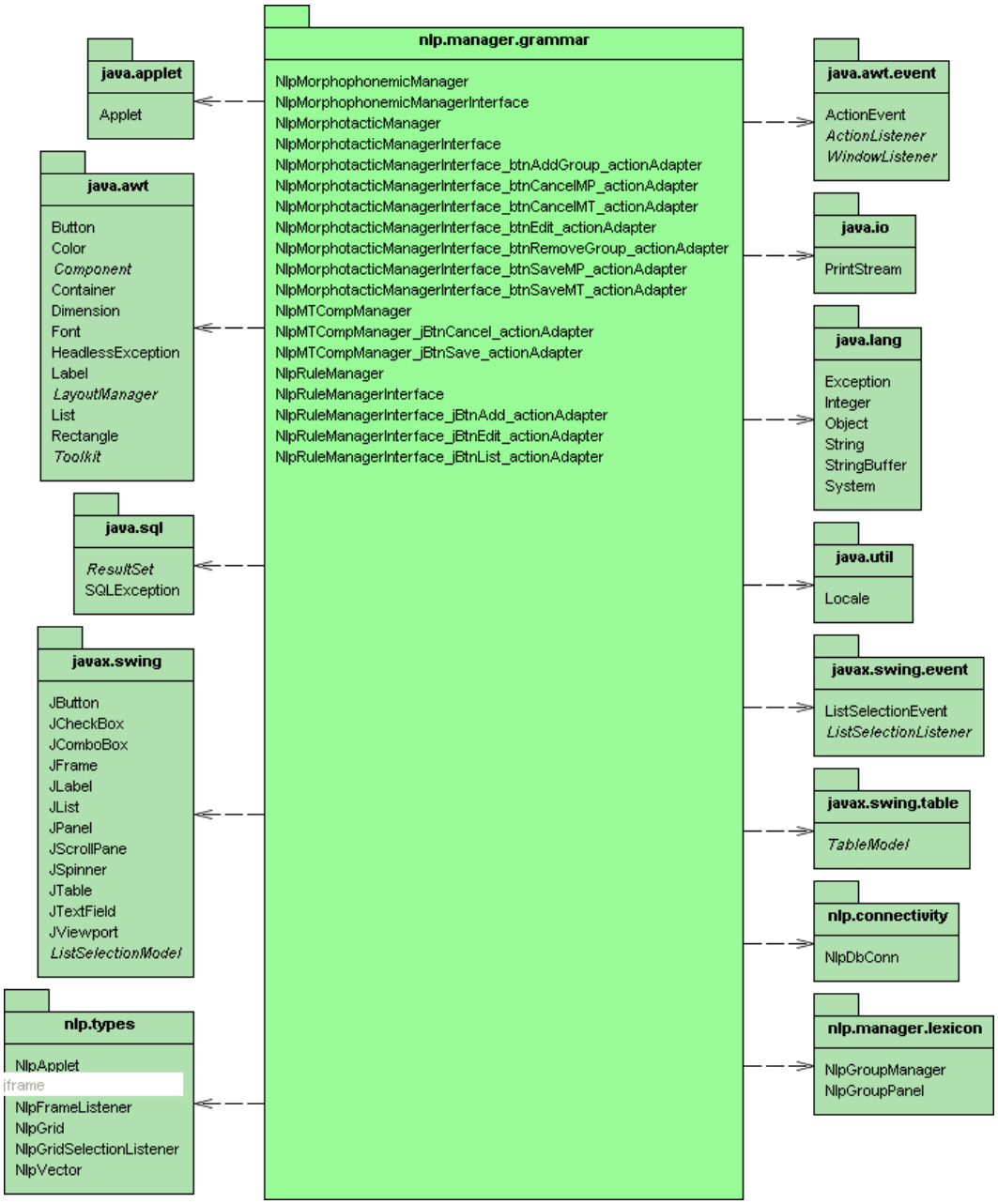

Figure 16 UML Diagram for *nlp.manager.grammar* package

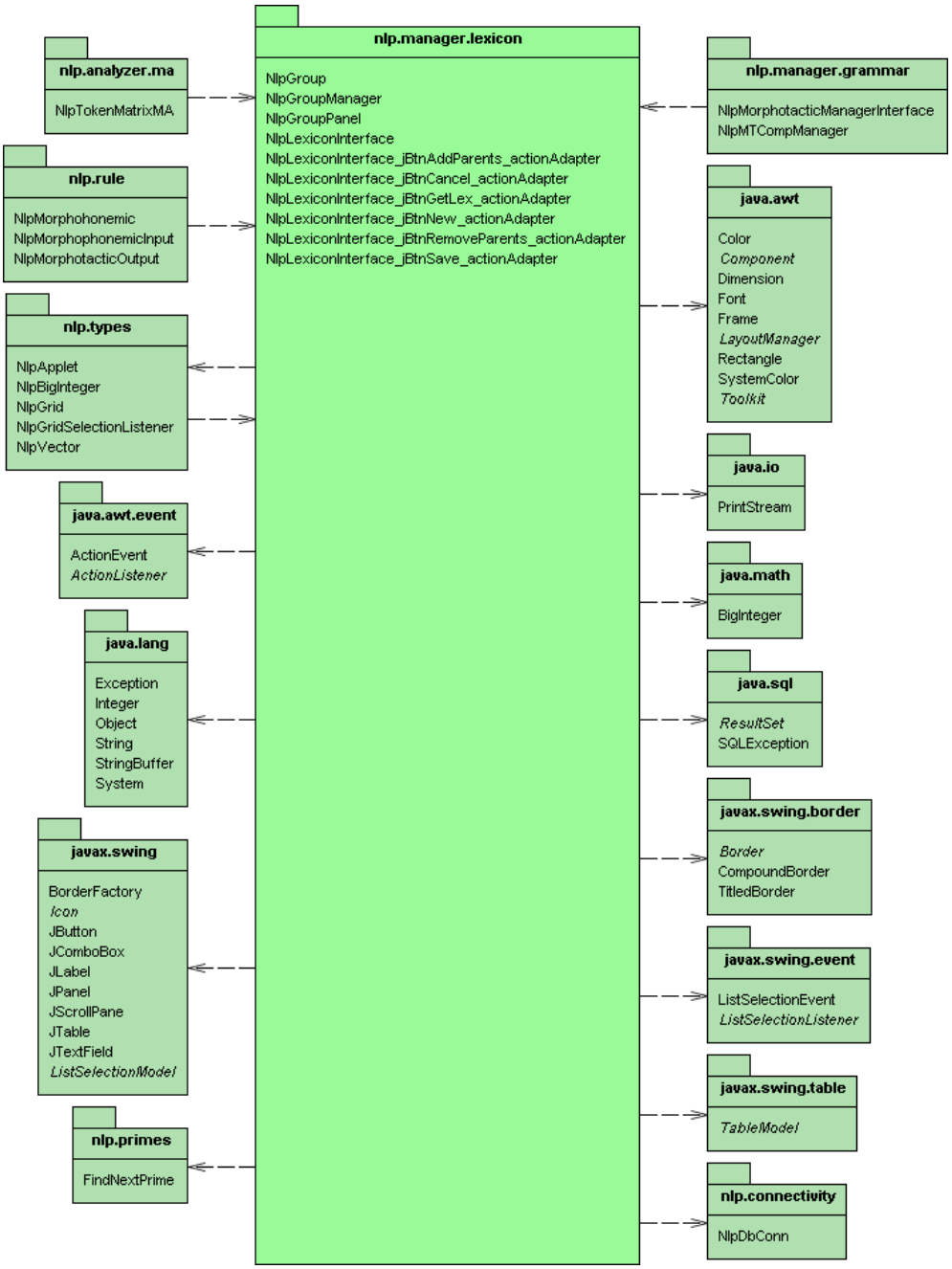

Figure 17 UML Diagram for *nlp.manager.lexicon* package

Figure 18 UML Diagram for *nlp.rule* package

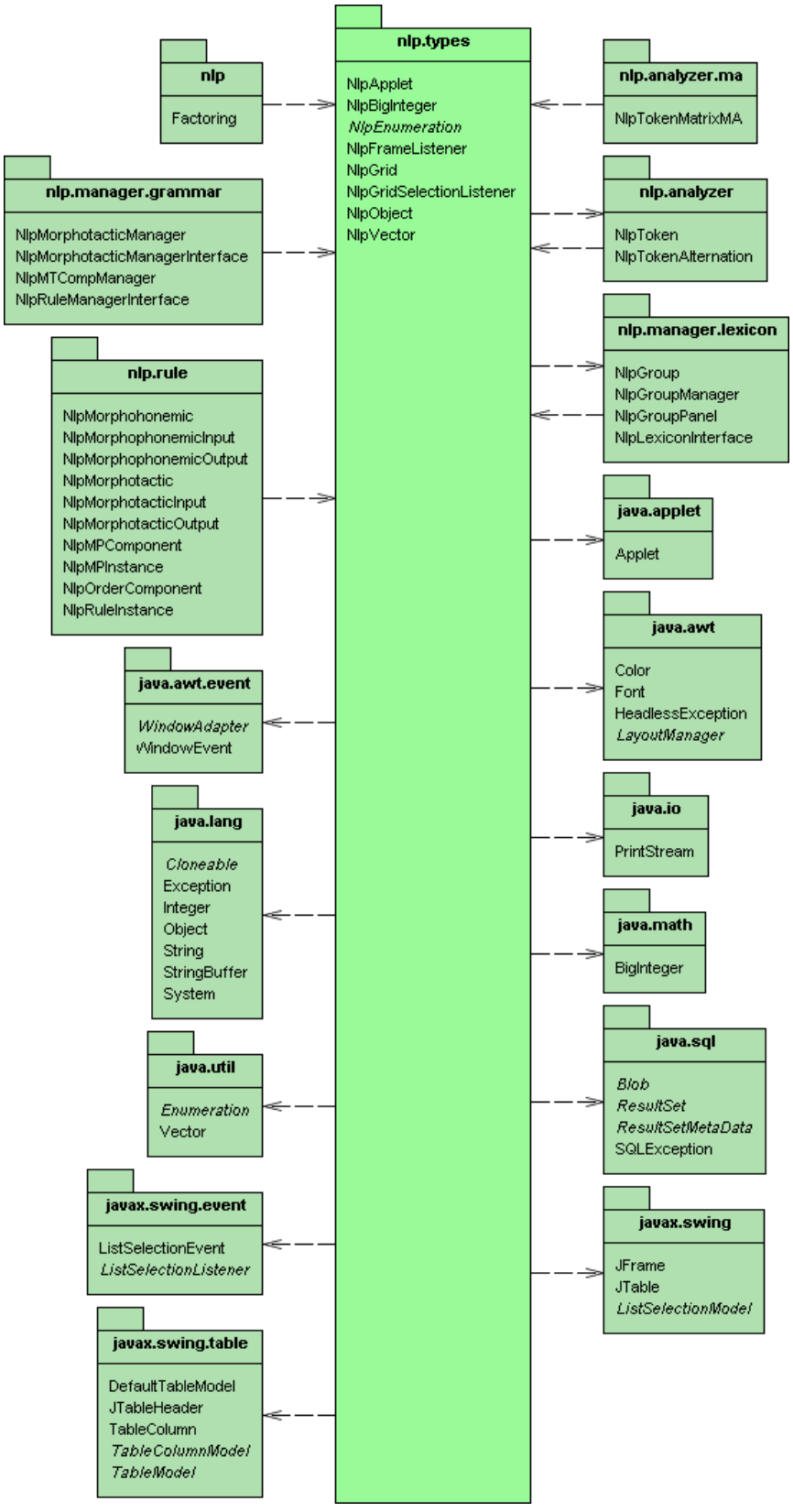

Figure 19 UML Diagram for *nlp.types* package

APPENDIX D

ER DIAGRAMS

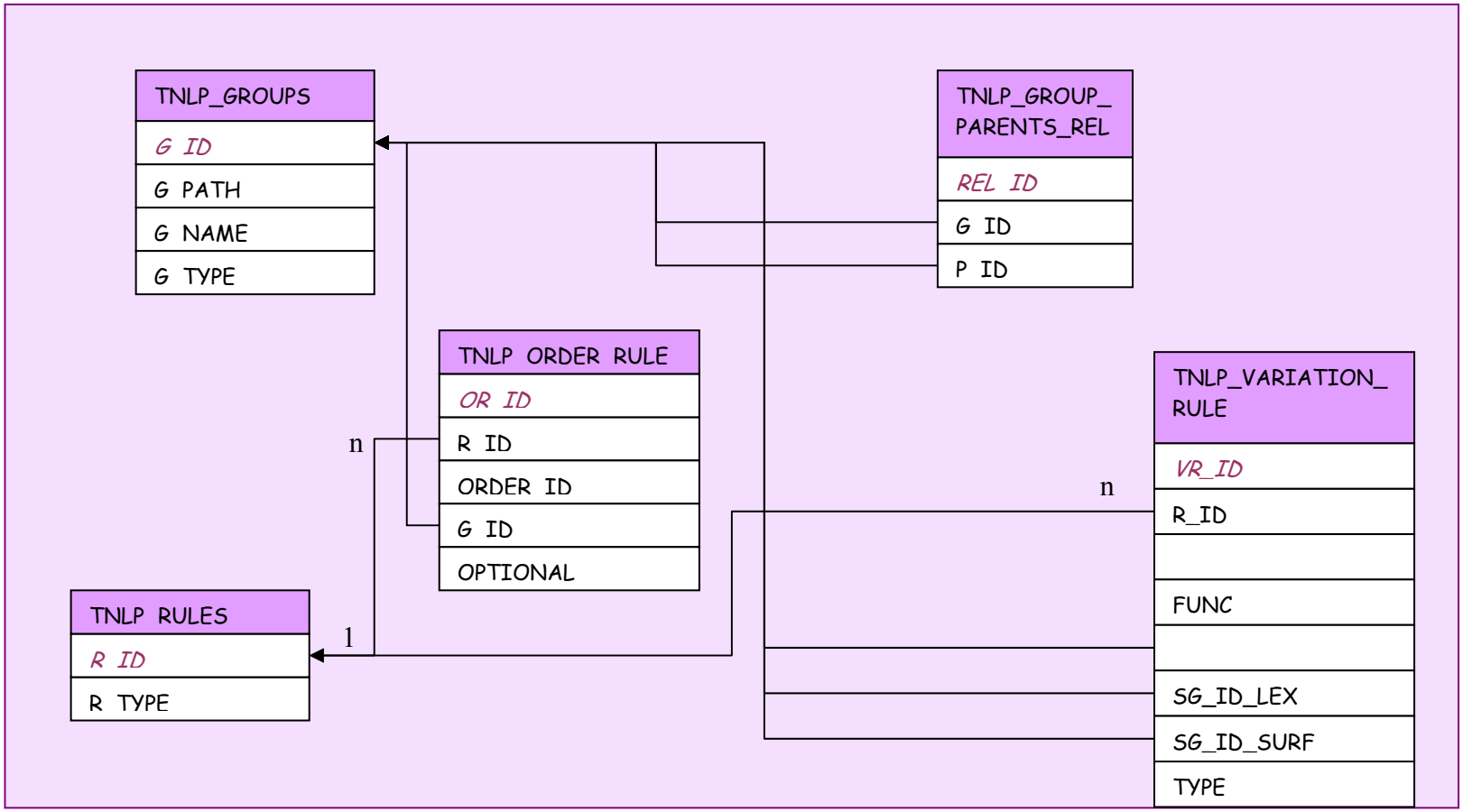

Figure 20 Entity Relationship Diagram for *lexicon* database

TNLP_GROUPS			
G_ID	Unsigned int	Primary key, Auto increment	unique identifier of the group, Prime number
G_PATH	Unsigned int		Multiplication of the group identifiers of parents (all parents till root)
G_NAME	Varchar(20)		Name of the group
G_TYPE	Tinyint		Type of group ("0" for normal groups, "1" for words and affixes; the real data, "2" for conceptual type; to use during parse)

TNLP_GROUP_PARENTS_REL			
REL_ID	int	Primary key, Auto increment	Unique identifier of the relation (group, parent). A group can have more than one child or parent (multiple inheritance).
G_ID	Int		Group identifier of the child
P_ID	int		Group identifier of the parent

TNLP_RULES			
R_ID	Int	PK, AI	Unique identifier of the rule
R_TYPE	Tinyint		The type of the rule. Variation and order are the implemented rules. Type information is hold as an enumeration. "1" for Variation, "2" for Order rules. The class name of the rule, the enumeration is known by the RulesList class.

TNLP_ORDER_RULE			
OR_ID	int	Primary key, Auto increment	The unique identifier of the rule component. A rule has more than one entry in this table so this auto increment id is used to achieve unique of the entry.
R_ID	int		The identifier of the related rule.
ORDER_ID	Int		The order of the component (for example: if the rule is "a+b+c", order of "a" is 1, "c" is 3).
G_ID	Unsigned Int		The identifier of the group that must be met in the specified order.
OPTIONAL	tinyint		Binary flag to indicate if the specified group entry is optional or not.

TNLP_VARIATION_RULE			
VR_ID	Int	Primary key, Auto increment	The unique identifier of the rule component. A rule has more than one entry in this table so this auto increment id is used to achieve unique of the entry.
R_ID	Int		The identifier of the related rule.
LOC	Tinyint		The location of the element to find, in this substring "1", previous "0" or next "2" (Enumeration is defined by the specific rule class).
FUNC	Tinyint		The function which is used for search; "0" for exist, "1" for first, "2" for last (Enumeration is defined by the specific rule class).
G_ID	Unsigned Int		The identifier of the group whom an element is searched for.
SG_ID_LEX	Int		For condition: it is the group id which the element searched must be in (in the lexical form). For action: the group identifier of the lexical form.
SG_ID_SURF	Int		The SURFACE form of SG_ID_LEX.
TYPE	Tinyint		The job specifier, is this entry for defining condition or action. "1" for condition, "2" for action