A Natural Language Processing Infrastructure for Turkish

A. C. Cem SAY
Department of Computer Engineering,
Bogaziçi University,
Bebek,İstanbul
say@boun.edu.tr

Özlem ÇETİNOĞLU
Faculty of Engineering and Natural Sciences,
Sabancı University,
Tuzla,İstanbul
ozlemc@sabanciuniv.edu

Şeniz DEMİR
Department of Computer Engineering,
Bogaziçi University
Bebek,İstanbul
sdemir@cse.yeditepe.edu.tr

Fatih ÖĞÜN
Department of Computer Engineering,
Bogaziçi University
Bebek,İstanbul
fatih_ogun@yahoo.com

Abstract

We built an open-source software platform intended to serve as a common infrastructure that can be of use in the development of new applications involving the processing of Turkish. The platform incorporates a lexicon, a morphological analyzer/generator, and a DCG parser/generator that translates Turkish sentences to predicate logic formulas, and a knowledge base framework. Several developers have already utilized the platform for a variety of applications, including conversation programs and an artificial personal assistant, tools for automatic analysis of rhyme and meter in Turkish folk poems, a prototype sentence-level translator between Albanian, Turkish, and English, natural language interfaces for generating SQL queries and JAVA code, as well as a text tagger used for collecting statistics about Turkish morpheme order for a speech recognition algorithm. The results indicate the adaptability of the infrastructure to different kinds of applications and how it facilitates improvements and modifications.

Introduction

The obvious potential of natural language processing technology for economic, social and cultural progress can be realized more comprehensively if NLP techniques applicable to a wider selection of the languages of the world are developed. Before the full-scale treatment of a new language can start, a considerable amount of effort has to be invested to computerize the lexical, morphological and syntactic specifics of that language, which would be required by any nontrivial application.

We built an open-source software platform intended to serve as a common infrastructure that can be of use in the development of new applications involving the processing of Turkish. The platform, named TOY (Çetinoğlu 2001), is essentially a big set of predicates in the logic programming language Prolog. The choice of Prolog, which was designed specifically with computational linguistics applications in mind, as the implementation language for our software has natural consequences for the knowledge representation setup to be used by other programs built on our platform. Prolog is based on first-order predicate calculus, it allows knowledge items to be represented in terms of logic-style facts and rules, and a built-in theorem prover drives the execution of Prolog queries.

The TOY program's internal organization into source files reflects the three different levels (see Figure 1) on which text-based NLP applications can be based. In terms of that figure, processing at a "deeper" level necessitates all components of "shallower" levels.

In this paper, we describe this infrastructure and how it was adapted to a variety of applications. Section 2 gives a brief overview of the infrastructure. Section 3 presents the applications based on it.

1 Infrastructure

The TOY platform is formed of a lexicon that contains most of the Turkish morphemes (either root or suffix), a Turkish morphological analyzer/generator, a DCG parser/generator for Turkish, and a semantic processor which interfaces the aforementioned subunits with the underlying knowledge base for knowledge addition and extraction.

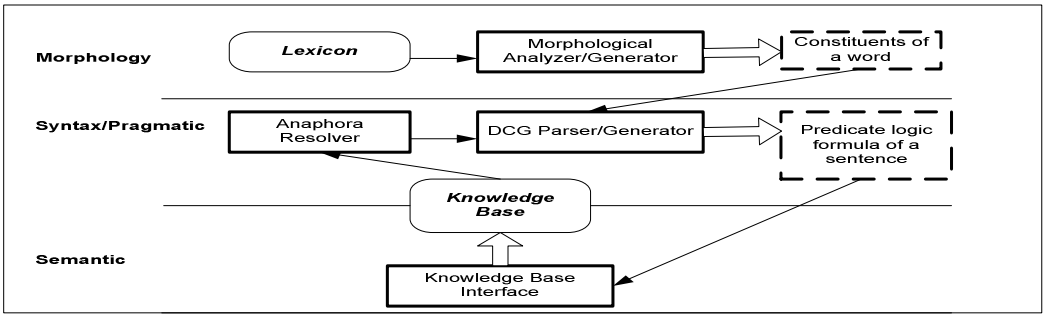

Figure 1. TOY's Internal Organization.

1.1 Lexicon

A complete lexicon is supposed to contain entries for all morphemes (meaningful units that make up words) for the language in question. There are two kinds of morphemes: roots and affixes. In Turkish, all affixes follow the root, that is, they are suffixes. Our lexicon contains entries for over 29000 roots and 157 suffixes. A single morpheme may have more than one entry, corresponding to its different allomorphs. A morpheme definition example for the word "çocuk" ("child") is shown in Figure 2.

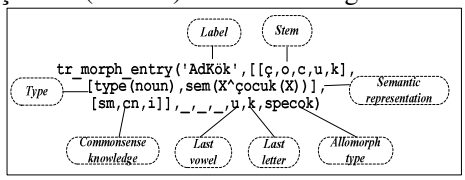

Figure 2. Morpheme Definition.

The semantic representation slot gives a description of the contribution of the morpheme to the meaning of full-size sentences in which it appears: Since the meanings of sentences are represented by predicate logic formulas in our setup, roots contribute "partial" versions of such formulas, with "holes" to be filled by the contributions of the other words of the sentence. For instance, the semantic representation entry of the noun "çocuk" is *çocuk(_)*. In the sentence "Ali çocuktur" ("Ali is a child"), the value of the missing argument is supplied by the name Ali, resulting in the formula *çocuk('Ali')* for the overall sentence.

For noun entries, the commonsense knowledge slot contains a pointer to the location of the thing described by this noun in the taxonomy tree (see Figure 4) used by the program. (The entry for "çocuk", for instance, indicates that it can be reached by descending from the root along the "concrete entity"-"animate-"human being" arcs.) For verbs, this slot contains a set of restrictions on the various argument slots of the verb. As an example, the agent of the verb "ye-" ("eat") is restricted to be a living thing, and its theme is restricted to be a solid.

1.2 Morphological Analyzer/ Generator

In the infrastructure, possible legal orderings of Turkish morphemes are represented by a large finite state diagram, a small part of which can be seen in Figure 3. The morphological component of the platform is a finite state transducer that makes use of the lexicon for traversing the arcs of the diagram (adopted, with changes, from Kemal Oflazer's work on Turkish morphology (Oflazer 1993)) to associate a character string with the list of meaning contributions of its morphemes. This traversal is complicated by the vowel harmony constraint of Turkish morphology. This rule means that, when adding a suffix to a word, the allomorph to be added is a function of the last vowel of the word to be extended. For instance, the plural suffix has the two allomorphs "-ler" and "-lar". The Turkish word for "children" is "çocuklar", while "olives" is "zeytinler", since "back vowels" like "a" and "u" require a back vowel in the suffix, whereas "front vowels" like "e" and "i" require a front vowel there. The program keeps track of the vowels during the transduction to enforce these constraints. Words of foreign origin which violate vowel harmony are flagged appropriately in the lexicon.

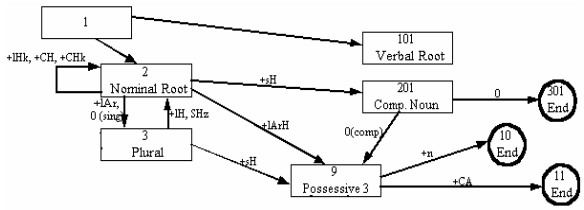

Figure 3. A Subgraph of the Morphological FST Employed by TOY.

Like most Prolog predicates the morphological component is reversible, that is, the same piece of code can be used both to analyze a given word to obtain its underlying constituents, and to generate a word when given a list of such constituents. Another built-in feature of Prolog makes it very easy for the program to compute all results associated with a particular input when more than one legal output is possible, as in the case of the analysis of the morphologically ambiguous word "yedi", where our

morphological parser produces, through backtracking, two alternative analyses: the third-person/singular past tense inflection of the verb "ye-" ("eat"), and the Turkish number word for "seven".

1.3 DCG Parser/Generator

We encoded a subset of the syntax rules of Turkish in Prolog's DCG notation. The DCG formalism allows the computation of the meaning formula of a constituent to be performed in parallel with its syntactic parsing; each DCG rule is written to indicate how the partial meanings of the elements on its right-hand side fit together to produce the semantic expression for the constituent on the left-hand side.

Certain language constructs correspond naturally to the notion of quantification in predicate calculus. For instance, the sentence "Bir çocuk zeytin yedi." ("A child ate (an/some) olive(s)") can be represented by the logical formula

$\exists X \exists Y($ çocuk$(X) \wedge$ zeytin$(Y) \wedge$ ye*(Event, X,Y, Location, Time, Goal, Source, Instrument, definite_past, none, positive))*.

In the Prolog program, existentially quantified expressions like this one have the form *some(X,Restrictor,Scope),* where X is the quantified variable, and Restrictor and Scope are the two sides of the conjunction (Covington, 1994)

*some(X,*çocuk*(X),some(Y,*zeytin*(Y)*,ye*(EventMarker, X,Y,Location,Time,Goal,Source,Instrument, definite_past,none,positive)))*

The successful DCG parsing of a sentence also results in a field being instantiated to a symbol representing the sentence's mood. Possible values for the mood field are "statement," "yes_no_question," and "wh_question."

This component of the program is also designed to be reversible, that is, it can produce the corresponding sentence when given a logical formula, but yet another peculiarity of the language complicates the solution: Turkish word order is (almost) free, which basically means that the sentence constituents can be shuffled around without changing the meaning. Therefore, a single semantic formula usually corresponds to several different sentences, even without taking synonymity of words into account. Our software has the capability of producing multiple alternative sentences as output in such cases.

1.4 Anaphora Resolver

In general, the full-scale processing of all but very simple sentences necessitates information that is not present in the sentence itself, the most obvious examples being question sentences. This additional information is either pre-encoded in the knowledge base as part of a big store of commonsense knowledge, or, when the agent is involved in a dialogue or a story understanding task, it is gleaned from the other sentences in the input.

One example where the computation of the meaning of a declarative sentence requires access to knowledge obtained from previous sentences is the process of anaphora resolution. Resolving an anaphor is the job of finding out which discourse marker (unique internal name) to use for the entity referred to by this phrase in the knowledge base. There is no "correct" algorithm for this task because of the inherent ambiguity of natural language (Lenat 1995). Our resolver selects a discourse marker for an anaphoric reference making use of the taxonomy tree (see Figure 4), semantic type information in its dictionary, pointers to the locations in this tree, and the positions of the original referents in their sentences.

Our resolver treats only definite clauses as anaphors and resolves direct anaphors. Gelbukh and Sidorov (1999) propose ways of solving indirect anaphors.

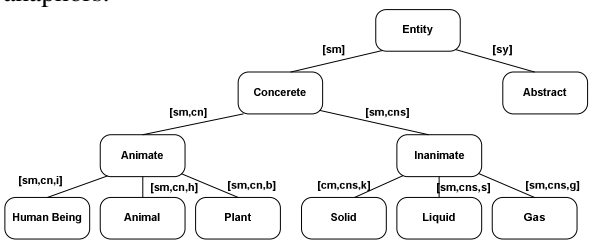

Figure 4. TOY's Taxonomy Tree.

An anaphor and its antecedent can be related if semantic type of the anaphor contains the semantic type of the antecedent or vice versa or their types intersect. Resolution of indirect anaphors will be added to TOY's anaphora resolver in the future. This taxonomy tree will also be used for this purpose.

1.5 Knowledge Base Interface

This module translates predicate logic formulas created by the DCG parser to Prolog facts and rules. As an example, the sentence "Ali çocuktur" ("Ali is a child") is eventually translated to the Prolog fact *çocuk('Ali')*, whose form enables it to take part in automatic proofs involving this knowledge item when necessary. In general, nouns and adjectives are represented as single-argument predicates standing for the invoked property.

Verbs other than "to be" have a considerably more complicated representation. The Prolog equivalent of the sentence *"Ali gitti"* ("Ali left") is

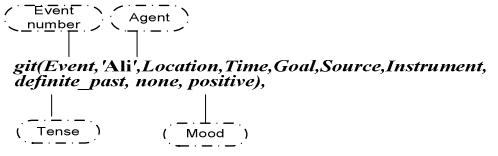

Figure 5. Prolog Representation Example.

Of course, from the point of view of the computer, (or, for that matter, of anybody who cannot speak Turkish,) a formula like *çocuk('Ali')* is just as opaque as *"Ali çocuktur"*. When we look up a strange word in the dictionary, we comprehend its meaning by mentally linking it in appropriate ways to the words appearing in its description. If a sufficiently large subgraph of this network of concepts that exists in our minds is replicated in the computer, it would be able to give the same response to an input sentence as a human utilizing the same network. For instance, the Prolog rule

çocuk(X):- insan(X), küçük(X).

(where "insan" means "human" and "küçük" means "small" in Turkish) relates these three concepts in a way similar to the picture in most people's minds.

The translation of a Turkish sentence to the corresponding predicate logic formula by the DCG rules is just an intermediate step in the processing of that sentence. "Understanding" a sentence necessitates a computation involving both this formula and the current contents of the knowledge base, possibly resulting in a change to the knowledge base, and the generation of an appropriate response.

Skolemization is used in the automatic transformation of the logical formulas of declarative sentences to actual Prolog code by means of replacing all the existentially quantified variables by special expressions called Skolem functions. The purpose of this operation is to assign a discourse marker to every entity which is mentioned but not named in the sentence. These markers are the atomic symbols used by the computer to model the world being described and referred to during the conversation, and keeping track of them is an essential part of the dialogue processing task.

For wh-question sentences, the DCG parser creates formulas in the form of Prolog predicates. For instance, the sentence "Kim zeytin yedi?" ("Who ate (an/some) olive(s)?") is translated to the formula

*which(X,*insan*(X),some(Y,*zeytin*(Y),*ye*(Event,X,Y,*
Loc, Time,Goal,Source, Instrument,
definite_past, none, positive))

whose form matches the already available logic program *which(Item,Property1*$_{Item}$*,Property2*$_{Item}$*)*. See the next section for a discussion of these "question-word" routines.

2 Applications Based on TOY

In this section, we will present some applications that were developed using the TOY infrastructure. Each subsection will briefly explain the application, the TOY components used, and the modifications done on the infrastructure.

2.1 Conversational agent – TOYagent

Smith (1994) classifies dialogue styles that can be adopted by the computer during human-computer interaction into four modes, depending on the degree of control that the computer has on the dialogue: Directive, suggestive, declarative, and passive. TOYagent's original approach mostly suits the passive mode, where the user has complete control, and the computer passively acknowledges user statements, and provides information only as a response to direct user requests.

TOYagent (Demir 2003) enables users to make on-line additions to the lexicon without the need to know Prolog. When faced with a word that it is unable to parse morphologically, TOYagent engages in a (mostly menu-driven) subdialogue with the user to identify the root, category, and morphophonemic properties of the word, and adds the appropriate entries to the lexicon. The meanings of these new words can be incorporated to the system by the logic program synthesis facility, which enables the user to provide natural language descriptions for new predicates in terms of existing predicates. These descriptions are automatically converted to Prolog clauses and added to the knowledge base of the program for future use.

The original dialogue algorithm embedded in TOYagent can be summarized as follows:

1. Read a sentence (this may cause a "word learning" subdialogue if one or more words in the sentence cannot be parsed by the morphological analyzer)
2. Analyze the sentence using the DCG parser, resolving anaphors if necessary. If the syntactic parse is unsuccessful, report this to the user and GOTO 1.
3. If the mood is "statement", then the user is making a declarative statement; use the built-in theorem prover to try to prove the logical formula corresponding to the sentence. There are two possibilities: (In the following, all the "canned" responses are in Turkish, of course.)
   a. If the formula can be proven using the current contents of the knowledge base, the information contained in the sentence is already there; respond with "Thanks, I know that"
   b. If Prolog fails to prove the formula with its current knowledge, then negate the formula and try to prove this negation. There are two possibilities:
      i. If this new formula can be proven using the current contents of the knowledge base, the information contained in the sentence is contradictory with what we already know; respond with "I do not think so"

ii. If Prolog fails to prove this new formula with its current knowledge, create the necessary discourse and event markers and assert the Prolog clauses representing the input sentence to the knowledge base, responding with "Thanks for the information"

4. If the mood is "yes_no_question", the user has asked a yes-no question; use the built-in the prover to try to prove the sentence's logical formula. There are two possibilities:
   a. If the formula can be proven using the current contents of the knowledge base, respond with "Yes"
   b. If Prolog fails to prove the formula with its current knowledge, then negate the formula and try to prove this negation. There are two possibilities:
      i. If this new formula can be proven using the current contents of the knowledge base, respond with "No"
      ii. If Prolog fails to prove this new formula with its current knowledge, respond with "I do not know."
5. If the mood is "wh_question", the user has asked a wh-question; use the built-in theorem prover on the sentence's logical formula. The associated program of each question word scans the knowledge base and produces the relevant answer. The answer can be printed out directly, or, if required, in the form of a grammatical sentence generated by a procedure that first prepares a new logical formula from the produced knowledge items and then uses the syntax and morphology components to form the statement corresponding to this formula. GOTO 1.

The following conversation fragments, in which user entries are shown in boldface, illustrate several aspects of TOYagent. (The English translations are not part of TOYagent's input-output, and have been added "manually.")

As an example to "online" learning of lexical entries, we deleted the word "ana" ("mother") from the lexicon, and carried out the following dialogue with the program:

Her ana güzeldir. (Every mother is beautiful)
"ana" kelimesini bilmiyorum. (I do not know the word "ana")
Kelimenin kökü nedir? (What is the stem of this word?)
1: a
2: an
3: ana
Hangisi: 3.
(Please enter: 3*.)*
"ana" kelimesinin tipi nedir? (What is the type of the word "ana"?)
1: Cins isim (1: Common noun)
2: Sıfat (2: Adjective)
3: Özel isim(3: Proper noun)
Hangisi: 3.
(Please enter: 3*)*
Sözlüğe eklendi. (It has been added to the lexicon)
"ana" ne demek? Anlatır mısın? (Could you explain the word "ana"?)
Çocuğu olan bir bayan anadır. (A female who has a child is a mother)
Teşekkürler, öğrendim. (Thanks for the information)

Since the unknown word could have appeared in an inflected form in the input sentence, TOYagent first asks about the actual stem. A Prolog rule corresponding to the relevant universally quantified logic formula is prepared and asserted for each of the user's sentences read in this dialogue. The last line is the program's response to the original input sentence.

Ayşe bir anadır. (Ayşe is a mother)
Teşekkürler, öğrendim. (Thanks for the information)
Ayşe güzel midir? (Is Ayşe beautiful?)
Evet. (Yes)
Neden? (Why?)
Her ana güzeldir. (*Every mother is beautiful*)
Ayşe anadır. (*Ayşe is a mother*)

The affirmative answer requires stepping through the implication translated to Prolog during the processing of the original user entry. The question "why?" is answered by translating the Prolog formulas used for the previous answer back to sentence form.

Definite noun phrases are treated as anaphors:

Canan küçük bir çocuk. (Canan is a small child)
Teşekkürler, öğrendim. (Thanks for the information)
Çocuk kahvaltıda zeytin yedi. (The child ate olives for breakfast)
Teşekkürler, öğrendim. (Thanks for the information)
Kim zeytin yedi? (Who ate olives?)
Canan zeytin yedi. (Canan ate olives)
Kaç kişi kahvaltıda zeytin yedi? (How many people ate olives for breakfast?)
Bir kişi kahvaltıda zeytin yedi. (One person ate olives for breakfast)

The definite noun phrase in the second user entry ("the child") is matched to the most recently mentioned child. As mentioned earlier, question words have small Prolog programs corresponding to them. The answer extracted from the knowledge base is presented in the form of a grammatically correct sentence. (The fact that every child is also a person is one of the commonsense items that have been preencoded in the knowledge base.)

A rudimentary capability of commonsense reasoning about time is implemented: The "time" argument in verb predicates has a substructure with slots for the beginning and ending points of the interval corresponding to the event. (In the present version, only a small subset of the verbal lexicon entries have their time subslots manually encoded for this purpose.) Hours are used as the unit interval.

Kemal küçük bir çocuk. Bütün küçük çocuklar 10 saat uyurlar. (Kemal is a small child. All small children sleep for 10 hours)
Teşekkürler, öğrendim. (Thanks for the information)
Kemal saat 23'te uyudu. (Kemal fell asleep at 23 hours)
Teşekkürler, öğrendim. (Thanks for the information)
Kemal ne zaman uyudu? (When did Kemal fall asleep?)
Kemal yirmiüçte uyudu. (Kemal fell asleep at twenty three)
Kemal ne zaman uyandı? (When did Kemal wake up?)
Kemal dokuzda uyandı. (Kemal woke up at nine)

Note that the program is able to do the "modulo 24" calculation required for producing the appropriate answer.

To find pronominal references in the absence of gender information, the semantic network is utilized. In the following excerpt, the pronoun "o" ("he/she/it") is correctly deduced to correspond to "çay" ("tea"), since the network does not allow "Kemal", a human name, to be the agent of the word "bit-" ("to be consumed entirely"), which can have only inanimate material at that role.

Kemal kahvaltıda ne içti? (What did Kemal drink for breakfast?)
Bilmiyorum. (I do not know)
Kemal çay içti ise o bitmiştir. (If Kemal drank tea, (he/she/it) must have been consumed entirely)
Teşekkürler, öğrendim. (Thanks for the information)
Kemal çay içti. (Kemal drank tea)
Teşekkürler, öğrendim.(Thanks for the information)
Çay bitmiş midir? (Has the tea been consumed entirely?)
Evet (Yes)

The latest release of TOYagent (Öğün 2003) is able to manage conversations with multiple agents, can adapt different "attitudes" about whether to believe what a user says depending on the user's profile, and has the capability of detecting and pointing out inconsistencies among the statements made by different users. This version also supports an optional "inquisitive" dialogue mode, where the computer questions the user about the values of currently empty slots in the verb predicates corresponding to previous user statements.

2.2 Turkish Natural Language Interface For SQL Queries (NALAN-TS)

NALAN-TS (Maden, Demir and Özcan 2003) is a Turkish natural language query interface for SQL databases, formed of a syntactic parser, semantic analyzer, meaning extractor, SQL constructor and executer. It is a dictionary based application and includes Turkish and database dictionaries.

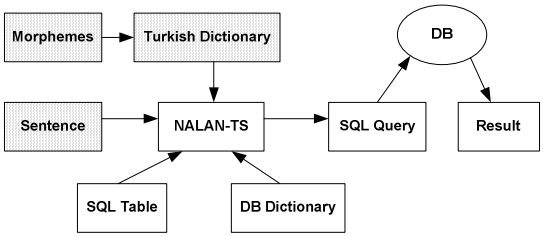

Figure 6. NALAN-TS Flow Diagram.

The shaded modules in Figure 6 were taken completely from the TOY infrastructure, except for a few modifications like the addition of new Turkish syntax rules and a different format for the semantic representation of the words in the dictionary. TOY's knowledge base interface is taken as the basis by NALAN-TS.

2.3 Turkish Speaking Assistant -TUSA

TUSA (Şeker, 2003) is a natural language interface for an online personal calendar. The morphological analyzer/generator of TOY was taken as a basis in this project with modifications made for utilizing.

2.4 Generating Java Class Skeleton Using a Natural Language Interface- TUJA

TUJA (Özcan, Şeker and Karadeniz 2004) is a natural language interface for generating Java source code and creating an object-oriented semantic network. This program uses TOY's morphological analyzer/generator as the starting point.

2.5 Other Applications

Ballhysa (2000) used TOY to produce a prototypical sentence-level translator between Albanian, English, and Turkish. (To our knowledge, this is the first NLP work ever done on Albanian) Dutağacı (2002) used the morphological component to tag a Turkish corpus of nearly ten million words to collect statistics and compared the performance of an N-gram model of speech recognition based on morphemes with those based on words or syllables. Tekeli (2002) made use of the word-level components to build an "ELIZA-like" (Covington 1994) dialogue program which caricaturizes Fatih Terim, a famous soccer coach and an idiosyncratic Turkish speaker.

The program's "bag of tricks" includes coming up with rhyming responses to user sentences. Bilsel (2000) developed a "poem expert" for analyzing Turkish folk poems for their rhyme and meter properties, a demanding task which is part of the high-school curriculum in Turkey.

Conclusion

Our work on TOY is continuing on many fronts: The DCG component is currently being extended to cover both a bigger subset of Turkish syntax, and some types of agrammatical sentences. We hope that TOY will be useful in the development of many other applications in the near future.

References

Can Tekeli 2002. *TERIM_SON.* B.S. Thesis, Department of Computer Engineering, Bogazici University.

Douglas Lenat 1995. *CYC: A Large-Scale Investment in Knowledge Infrastructure*. In "Comm. ACM, 38/11", pages 33-38.

Eda Bilsel 2000. *Poem Analyzer*, B.S. Thesis, Department of Computer Engineering, Bogazici University.

Elton Ballhysa 2000. *Albanian-Turkish-English Translator.* B.S. Thesis, Department of Computer Engineering, Bogazici University.

Ender Özcan, Şadi E. Şeker and Zeynep. I. Karadeniz 2004. *Generating Java Class Skeleton Using A Natural Language Interface*. In "First International Workshop on Natural Language Understanding and Cognitive Science-NLUCS".

Fatih Öğün 2003. *Design and Implementation of an Improved Conversational Agent Infrastructure for Turkish.* M.S. Thesis, Department of Computer Engineering, Bogazici University.

Helin Dutağacı 2002. *Statistical Language Models for Large Vocabulary Turkish Speech Recognition.* M.S. Thesis, Department of Computer Engineering, Bogazici University.

Ibrahim Maden, Şeniz Demir and Ender Özcan 2003. *Turkish Natural Language Interface for Generating SQL Queries.* In "TBD 20. Ulusal Bilişim Kurultayi".

Kemal Oflazer 1993. *Two-Level Description of Turkish Morphology*. In "Proc. Second Turkish Symposium on Artificial Intelligence and Neural Networks", Bogazici University Press, pages 86-93, Istanbul.

Michael A .Covington 1994. *Natural Language Processing for Prolog Programmers.* Prentice Hall, Englewood Cliffs, NJ.

Özlem Çetinoğlu 2001. *A Prolog Based Natural Language Processing Infrastructure for Turkish.* M.S. Thesis, Department of Computer Engineering, Bogazici University.

Ronnie W. Smith 1994. *Spoken Variable Initiative Dialog: An Adaptable Natural-Language Interface. In "*IEEE Expert: Intelligent Systems and Their Applications 9/1",pages 45-50.

Şadi E. Şeker 2003. *Design and Implementation of a Personal Calendar with a Natural Language Interface in Turkish.* M.S. Thesis, Department of Computer Engineering, Yeditepe University.

Şeniz Demir 2003. *Improved Treatment of Word Meaning in a Turkish Conversational Agent.* M.S. Thesis, Department of Computer Engineering, Bogazici University.