Massachusetts Institute of Technology
Department of Electrical Engineering Computer Science
6.863 Natural Language Processing
Final Project: Turkish Morphological Analyzer

by Eren Sila Sayan, Burkay Gur
August 17, 2011

1. Introduction

In this project, we have implemented a morphological analyzer for the Turkish language using the KIMMO architecture. Due to its agglutinative nature, Turkish is a very interesting language in terms of its morphological features. Single words (single words could also be whole sentences) can get extremely long by adding a number of suffixes. For example: “kitap” (book), “kitaplar” (books), “kitaplarım” (my books), “kitaplarım da” (in my books), “Kitaplarım da dair.” (It is in my books.) This and other many interesting properties of the Turkish language make it a great challenge for creating a morphological analyzer. In the following sections, we will be explaining how the KIMMO system operates, what automata are and how each automaton handles the morphological rules, what is the lexicon and how the lexicon works and what kind of issues we have run into while implementing this analyzer. Finally, we will show how the system could be extended to handle more cases.

2. How the System Operates

In order to implement the morphological analyzer, we have used a Python port of the KIMMO system. KIMMO uses a two level approach to analyze an input. According to this system, an input has an underlying and a surface form. The surface form is how we see the word in the dictionary while the underlying form is
what is structurally going on in the word when it is under certain changes. For example, the word “kitap” (book) becomes “kitabım” (my book):

\[
\begin{align*}
\text{kitab0ım # (surface form)} \\
\text{kitab + I m # (underlying form)}
\end{align*}
\]

To explain the details of the system further, we will go over the special characters that appear in this example. The “0” character in the surface form means empty character. This is to keep the same length principle, which requires the underlying and the surface form to be the same character length for the machine to operate correctly. Therefore, the “+” surfaces as a 0. Next, the capital I in the underlying form (any capital letters as we will see in the following sections), represents a character that can surface as multiple characters. In this example it surfaces as an “ı”, however in other examples, it could surface as an “i”. (Note: it is important to mention the special characters in Turkish: “ç ğ ö ş ü”). Finally “#” represents the end of a word.

Now that we are familiar with the notation, we can explain how KIMMO works. KIMMO requires two files in order to operate, a .lex file (for the lexicon) and a .yaml file (for the automata). The automata in the .yaml file consist of rules for each morphological concept (such as vowel dropping, consonant harmony), made of states and the conditions for transitions between those states. The lexicon, on the other hand, can be seen as a road map of valid structural changes that an input can go through. KIMMO can either recognize or generate words, “kitaplar” (books) would be recognized as “kitab (book, NounRoot) + lEr (PluralSuffix)” and the input “kitab + lEr” would generate “kitaplar”. While generating, KIMMO only goes through the .yaml file (the automata) and applies the rules of underlying to surface forms. While recognizing, KIMMO uses the lexicon as guidance and figures out which words and the suffixes in the lexicon could surface as the given input.
Further details about the automata and the lexicon will be revealed in the following sections.

2.1. Additional Lexical Characters

In order to incorporate the special Turkish characters “ç ı ğ ö ş ü” into the system, we have notated them as “c_ i_ g_ o_ s_ u_” respectively. This is because the Python implementation of KIMMO did not handle Unicode characters. We have also introduced special characters for the underlying form that are allowed to surface as multiple lexical characters. These are “XI XI_ XU XU_ Y E I D C G”. “XI XI_ XU XU_” can appear in the underlying forms of words, whereas “Y E I D C G” only appears in suffixes.

“XI XI_ XU XU_” can surface as the null character “0” or “i i_ u u_” respectively. For example, “alXI_n” (forehead) surfaces as “ali_n” when there is no suffix, but “alni_m” (my forehead) when +Im is attached (notice the “XI_” between “I” and “n” being dropped). q.v. Section 3.6.

“Y” can surface as the null character “0” or “y”. In our implementation, it is mentioned in two automata, Vowel Harmony and Buffer Consonant. For suffixes such as +YIm (I am), +YE (to) Y functions as a potential buffer consonant “y” or a null character depending on what the word ending is whether a vowel or a consonant. In Vowel Harmony, “Y” is allowed to surface as a “0” in order to keep track of the last letter if it happened to be a vowel. q.v. Section 3.1 and 3.2.

“E and I” can surface as the null character “0” or “e a” and “i_ i u u_” respectively. For Vowel Harmony, “E and I” surfaces as “e or a” and one of “i_ i u u_” depending on the previous vowel in the word. For Vowel Dropping, “E and I” might surface as “0” if the word it is attached to has a vowel at the end. q.v. Section 3.1 and 3.5.
“C, D, G” can surface as “c c_”, “d t”, “k g g_” respectively, depending on whether the word they are attached to are ending with voiced or voiceless consonants, therefore maintaining the Consonant Harmony rule in the Turkish language. q.v. Section 3.3.

2.2. Defining the Subsets

The Python implementation of KIMMO allows us to use subsets in order to have easy access to a group of characters, without writing duplicate code. The additional subsets we have created are:

- CONS: “b c c_ d f g g_ h j k l m n p r s s_ t v y z D Y C G”
- VOWEL: “a e i _ o o _ u u _ E I”
- VOICELESS: “f s t k c_s_h p”
- VOICED: “a b c d e g g_ i i j l m n o o r u u_ v y z”
- NOT_STOP_CONS: “f g_ h j k l m n r s s_ v y z”
- PCTK: “p c_t k”

3. The Automata

3.1. Vowel Harmony

Turkish has certain constraints on which vowels can be found near each other. Suffixes change to ensure these constraints are satisfied.

We noticed that for the purposes of building an automaton for Vowel Harmony, Turkish vowels can be divided into the four following categories:

- oustate: The automaton enters this state if the input is an “o” or “u”.
- o_u_state: The automaton enters this state if the input is an “o_” or “u_”.
- eistate: The automaton enters this state if the input is an “e” or “i”.
The automaton enters this state if the input is an “a” or “i_”.

These four categories of vowels form the backbone of our Vowel Harmony automaton, because the rules of Turkish vowel harmony dictate how an undetermined high vowel, I, and an undetermined low vowel, E, can lawfully surface following a vowel that belongs to any of these categories:

<table>
<thead>
<tr>
<th>If last vowel is....</th>
<th>An “E” can surface as...</th>
<th>An “I” can surface as...</th>
<th>An “Y” can surface as...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“o” or “u”</td>
<td>0,a</td>
<td>0, u</td>
<td>0</td>
</tr>
<tr>
<td>“o_” or “u_”</td>
<td>0,e</td>
<td>0, u_</td>
<td>0</td>
</tr>
<tr>
<td>“e” or “i”</td>
<td>0,e</td>
<td>0,i</td>
<td>0</td>
</tr>
<tr>
<td>“a” or “i_”</td>
<td>0,a</td>
<td>0,i_</td>
<td>0</td>
</tr>
</tbody>
</table>

(The state diagram for this automaton is not included, because due to its large number of states and transitions, it is best to investigate it in the GUI.)

3.2. Buffer Consonants

Turkish has certain rules about adjacent vowels. In certain cases, adjacent vowels must be avoided and one way to do so is to insert what’s called a “buffer consonant”. Turkish has the buffer consonants “y”, “s”, “s_” and “n”, but for this lab we have only considered “y”, which we have represented with a lexical “Y”.

We designed the automaton for “Buffer Consonants” with only two states, because all that matters is whether the most recently seen letter is a vowel or a consonant.

If the word ends with a consonant, a suffix that begins with a “Y” will have the “Y” surface as a 0, because a bugger consonant is not needed. The “Y” will surface as a “y” if the word ends with a vowel.
3.3. Consonant Harmony

“Consonant Harmony”, in Turkish, refers to compliance with certain constraints about having voiced versus voiceless consonants adjacent to one another. A suffix beginning with a consonant attached to a word that ends with a consonant must adapt so that if the word ends with a voiceless consonant the suffix must begin with a voiceless consonant, and if the word ends with a voiced consonant the suffix must begin with a voiced consonant.

We used “D”, “C” and “G” as lexical representations for undetermined consonants that can take either the voiced or voiceless version of a certain consonant.
Similarly to the “Buffer Consonant” automaton, we only have two states in the “Consonant Harmony” automaton. We only need to know whether the preceding consonant is a voiced or voiceless consonant to determine how a given undetermined consonant among “D”, “C”, and “G” must surface.

“Consonant Harmony” Automaton
3.4. **Final Voiced Stop**

Turkish doesn’t allow a word to end in “b”, “c”, “d”, or “g” (called voiced stops). If a word ends in any of these letters, the voiced stop surfaces as its voiceless counterparts. The same happens if a suffix beginning with a consonant comes after a word ending in a voiced stop. There are also certain rules surrounding whether a “G”, when necessary, surfaces as a “k” or a “g” (soft g).

The “Final Voiced Stop” automaton embodies all these rules and is quite complicated, so we haven’t included the state diagram here. We suggest it be explored in the GUI.

The main idea surrounding the automaton is that given whether the most recently observed letter is a vowel, a voiced stop (except ‘g’), an unvoiced stop (i.e. a consonant that is not b,c,d,g), or a “g”, we form different expectations for what kind of letter should follow. The automaton keeps running if the expectation is met, rejects if not. One important point to emphasize is that “G” is treated differently than the rest of the lexical forms “B”, “C” and “D”. “G” has two different voiceless versions and more complex rules govern the changes that happen to it.

3.5. **Vowel Dropping**

While one way to avoid adjacent vowels in Turkish is buffer consonants, another way is “vowel dropping”. “Vowel Dropping” dictates that if a suffix (without a buffer consonant) beginning with a vowel is added after a word ending in a vowel, the vowel at the beginning of the suffix drops. This rule can be represented with a simple automaton consisting of two states (in addition to the ‘start’ and ‘reject’ states).
All we need to keep track of is whether the most recently seen letter is a vowel or consonant. Then, if we encode the first vowels of suffixes beginning with a vowel as an undetermined lexical letter (i.e. “E” or “I”), then we can simply check whether these vowels have surfaced as a ‘0’ or an actual vowel. If the word ends in a vowel, an “E” or “I” at the start of a suffix should surface as a ‘0’.

“Vowel Dropping” Automaton
3.6. **Vowel Insertion**

Vowels may be inserted into a Turkish word if the word ends in a cluster of consonants that violate the rules dictating what group of consonants can end a word.

We managed to represent this rule with a relatively simple automaton by having some of the lexicon shoulder some of the weight. Those words that go through “Vowel Insertion” are handled in a special way in the lexicon. Namely, the vowel that is inserted into the word is represented in a special lexical form that indicates the fact that that particular vowel is not present in the underlying stem of the word, but must appear in the word’s surface form.

The word for “son”, for example, is really “og_l”. However, this cluster of consonants is not allowed to end a word in Turkish, so on the surface, the word appears as “og_ul”, with the vowel “u” inserted between the last two consonants. In the lexicon for our system, we represent this word as “og_XUl”. The “XU” lexical form indicates that this is an inserted “u” that is not present in the underlying stem of the word.

If on the surface, an “inserted vowel” surfaces as 0, then the word must have received a suffix beginning with a vowel. While if the inserted vowel surfaces as the associated vowel, the word must be either without a suffix, or with a suffix beginning with a consonant. These conditions are what our automaton checks for.
3.7. **Extra Automaton**

We added two automata in addition to the rules we were asked to handle. The additional automata encode the following rules:

i) No word can begin with “g_”

ii) “g_” cannot follow a consonant

Both of these automata are straightforward and their operation is self-evident from the state diagrams.
No word can begin with “g”

“g_” cannot follow a consonant
4. The Lexicon

Lexicon is the roadmap that KIMMO’s recognition system uses. It includes all of the possible prefixes and suffixes (Turkish has suffixes most of the time), as well as all the root words that the morphological analyzer is supposed to be able to recognize. The lexicon consists of a tree structure that allows the root word to traverse this tree in order to morph to other forms of that root word within the rules of the language. To demonstrate how the lexicon works, here is a simple snippet from our implementation of the Turkish lexicon:

```
Begin: NOUN VERB ADJECTIVE

AfterNoun: POSSESIVE CASES PLURAL WHILE PAST_TENSE
NONVERBAL_PERSONAL_CONJ WITHOUT End

AfterWithout: End

NOUN:
el  AfterNoun Noun(hand)
it   AfterNoun Noun(dog)

WITHOUT:
+sz Iz AfterWithout +WithoutSuffix

When the recognizer is analyzing a given word, it starts parsing the word letter by letter from left to right. Therefore the first valid chunk of the given input “elsiz” would be “el”, since “el” is a valid word in the lexicon, given by the lines:

NOUN:
el  AfterNoun Noun(hand)
```
Also, this is a valid beginning of a word, because:

**Begin: NOUN VERB ADJECTIVE**

states that the word type NOUN could be a valid Begin. In order to decide what could follow after nouns, the machinery looks at:

**AfterNoun: POSSESSIVE CASES PLURAL WHILE PAST_TENSE NONVERBAL_PERSONAL_CONJ WITHOUT End**

These are the possible suffixes that could follow a noun. Continuing on the word “elsiz”, we see “siz” and it is under type WITHOUT which is a valid type after NOUN given by the AfterNoun rule. This completes the traversing since we are at the end of the word. Finally we check if the word is allowed the finish after WITHOUT:

**AfterWithout: End**

and this rule suggests that the word is allowed the end.

Even though this is a very simple example, it gives great insight on how the lexicon works. One can think about the lexicon as a high level automaton that manages the order of suffixes with its own states and possible transitions.

In our implementation of the Turkish lexicon, we have included countless suffixes along with their valid transitions. These suffixes include: plurality, cases, possessive, while, without, infinitive, verbal and non-verbal tenses, negation and conditionals. For a list of examples of the words that can be parsed by the system, please see the attachment.
5. System Extendibility

The modular structure of the KIMMO system allows the users to easily extend existing morphological analyzers. An extension could be done in two ways: extending the lexicon and extending the automaton. If we want to add more words to the dictionary under existing word types such as nouns, verbs or adjectives, we can go under the respective type and add the line in the lexicon file:

NOUN:
newnoun  AfterNoun Noun(newnoun description)

If we want to introduce new word types or new suffixes:

NEWSUFFIX:
+newsuffix  AfterNewSuffix Suffix(description)

We will also have to specify after where this suffix could be used, and what could follow this suffix:

AfterWithout: NEWSUFFIX End

AfterNewSuffix: End

In this example, NEWSUFFIX can follow WITHOUT, and it will end the given word. While extending the lexicon, we should pay attention to the underlying structures of the newly added words or suffixes so that they are consistent with the rest of the lexicon.

We can also extend the automaton in order to handle more morphological rules. To add a new automaton, we add a new set of rules to the .yaml file including states and transitions.
6. Issues During Development

During the development of the morphological analyzer, the most challenging issue was implementing Final Voiced Stops and Vowel Insertion. Final Voiced Stops required a lot of debugging and editing, since we had to handle many cases. Specifically, handling the underlying character “C” was the hardest, because it surfaced as “g_” or “k” depending on what was following. The detailed nature of this machinery resulted in a more complicated design. Secondly, as we designed the automaton for Vowel Insertion, we realized that we were introducing too many states and transitions and we decided that it is not an efficient design. We resolved the issue by introducing new underlying characters to the lexicon, namely “XI XI_ XU XU_”, since this morphological rule was a very specific one for exceptional words.

Another issue that would go into the to-do list was the question form suffix. Turkish question suffix “+mI” is written with a whitespace after the word, and this is not yet implemented current KIMMO system. For future, we might be able to write an extension to the KIMMO system in order to handle the whitespace.

Since Turkish had some special characters “ç ı ğ ö ş ü”, that could only be encoded by Unicode, the Python implementation could not handle those characters out of the box. For future work, several modifications can be made to handle those characters, which would yield a better user experience.
7. How to Run the Code

Assuming the user has Python version 2.5, executing these commands within the attached folder using the command line will bring up the KIMMO user interface:

```python
>>> from kimmo import *

>>> k=load("turkish.yaml")

>>> k.gui()
```

![Diagram of the KIMMO user interface](image_url)
From this screen, either the recognizer or the generator can be used. Recognizer will accept an input of the form: “el+lEr+Im+dEn” and generate possible outputs on the box across. The steps for each rule automaton can be selected on respective boxes.

Generator will accept an input of the form: “kitaplar” and return possible parses of the given input. Similarly, the steps for each rule automaton can be selected on respective boxes in order to visualize what is going on underneath the KIMMO system.
Appendix: List of Examples

We already have a file (‘turkish.rec’) containing all the words that our system was required to handle, as well as some others in addition to the requirements. Nevertheless, we thought it useful to include here a list of a few interesting examples beyond the requirements that our Turkish morphological analyzer is able to handle, and to offer short explanations for them.

Additional Examples of Vowel Insertion

alin - forehead

burun - nose

gönül - source and carrier of emotions and desires (closest translation in English might be heart/mind)

karın - belly

Samples of Additional Tenses Covered by Morphological Analyzer

görür (go_ru_r) -

‘[he/she] sees’. This is a form of present tense, however it has a meaning that is more spread out in time -- a sense in which the person being talked about just generally sees, “it’s his thing”. Conjugation available for all 3 singulars and all 3 plurals.
görürse (go_ru_rse) -

“if she sees”. This is the **conditional tense**. Conjugation available for all 3 singualrs and all 3 plurals.

görse (go_rse) -

Again, translates as “if she sees” into English, but **this form of the conditional tense expresses a state of wishfulness or doubtfulness**. Conjugation available for all 3 singualrs and all 3 plurals.

görüyorsunuz (go_ru_yorunuz) -

“you (plural you) are seeing”. We have simply extended the verbal progressive suffix +́yor to cover all conjugations (i.e. for the 3 singular and 3 plural).

görüyorsa (go_ru_yorsa) -

“if she is seeing”. Another form of the conditional tense. (Perhaps could be called **“progressive conditional”**). Conjugation available for all 3 singualrs and all 3 plurals.

görüyörler (go_ru_yorlarsa) -

“if they are seeing”. This is another example of the “progressive conditional”, but we wanted to include this example anyway so that it may be contrasted with the example below. Turkish has a certain order in which suffixes can appear. As this example and the one below (which is incorrect) demonstrates, the plural suffix +́Er must precede the conditional suffix +́sE.
görüyorsalar (go_ru_yorsalar) -

This is an incorrect way of trying to say “if they are seeing”. The order of the suffixes are incorrect and the system correctly rejects this example.

görüyordu (go_ru_yordu) -

“she was seeing”. Past Progressive Tense. Conjugation available for all 3 singulars and all 3 plurals.

görüyorlardı (go_ru_yorlardı_) -

“they were seeing”. Again, this is the past progressive tense but the order of suffixes matter. This is the correct ordering, while the example below has an incorrect order of suffixes.

görüyordular (go_ru_yordular) -

Incorrect way of trying to say “they were seeing”. System correctly rejects this example.

görüyorlardıysa (go_ru_yorlardı_ysa) -

“If they were seeing”. This is another form of the conditional -- could be called the past progressive conditional tense. This example and the few preceding examples demonstrate the agglutinative nature of Turkish. The order of the suffixes, however, matter and the example below has incorrect ordering. Conjugation available for all 3 singulars and all 3 plurals.

görüyordularsa (go_ru_yordularsa) -

Incorrect way of trying to say “if they were seeing”. System correctly rejects this example.
gördülerse (go_rdu_lerse) -

“if they saw”. Past Conditional. Conjugation available for all 3 singulars and all 3 plurals. This example represents the correct ordering of suffixes for the tense, while the example below is incorrectly ordered.

görüyseler (go_rdu_ysele) -

Incorrect way of trying to say “if they saw”. System correctly rejects this example.