Massachvsetts Institvte of Technology
Department of Electrical Engineering and Computer Science
6.863J/9.611J: Natural Language Processing Spring 2018
Lab 1 Main: A Morphological Analyzer for Turkish

Name: Mehmet Tugrul Savran
KerberosID: tugrul
Collaborators: None

Disclaimer: I am a native Turkish speaker.

Problem 1. A brief description of how your system operates.

The system pretty much utilizes a config file that is consumed by a wrapper library named Kimmo, which is the *engine* of our word parsing machine. This config file, named turkish.yaml, is aptly the brain of the 2-stage machine. It is a finite state machine with multiple rules representing distinct phenomena of the spelling change principles of Turkish. The 2nd stage of the machine is the lexicon file with the Turkish vocabulary and groupings of verbs, nouns and adjectives though this 2nd stage is less relevant since we are more concerned with generation rather than recognition. The crucial part is the Finite State Automata (FSA) contained within the yaml file.

Shortly speaking, my FSA consisted of distinct rules for **vowel harmony, buffer consonant, consonant harmony, vowel dropping and 2 exceptions that were later discovered.** These distinct sets of rules each act as a finite state machine dealing with their own "assigned" phenomenon. For example, vowel harmony automata deals with the mapping of the lexical forms I and E into feasible surface options of a,e,i, <i>, o, <o..>, u, <u..> while buffer consonant automata deals with mapping of Y into a y or the null character. Surface forms that finish the matching process at an accepting state for every single rule are returned as possible generations.

Problem 2. What additional lexical (underlying) characters have you introduced (if any)?

I didn't add any lexical characters.

Problem 3. What do your subset definitions represent?

I added the following subsets to my system, as well as the @ subset that contains everything.

- **"VOICELESS": "<c,> f h k p s <s,> t"**

- **"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 <i> o <o..> u <u..>"**
- **"BACKROUND": "o u"**
- **"BACKUNROUND": "a <i>"**
- **"FRONTROUND": "<o..> <u..>"**
- **"FRONTUNROUND": "e i"**

As their names suggest, they represent possible lexical forms grouped according to certain features (I am referring to them as lexical because the union of lexical and surface forms is the lexical set).

Problem 4. What is the purpose of each automaton? Briey, how does each automaton work?

The purpose of each automaton is to filter in possible surface forms for the spelling change phenomenon they're purposed for.

The **vowel harmony automaton** serves to correctly map the lexical forms I and E into feasible surface options of a,e,i, <i>, o, <o..>, u, <u..>.

Buffer consonant automaton deals with mapping of Y into a y or the null character.

Consonant harmony deals with the conversion of the lexical form D into either d or t depending on the type of the last consonant seen (whether it was voiceless).

Vowel dropping automaton deals with the mapping of the lexical form I into a null character if the last character seen was a vowel.

I also added 2 additional automata that *"overfit"* to the exceptions of words ending with g **and** the very specific case of kitab+lEr+Im+YI → kitaplarimi. I would also like to note that I am Turkish and this test case is wrong, because **no word in Turkish ends with a b.** The lexicon verified that "kitab" was indeed registered as the noun that referred to the word book. However, the correct version is indeed **kitap** with a p at the end. So this overfitting, I believe, was redundant. I do not know how kitab was registered to the lexicon as a valid word. Again, no word in Turkish ends with a b. **Not even kebap (so nope, it is not kebab).**

Problem 5. Describe what problems arose in the design of your system.

I experienced numerous issues while implementing the system.

First one was in vowel harmony. My first version produced multiple outputs for gel+Iyor+sInIz because the last "I" was oblivious of the previous I's conversion (the initial version would just go back to the start state after adding the proper version of I). I changed it so that it **goes to the state of proper version of the added character.** For example, if I:u was the last transition, it goes to the state representing back and round vowels instead of going to the start state. This solved the multiple output problem (I anyways learned later that producing multiple outputs was fine…).

The second major issue I experienced happened when integrating the automata. The vowel dropping automata caused the vowel harmony to fail. This was largely due to the fact that the vowel harmony automata didn't keep track of whether the last seen character was a vowel or consonant. It solely changed states according to vowel types and stayed there as long as no new type of vowel was seen. To remedy, I made copies of each state pertaining to the groups of vowels. For example, a copied state pair would be frontroundseen and frontroundseenconsonant. Inside the frontroundseen, it means we literally just saw a consonant so any incoming I should be replaced by a null character. We switch to the frontroundseenconsonant state if we see any consonant while in frontroundseen state. We add the proper version of I in that "consonant" state. I repeated this state pairing process for all the other vowel types. This resolved the vowel dropping integration issue!

Of course, a third issue went up while dealing with the *exceptions* of oku+mEg and gel+mEg . How I resolved it was to literally write a rule to *overfit* to the words ending with a g (actually, no word in Turkish ends with a g). Indeed, **no word in Turkish ends with b, c, d, g, or j.** I still do not know how **kitab** was put there. Speaking of which, the case kitab+lEr+Im+YI caused another exception issue. I resolved the case by also writing another overfitting rule, however as I also mentioned in part 4 no word in Turkish ends with a b. So I found this test case to be ill-formed.

Problem 6. Include a brief discussion on how you would extend your system if one had to add more nouns and verbs

This is more of an engineering vs acquisition question. Most of the verbs or nouns will not bring in exceptions, so solely modifying the lexicon file should suffice in most of the cases. However, <u>quantitatively thinking, adding new nouns and verbs bring a non-decreasing probability distribution of running into exceptions, which can't be handled inside the lexicon</u>. Even if I am not to extend the morphological processes in my system, exceptions will continue to cause issues. For example, in Turkish sen+YE (you + DativeCase) becomes **sana**. **Not sene. (**Actually, "sene" means "year" in Turkish which is super different than sana). This case fails in my current system, because of the current implementation. Hence, the word "sen" would require an additional rule even if we don't change the underlying morphological process. Another issue I know is that currently consonant harmony is ignoring a half of the consonant harmony phenomena of Turkish. The current version handles "consonant hardening" that replaces the consonant d with a t if it comes after a voiceless consonant. However, there is also the extremely important "consonant softening" phenomenon that needs to be handled within the consonant harmony rule. This phenomenon says that any vowel coming after a voiceless consonant *softens* the consonant. For example, yemek+YI becomes yemeği in Turkish, **and not yemeki.** "Ğ" is called the silent g and it extends the spelling of the previous vowel. For example, my name is Tuğrul and it is pronounced as Tuurul. Anyways, without adding any new rule, we could modify the consonant harmony with the new phenomenon.