Emre Yılmaz

1901042606

Gebze Technical University

Computer Engineering

NLP HOMOWORK-1 REPORT

1. Part 1 of Homework (Plural word derivation)
2. Part 2 of Homework (Possessive word derivation)
3. Test and Results

Note:

Finite State Transducer’s .dot files are generated and zipped. I convert them into PNG files and showed them in both notebook and this report. If you want, you can generate this PNG files using this command:

```
dot file.dot -Tpng -o image.png
```

Note:

I need to explain how I understood this homework. I derived correct words with suffixes that is given by user. For example, my input is: “köpek + 3sp”. This means that “derive the word köpek with third singular person possessive suffix and give me ‘köpekleri’ “.

Part – 1 of Homework:

First part is creating plural FST.

FST:

We are going to build a Finite State Transducer to derive Turkish Language plural word. The main problem in this application is determining which suffixes to use after a Turkish word. For example, after "kedi", we must put there "ler" suffix. After "Papağan", we must put there a "lar" suffix.

Let's build the FST.

We have 4 input symbols:

1. sessiz -> (Represents consonent letters)

2. "a,u,o,ı" -> (Represents front vowel letters)

3. "e,ü,ö,i" -> (Represents back vowel letters)

4. "+pl" -> (Represents plural suffix)

We have 5 output symbols:

1. sessiz -> (Represents consonent letters)

2. "a,u,o,ı" -> (Represents front vowel letters)

3. "e,ü,ö,i" -> (Represents back vowel letters)

4. "lar" -> (Represents "lar" suffix)

5. "ler" -> (Represents "ler" suffix)

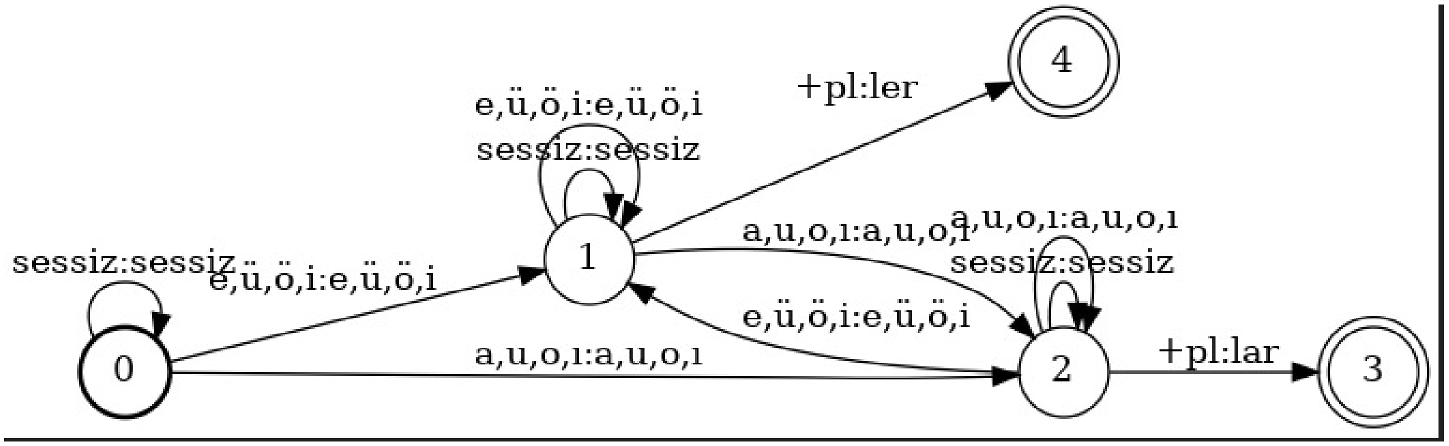
We have 4 states. The main thing in the machine is:

- if we put plural suffix after a front vowel letter, the correct plural suffix must be "lar".
- If we put plural suffix after a back vowel letter, the correct plural suffix must be "ler".

In Turkish, after a front vowel, the suitable plural suffix is "ler"

In Turkish, after a back vowel, the suitable plural suffix is "lar"

Visualized FST



Preprocessing

When we are getting input. There is a small preprocessing step. To bring the input into a form suitable for the fst that I designed, I implemented a function:

1. If the letter is front vowel, I indicate it as "e,i,ö,ü"

2. If the letter is back vowel, I indicate it as "a,ı,o,u"

3. If the letter is consonant, I indicate it as "cnsnt"

The need of this preprocessing step comes from complex visualization of FST. If I take all vowels separately, FST looks very big and complex. So, I grouped them. However, to use this group rules, I need a preprocessing step.

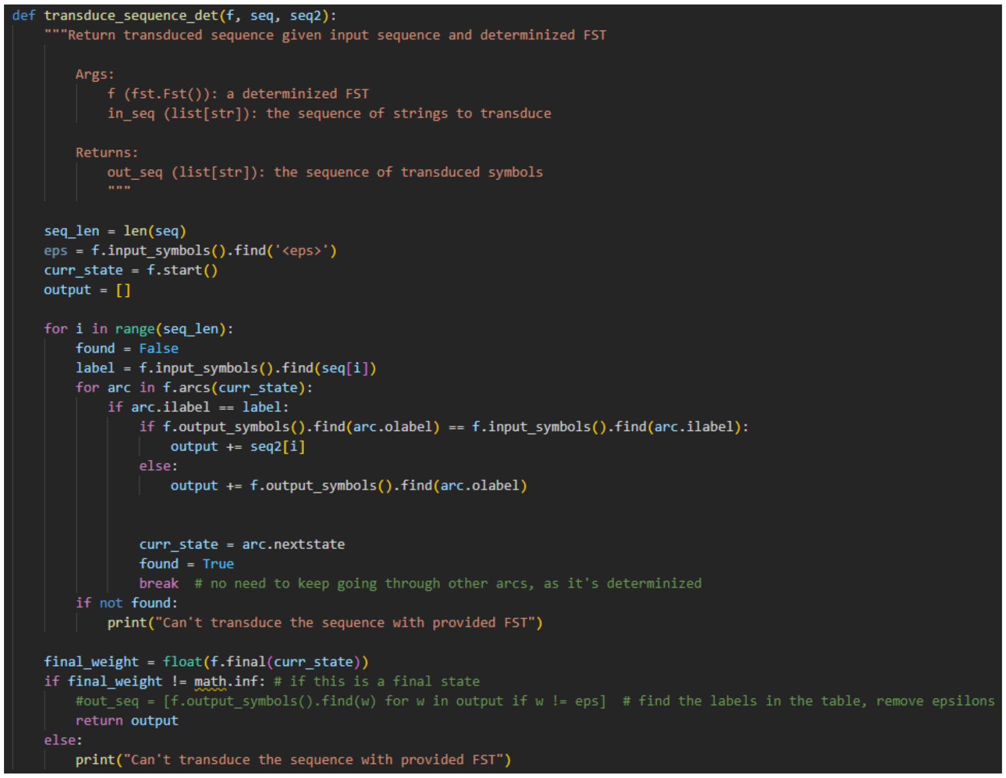
Generating Output Using FST

A function is implemented that allows us to obtain the output we get as the fst states change.

I got that function from Openfst-Python documentation:
https://github.com/edemattos/asr/blob/master/asr_lab5_solutions.ipynb

I modified existing function a little bit. Let me explain what I change:

- I am giving 2 parameter to function instead of 1. One of it is correct form of input for FST, other one is unchanged version of it. The algorithm is:
  - If input and output of a state is the same, give the input as output. Actually here we are solving this problem: If the output is "a,u,ı,o", we should pick the correct vowel of this string. And while picking the correct one, we should find it in the original input.
  - Else, give the original output of the current state.
  - I return the output as a list of string



```
def transduce_sequence_det(f, seq, seq2):
    """Return transduced sequence given input sequence and determinized FST

        Args:
            f (fst.Fst()): a determinized FST
            in_seq (list[str]): the sequence of strings to transduce

        Returns:
            out_seq (list[str]): the sequence of transduced symbols
            """

    seq_len = len(seq)
    eps = f.input_symbols().find('<eps>')
    curr_state = f.start()
    output = []

    for i in range(seq_len):
        found = False
        label = f.input_symbols().find(seq[i])
        for arc in f.arcs(curr_state):
            if arc.ilabel == label:
                if f.output_symbols().find(arc.olabel) == f.input_symbols().find(arc.ilabel):
                    output += seq2[i]
                else:
                    output += f.output_symbols().find(arc.olabel)


                curr_state = arc.nextstate
                found = True
                break  # no need to keep going through other arcs, as it's determinized
        if not found:
            print("Can't transduce the sequence with provided FST")

    final_weight = float(f.final(curr_state))
    if final_weight != math.inf: # if this is a final state
        #out_seq = [f.output_symbols().find(w) for w in output if w != eps]  # find the labels in the table, remove epsilons
        return output
    else:
        print("Can't transduce the sequence with provided FST")
```

Part – 2 of Homework:

Second part is creating possessive FST. I implemented 2 FST to solve this problem. First one is an intermediate Finite State Transducer to detect softening (yumuşama) problem in Turkish. It fixes the softening in the words and give an output. This output will be input of second FST. Let's begin with first intermediate FST to detect softening.

INTERMEDIATE FSM:

This is the generation of FST that applies softening operation.

Input symbols:

1. "sesli"

2. "p"

3. "ç"

4. "t"

5. "k"

6. "+1sp" -> First Person Singular Possessive

7. "+2sp" -> Second Person Singular Possessive

8. "+3sp" -> Third Person Singular Possessive

9. "+1pp" -> First Person Plural Possessive

10. "+2pp" -> Second Person Plural Possessive

11. "+3pp" -> Third Person Plural Possessive

12. "other"

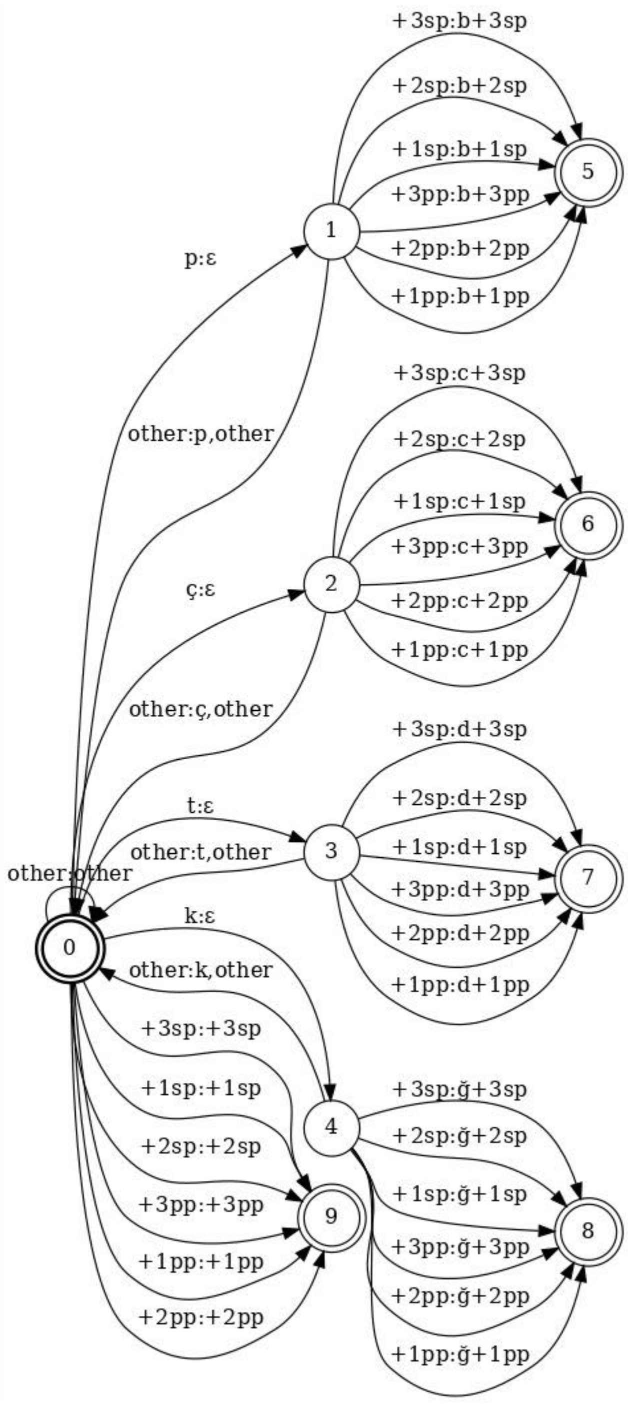
Output symbols:

1. "sesli"
2. "b+3pp" -> This is a multiple output. Softened letter and possessive suffix.
3. "b+2pp"
4. "b+1pp"
5. "b+3sp"
6. "b+2sp"
7. "b+1sp"
8. "c+3pp"
9. "c+2pp"
10. "c+1pp"
11. "c+3sp"
12. "c+2sp"
13. "c+1sp"
14. "d+3pp"
15. "d+2pp"
16. "d+1pp"
17. "d+3sp"
18. "d+2sp"
19. "d+1sp"
20. "ğ+3pp"
21. "ğ+2pp"
22. "ğ+1pp"
23. "ğ+3sp"
24. "ğ+2sp"
25. "ğ+1sp"
26. "p,other" -> This is multiple output. It produces "p" letter and "other". Here, the "other" represents what we read.
27. "ç,other"
28. "t,other"
29. "k,other"
30. "epsilon" -> This is empty output.
31. "other" -> Represents what we read.

Algorithm is here:

If the word ends with "p" or "ç" or "t" or "k", it means that there is a softing operation and we need to change these letters as "b" or "c" or "d" or "ğ" letters. FST also adds the possessive suffix of input to the end of result string. It is going to be given to other level FST as input.

Visualized FST:



Preprocessing

When we are getting input. There is a small preprocessing step. To bring the input into a form suitable for the FST that I designed, I implemented a function:

If a character in the input is not any of these {"+1pp”, "+2pp”, "+3pp" ,"+1sp", "+2sp", "+3sp", "p", "ç", "t", "k"}, I labelled it as “other”

The need of this preprocessing step comes from complex visualization of FST. If I take all vowels separately, FST looks very big and complex. So, I grouped them. However, to use this group rules, I need this preprocessing step.

Generating Output Using FST

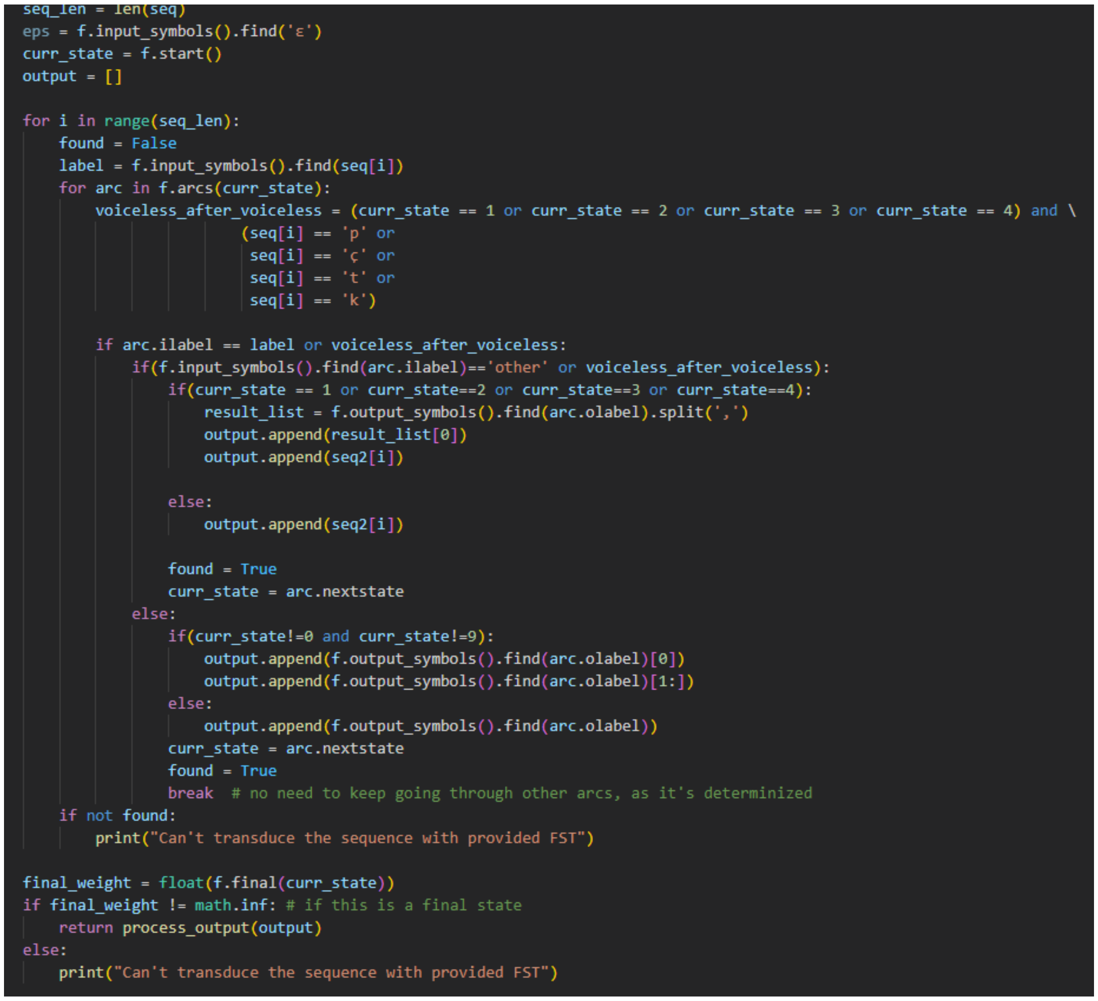
The next function allows us to obtain the output we get as the fst states change. It is very similar to plural FST version.

I got this function from Openfst-Python documentation:
https://github.com/edemattos/asr/blob/master/asr_lab5_solutions.ipynb

I modified existing function a little bit. Let me explain what I change:

I am giving 2 parameters to function instead of 1. One of it is correct form of input for FST, other one is unchanged version of it. The algorithm is:

1. If the input is "other", we should check the current state. If the current state is 1 or 2 or 3 or 4, it means that there is NO a softening operation. So, we should produce original form of previous read character before the possessive indicator. If the input is "other", we should check the current state. If the current state is 1 or 2 or 3 or 4, it means that there is NO a softening operation. So, we should produce original form of previous read character before the possessive indicator. The "other" input in the state of 1,2,3 and 4 means that any other input except possessive suffix. So, I put an additional condition to detect if a voiceless consonant comes after a voiceless consonant. It must be considered as "other”.
2. If the input is "other" and the current state is not 1 or 2 or 3 or 4, we should produce what we read.
3. If the input is not "other", again we should check the current state. If it is not 0 and 9, it means that there is a softening operation. So, we should produce soft form of previous read character before the possessive indicator.
4. If the input is not "other", and the state is 0 or 9, we should produce output of the state as what it is.



```
seq_len = len(seq)
eps = f.input_symbols().find('ε')
curr_state = f.start()
output = []

for i in range(seq_len):
    found = False
    label = f.input_symbols().find(seq[i])
    for arc in f.arcs(curr_state):
        voiceless_after_voiceless = (curr_state == 1 or curr_state == 2 or curr_state == 3 or curr_state == 4) and \
                                    (seq[i] == 'p' or
                                     seq[i] == 'ç' or
                                     seq[i] == 't' or
                                     seq[i] == 'k')

        if arc.ilabel == label or voiceless_after_voiceless:
            if(f.input_symbols().find(arc.ilabel)=='other' or voiceless_after_voiceless):
                if(curr_state == 1 or curr_state==2 or curr_state==3 or curr_state==4):
                    result_list = f.output_symbols().find(arc.olabel).split(',')
                    output.append(result_list[0])
                    output.append(seq2[i])

                else:
                    output.append(seq2[i])

                found = True
                curr_state = arc.nextstate
            else:
                if(curr_state!=0 and curr_state!=9):
                    output.append(f.output_symbols().find(arc.olabel)[0])
                    output.append(f.output_symbols().find(arc.olabel)[1:])
                else:
                    output.append(f.output_symbols().find(arc.olabel))
                curr_state = arc.nextstate
                found = True
                break  # no need to keep going through other arcs, as it's determinized
    if not found:
        print("Can't transduce the sequence with provided FST")

final_weight = float(f.final(curr_state))
if final_weight != math.inf: # if this is a final state
    return process_output(output)
else:
    print("Can't transduce the sequence with provided FST")
```

MAIN FST TO GENERATE POSSESSIVE SUFFIX:

We get input from intermediate FST to fix softening problems. Output of intermediate FST is input for this FST. It seems complicated by looking at the image but it is not. I am going to clarify the FST.

Input Symbols:

1. "cnsnt" -> Consonant letters of Turkish

2. "a,ı" -> "a" and "ı" letters

3. "u,o"

4. "e,i"

5. "ü,ö"

6. "+3sp" -> Third person singular possessive suffix

7. "+2sp" -> Second person singular possessive suffix

8. "+1sp" -> First person singular possessive suffix

9. "+3pp" -> Third person plural possessive suffix

10. "+2pp" -> Second person plural possessive suffix

11. "+1pp" -> First person plural possessive suffix

Output Symbols:

1. "cnsnt"
2. "a,ı" -> "a" and "ı" letters
3. "u,o"
4. "e,i"
5. "ü,ö"
6. "sı" -> 3sp and 3pp possessive suffix
7. "si" -> 3sp and 3pp possessive suffix
8. "ı" -> 3sp and 3pp possessive suffix
9. "i" -> 3sp and 3pp possessive suffix
10. "su" -> 3sp and 3pp possessive suffix
11. "sü" -> 3sp and 3pp possessive suffix
12. "u" -> 3sp and 3pp possessive suffix
13. "ü" -> 3sp and 3pp possessive suffix
14. "in" -> 2sp possessive suffix
15. "ın" -> 2sp possessive suffix
16. "ün" -> 2sp possessive suffix
17. "un" -> 2sp possessive suffix
18. "n" -> 2sp possessive suffix
19. "iniz" -> 2pp possessive suffix
20. "ınız" -> 2pp possessive suffix
21. "niz" -> 2pp possessive suffix
22. "nız" -> 2pp possessive suffix
23. "nuz" -> 2pp possessive suffix
24. "nüz" -> 2pp possessive suffix
25. "unuz" -> 2pp possessive suffix
26. "ünüz" -> 2pp possessive suffix
27. "im" -> 1sp possessive suffix
28. "um" -> 1sp possessive suffix
29. "üm" -> 1sp possessive suffix
30. "ım" -> 1sp possessive suffix
31. "m" -> 1sp possessive suffix
32. "imiz" -> 1pp possessive suffix

33. "ımız" -> 1pp possessive suffix

34. "miz" -> 1pp possessive suffix

35. "mız" -> 1pp possessive suffix

36. "ümüz" -> 1pp possessive suffix

37. "umuz" -> 1pp possessive suffix

38. "muz" -> 1pp possessive suffix

39. "müz" -> 1pp possessive suffix

Let me tell the algorithm:

Firstly, if the word ends with a consonant letter, we must add an additional letter there. For example, possessive form of asa "asa + m" but possessive form of "kalem" is "kalem + im". So, we need to check which letter we read before the possessive suffix. If we are in the state of consonant letter, we are going to add this additional character. That is why we have some states after each vowel letter state.

Secondly, we need to indicate which possessive suffix will be used after a vowel letter:

1. If the possessive suffix comes after "a" or "ı", we are gonna put there "mız" "ınız" "ı" as examples. As you see suffixes are formed with "ı" letter. Example: "can'ım"

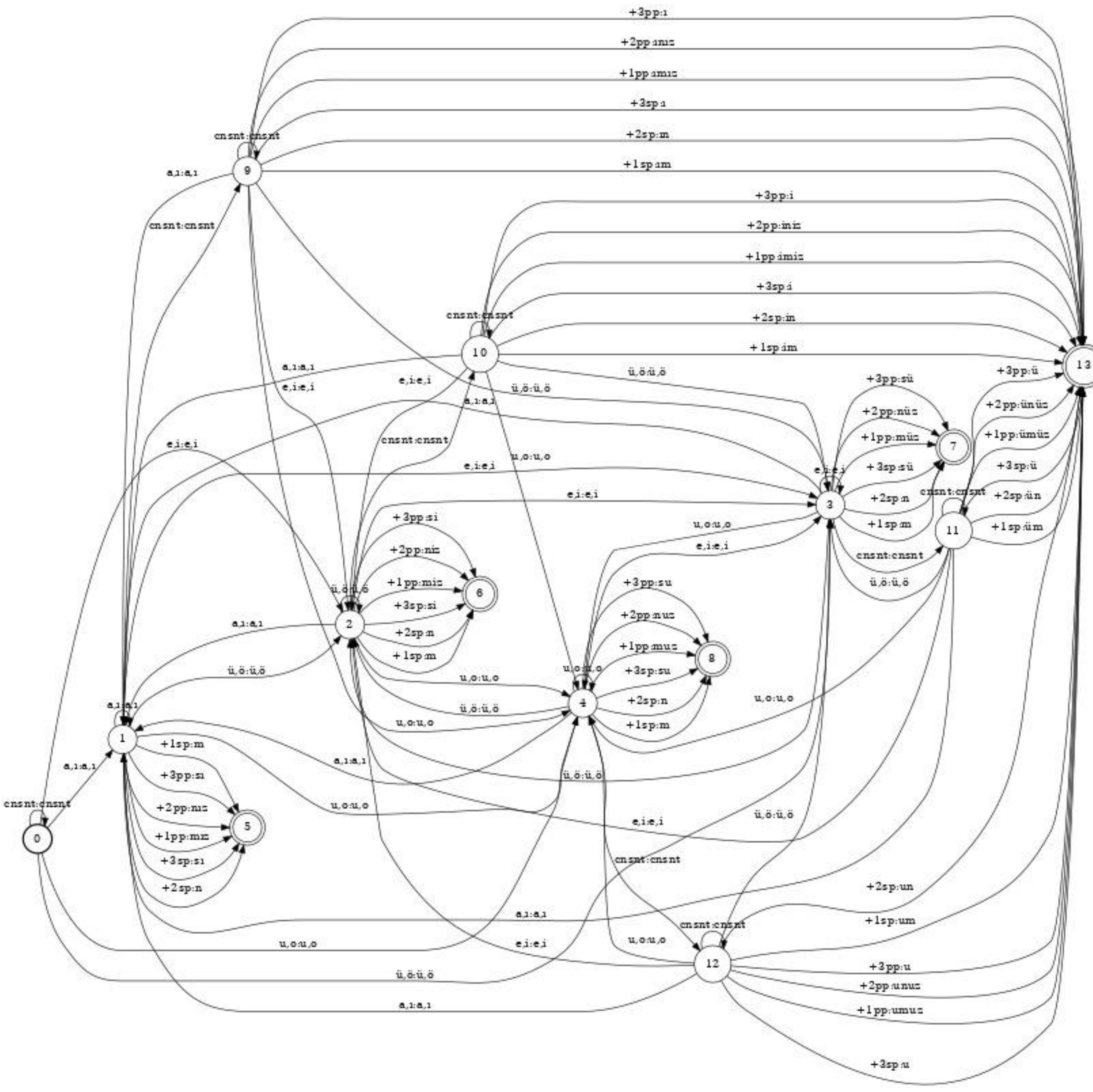
2. If the possessive suffix comes after "e" or "i", we are gonna put there "miz" "iniz" "i" as examples. As you see suffixes are formed with "i" letter. Example: "güzel'im"

3. If the possessive suffix comes after "o" or "u", we are gonna put there "muz" "unuz" "u" as examples. As you see suffixes are formed with "u" letter. Example: "kuzu'm"

4. If the possessive suffix comes after "ö" or "ü", we are gonna put there "müz" "ünüz" "ü" as examples. As you see suffixes are formed with "ü" letter. Example: "bücür'üm"

So, at the end we need 4 states for vowel cases. And for each vowel case, we need a consonant case. At the end, we will have 8 states excluding final and start states. As I said, this consonant states for deciding we are going to add an additional character or not to the possessive suffix. The main thing we did is to decide which type of letter we read: "consonent", "a,ı", "e,i", "u,o", "ü,ö" or possessive suffix inputs. We just change the states according to this information. In summary, there are too many edges between states but as I said, in fact it is very simple.

Visualized FST:



Generating Output Using FST

The next function allows us to obtain the output we get as the fst states change. It is very similar to plural FST version.

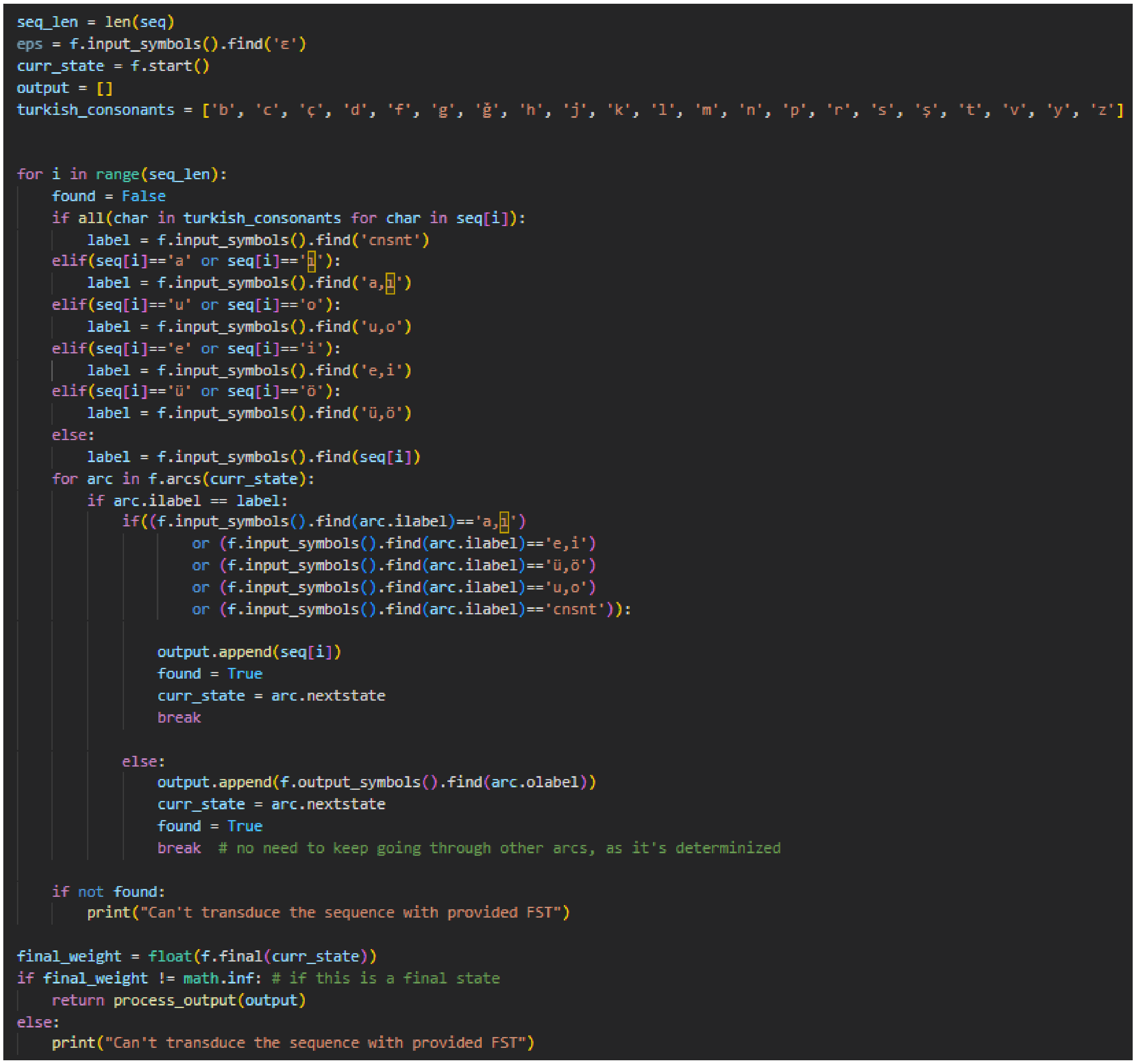
I got this function from Openfst-Python documentation:
https://github.com/edemattos/asr/blob/master/asr_lab5_solutions.ipynb

I modified existing function a little bit. Let me explain what I change:

This time, I did NOT change the parameter number of this functon. I process the input at the beginning of the loop to convert the input correct form for my FST:

- I change the input to "a,ı" if the input is "a" or "ı"
- I change the input to "o,u" if the input is "o" or "u"
- I change the input to "e,i" if the input is "e" or "i"
- I change the input to "ö,ü" if the input is "ö" or "ü"

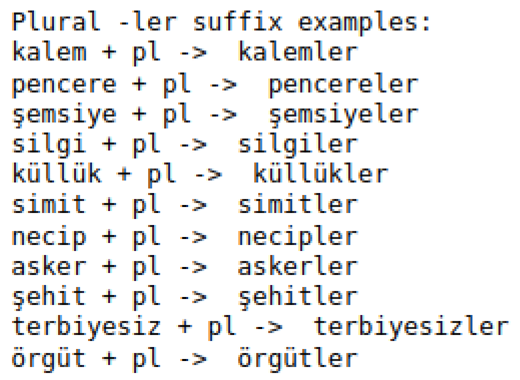
After that, we produce correct outputs. If the input is "a,ı" or "o,u" or "e,i" or "ö,ü" or "cnsnt" (sessiz harf) it means that we should produce exact character of input according to FST. Else, we should produce exactly the same as the FST output.



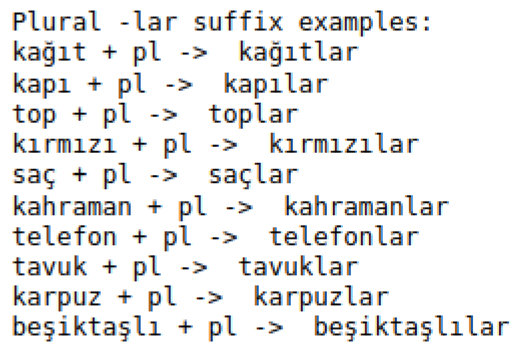
```python
seq_len = len(seq)
eps = f.input_symbols().find('ε')
curr_state = f.start()
output = []
turkish_consonants = ['b', 'c', 'ç', 'd', 'f', 'g', 'ğ', 'h', 'j', 'k', 'l', 'm', 'n', 'p', 'r', 's', 'ş', 't', 'v', 'y', 'z']


for i in range(seq_len):
    found = False
    if all(char in turkish_consonants for char in seq[i]):
        label = f.input_symbols().find('cnsnt')
    elif(seq[i]=='a' or seq[i]=='ı'):
        label = f.input_symbols().find('a,ı')
    elif(seq[i]=='u' or seq[i]=='o'):
        label = f.input_symbols().find('u,o')
    elif(seq[i]=='e' or seq[i]=='i'):
        label = f.input_symbols().find('e,i')
    elif(seq[i]=='ü' or seq[i]=='ö'):
        label = f.input_symbols().find('ü,ö')
    else:
        label = f.input_symbols().find(seq[i])
    for arc in f.arcs(curr_state):
        if arc.ilabel == label:
            if((f.input_symbols().find(arc.ilabel)=='a,ı')
                    or (f.input_symbols().find(arc.ilabel)=='e,i')
                    or (f.input_symbols().find(arc.ilabel)=='ü,ö')
                    or (f.input_symbols().find(arc.ilabel)=='u,o')
                    or (f.input_symbols().find(arc.ilabel)=='cnsnt')):

                output.append(seq[i])
                found = True
                curr_state = arc.nextstate
                break

            else:
                output.append(f.output_symbols().find(arc.olabel))
                curr_state = arc.nextstate
                found = True
                break  # no need to keep going through other arcs, as it's determinized

    if not found:
        print("Can't transduce the sequence with provided FST")

final_weight = float(f.final(curr_state))
if final_weight != math.inf: # if this is a final state
    return process_output(output)
else:
    print("Can't transduce the sequence with provided FST")
```

Test and Results:

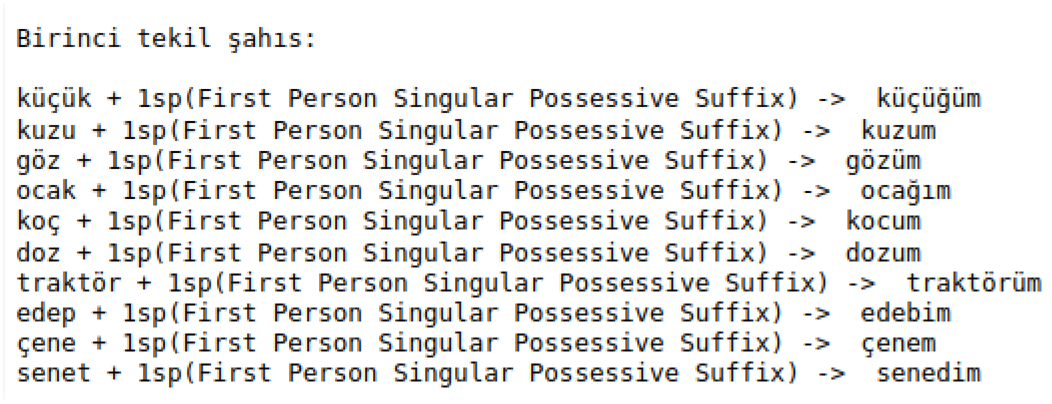
You can see the outputs in the notebook, also.



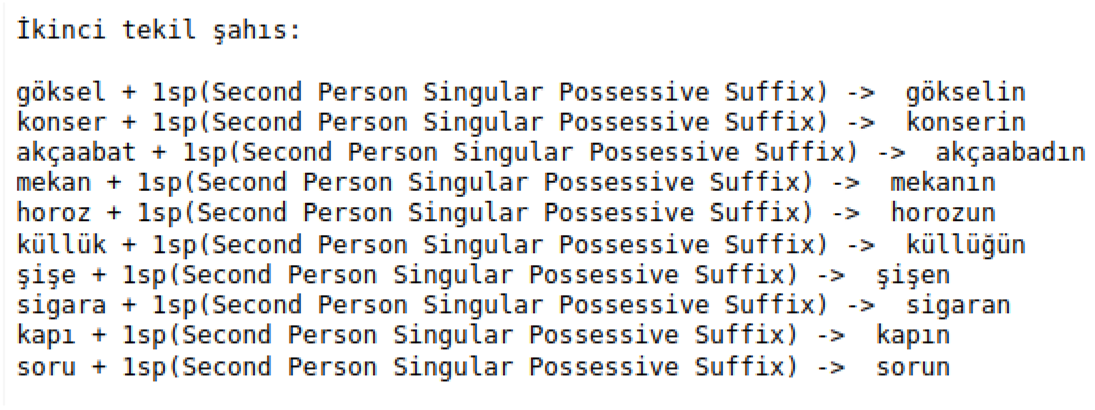
```
Plural -ler suffix examples:
kalem + pl ->  kalemler
pencere + pl ->  pencereler
şemsiye + pl ->  şemsiyeler
silgi + pl ->  silgiler
küllük + pl ->  küllükler
simit + pl ->  simitler
necip + pl ->  necipler
asker + pl ->  askerler
şehit + pl ->  şehitler
terbiyesiz + pl ->  terbiyesizler
örgüt + pl ->  örgütler
```



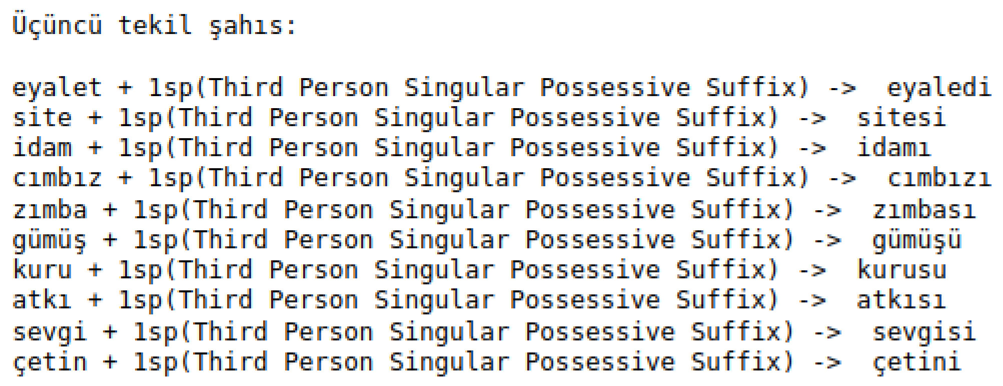
```
Plural -lar suffix examples:
kağıt + pl ->  kağıtlar
kapı + pl ->  kapılar
top + pl ->  toplar
kırmızı + pl ->  kırmızılar
saç + pl ->  saçlar
kahraman + pl ->  kahramanlar
telefon + pl ->  telefonlar
tavuk + pl ->  tavuklar
karpuz + pl ->  karpuzlar
beşiktaşlı + pl ->  beşiktaşlılar
```



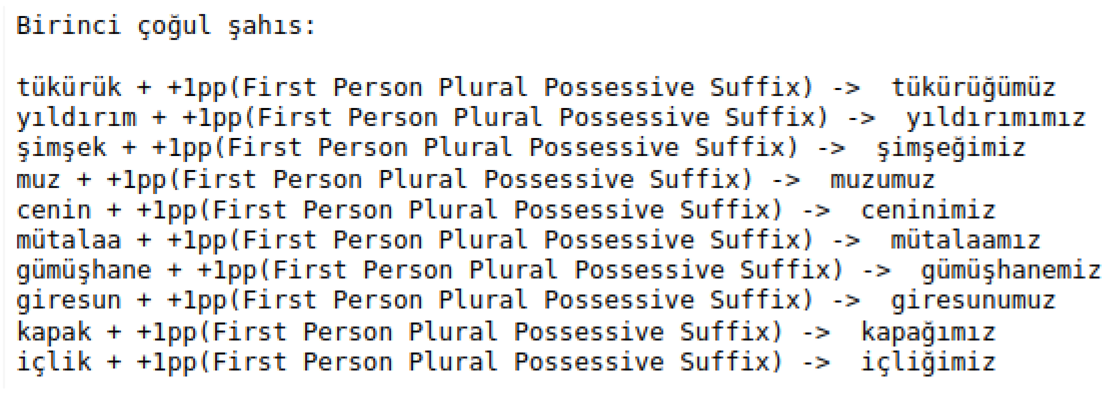
```
Birinci tekil şahıs:

küçük + 1sp(First Person Singular Possessive Suffix) ->  küçüğüm
kuzu + 1sp(First Person Singular Possessive Suffix) ->  kuzum
göz + 1sp(First Person Singular Possessive Suffix) ->  gözüm
ocak + 1sp(First Person Singular Possessive Suffix) ->  ocağım
koç + 1sp(First Person Singular Possessive Suffix) ->  kocum
doz + 1sp(First Person Singular Possessive Suffix) ->  dozum
traktör + 1sp(First Person Singular Possessive Suffix) ->  traktörüm
edep + 1sp(First Person Singular Possessive Suffix) ->  edebim
çene + 1sp(First Person Singular Possessive Suffix) ->  çenem
senet + 1sp(First Person Singular Possessive Suffix) ->  senedim
```



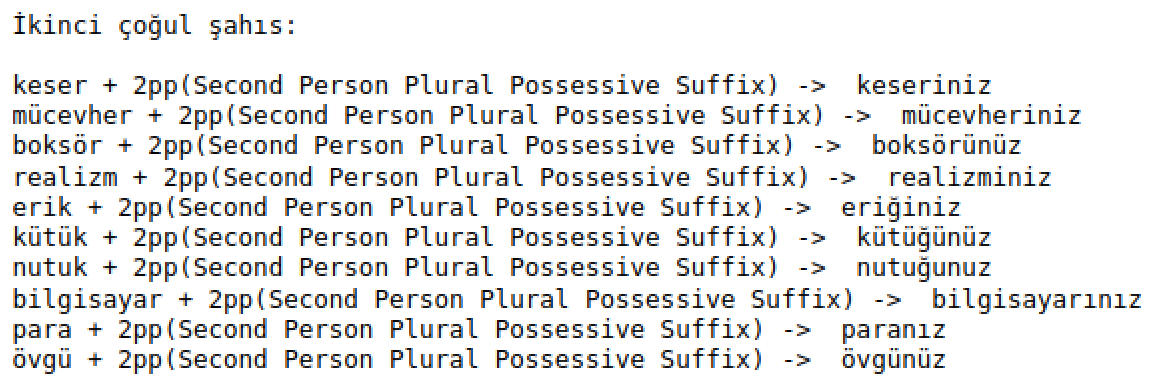
```
İkinci tekil şahıs:

göksel + 1sp(Second Person Singular Possessive Suffix) ->  gökselin
konser + 1sp(Second Person Singular Possessive Suffix) ->  konserin
akçaabat + 1sp(Second Person Singular Possessive Suffix) ->  akçaabadın
mekan + 1sp(Second Person Singular Possessive Suffix) ->  mekanın
horoz + 1sp(Second Person Singular Possessive Suffix) ->  horozun
küllük + 1sp(Second Person Singular Possessive Suffix) ->  küllüğün
şişe + 1sp(Second Person Singular Possessive Suffix) ->  şişen
sigara + 1sp(Second Person Singular Possessive Suffix) ->  sigaran
kapı + 1sp(Second Person Singular Possessive Suffix) ->  kapın
soru + 1sp(Second Person Singular Possessive Suffix) ->  sorun
```



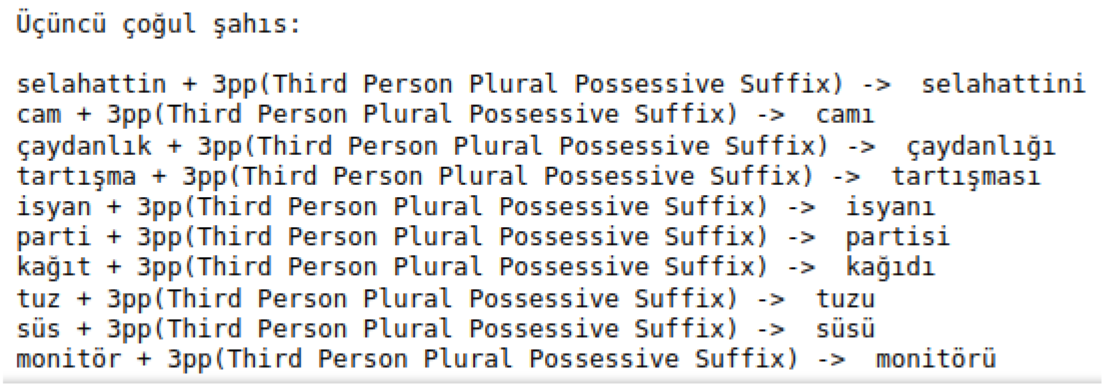
```
Üçüncü tekil şahıs:

eyalet + 1sp(Third Person Singular Possessive Suffix) ->  eyaledi
site + 1sp(Third Person Singular Possessive Suffix) ->  sitesi
idam + 1sp(Third Person Singular Possessive Suffix) ->  idamı
cımbız + 1sp(Third Person Singular Possessive Suffix) ->  cımbızı
zımba + 1sp(Third Person Singular Possessive Suffix) ->  zımbası
gümüş + 1sp(Third Person Singular Possessive Suffix) ->  gümüşü
kuru + 1sp(Third Person Singular Possessive Suffix) ->  kurusu
atkı + 1sp(Third Person Singular Possessive Suffix) ->  atkısı
sevgi + 1sp(Third Person Singular Possessive Suffix) ->  sevgisi
çetin + 1sp(Third Person Singular Possessive Suffix) ->  çetini
```



```
Birinci çoğul şahıs:

tükürük + +1pp(First Person Plural Possessive Suffix) ->  tükürüğümüz
yıldırım + +1pp(First Person Plural Possessive Suffix) ->  yıldırımımız
şimşek + +1pp(First Person Plural Possessive Suffix) ->  şimşeğimiz
muz + +1pp(First Person Plural Possessive Suffix) ->  muzumuz
cenin + +1pp(First Person Plural Possessive Suffix) ->  ceninimiz
mütalaa + +1pp(First Person Plural Possessive Suffix) ->  mütalaamız
gümüşhane + +1pp(First Person Plural Possessive Suffix) ->  gümüşhanemiz
giresun + +1pp(First Person Plural Possessive Suffix) ->  giresunumuz
kapak + +1pp(First Person Plural Possessive Suffix) ->  kapağımız
içlik + +1pp(First Person Plural Possessive Suffix) ->  içliğimiz
```



```
İkinci çoğul şahıs:

keser + 2pp(Second Person Plural Possessive Suffix) ->  keseriniz
mücevher + 2pp(Second Person Plural Possessive Suffix) ->  mücevheriniz
boksör + 2pp(Second Person Plural Possessive Suffix) ->  boksörünüz
realizm + 2pp(Second Person Plural Possessive Suffix) ->  realizminiz
erik + 2pp(Second Person Plural Possessive Suffix) ->  eriğiniz
kütük + 2pp(Second Person Plural Possessive Suffix) ->  kütüğünüz
nutuk + 2pp(Second Person Plural Possessive Suffix) ->  nutuğunuz
bilgisayar + 2pp(Second Person Plural Possessive Suffix) ->  bilgisayarınız
para + 2pp(Second Person Plural Possessive Suffix) ->  paranız
övgü + 2pp(Second Person Plural Possessive Suffix) ->  övgünüz
```



```
Üçüncü çoğul şahıs:

selahattin + 3pp(Third Person Plural Possessive Suffix) ->  selahattini
cam + 3pp(Third Person Plural Possessive Suffix) ->  camı
çaydanlık + 3pp(Third Person Plural Possessive Suffix) ->  çaydanlığı
tartışma + 3pp(Third Person Plural Possessive Suffix) ->  tartışması
isyan + 3pp(Third Person Plural Possessive Suffix) ->  isyanı
parti + 3pp(Third Person Plural Possessive Suffix) ->  partisi
kağıt + 3pp(Third Person Plural Possessive Suffix) ->  kağıdı
tuz + 3pp(Third Person Plural Possessive Suffix) ->  tuzu
süs + 3pp(Third Person Plural Possessive Suffix) ->  süsü
monitör + 3pp(Third Person Plural Possessive Suffix) ->  monitörü
```

As you see, it can produce all plural and possessive Turkish words correctly. However, there are many exceptions to the softening rule in Turkish, and since these exceptions do not have a specific rule, this situation could not be handled in FST. For example, the sentence that should be "Akçaabatımız" is generated as "Our Akçaabadımız". Except for this problem, all cases work successfully.