# A Simple Tutorial on Finite State Morphology Tools

Burak Kerim Akkuş

## 1 Introduction

This document provides a brief tutorial writing two-level morphology rules for Turkish using Finite State Morphology Tools. The examples cover a very tiny subset of nominal morphology with plurality and possession suffixes. The rules can handle vowel harmony; however, consonant assimilation, softening and filler drop rules are necessary for a fully working model in this context. 4 binary files, *twolc*, *lexc*, *xfts* and *lookup*, can be downloaded from the provided link in the References section. It is important to read the documentation for the syntax of the rules and the lexicon.

## 2 Two-Level Rules

The grammar file contains 5 sections. Two of them are necessary for every application. The grammar must start with the alphabet declaration and must end with rules. Other three sections are optional but makes the grammar more manageable, easy to write and read.

The Alphabet section is a list of elements, single symbols, complete or incomplete pairs terminated with a semicolon. Pairs are separated with :'s. Lexical forms or upper forms are on the left and surface or lower forms are on the right:

```
ALPHABET
a b c ç d ... %+:0 %(:0 %):0 ;
```

Diacritics are typically used as features that force or block the application of particular rules:

```
DIACRITICS
! Optional lexical symbols as features for particular rules
```

The optional Sets declaration serves the same purpose as distinctive features in generative phonology: it enables the user to define classes of symbols that are equivalent in some respect:

```
SETS
! Classes of symbols - also optional

! consonants - sessiz harfler
C = b c ç d f g h ... ;

! vowels - sesli harfler
V = a e ı i ... ;
```

The optional Definitions section allows the user to write regular expressions to create and name two-level relations:

```
DEFINITIONS
! Optional place holders for regular expressions

! context for vowel harmony, where C is the set of consonants
VOWEL = [:C* :0*]* ;
```

Each rule must have a name, written within double quotes, a lexical:surface correspondence, a rule operator that defines the nature of the constraint, one or more contexts, divided between a left and a right context by an underscore:

```
RULES
! Name of the rule
"1. Vowel Harmony A:a"
! Explain your rule in detail
! if last vowel in stem is a back vowel
A:a => :Vb [:C* (:0)*]* _ ;
```

# 3 Two-Level Rule Compiler - twolc

Listing 1: Read the grammar with two-level rules

```
twolc> read-grammar tr_simple_rules.txt
opening "tr_simple_rules.txt"
reading from "tr_simple_rules.txt"...
Alphabet... Sets... Definitions... Rules...
  "1. Vowel Harmony A:a" "2. Vowel Harmony A:e" "3. Vowel Harmony H:ı" "4. Vowel ↩
    Harmony H:i"
  "5. Vowel Harmony H:u" "6. Vowel Harmony H:ü"
```

Listing 2: Compile rules

```
twolc> compile
Compiling "tr_simple_rules.txt"
Expanding... Analyzing...
Compiling rule components:
  "1. Vowel Harmony A:a" "2. Vowel Harmony A:e" "3. Vowel Harmony H:ı" "4. Vowel ↩
    Harmony H:i"
  "5. Vowel Harmony H:u" "6. Vowel Harmony H:ü"
Compiling rules:
  "1. Vowel Harmony A:a"
    A:a =>
  "2. Vowel Harmony A:e"
    A:e =>
  "3. Vowel Harmony H:ı"
    H:ı =>
  "4. Vowel Harmony H:i"
    H:i =>
  "5. Vowel Harmony H:u"
    H:u =>
  "6. Vowel Harmony H:ü"
    H:ü =>
Done.
```

Read and compile with single command if you update your rules

```
twolc> redo
reading from "tr_simple_rules.txt"...
Alphabet... Sets... Definitions... Rules...
  "1. Vowel Harmony A:a" "2. Vowel Harmony A:e" "3. Vowel Harmony H:ı" "4. Vowel ↩
    Harmony H:i"
  "5. Vowel Harmony H:u" "6. Vowel Harmony H:ü"
Compiling "tr_simple_rules.txt"
Expanding... Analyzing...
Compiling rule components:
  "1. Vowel Harmony A:a" "2. Vowel Harmony A:e" "3. Vowel Harmony H:ı" "4. Vowel ↩
    Harmony H:i"
  "5. Vowel Harmony H:u" "6. Vowel Harmony H:ü"
Compiling rules:
  "1. Vowel Harmony A:a"
    A:a =>
  "2. Vowel Harmony A:e"
    A:e =>
  "3. Vowel Harmony H:ı"
    H:ı =>
  "4. Vowel Harmony H:i"
    H:i =>
  "5. Vowel Harmony H:u"
    H:u =>
  "6. Vowel Harmony H:ü"
    H:ü =>
Done.
```

You can also see the transition table for your rules

```
twolc> show-rules

"1. Vowel Harmony A:a"
   ? a b A:a
1: 1 2 1
2: 1 2 2  1
Equivalence classes:
((? e i ö ü A:e H:i H:u H:ü H:ı) (a o u ı) (b c d f g h j k l m n p r
  s t v y z ç ş (:0 ):0 +:0) (A:a))

Type <CR> to continue...
```

Also check for individual rules by providing their names

```
twolc> show
Rule name: "2. Vowel Harmony A:e"

   ? b e A:e
1: 1 1 2
2: 1 2 2  1
Equivalence classes:
((? a o u ı A:a H:i H:u H:ü H:ı) (b c d f g h j k l m n p r s t v y z ç
  ş (:0 ):0 +:0) (e i ö ü) (A:e))
```

Listing 3: Test your rules to see they can match a lexical form with its surface representation

```
twolc> lex-test

Lexical string ('q' = quit): ev+Hm
                             evim
e
v
+:0
H:i
m
```

Listing 4: Save your transducer so that you can use it with lexicon compiler

```
twolc> save-binary
Output file [cancel]: tr_simple.rules
opening "tr_simple.rules"
Writing "tr_simple.rules"
```

For this example to fully work, we also need to add voicing (softening), devoicing (assimilation, hardening) and filler drop rules.

# 4 Lexicon

The lexicon contains the lexemes. There may be any number of sublexicons, but one of them must have the name *Root*. Entries of a lexicon are separated by semicolons. Each entry consists of a form, possibly null, and a continuation class. The continuation class must be the name of another sublexicon or #, a terminator symbol. The word END at the end of the file is optional.

The lexicon may optionally contain the declaration *Multichar_Symbols* at the beginning of the file. This section tells the compiler that these tags are to be parsed as single symbols rather than as a sequence of symbols.

Optional multiple character symbols

```
Multichar_Symbols
    +SNG +PLU P1S +P2S ...
```

Begin with Root, next state is Plu for ev and ağaç

```
LEXICON Root
    ev      Plu ;
    ağaç    Plu ;
```

Pos follows Plu

```
LEXICON Plu
    +SNG:0      Pos;
    +PLU:+lAr   Pos;
```

Pos is the final state

```
LEXICON Pos
    +Poo:0          #;
    +P1S:+(H)m      #;
```

# 5 Finite-State Lexicon Compiler - lexc

Listing 1: Read the transducer you saved before

```
lexc> read-rules tr_simple.rules
Opening 'tr_simple.rules'...
6 nets read.
```

Listing 2: Read the lexicon file

```
lexc> compile-source tr_simple_lex.txt
opening "tr_simple_lex.txt"
Opening 'tr_simple_lex.txt'...
Root...2, Plu...2, Pos...7
Building lexicon...Minimizing...Done!
SOURCE: 3.4 Kb. 40 states, 47 arcs, 28 paths.
```

Listing 3: Combine lexicon and rules

```
lexc> compose-result
No epenthesis.
No surrounding word boundaries added.
...Done.
4.5 Kb. 76 states, 88 arcs, 28 paths.
Minimizing...Done.
3.7 Kb. 48 states, 62 arcs, 28 paths.
```

Listing 4: Test your work, notice the difference between result & source

```
lexc> lookdown
Use (s)ource or (r)esult? [r]: r
NOTE:  Using RESULT.
Word: ev+SNG+P1S
evim

lexc> lookdown
Use (s)ource or (r)esult? [r]: s
NOTE:  Using SOURCE.
Word: ev+SNG+P1S
ev+(H)m
```

```
lexc> lookup
Use (s)ource or (r)esult? [r]: r
NOTE:  Using RESULT.
Word: evim
ev+SNG+P1S

lexc> lookup
Use (s)ource or (r)esult? [r]: s
NOTE:  Using SOURCE.
Word: evim
***************

lexc> lookup
Use (s)ource or (r)esult? [r]: s
NOTE:  Using SOURCE.
Word: ev+(H)m
ev+SNG+P1S
```

Listing 5: Save your work so that you can use the transducer with lookup application

```
lexc> save-result
Output file [cancel]: tr_simple.fst
opening "tr_simple.fst"
Opening 'tr_simple.fst'...
Done.
```

Samples from lexical level

```
lexc> random-lex
Use (s)ource or (r)esult? [r]: s
NOTE:  Using SOURCE.
ağaç+SNG+P1S
ağaç+SNG+P1P
ev+PLU+P2S
ağaç+SNG+P2P
ev+SNG+P1S


lexc> random-lex
Use (s)ource or (r)esult? [r]: r
NOTE:  Using RESULT.
ev+SNG+Poo
ağaç+PLU+Poo
ev+SNG+Poo
ağaç+SNG+P1S
ev+SNG+Poo
```

Samples from surface level

```
lexc> random-surf
Use (s)ource or (r)esult? [r]: s
NOTE:  Using SOURCE.
ev+(H)nHz
ağaç
ev+lAr+(s)HN
ağaç+lAr+(H)nHz
ağaç+(s)HN


lexc> random-surf
Use (s)ource or (r)esult? [r]: r
NOTE:  Using RESULT.
evim
evin
ağaçsıN
evler
ağaçlar
```

# 6 Xerox Finite-State Tool - xfst

Optionally you can use xfst to display all lexical and surface forms of the strings in your lexicon. You can use this feature to check whether your grammar generates strings out of the language or misses some strings.

Listing 1: Read the lexicon file

```
xfst[0]: read lexc tr_simple_lex.txt
Opening input file 'tr_simple_lex.txt'
March 18, 2015 03:16:33 GMT
Reading UTF-8 text from 'tr_simple_lex.txt'
Root...2, Plu...2, Pos...7
Building lexicon...Minimizing...Done!
3.4 Kb. 40 states, 47 arcs, 28 paths.
Closing 'tr_simple_lex.txt'
```

Listing 2: Print lexical / upper strings (to be used with lookdown in lexc)

```
xfst[1]: print upper-words
ev+SNG+Poo
ev+SNG+P1S
ev+SNG+P2S
ev+SNG+P3S
ev+SNG+P1P
ev+SNG+P2P
ev+SNG+P3P
ev+PLU+Poo
ev+PLU+P1S
ev+PLU+P2S
ev+PLU+P3S
ev+PLU+P1P
ev+PLU+P2P
ev+PLU+P3P
```

Listing 3: Print surface / lower strings (to be used with lookup in lexc)

```
xfst[1]: print lower-words
ev
ev+(H)m
ev+(H)n
ev+(s)H
ev+(H)mHz
ev+(H)nHz
ev+LArH
ev+lAr
ev+lAr+(H)m
ev+lAr+(H)n
ev+lAr+(s)H
ev+lAr+(H)mHz
ev+lAr+(H)nHz
ev+lAr+LArH
```

# 7 Lexical Lookup - lookup

After building your finite state transducer, you can use lookup application similar to the lookup command in lexc.

```
$ echo evim | ./lookup tr_simple.fst

  *****  LEXICON LOOK-UP  *****

evim  ev  +SNG+P1S


LOOKUP STATISTICS (success with different strategies):
strategy 0: 1 times   (100.00 %)
not found:  0 times   (0.00 %)

corpus size:  1 word
execution time: 0 sec
speed:    1 word/sec

  *****  END OF LEXICON LOOK-UP  *****
```

```
$ echo evim | ./lookup tr_simple.fst 2> /dev/null
evim  ev  +SNG+P1S
```

# 8 References


- Download links for the software and the book:

  `http://web.stanford.edu/~laurik/fsmbook/home.html`

  `http://web.stanford.edu/~laurik/.book2software/`

- Documentation for the software:

  previously at `http://www.cis.upenn.edu/~cis639/docs/`
  now `ftp://ftp.cis.upenn.edu/pub/cis639/public_html/docs/`

  `ftp://ftp.cis.upenn.edu/pub/cis639/public_html/docs/twolc.html`

  `ftp://ftp.cis.upenn.edu/pub/cis639/public_html/docs/lexc.html`

  `ftp://ftp.cis.upenn.edu/pub/cis639/public_html/docs/lookup.html`

  `ftp://ftp.cis.upenn.edu/pub/cis639/public_html/docs/xfst.html`

- Two-level description of Turkish morphology, Kemal Oflazer, 1993

- An Outline of Turkish Morphology, Kemal Oflazer, Elvan Göçmen and Cem Bozşahin, 1994

- A Word Grammar of Turkish with Morphophonemic Rules, Serdar Murat Öztaner, 1996