Morphological Processing

Morphology

- Morphology is the study of the way words are built from smaller meaningful units called **morphemes**.
- We can divide morphemes into two broad classes.
  - **Stems** – the core meaningful units, the root of the word.
  - **Affixes** – add additional meanings and grammatical functions to words.
- Affixes are further divided into:
  - **Prefixes** – precede the stem: do / undo
  - **Suffixes** – follow the stem: eat / eats
  - **Infixes** – are inserted inside the stem
  - **Circumfixes** – precede and follow the stem
- English doesn’t stack more affixes.
- But Turkish can have words with a lot of suffixes.
- Languages, such as Turkish, tend to string affixes together are called **agglutinative** languages.

Surface and Lexical Forms

- The **surface level** of a word represents the actual spelling of that word.
  - geliyorum eats cats kitabım
- The **lexical level** of a word represents a simple concatenation of morphemes making up that word.
  - gel +PROG +1SG
  - eat +AOR
  - cat +PLU
  - kitap +P1SG
- Morphological processors try to find correspondences between lexical and surface forms of words.
  - **Morphological recognition** – surface to lexical
  - **Morphological generation** – lexical to surface

Inflectional and Derivational Morphology

- There are two broad classes of morphology:
  - **Inflectional morphology**
  - **Derivational morphology**
- After a combination with an **inflectional morpheme**, the meaning and class of the actual stem usually do not change.
  - eat / eats pencil / pencils
  - gel / geliyorum masa / masam
- After a combination with an **derivational morpheme**, the meaning and the class of the actual stem usually change.
  - compute / computer do / undo friend / friendly
  - Uygar / uygarlaş kapı / kapıcı
- The irregular changes may happen with derivational affixes.

English Inflectional Morphology

- Nouns have simple inflectional morphology.
  - plural -- cat / cats
  - possessive -- John / John's
- Verbs have slightly more complex inflectional, but still relatively simple inflectional morphology.
  - past form -- walk / walked
  - past participle form -- walk / walked
  - gerund -- walk / walking
  - singular third person -- walk / walks
- Verbs can be categorized as:
  - main verbs
  - modal verbs -- can, will, should
  - primary verbs -- be, have, do
- Regular and irregular verbs: walk / walked -- go / went

English Derivational Morphology

- Some English derivational affixes
  - -ation : transport / transportation
  - -er : kill / killer
  - -ness : fuzzy / fuzziness
  - -al : computation / computational
  - -able : break / breakable
  - -less : help / helpless
  - un : do / undo
  - re : try / retry

Turkish Inflectional Morphology

- Some of inflectional suffixes that Turkish nouns can have:
  - singular/plural : masa / masalar
  - possessive markers : masam / masan / masası / masamız / masanız / masaları
  - case markers :
    - ablative : masadan
    - accusative : masayı
    - dative : masaya
- Some of inflectional suffixes that Turkish verbs can have:
  - tense : gel / geldi / geliyor / gelmiş / gelecek
  - second tense : geliyordu / gelmişti / gelecekti
  - agreement marker : geldim / geldin / geldi / geldik / geldiniz / geldiler
- There are order among inflectional suffixes (**morphotactics**)
  - masalarımdan -- masa +PLU +P1SG +ABL
  - geliyordum -- gel +PROG +PAST +1SG

Turkish Derivational Morphology

- Turkish derivational morphology is very rich.
- Some of derivational suffixes in Turkish:
  - -cı : kapı / kapıcı
  - -laş : uygar / uygarlaş
  - -mek : gel / gelmek
  - -cik : mini / minicik
  - -li : Ankara / Ankaralı

Morphological Parsing

- Morphological parsing is to find the lexical form of a word from its surface form.
  - cats -- cat +N +PLU
  - cat -- cat +N +SG
  - goose -- goose +N +SG or goose +V
  - geese -- goose +N +PLU
  - gooses -- goose +V +3SG
  - catch -- catch +V
  - caught -- catch +V +PAST or catch +V +PP

  - geliyorum -- gel +V +PROG +1SG
  - masalardan -- masa +N +PLU +ABL

- There can be more than one lexical level representation for a given word. (ambiguity)

Parts of A Morphological Processor

- For a morphological processor, we need at least followings:

- **Lexicon** : The list of stems and affixes together with basic information about them such as their main categories (noun, verb, adjective, …) and their sub-categories (regular noun, irregular noun, …).

- **Morphotactics** : The model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word.

- **Orthographic Rules (Spelling Rules)** : These spelling rules are used to model changes that occur in a word (normally when two morphemes combine).

Lexicon

- A lexicon is a repository for words (stems).
- They are grouped according to their main categories.
  - noun, verb, adjective, adverb, …
- They may be also divided into sub-categories.
  - regular-nouns, irregular-singular nouns, irregular-plural nouns, …
- The simplest way to create a morphological parser, put all possible words (together with its inflections) into a lexicon.
  - We do not this because their numbers are huge (theoretically for Turkish, it is infinite)

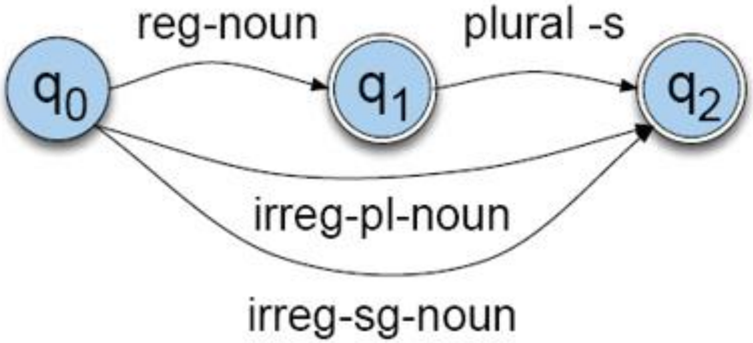
Morphotactics

- Which morphemes can follow which morphemes.

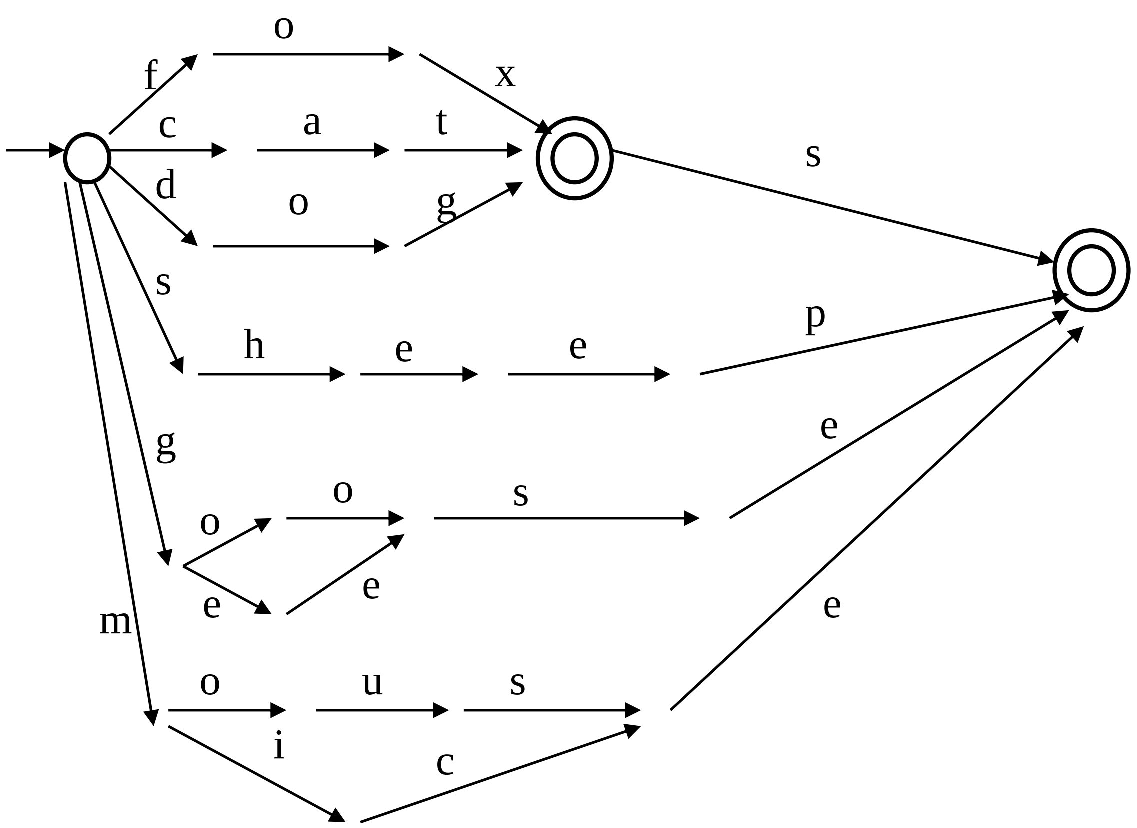
Lexicon:

regular-noun	irregular-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	

- Simple English Nominal Inflection (Morphotactic Rules)



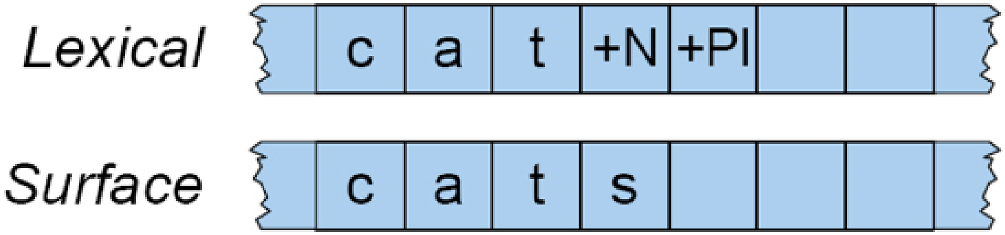
Combine Lexicon and Morphotactics



- This only says yes or no. Does not give lexical representation.
- It accepts a wrong word (foxs).

Two-Level Morphology

- Two-level morphology represents the correspondence between lexical and surface levels.
- We use a finite-state transducer to find mapping between these two levels.
- A FST is a two-tape automaton:
  - Reads from one tape, and writes to other one.
- For morphological processing, one tape holds lexical representation, the second one holds the surface form of a word.



Formal Definition of FST (Mealey Machine)

FST is $\mathbf{Q} \times \boldsymbol{\Sigma} \times \mathbf{q_0} \times \mathbf{F} \times \boldsymbol{\delta}$

- **Q** : a finite set of N states q_0, q_1, … q_N
- **Σ** : a finite input alphabet of complex symbols.
  - Each complex symbol is a pair of an input and an output symbol **i:o**
  - where **i** is a member of I (an input alphabet),
  - and **o** is a member of O (an output alphabet).
  - I and O may contain empty string.
  - So, Σ is a subset of IxO.
- $\mathbf{q_0}$: the start state
- **F** : the set of final states -- F is a subset of Q
- **δ(q,i:o)** : transition function

FST (cont.)

- Σ may not contain all possible pairs from IxO.
- For example:
  - I = {a, b, c}   O={a,b,c, ϵ}
  - Σ = {a:a, b:b, c:c, a:ϵ, b: ϵ, c: ϵ}
- **feasible pairs** – In two-level morphology terminology, the pairs in Σ are called as feasible pairs.
- **default pair** – Instead of a:a we can use a single character for this default pair.
- FSAs are isomorphic to regular languages, and FSTs are isomorphic to regular relations (pair of strings of regular languages).

FST Properties

- FSTs are closed under: union, inversion, and composition.

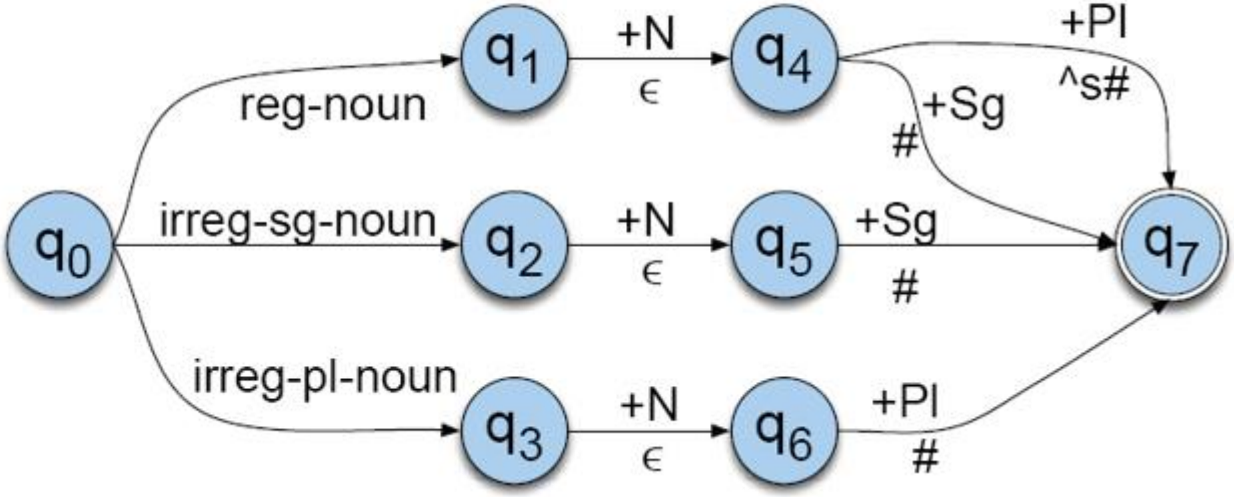
- **union** : The union of two regular relations is also a regular relation.

- **inversion** : The inversion of a FST simply switches the input and output labels.
  - This means that the same FST can be used for both directions of a morphological processor.

- **composition** : If T_1 is a FST from I_1 to O_1 and T_2 is a FST from O_1 to O_2, then composition of T_1 and T_2 ($T_1 o T_2$) maps from I_1 to O_2.

- We use these properties of FSTs in the creation of the FST for a morphological processor.

A FST for Simple English Nominals



FST for stems

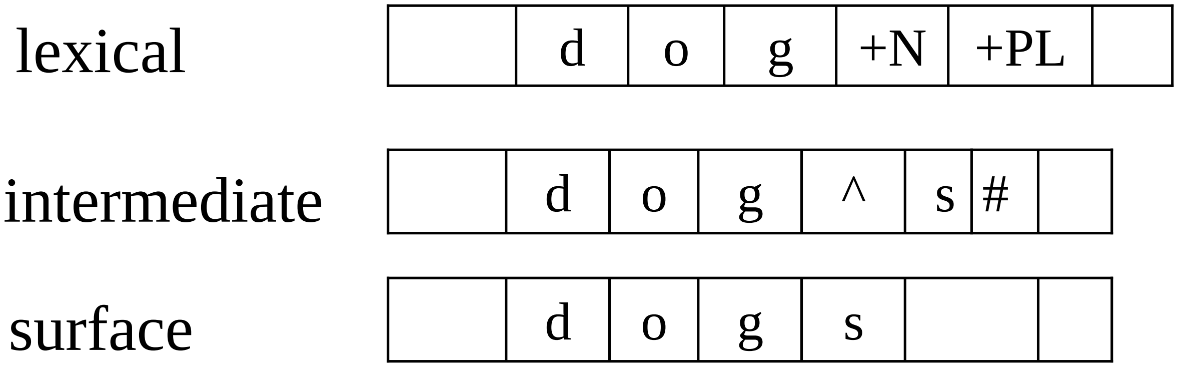
- A FST for stems which maps roots to their root-class

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e se	goose
cat	sheep	sheep
dog	m o:i u:є s:c e	mouse

- fox stands for f:f o:o x:x
- When these two transducers are composed, we have a FST which maps lexical forms to intermediate forms of words for simple English noun inflections.
- Next thing that we should handle is to design the FSTs for orthographic rules, and combine all these transducers.

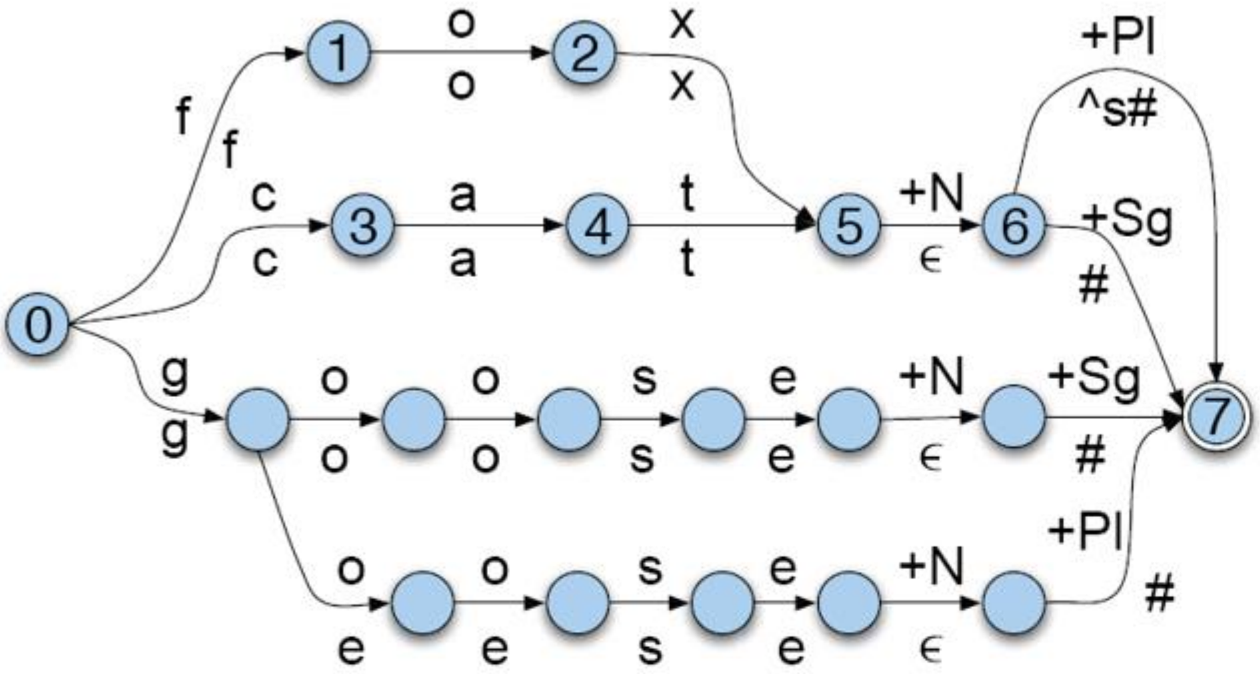
Multi-Level Multi-Tape Machines

- A frequently use FST idiom, called **cascade**, is to have the output of one FST read in as the input to a subsequent machine.
- So, to handle spelling we use three tapes:
  - **lexical, intermediate** and **surface**
- We need one transducer to work between the lexical and intermediate levels, and a second (a bunch of FSTs) to work between intermediate and surface levels to patch up the spelling.



lexical		d	o	g	+N	+PL		
intermediate		d	o	g	^	s	#	
surface		d	o	g	s			

Lexical to Intermediate FST



Orthographic Rules

- We need FSTs to map intermediate level to surface level.
- For each spelling rule we will have a FST, and these FSTs run parallel.

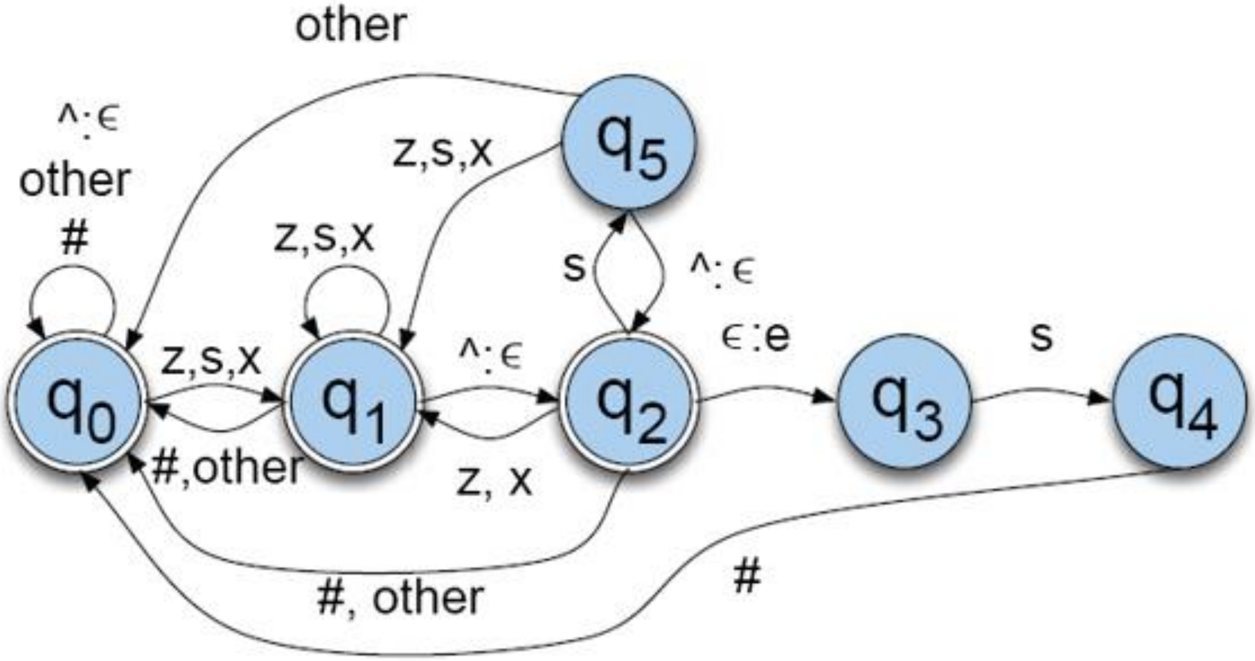
- Some of English Spelling Rules:
  - consonant doubling -- 1-letter consonant doubled before ing/ed -- beg/begging
  - E deletion -- Silent e dropped before ing and ed -- make/making
  - E insertion -- e added after s, z, x, ch, sh before s -- watch/watches
  - Y replacement -- y changes to ie before s, and to i before ed -- try/tries
  - K insertion -- verbs ending with vowel+c we add k -- panic/panicked

- We represent these rules using two-level morphology rules:
  - `a => b / c __ d` rewrite a as b when it occurs between c and d.

FST for E-Insertion Rule

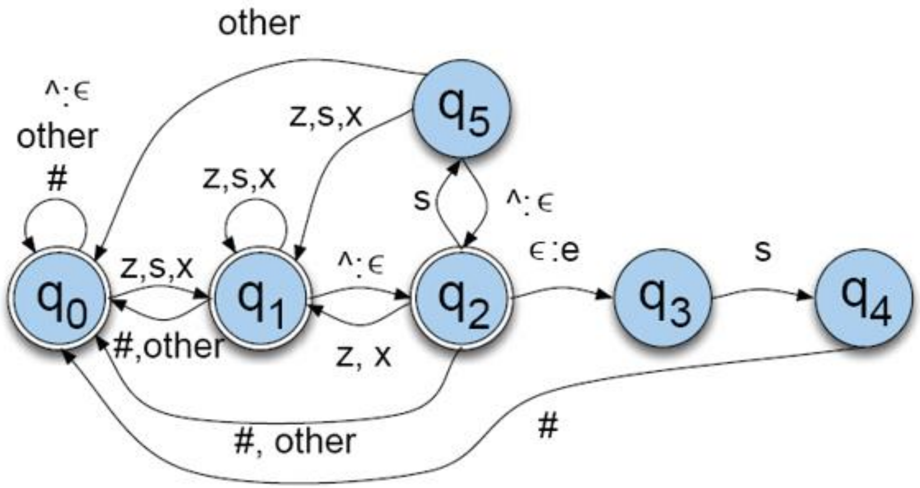
E-insertion rule: ϵ => e / {x,s,z}^ __ s#

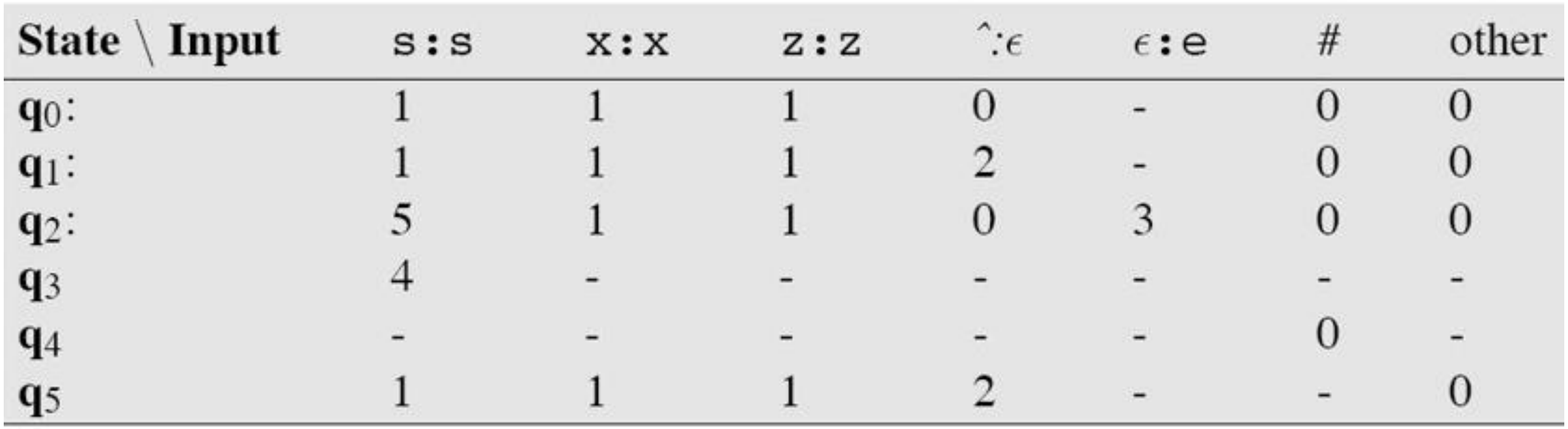
- ^ (morpheme boundary) means ^: ϵ



FST for E-Insertion Rule

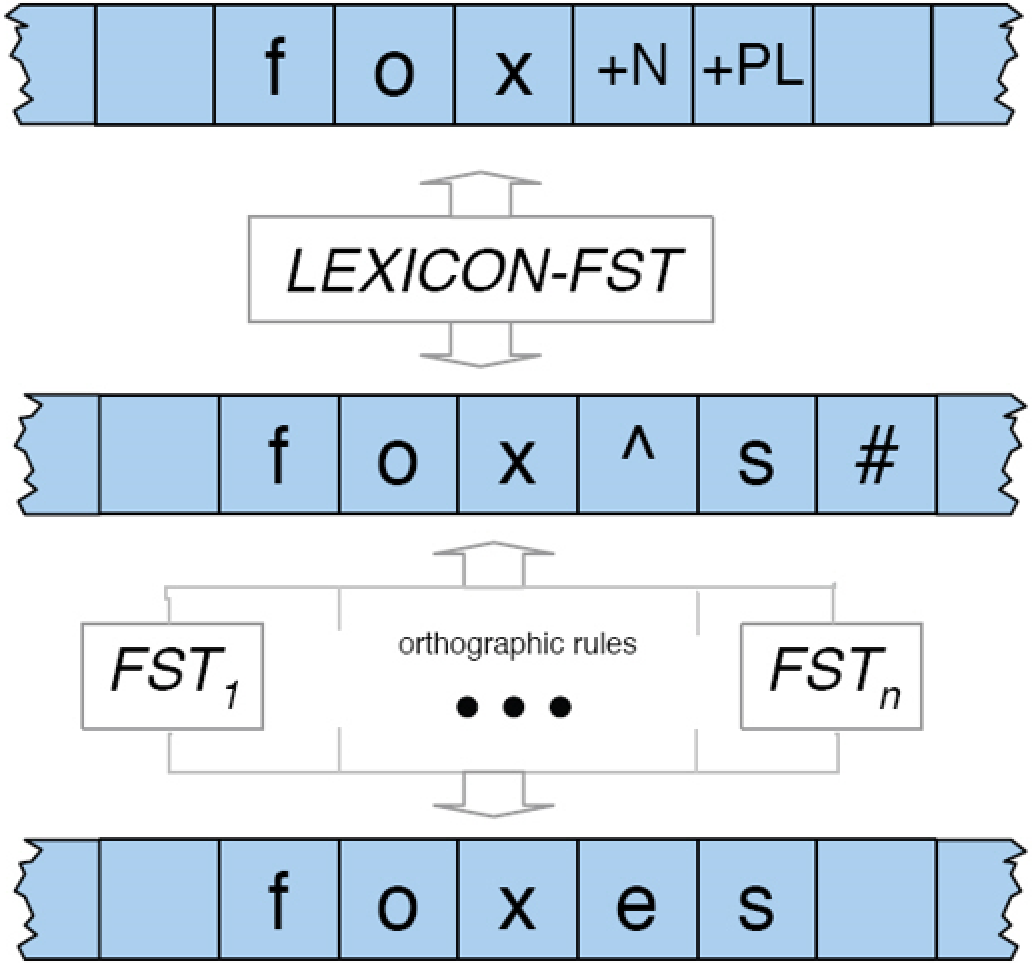
E-insertion rule: ϵ => e / {x,s,z}^ __ s#



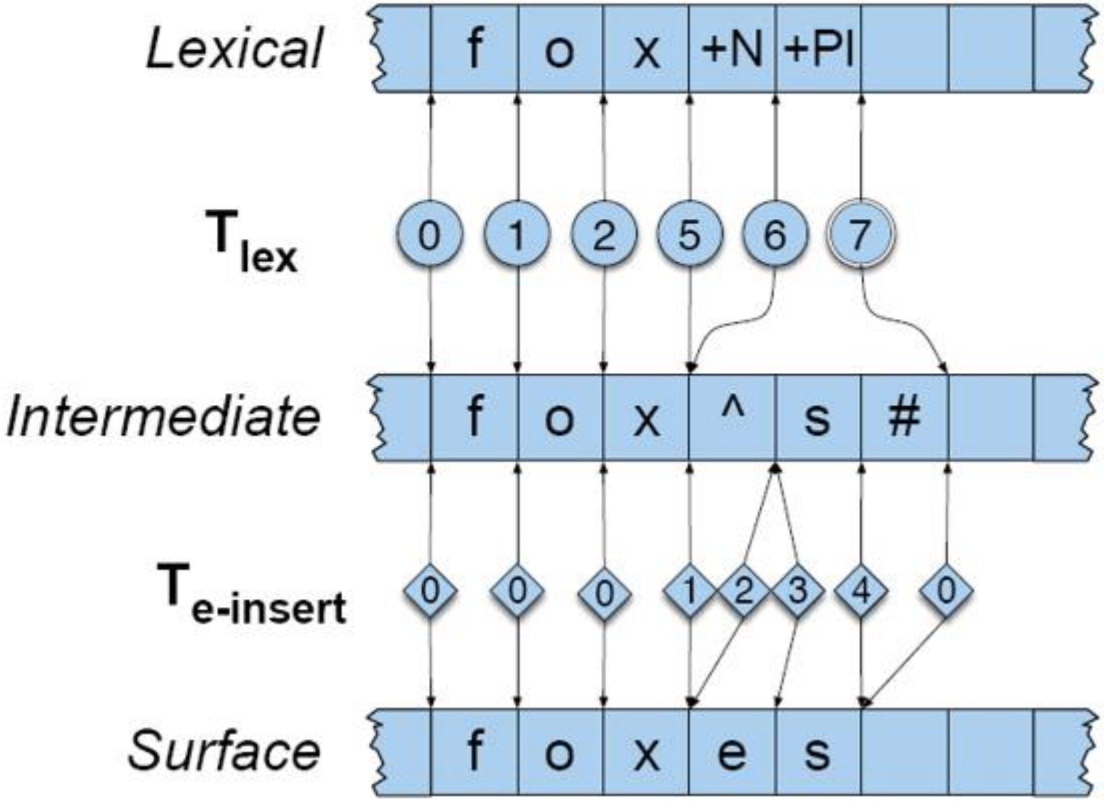


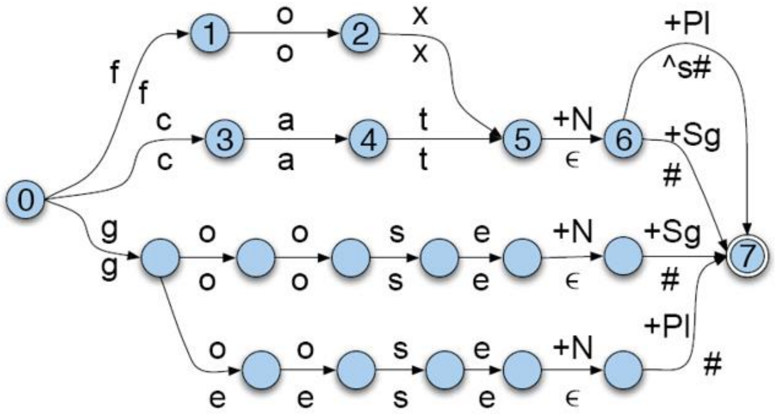
State \ Input	s:s	x:x	z:z	^:ϵ	ϵ:e	#	other
q0:	1	1	1	0	-	0	0
q1:	1	1	1	2	-	0	0
q2:	5	1	1	0	3	0	0
q3	4	-	-	-	-	-	-
q4	-	-	-	-	-	0	-
q5	1	1	1	2	-	-	0

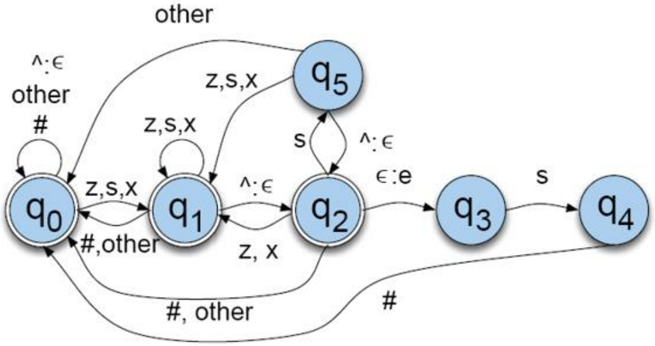
Generating or Parsing with FST Lexicon and Rules



Accepting foxes







Intersection

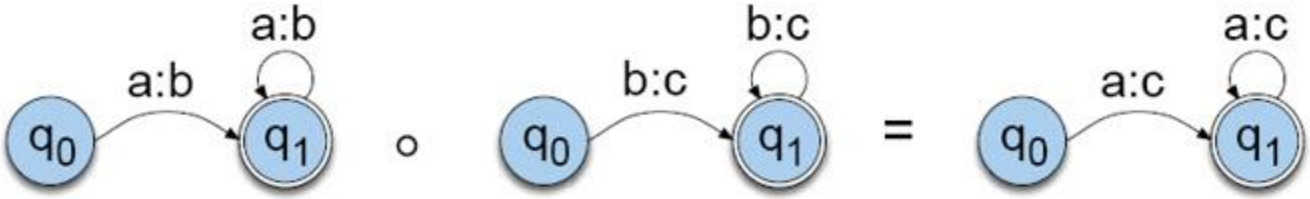
- ***We can intersect all rule FSTs to create a single FST.***
- Intersection algorithm just takes the Cartesian product of states.
  - For each state $\mathbf{q_i}$ of the first machine and $\mathbf{q_j}$ of the second machine, we create a new state $\mathbf{q_{ij}}$
  - For input symbol **a**, if the first machine would transition to state $\mathbf{q_n}$ and the second machine would transition to $\mathbf{q_m}$ the new machine would transition to $\mathbf{q_{nm}}$.

Composition

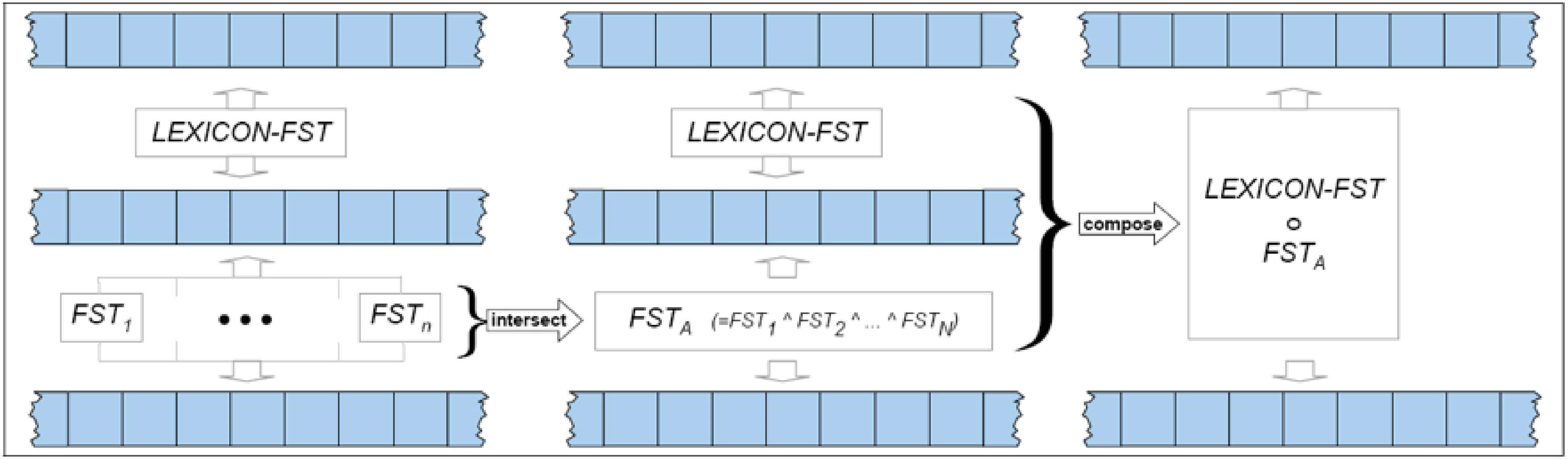
- Cascade can turn out to be somewhat pain.
  - it is hard to manage all tapes
  - it fails to take advantage of restricting power of the machines
- So, it is better to compile the cascade into a single large machine.
- Create a new state (x,y) for every pair of states x ϵ Q_1 and y ϵ Q_2.
- The transition function of composition will be defined as follows:

$\delta((x,y),i{:}o) = (v,z)$ if

there exists c such that $\delta_1(x,i{:}c) = v$ and $\delta_2(y,c{:}o) = z$



Intersect Rule FSTs



Simplified Turkish Noun Morphotactics

in Foma Environment

```
LEXICON NOUNS
aba    POST-NOUN;
aday   POST-NOUN;
benzin POST-NOUN;
…


LEXICON POST-NOUN
+Noun:0 POST-NOUNR;
```

```
LEXICON POST-NOUNR
+A3pl:+lAr PLURAL;
+A3sg:0    PLURAL;


LEXICON PLURAL
+P1sg:+Hm   POSSESSIVE;
+P2sg:+Hn   POSSESSIVE;
+P1pl:+HmHz POSSESSIVE;
+P2pl:+HnHz POSSESSIVE;
+Pnon:0     POSSESSIVE;
+P3sg:+sH   POSSESSIVE;


LEXICON POSSESSIVE
+Acc:+yH  End;
+Dat:+yA  End;
+Loc:+DA  End;
+Abl:+DAn End;
+Gen:+nHn End;
+Ins:+ylA End;
+Nom:0    End;
```

Simplified Turkish Orthographic Rules

in Foma Environment

```
##### Turkish Foma 2016  ####
define ALPHABET [a | e | ı | i | o | ö | u | ü | A | H | … | b | c | ç | d
| f | g|ğ|h|j| k | l | m | n | p | r | s | ş | t | v | y | z | D | … ];
define CONS [b | c | ç | d | f | g | ğ | h | j | k | l | m | n | p | r | s
| ş | t | v | y | z | D | Z | Y | K | J | B];
define VOWEL    [a | e | ı | i | o | ö | u | ü | A | H | … ];
define SVOWEL   [a | e | ı | i | o | ö | u | ü];
define BACKV    [a | ı | u | o];    #kalın ünlüler
define FRONTV   [e | i | ö | ü];    #ince ünlüler
define HIGHV    [ı | i | u | ü];    #dar ünlüler
define FRUNRV   [i | e];            #düz ince
define FRROV    [ö | ü];            #yuvarlak ince
define BKROV    [u | o];            #yuvarlak kalın
define BKUNRV   [a | ı];            #düz kalın
define Xsyn     [s | y | n];
define NDCONS   [c | Z | l | d | D];
```

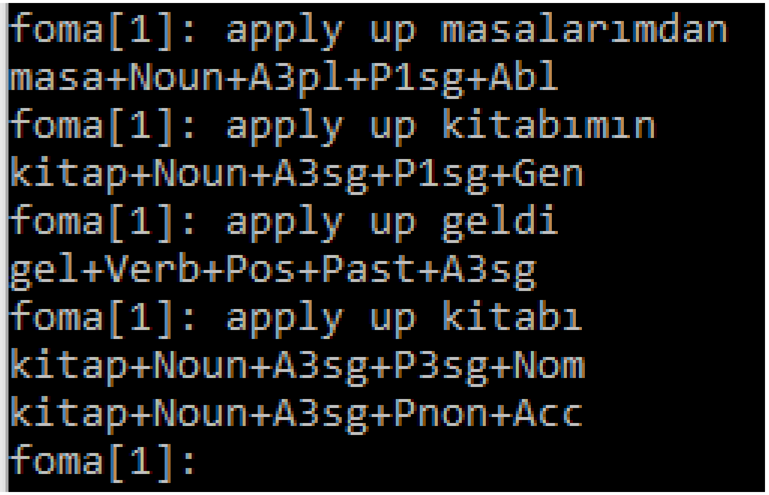
Simplified Turkish Orthographic Rules

in Foma Environment

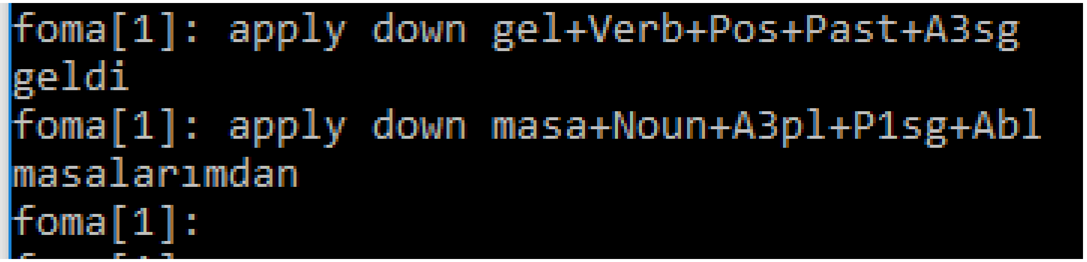
```
#---------------ALTERNATION RULE SECTION----------------------
define AReplacement
   A -> a || [BACKV | … ] [CONS | … | "+"]* _ ;
   A -> e || [FRONTV | …] [CONS | … | "+"]* _ ;

define HReplacement
   H -> u || [BKROV | … ]  [CONS | "+" | … ]* _ ,,
   H -> ü || [FRROV | … ]  [CONS | "+" | … ]* _ ,,
   H -> ı || [BKUNRV | … ] [CONS | "+" | … ]* _ ,,
   H -> i || [FRUNRV | … ] [CONS | "+" | … ]* _ ,,
   H -> 0 || VOWEL "+" _ ;
```

Morphological Processing in Foma Environment



```
foma[1]: apply up masalarımdan
masa+Noun+A3pl+P1sg+Abl
foma[1]: apply up kitabımın
kitap+Noun+A3sg+P1sg+Gen
foma[1]: apply up geldi
gel+Verb+Pos+Past+A3sg
foma[1]: apply up kitabı
kitap+Noun+A3sg+P3sg+Nom
kitap+Noun+A3sg+Pnon+Acc
foma[1]:
```



```
foma[1]: apply down gel+Verb+Pos+Past+A3sg
geldi
foma[1]: apply down masa+Noun+A3pl+P1sg+Abl
masalarımdan
foma[1]:
```