Morphological Analysis and Generation using FST

Finite State Morphological Parsing

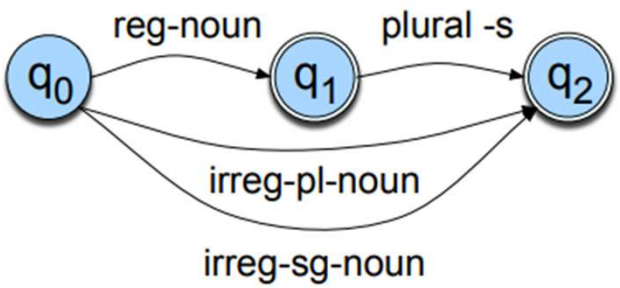
Finite State Morphological parsing

English	
Input	Morphologically Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +Pl
geese	goose +N +Pl
goose	goose +N +Sg
goose	goose +V
gooses	goose +V +1P +Sg
merging	merge +V +PresPart
caught	catch +V +PastPart
caught	catch +V +Past

In order to build a morphological parser, we'll need at least the following:

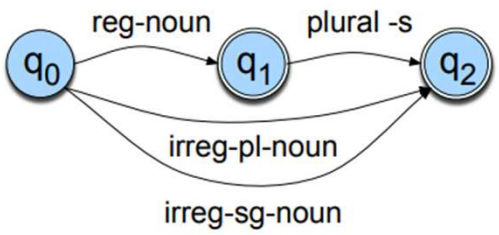
1. **lexicon:** the list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc.).
2. **morphotactics:** the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the fact that the English plural morpheme follows the noun rather than preceding it is a morphotactic fact.
3. **orthographic rules:** these **spelling rules** are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the $y \rightarrow ie$ spelling rule discussed above that changes *city* + *-s* to *cities* rather than *citys*).

1. BUILDING A FINITE-STATE LEXICON

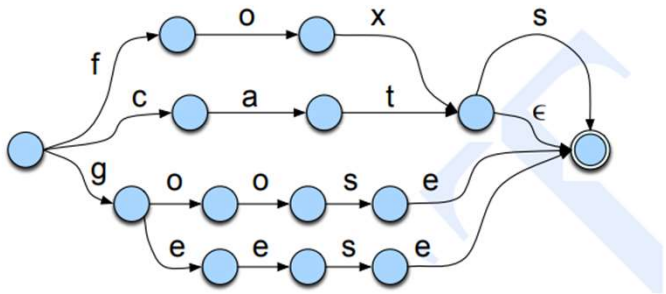


- Consider

reg-noun	irreg-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
aardvark	mice	mouse	



Expand each arc (reg-noun) stem arc with all the morphemes that makeup the set of regular noun stem.

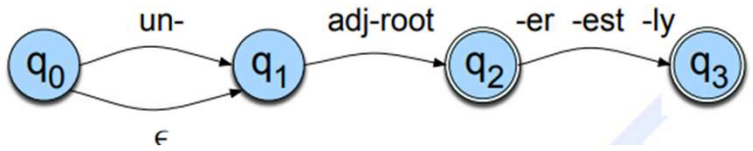


Incorrectly accepts (foxs)

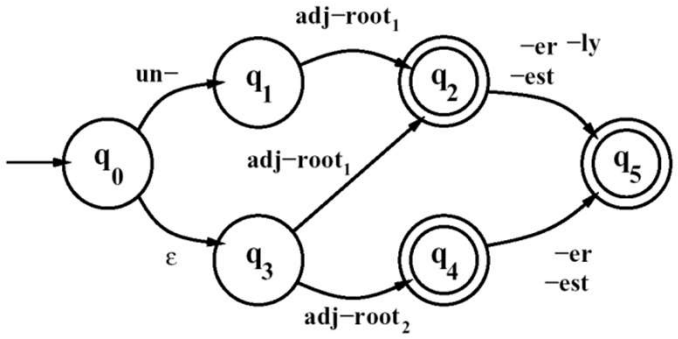
Incorrectly accepts (foxs)

FSAs for Adjectives – Lexicons and Morphotactics

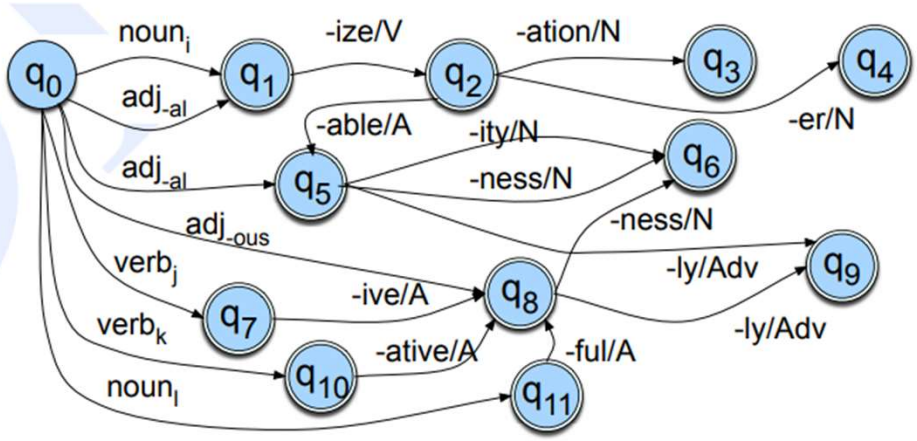
big, bigger, biggest,
happy, happier, happiest, happily
unhappy, unhappier, unhappiest, unhappily
clear, clearer, clearest, clearly, unclear, unclearly
cool, cooler, coolest, coolly
red, redder, reddest
real, unreal, really



The FSA recognizes all the listed adjectives, and ungrammatical forms like unbig, redly, and realest.



FSA for a fragment of Derivational Morphology



Examples:

- noun$_i$: fossil
- verb$_j$: pass
- verb$_k$: conserve
- noun$_l$: wonder

Exceptions:

- noun$_i$: *apology* accepts *-ize* but *apologization* sounds odd
- verb$_j$: *detect* accepts *-ive* but it becomes a noun, not an adjective
- verb$_k$: *cause* accepts *-ative* but *causitiveness* sounds odd
- noun$_l$: *arm* accepts *-ful* but it becomes a noun, not an adjective

Finite State Transducers(FST)

- A transducer maps between one representation and another
- A Finite State Transducer(FST) maps between two sets of symbols.

FST as recognizer:

- a transducer that takes a pair of strings as input and output *accept* if the string-pair is in the string-pair language, and a *reject* if it is not.

FST as generator:

- a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.

FST as transducer:

- A machine that reads a string and outputs another string.

FST as set relater:

- A machine that computes relation between sets.

Let's begin with a formal definition. An FST can be formally defined with 7 parameters:

Q	a finite set of N states $q_0, q_1, \ldots, q_{N-1}$
Σ	a finite set corresponding to the input alphabet
Δ	a finite set corresponding to the output alphabet
$q_0 \in Q$	the start state
$F \subseteq Q$	the set of final states
$\delta(q,w)$	the transition function or transition matrix between states; Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\delta(q,w)$ returns a set of new states $Q' \in Q$. δ is thus a function from $Q \times \Sigma^*$ to 2^Q (because there are 2^Q possible subsets of Q). δ returns a set of states rather than a single state because a given input may be ambiguous in which state it maps to.
$\sigma(q,w)$	the output function giving the set of possible output strings for each state and input. Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\sigma(q,w)$ gives a set of output strings, each a string $o \in \Delta^*$. σ is thus a function from $Q \times \Sigma^*$ to 2^{Δ^*}

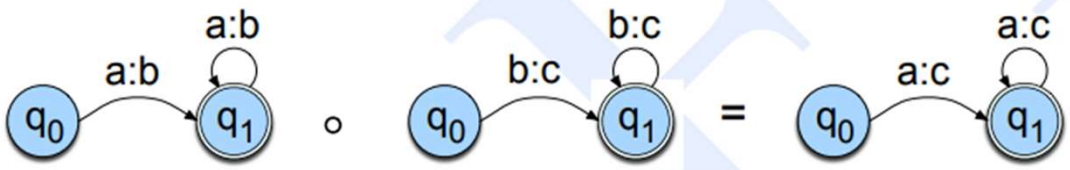
Properties of FSTs

INVERSION

- **inversion**: The inversion of a transducer T (T^{-1}) simply switches the input and output labels. Thus if T maps from the input alphabet I to the output alphabet O, T^{-1} maps from O to I.

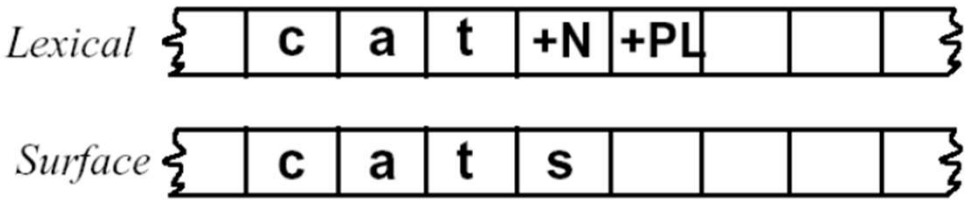
COMPOSITION

- **composition**: If T_1 is a transducer from I_1 to O_1 and T_2 a transducer from O_1 to O_2, then $T_1 \circ T_2$ maps from I_1 to O_2.



Finite-State Morphological Parsing
Morphological Parsing with FST

- Given the input, for example, *cats*, we would like to produce `cat +N +PL`.
- Two-level morphology, by Koskenniemi (1983)
  - Representing a word as a correspondence between a **lexical level**
    - Representing a simple concatenation of morphemes making up a word, and
  - The **surface level**
    - Representing the actual spelling of the final word.
- Morphological parsing is implemented by building mapping rules that maps letter sequences like *cats* on the surface level into morpheme and features sequence like `cat +N +PL` on the lexical level.



Formalization of FST

- A finite state transducer $T = L_{in} \times L_{out}$
- Defines a relation between two languages
- L_{in} = { cat, cats, fox, foxes……}
- L_{out} = {cat, cat + N+ PL,…….}

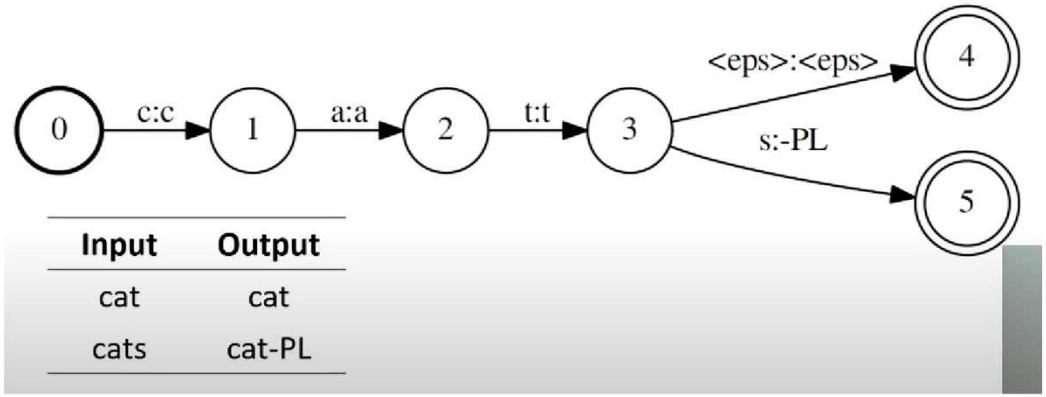
English	
Input	Morphologically Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +Pl
geese	goose +N +Pl
goose	goose +N +Sg
goose	goose +V
gooses	goose +V +1P +Sg
merging	merge +V +PresPart
caught	catch +V +PastPart
caught	catch +V +Past

Finite-State Morphological Parsing

Morphological Parsing with FST

- The automaton we use for performing the mapping between these two levels is the **finite-state transducer** or **FST**.
  - A transducer maps between one set of symbols and another;
  - An FST does this via a finite automaton.
- Thus an FST can be seen as a two-tape automaton which **recognizes** or **generates *pairs*** of strings.
- The FST has a more general function than an FSA:
  - An FSA defines a formal language
  - An FST defines a relation between sets of strings.
- Another view of an FST:
  - A machine reads one string and generates another.

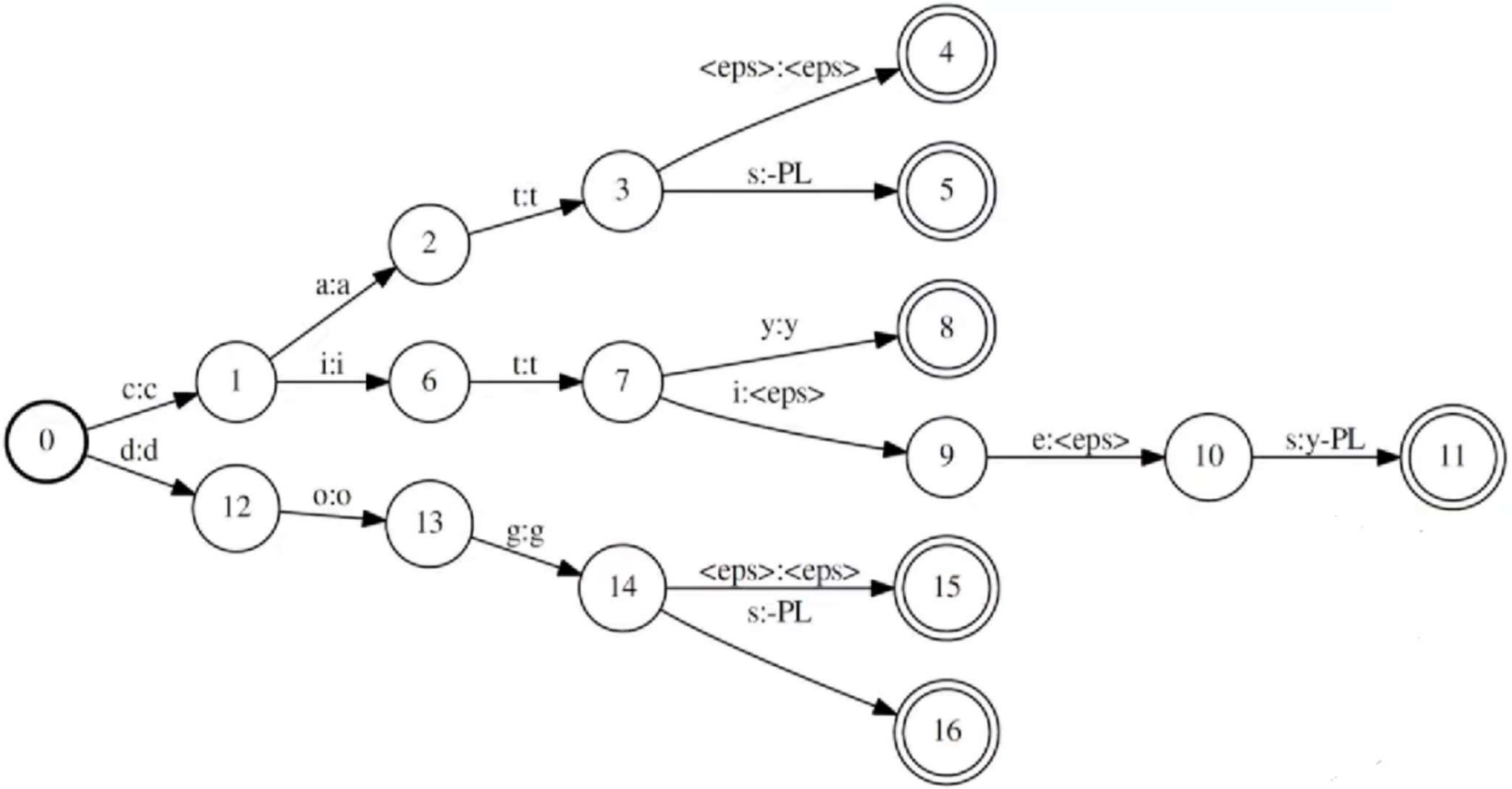
Example FST



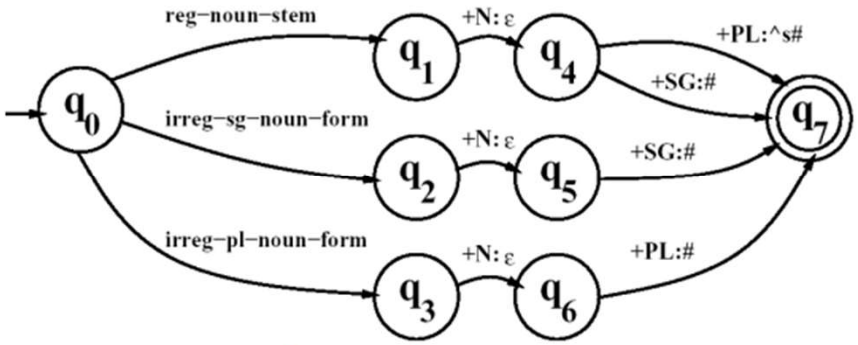
Input	Output
cat	cat
cats	cat-PL

Expand this FST to perform the following transformations:

Input	Output
cat	cat
cats	cat-PL
dog	dog
dogs	dog-PL
city	city
cities	city-PL


0
c:c
d:d
1
12
a:a
i:i
o:o
2
6
13
t:t
t:t
g:g
3
7
14
<eps>:<eps>
s:-PL
y:y
i:<eps>
<eps>:<eps>
s:-PL
4
5
8
9
15
16
e:<eps>
10
s:y-PL
11

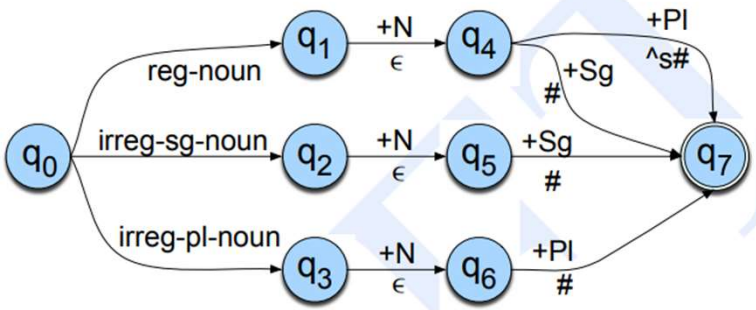
- Inversion is useful because it makes it easy to convert a FST-as-parser into an FST-as-generator.
- Composition is useful because it allows us to take two transducers than run in series and replace them with one complex transducer.
  - $T_1 \circ T_2(S) = T_2(T_1(S))$



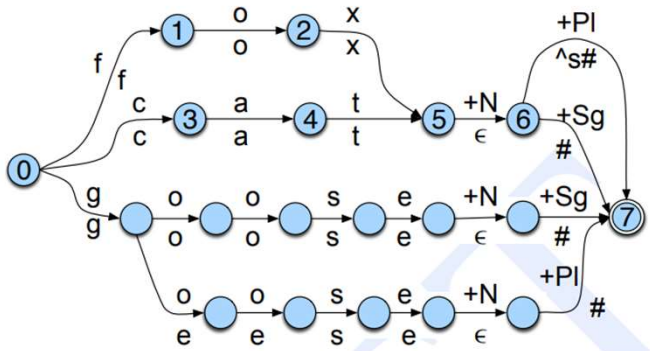
A transducer for English nominal number inflection T_{num}

Reg-noun	Irreg-pl-noun	Irreg-sg-noun
fox	g o:e o:e s e	goose
fat	sheep	sheep
fog	m o:i u:ɛs:c e	mouse
aardvark		

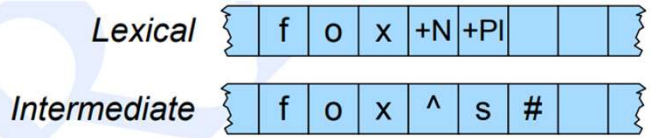
Finite State Transducers



Schematic Transducer



Equivalent Noun Parser



A schematic view of the lexical and intermediate tapes.

```
c:c  a:a  t:t  +N:ϵ  +Pl:^s#
```

Orthographic Rules / Spelling rules

Name	Description of Rule	Example
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*, *-i* before *-ed*	try/tries
K insertion	verbs ending with *vowel* + *-c* add *-k*	panic/panicked

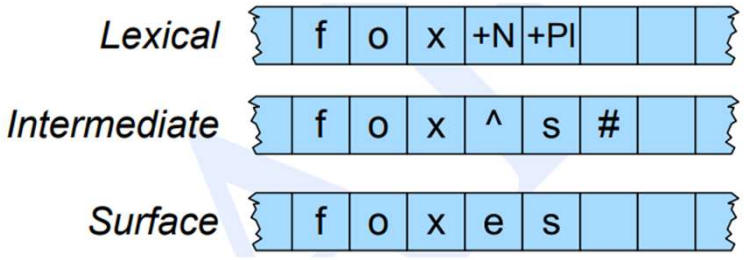
English plural -s:

- cat ⇒ cats, dog ⇒ dogs
- but: fox ⇒ foxes, buzz ⇒ buzzes

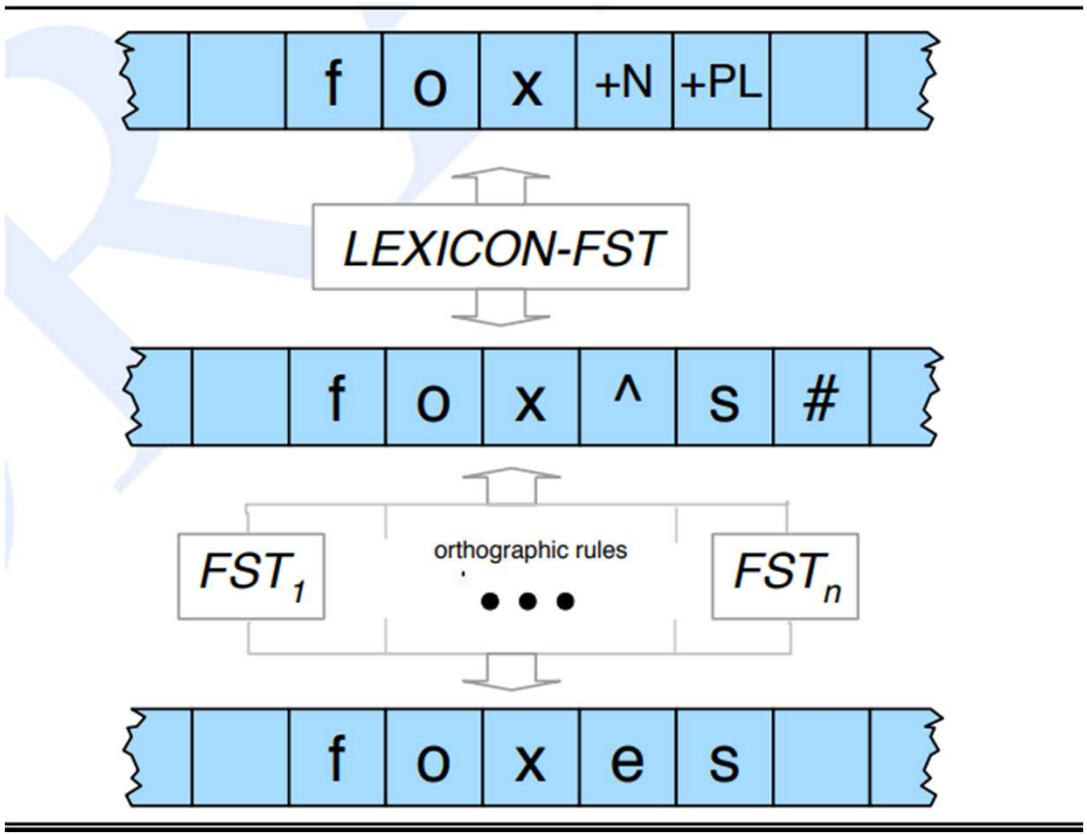
We define an **intermediate representation** which captures **morpheme boundaries** (^) and **word boundaries** (#):

Lexicon:	cat+N+PL	fox+N+PL
Intermediate representation:	cat^s#	fox^s#
Surface string:	cats	foxes

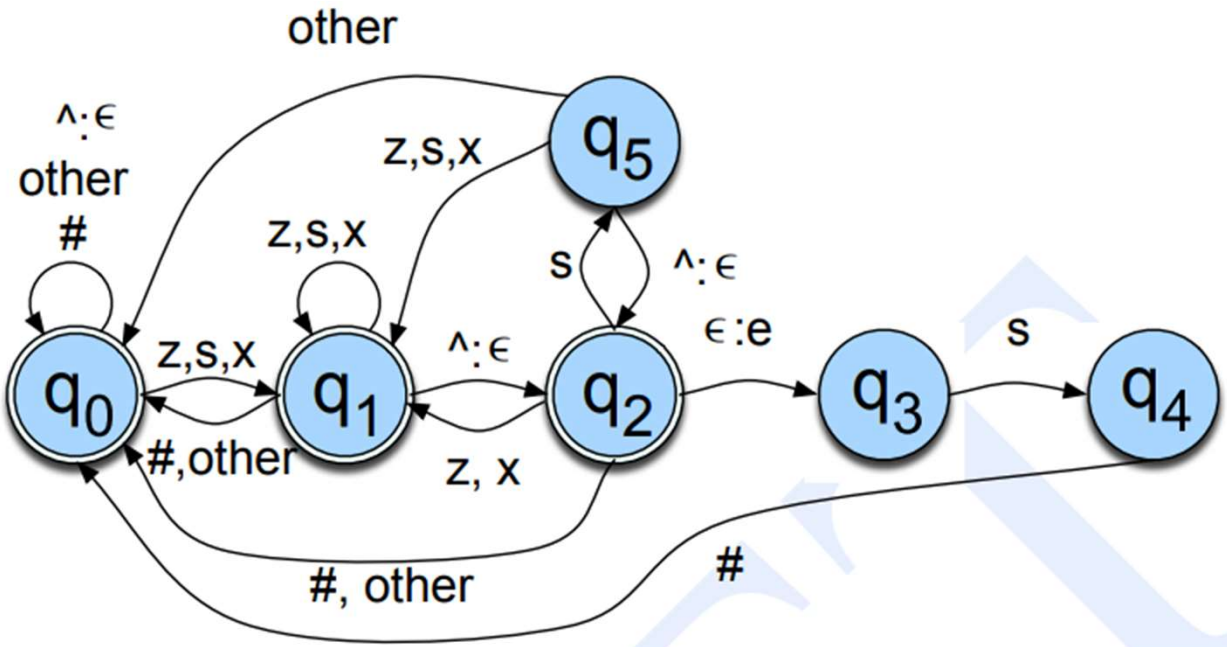
Intermediate-to-Surface Spelling Rule:
If **plural 's'** follows a morpheme ending in **'x','z'** or **'s'**, ***insert 'e'***.



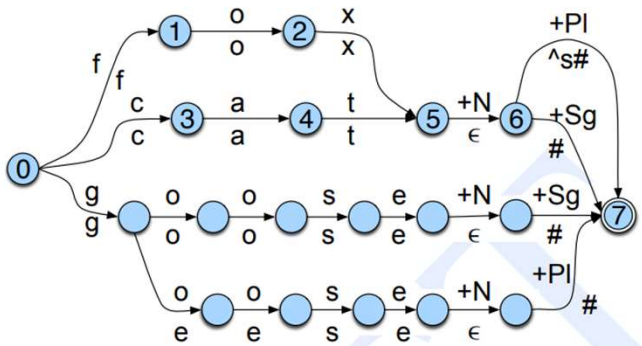
$$\epsilon \rightarrow e / \left\{ \begin{matrix} x \\ s \\ z \end{matrix} \right\} \hat{} \_\_ s\#$$

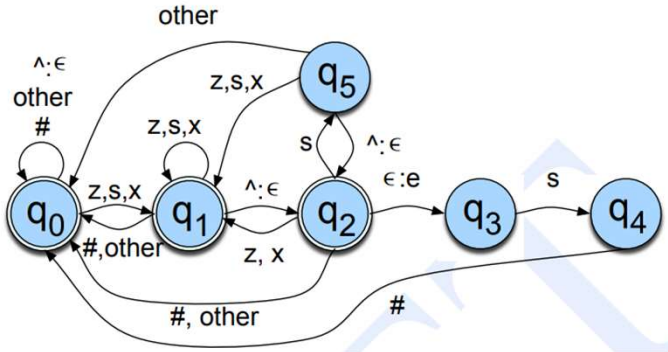

f o x +N +PL
LEXICON-FST
f o x ^ s #
FST1
orthographic rules
FSTn
f o x e s

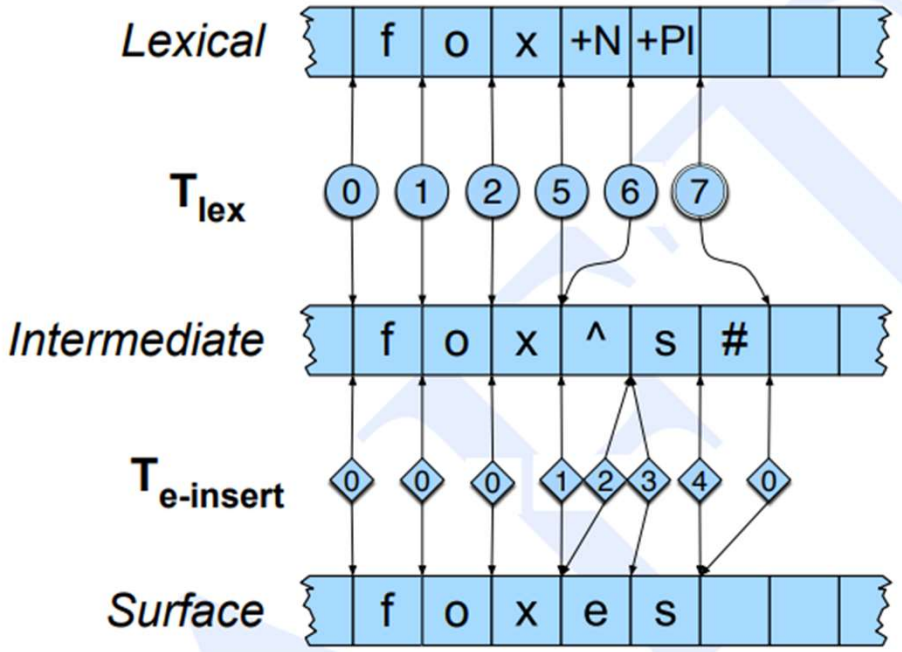
Transducer for e-insertion



Transducer accepting “Foxes”







Applications of FST

- Spell Checking and Correction: FSTs are utilized to create efficient spell-checking systems that can automatically correct misspelled words by comparing input text against a dictionary of correctly spelled words.
- Grammar Checking: FSTs can assist in grammar checking by analyzing the syntax and structure of sentences, identifying grammatical errors, and suggesting corrections or improvements.
- Morphological Analysis: FSTs are valuable for analyzing the morphology of words, including inflectional and derivational morphemes. They can segment words into their root forms and apply morphological rules to generate different word forms.
- Part-of-Speech Tagging: FSTs are used in part-of-speech tagging systems to assign grammatical categories (such as noun, verb, adjective, etc.) to words in a sentence based on their context and syntactic properties.

Applications of FST

- Named Entity Recognition (NER): FSTs play a role in named entity recognition tasks by identifying and classifying named entities such as names of people, organizations, locations, and dates within text data.
- Machine Translation: FSTs are employed in machine translation systems to model the translation process between different languages. They can handle linguistic transformations such as word reordering, phrase translation, and morphological changes.
- Speech Recognition: FSTs are utilized in speech recognition systems to transcribe spoken language into text. They model phonetic patterns and language rules to accurately convert spoken utterances into written form.
- Text Normalization: FSTs help in text normalization tasks by standardizing text data, including handling variations in spelling, punctuation, and formatting to improve the accuracy of downstream NLP tasks.
- Information Extraction: FSTs can extract structured information from unstructured text data by identifying relevant entities, relationships, and events mentioned within the text.
- Dialogue Systems: FSTs are employed in dialogue systems, including chatbots and virtual assistants, to process user queries, generate responses, and maintain conversational context.

References

- Daniel Jurafsky and James H Martin “Speech and Language Processing”, Prentice Hall 2017.
- Weblink for Applications of FST - GeeksforGeeks