

# Natural Language Processing and Machine Translation

## Morphology, Computational Phonology and Speech Processing

Abhishek Koirala

M.Sc. in Informatics and  
Intelligent Systems  
Engineering

# Theory of Automata

- Related Terminologies
  - Symbols
  - Alphabets
  - String
  - Length of String
  - Kleene Star
  - Kleene Closure/Plus
  - Language
- Types of automata
  - Deterministic Finite Automata
  - Non-deterministic Finite Automata

# Regular Expressions

- Language accepted by finite automata
- Usage
  - Data Pre-processing
  - Rule based information mining systems
  - Pattern matching
  - Text feature Engineering
  - Web Scraping
  - Data Extraction, etc

# Regular Expressions

```
# re.search()
```

```
import re
result=re.search('students',r'All of the students of NLPMT are genius')
print(result.group())
```

```
students
```

```
# re.match()
```

```
import re
result=re.match('All',r'All of the students of NLPMT are genius')
print(result)
print(result.group())
```

```
<re.Match object; span=(0, 3), match='All'>
All
```

```
# re.sub()
```

```
import re
string="abc xxx abc yyy"
result=re.sub(r"xxx|yyy", "abc", string)
print(result)
```

```
abc abc abc abc
```

```
# re.findall()
```

```
import re
result1=re.findall('founded','Andrew NG founded Coursera, He also founded deeplearning.ai')
print(result1)
```

```
['founded', 'founded']
```



# Words

Text Sample from a brown corpus:

He stepped out into the hall, was delighted to encounter a water bottle.

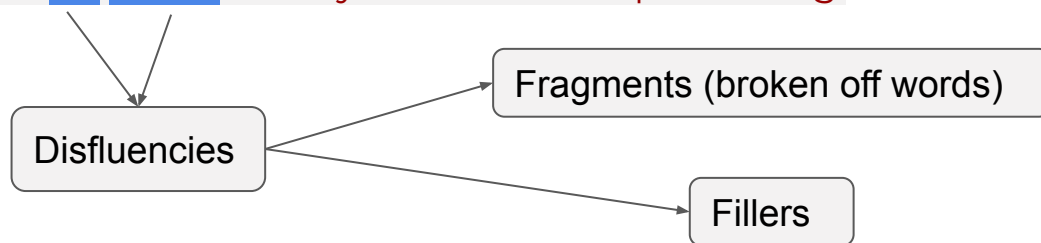
=> 13 words (if we don't count punctuations)

=> 15 words (if we count punctuations)

But are punctuations critical?

Utterance sample from Switchboard corpus of American telephone conversation

I do uh main- mainly business data processing.



# Wordform

- Full inflected or derived form of the word

How many words are there in English language ?

- Types
- Tokens

We arrived at the class early today because the roads were empty.

⇒ 11 types

⇒ 12 tokens

Number of words in a language refers to word types.

# Heap's law

- Larger the **corpora**, the more word types we find

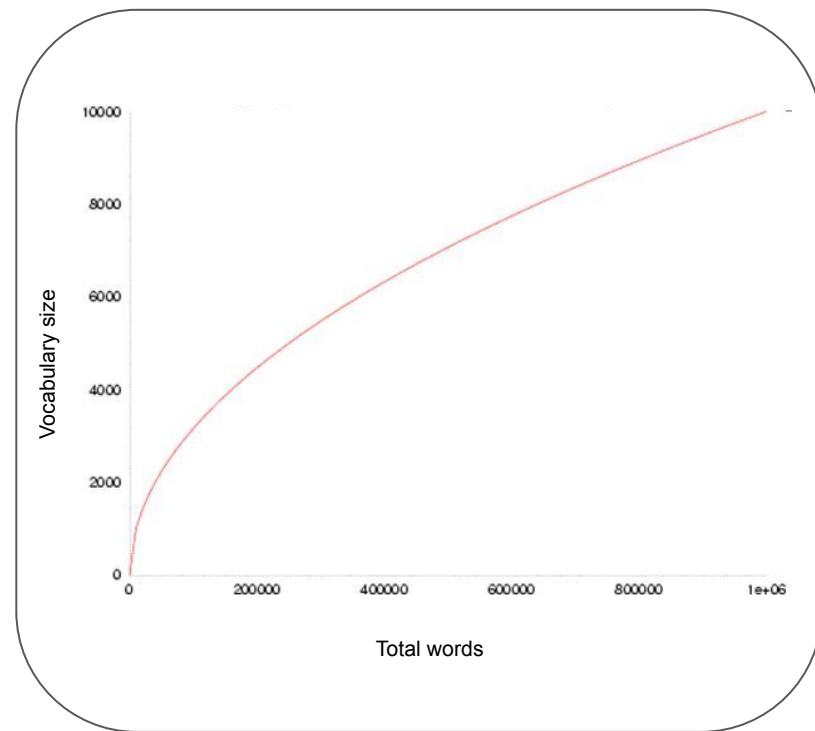
Experiment:

- 1) Read a book/newspaper/website
  - a) Record every time you see a new word
    - i)  $v \rightarrow$  number of new words seen
    - ii)  $n \rightarrow$  total words
  - b) Plot  $v$  vs  $n$

Vocab growth

$$v = k \times n^b$$

$b \approx 0.5$



# Corpora

- Large collection of texts which represent a sample of a particular variety or use of language
- How does corpus vary?
  - Algorithms
  - Code switching
  - Genre of text
  - Change of language over time

How can one create a useful corpus?

**Datasheet / data statement :** Motivation, Situation, Language variety, Speaker  
Demographics, Collection process, Annotation process, Distribution



# Tokenization

Commonly divided into 3 types:

- Word
- Character
- Subword (n-gram characters)

Never give up

Word tokenization  $\Rightarrow$  Never - give - up

Smarter

Character tokenization  $\Rightarrow$  s - m - a - r - t - e - r

Subword tokenization  $\Rightarrow$  smart - er

## Byte pair encoding for tokenization

- Uses corpus statistics to decide how to segment a text into tokens
- Merges most frequently occurring character or character sequences iteratively.
- Contains two parts
  - Token learner
  - Token segmenter
- Steps to learn BPE:
  - Split the words in the corpus into characters after appending </w>
  - Initialize the vocabulary with unique characters in the corpus
  - Compute frequency of a pair of characters or character sequences in corpus
  - Merge the most frequent pair in corpus
  - Save the best pair to the vocabulary
  - Repeat the steps certain number of times.

# Byte Pair Encoding for Tokenization

## Token Learner

```
train_data = {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}
```

### Iteration 1

```
new merge: ('t', '</w>')  
train data: {'n e w e s t</w>': 6, 'l o w e r </w>': 2, 'l o w </w>': 5, 'w i d e s t</w>': 3}
```

### Iteration 2 ¶

```
new merge: ('s', 't</w>')  
train data: {'l o w e r </w>': 2, 'n e w e s t</w>': 6, 'w i d e s t</w>': 3, 'l o w </w>': 5}
```

### Iteration 3

```
new merge: ('e', 'st</w>')  
train data: {'l o w e r </w>': 2, 'l o w </w>': 5, 'n e w e s t</w>': 6, 'w i d e s t</w>': 3}
```

### Iteration 4

```
new merge: ('l', 'o')  
train data: {'w i d e s t</w>': 3, 'l o w </w>': 5, 'n e w e s t</w>': 6, 'l o w e r </w>': 2}
```

### Iteration 5

```
new merge: ('lo', 'w')  
train data: {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t</w>': 6, 'w i d e s t</w>': 3}
```

### Iteration 6

```
new merge: ('e', 'w')  
train data: {'w i d e s t</w>': 3, 'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t</w>': 6}
```

### Iteration 7

```
new merge: ('ew', 'est</w>')  
train data: {'n e w e s t</w>': 6, 'l o w </w>': 5, 'l o w e r </w>': 2, 'w i d e s t</w>': 3}
```

### Iteration 8

```
new merge: ('n', 'ewest</w>')  
train data: {'n e w e s t</w>': 6, 'l o w </w>': 5, 'l o w e r </w>': 2, 'w i d e s t</w>': 3}
```

### Iteration 9

```
new merge: ('low', '</w>')  
train data: {'n e w e s t</w>': 6, 'w i d e s t</w>': 3, 'l o w </w>': 5, 'l o w e r </w>': 2}
```

### Iteration 10

```
new merge: ('i', 'd')  
train data: {'l o w e r </w>': 2, 'n e w e s t</w>': 6, 'w i d e s t</w>': 3, 'l o w </w>': 5}
```

# Byte Pair Encoding for Tokenization

## Token Segmenter

```
encode("lowest")
```

**word split into characters:** ('l', 'o', 'w', 'e', 's', 't', '')

**Iteration 1:**

bigrams in the word: {('l', 'o'), ('t', '</w>'), ('o', 'w'), ('w', 'e'), ('e', 's'), ('s', 't')}

candidate for merging: ('t', '</w>')

word after merging: ('l', 'o', 'w', 'e', 's', 't</w>')

**Iteration 2:**

bigrams in the word: {('l', 'o'), ('w', 'e'), ('e', 's'), ('s', 't</w>'), ('o', 'w')}

candidate for merging: ('s', 't</w>')

word after merging: ('l', 'o', 'w', 'e', 'st</w>')

**Iteration 3:**

bigrams in the word: {('l', 'o'), ('w', 'e'), ('e', 'st</w>'), ('o', 'w')}

candidate for merging: ('e', 'st</w>')

word after merging: ('l', 'o', 'w', 'est</w>')

**Iteration 4:**

bigrams in the word: {('l', 'o'), ('w', 'est</w>'), ('o', 'w')}

candidate for merging: ('l', 'o')

word after merging: ('lo', 'w', 'est</w>')

**Iteration 5:**

bigrams in the word: {('w', 'est</w>'), ('lo', 'w')}

candidate for merging: ('lo', 'w')

word after merging: ('low', 'est</w>')

**Iteration 6:**

bigrams in the word: {('low', 'est</w>')}

candidate for merging: ('low', 'est</w>')

**Candidate not in BPE merges, algorithm stops.**

('low', 'est')

[https://ufal.mff.cuni.cz/~helcl/courses/npfl116/ipython/byte\\_pair\\_encoding.html](https://ufal.mff.cuni.cz/~helcl/courses/npfl116/ipython/byte_pair_encoding.html)

# Tokenization

## Problems during tokenization

- Internal punctuation

I have completed M.E. in Computer Engineering. I am going for Ph.D. soon.

- Numerical expressions

The total area of Nepal is 1,47,181 sq. km.

- Special characters

He earns \$5000 per month.

The value added tax rate for Nepal is 13%.

Tom Hanks was born on 9th July, 1956.

# Tokenization

## Tokenization libraries in use

- NLTK
  - word\_tokenize

```
s1= ''' I have completed M.E. in Computer Engineering. I am going for Ph.D. soon.'''  
s2= ''' The total area of Nepal is 1,47,181 sq. km.'''  
s3= ''' He earns $5000 per month.'''  
s4= '''The value added tax rate for Nepal is 13%.'''  
s5= '''Tom Hanks was born on 9th July, 1956.'''
```

```
['I',  
'have',  
'completed',  
'M.E',  
'',  
'in',  
'Computer',  
'Engineering',  
'',  
'I',  
'am',  
'going',  
'for',  
'Ph.D.',  
'soon',  
'']
```

```
['The', 'total', 'area', 'of', 'Nepal', 'is', '1,47,181', 'sq', '.', 'km', '.']
```

```
['He', 'earns', '$', '5000', 'per', 'month', '.']
```

```
['The', 'value', 'added', 'tax', 'rate', 'for', 'Nepal', 'is', '13', '%', '.']
```

```
['Tom', 'Hanks', 'was', 'born', 'on', '9th', 'July', ',', ',', '1956', '.']
```

# Tokenization

- Treebank Word Tokenizer

```
s1= ''' I have completed M.E. in Computer Engineering. I am going for Ph.D. soon.'''  
s2= ''' The total area of Nepal is 1,47,181 sq. km.'''  
s3= ''' He earns $5000 per month.'''  
s4= '''The value added tax rate for Nepal is 13%.'''  
s5= '''Tom Hanks was born on 9th July, 1956.'''
```

```
['I', 'have', 'completed', 'M.E.', 'in', 'Computer', 'Engineering.', 'I', 'am', 'going', 'for', 'Ph.D.', 'soon', '.']
```

```
['The', 'total', 'area', 'of', 'Nepal', 'is', '1,47,181', 'sq.', 'km', '.']
```

```
['He', 'earns', '$', '5000', 'per', 'month', '.']
```

```
['The', 'value', 'added', 'tax', 'rate', 'for', 'Nepal', 'is', '13', '%', '.']
```

```
['Tom', 'Hanks', 'was', 'born', 'on', '9th', 'July', ',', '1956', '.']
```

# Tokenization

- Textblob Word Tokenizer

```
s1= ''' I have completed M.E. in Computer Engineering. I am going for Ph.D. soon.'''  
s2= ''' The total area of Nepal is 1,47,181 sq. km.'''  
s3= ''' He earns $5000 per month.'''  
s4= '''The value added tax rate for Nepal is 13%.'''  
s5= '''Tom Hanks was born on 9th July, 1956.'''
```

```
['I', 'have', 'completed', 'M.E', 'in', 'Computer', 'Engineering', 'I', 'am', 'going', 'for', 'Ph.D', 'soon']
```

```
['The', 'total', 'area', 'of', 'Nepal', 'is', '1,47,181', 'sq', 'km']
```

```
['He', 'earns', '5000', 'per', 'month']
```

```
['The', 'value', 'added', 'tax', 'rate', 'for', 'Nepal', 'is', '13']
```

```
['Tom', 'Hanks', 'was', 'born', 'on', '9th', 'July', '1956']
```



# Tokenization

- Regex Tokenizer

```
text="That U.S.A. poster-print costs $12.40..."
```

```
['That', 'U.S.A.', 'poster-print', 'costs', '$', '12.40', '...']
```

Treebank word tokenizer

```
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40...']
```

WhiteSpace tokenizer

```
pattern = r'''(?x)          # set flag to allow verbose regexps
    (?:[A-Z]\.)+          # abbreviations, e.g. U.S.A.
    | \w+(?:-\w+)*         # words with optional internal hyphens
    | \$?\d+(?:\.\d+)?%?    # currency and percentages, e.g. $12.40, 82%
    | \.\.\.              # ellipsis
    | [][.,;"'()?():_`'-] # these are separate tokens; includes ], [
    ...
```

```
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

## Inflectional morphology

Adding grammatical information to a word via morphological transformation

Study of processes, including affixation and vowel change, that distinguish word forms in certain grammatical categories.

- nouns
  - plural marker: -s (dog + s = dogs)
  - possessive marker: -'s (dog + 's = dog's)
- verbs
  - 3<sup>rd</sup> person present singular: -s (walk + s = walks)
  - past tense: -ed (walk + ed = walked)
  - progressive: -ing (walk + ing = walking)
  - past participle: -en or -ed (eat + en = eaten)
- adjectives
  - comparative: -er (fast + er = faster)
  - superlative: -est (fast + est = fastest)

<https://slideplayer.com/slide/4697089/>

## Derivational morphology

Morphology that creates new lexemes, either by changing the syntactic category of a base or by adding substantial non grammatical meaning of both.

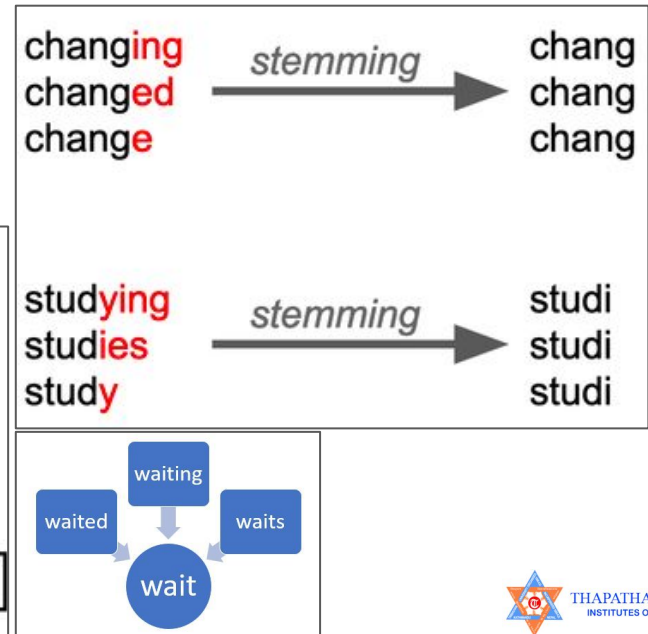
Inflection does not change the grammatical meaning of the word, it typically expresses distinctions like number, case, tense, aspect, among others.

Derivation can be distinguished from compounding, which also creates new lexemes, but by combining two or more bases rather by affixation, reduplication, subtraction or internal modification of various sorts.

- -ic : Noun → Adj ; alcohol → alcoholic
- -ance : Verb → Noun ; clear → clearance
- -ly : Adj → Adv ; exact → exactly
- -ity : Adj → Noun ; active → activity
- -able : Verb → Adj ; read → readable
- -ship : Noun → Noun ; friend → friendship
- re- : Verb → Verb ; cover → recover
- in- : Adj → Adj ; definite → indefinite

# Stemming

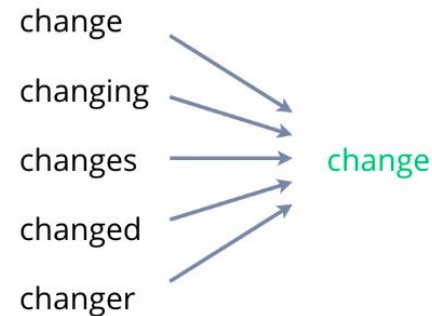
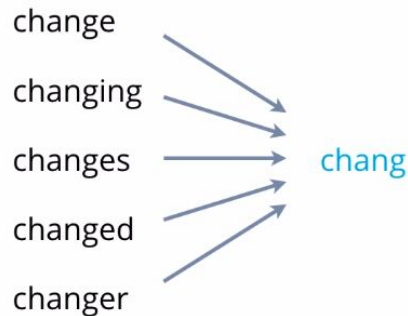
- Lowers inflection in words to their root forms , hence aiding in the preprocessing of text, words and documents for text normalization
- English language has several variants of a single term. These variants in a corpus leads to redundancies in NLP application



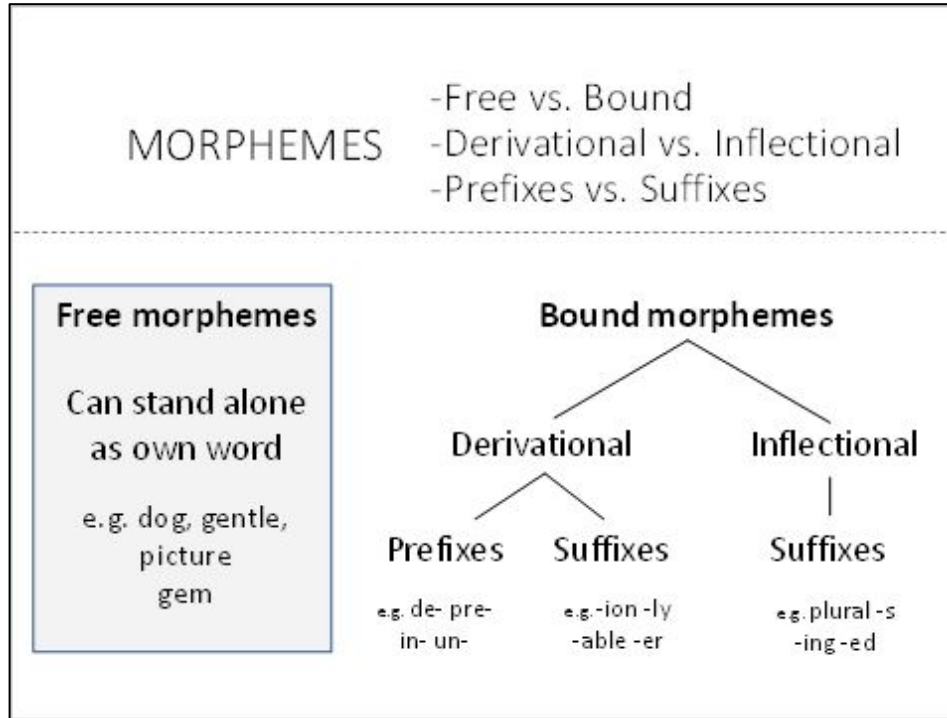
# Lemmatization

- Converts a word to its based form
- Lemmatization considers the context and converts the word into a meaningful base form

## Stemming vs Lemmatization



# Bound vs Free Morphemes



<https://www.education.vic.gov.au/school/teachers/teachingresources/discipline/english/literacy/readingviewing/Pages/litfocuswordmorph.aspx>

# Morphological Processes

- Concatenation
- Reduplication
- Suppletion
- Morpheme internal changes
- Compounding
- Acronyms
- Blending
- Clipping

# Morphological analysis

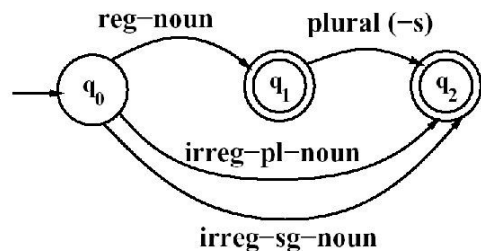
- Knowledge of stem or root is required
  - We need a dictionary (lexicon)
- Morphotactics
  - Eg. Which kind of morphemes follow other kind of morphemes inside the word
    - plural morphemes follows the noun
- Only some endings go on some words

do + er → ok  
be + er → not ok
- Spelling change rules
  - Adjust the surface form using spelling change rules
    - get +er → getter



# Finite State Automaton

## FSA for Inflectional Morphology: English Nouns



<https://slidetodoc.com/morphology-what-is-morphology-finite-state-transducers-two/>

Reg-noun +s  $\rightarrow$  boy + s  $\Rightarrow$  boys  
Irreg-pl-noun  $\rightarrow$  goose  $\Rightarrow$  geese

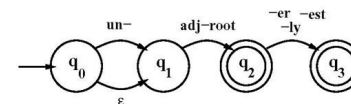


Fig. 3.4 Simple FSA for adjective inflection.

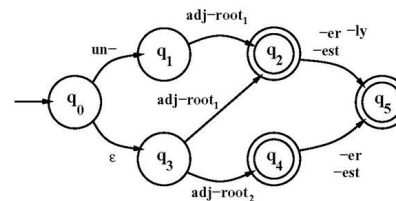


Fig. 3.5 More detailed FSA for adjective inflection.

<https://slideplayer.com/slide/4830914/>

un - happy - er  $\Rightarrow$  unhappier  
happy -ly  $\Rightarrow$  happily  
un - real  $\Rightarrow$  unreal

Can we acquire stem word from Finite State Automaton? Eg. retrieve boy from plural boys

# Finite State Transducers

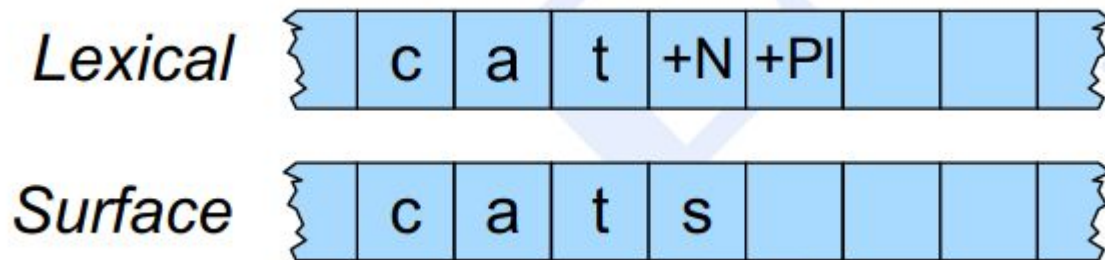
- FST represents a set of pair of strings (eg. i/p, o/p pairs)
  - {(walked, walk + V + PAST)}
- Transducer function: maps input to zero or more outputs
  - Transduce (walk)  $\rightarrow$  {walk + V + PL, walk + N + SG}

Can return multiple answers if ambiguity

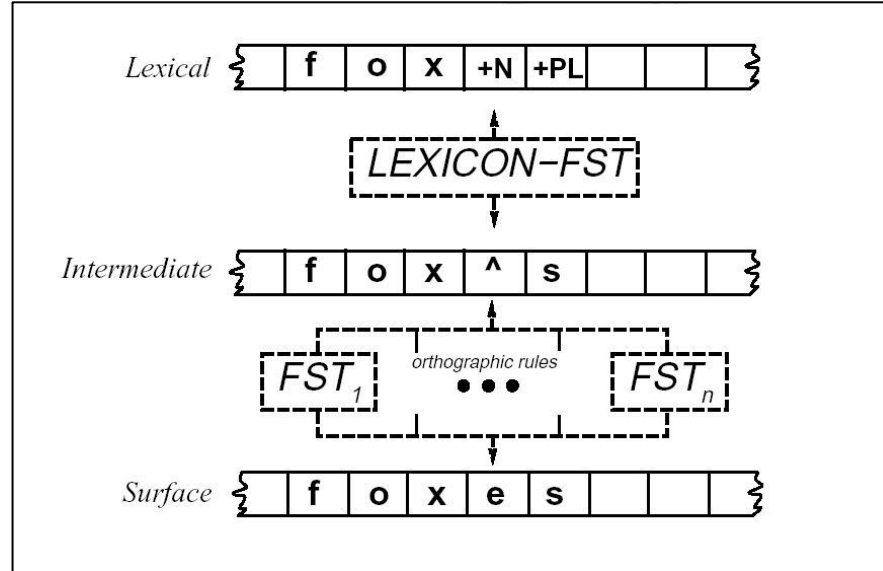
- FSA have input labels (one input tape) while FST have input:output pairs on labels

Parsing vs Recognition ?

# FST for morphological parsing



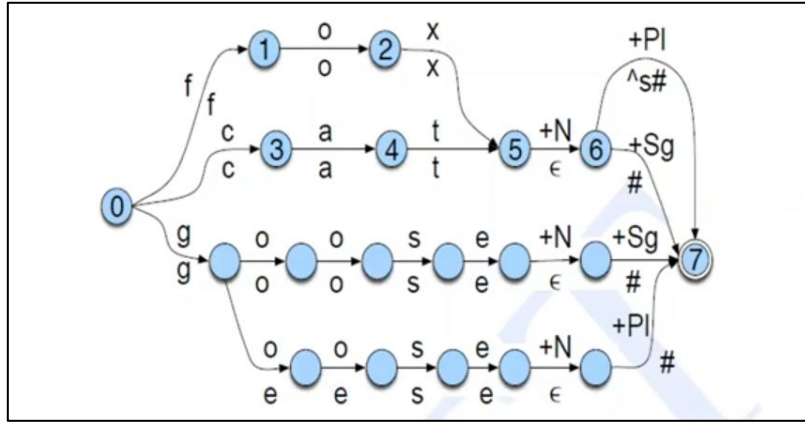
# Two level morphology



<https://slideplayer.com/slide/4818226/>

# Two level morphology

T-lex



T-E -insertion

