BBM-497
Week 4

Regular Expressions

expression	matches...
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}
\w+	a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1
100\s*mk	the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)
abc\b	abc when followed by a word boundary (e.g. in abc! but not in abcd)
perl\B	perl when *not* followed by a word boundary (e.g. in perlert but not in perl stuff)

- You can find some regular expression rules in http://www.rexegg.com/regex-quickstart.html

Anchors —^ and $

```
^The        matches any string that starts with The -> Try it!
end$        matches a string that ends with end
^The end$   exact string match (starts and ends with The end)
```

Quantifiers—* + ? and {}

```
abc*        matches a string that has ab followed by zero or more c
abc+        matches a string that has ab followed by one or more c
abc?        matches a string that has ab followed by zero or one c
abc{2}      matches a string that has ab followed by 2 c
abc{2,}     matches a string that has ab followed by 2 or more c
abc{2,5}    matches a string that has ab followed by 2 up to 5 c
a(bc)*      matches a string that has a followed by zero or more copies of the sequence bc
a(bc){2,5}  matches a string that has a followed by 2 up to 5 copies of the sequence bc
```

OR operator—| or []

```
a(b|c)     matches a string that has a followed by b or c
a[bc]      same as previous
```

Character classes—\d \w \s and .

```
\d         matches a single character that is a digit
\w         matches a word character (alphanumeric character plus underscore)
\s         matches a whitespace character (includes tabs and line breaks)
.          matches any character
```

\d, \w and \s also present their negations with \D, \W and \S respectively. Ex: \D will perform the inverse match with respect to that obtained with \d.

```
\D         matches a single non-digit character
```

you must escape the characters **^.[$()|*+?{** with a backslash \ as they have special meaning.

\d*matchesastringthathasa* before one digit -> Try it!

Grouping and capturing—()

```
a(bc)          parentheses create a capturing group with value bc
a(?:bc)*       using ?: we disable the capturing group
a(?<foo>bc)    using ?<foo> we put a name to the group
```

Bracket expressions—[]

```
[abc]           matches a string that has either an a or a b or a c -> is the same as a|b|c
[a-c]           same as previous
[a-fA-F0-9]     a string that represents a single hexadecimal digit, case insensitively
[0-9]%          a string that has a character from 0 to 9 before a % sign
[^a-zA-Z]       a string that has not a letter from a to z or from A to Z. In this case the ^ is used as n
egation of the expression
```

Example <.+?> matches any character one or more times included inside < and >, expanding as needed <[^<>]+> matches any character except < or > one or more times included inside < and >

Boundaries—\b and \B

```
\babc\b          performs a "whole words only" search
```

Example:

```
colours
colors
they're colours
they're colors
they are colours
they are colors
```

Example: Write a regular expression that defines the language L.

$L = \{ab, aab, abb, aaab, abab, abbb, aaaab, \ldots\}$

Possible Solution : a (a | b)* b

```python
import re
replacement_patterns = [(r'won\'t','will not'),
                        (r'can\'t','can not'),
                        (r'i\'m','i am'),
                        (r'isn\'t','is not'),
                        (r'(\w+)\'ll','\g<1> will'),
                        (r'(\w+)n\'t','\g<1> not'),
                        (r'(\w+)\'ve','\g<1> have'),
                        (r'(\w+)\'s','\g<1> is'),
                        (r'(\w+)\'re','\g<1> are'),
                        (r'(\w+)\'d','\g<1> would')]

class RegexReplacer(object):
    def __init__(self, patterns=replacement_patterns):
        self.patterns = [(re.compile(regex), repl) for (regex,repl) in patterns]

    def replace(self,text):
        s = text
        for (pattern,repl) in self.patterns:
            s = re.sub(pattern, repl, s)
        return s

#import RegexReplacer
rp = RegexReplacer()

print(rp.replace("can't is a contradiction"))
print(rp.replace("I should've done that thing I didn't do"))

from nltk.tokenize import word_tokenize

print(word_tokenize("can't is a contradiction"))
print(word_tokenize(rp.replace("can't is a contradiction")))
```

```
can not is a contradiction
I should have done that thing I did not do
['ca', "n't", 'is', 'a', 'contradiction']
['can', 'not', 'is', 'a', 'contradiction']
```

Finite State Automata (FSA's)

An **alphabet** $\sum$ is a set of symbols: e.g. $\sum = \{a, b, c\}$

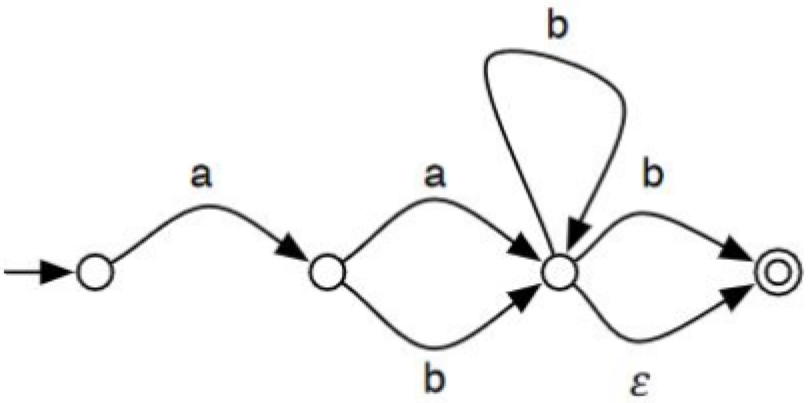
A **string** w is a sequence of symbols, e.g. $w = abcb$

The **Kleene closure** $\sum^*$ is the **infinite** set of all strings that can be generated from $\sum$

$$\sum^* = \{\epsilon, a, b, c, aa, ab, ba, aaa, bac, \cdots\}$$

A **language** $L \subseteq \sum^*$ over $\sum$ is also a set of strings (but finite)

Example:



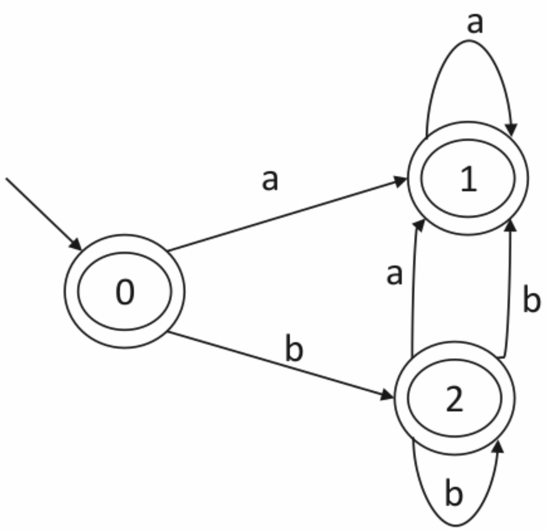
For each of the following strings, is it accepted or not accepted by this FSA?

1. ab
2. a
3. aabbb
4. aba
5. aa

- Write a regular expression that corresponds to the same regular languagerepresented by this FSA
- The FSA above is bigger than it needs to be. There exists an FSA with asmaller number of edges that represents the same regular language.

- Converting the regular expression (a*|b*)* to a FSA

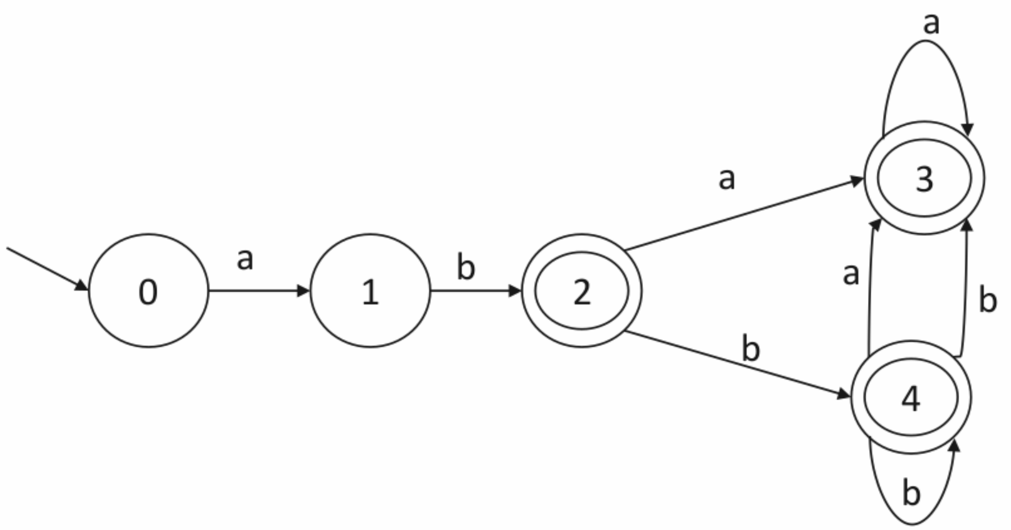
Possible Solution :



Example

- Converting the regular expression ab(a|b)* to a FSA

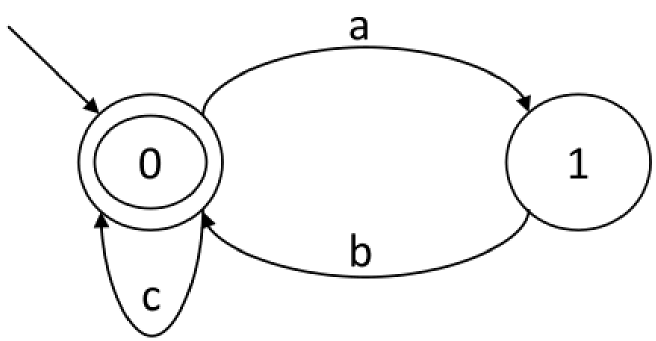
Possible Solution :

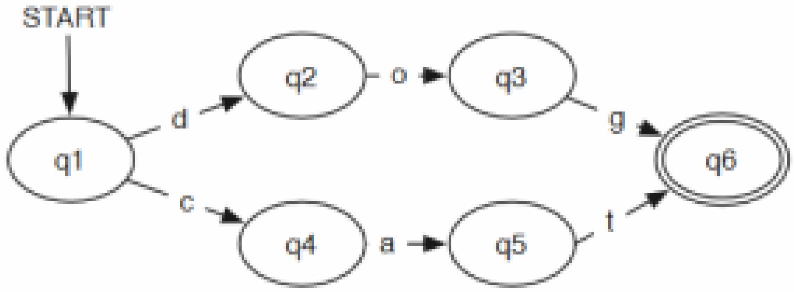


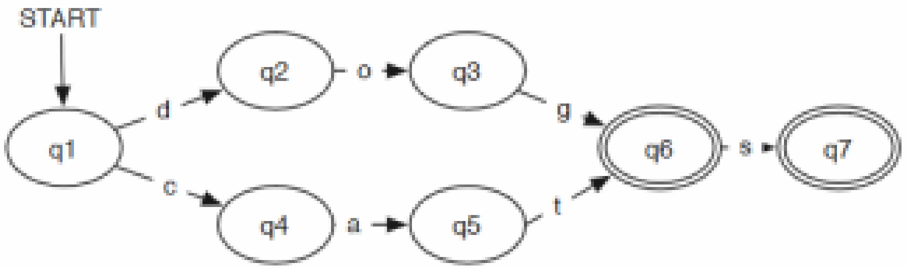
Example

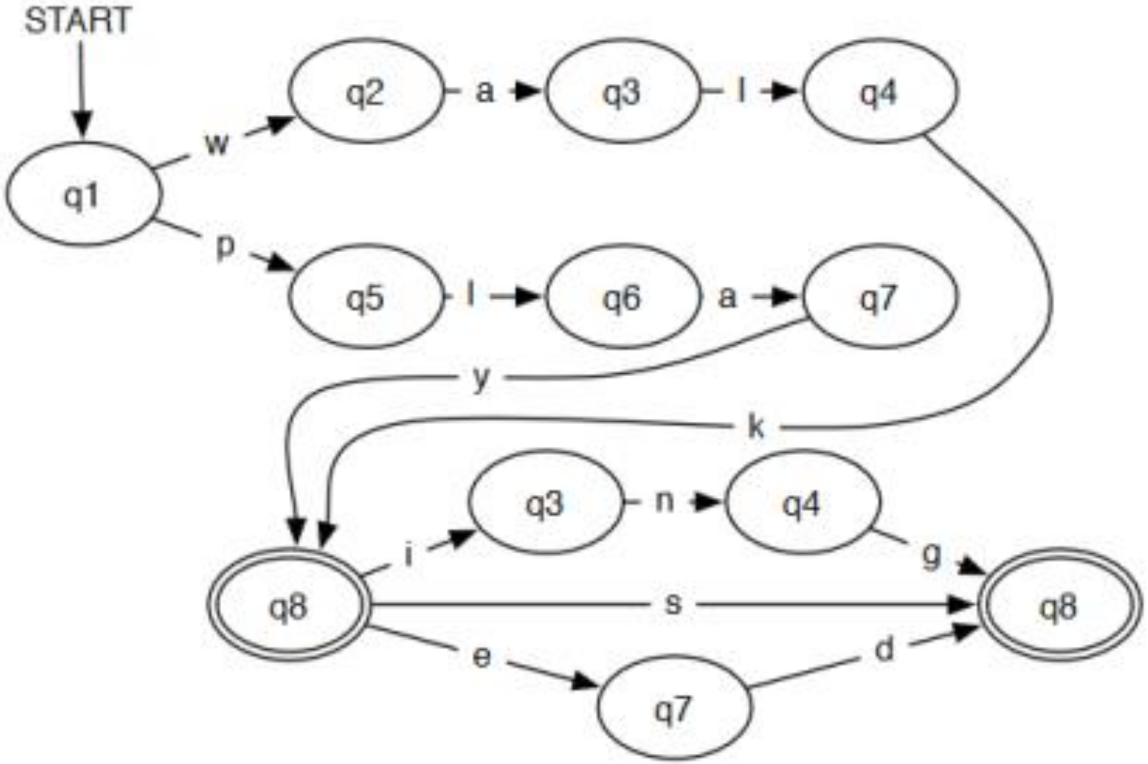
- Draw a FSA that accepts the regular expression ((ab)|c)* :

Possible Solution :

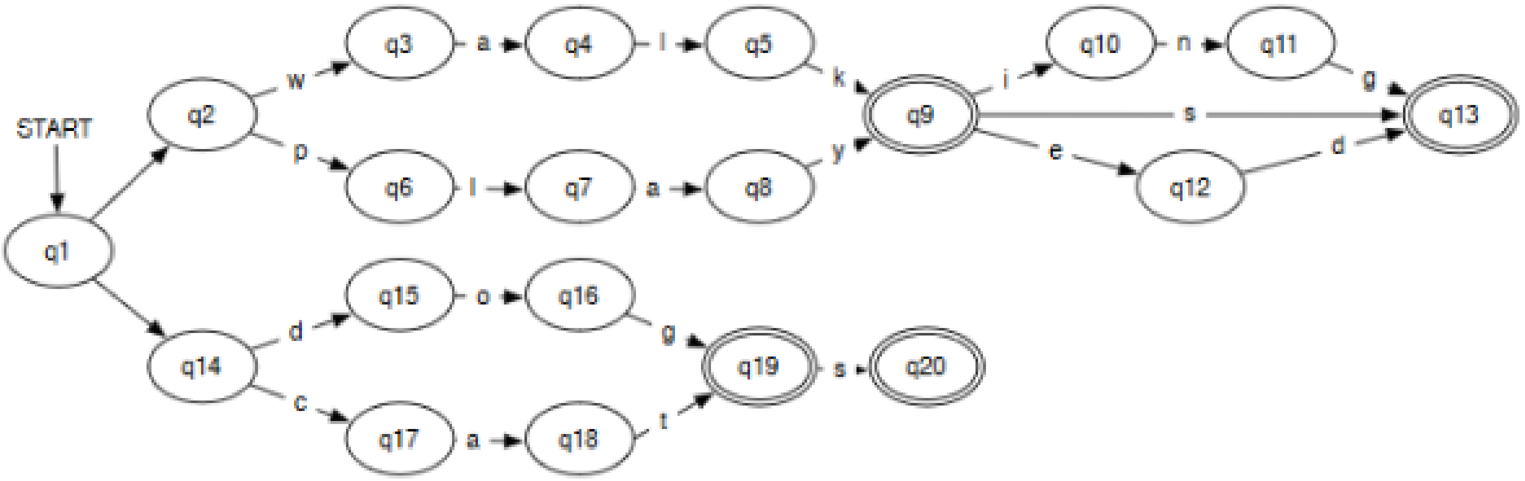



START
q1
d
c
q2
o
q3
g
q4
a
q5
t
q6


START
q1
d
c
q2
o
q3
g
q4
a
q5
t
q6
s
q7



START
q1
w
q2
a
q3
l
q4
p
q5
l
q6
a
q7
y
k
q8
i
q3
n
q4
g
s
e
q7
d
q8


START
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
q14
q15
q16
q17
q18
q19
q20
w
a
l
k
p
l
a
y
i
n
g
s
e
d
d
o
g
c
a
t
s

Morphology

Def. The study of how words are formed from minimal meaning-bearing units **(morphemes)**

We can usefully divide morphemes into two classes

```
Stems: The core meaning bearing units
Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical functions
```

English Morphology We can also divide morphology up into two broad classes:

```
Inflectional
Derivational
```

Inflectional Morphology The resulting word:

```
Has the same word class as the original
Serves a grammatical/semantic purpose different from the original
```

Nouns, Verbs and Adjectives (English) Nouns are simple (not really) : Markers for plural and possessive Verbs are only slightly more complex : Markers appropriate to the tense of the verb and to the person Adjectives : Markers for comparative and superlative

Regulars and Irregulars Some words misbehave (refuse to follow the rules)

```
Mouse/mice, goose/geese, ox/oxen
Go/went, fly/flew
```

Regulars...

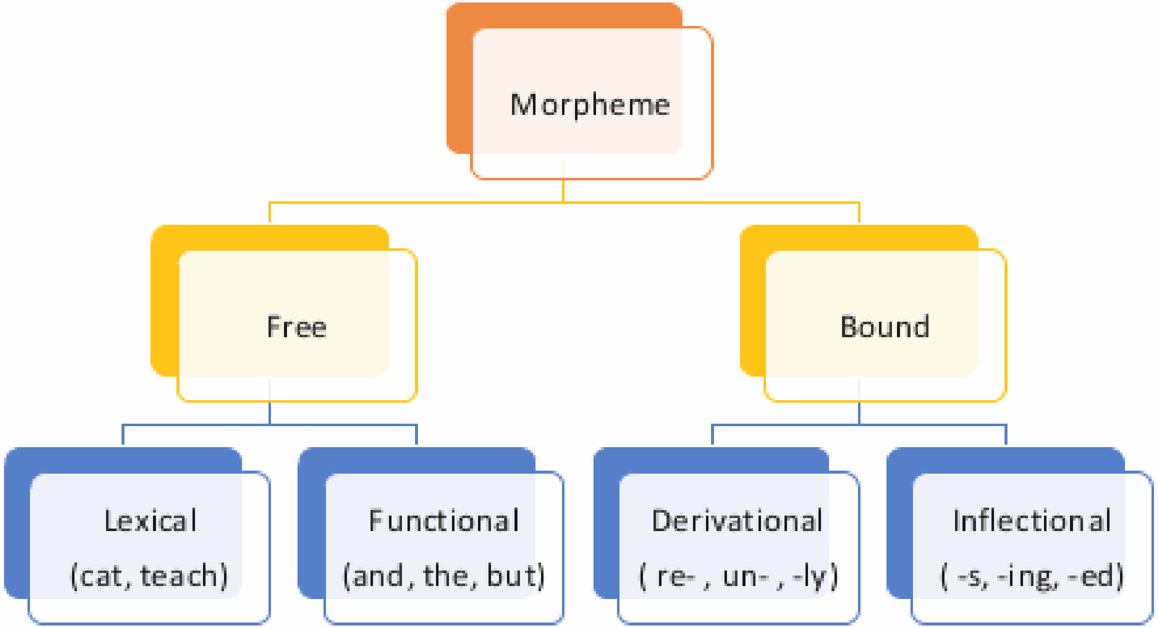
```
Walk, walks, walking, walked, walked
```

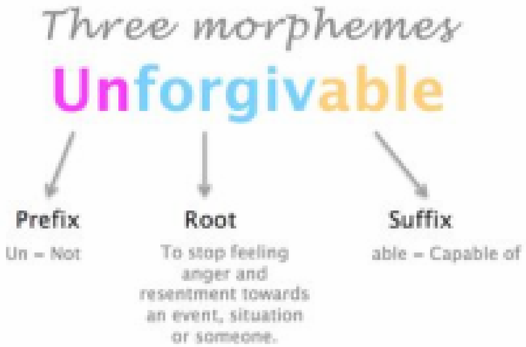
Irregulars

```
Eat, eats, eating, ate, eaten
Catch, catches, catching, caught, caught
Cut, cuts, cutting, cut, cut
```

Derivational Morphology Derivational morphology is the messy stuff that no one ever taught you.

```
Changes of word class
Less Productive ( -ant V -> N only with V of Latin origin!)
```


Morpheme
Free
Bound
Lexical
(cat, teach)
Functional
(and, the, but)
Derivational
(re- , un- , -ly)
Inflectional
(-s, -ing, -ed)


Three morphemes
Unforgivable
Prefix
Un = Not
Root
To stop feeling anger and resentment towards an event, situation or someone.
Suffix
able = Capable of

Example: Verb/Adj to Noun

-ation | computerize | computerization

-ee | appoint | appointee

-er | kill | killer

-ness |fuzzy | fuzziness

Example: Noun/Verb to Adj

-al | Computtaion | Computational

-able | Embrace | Embraceable

-less | Clue | Clueless

Example: Compute

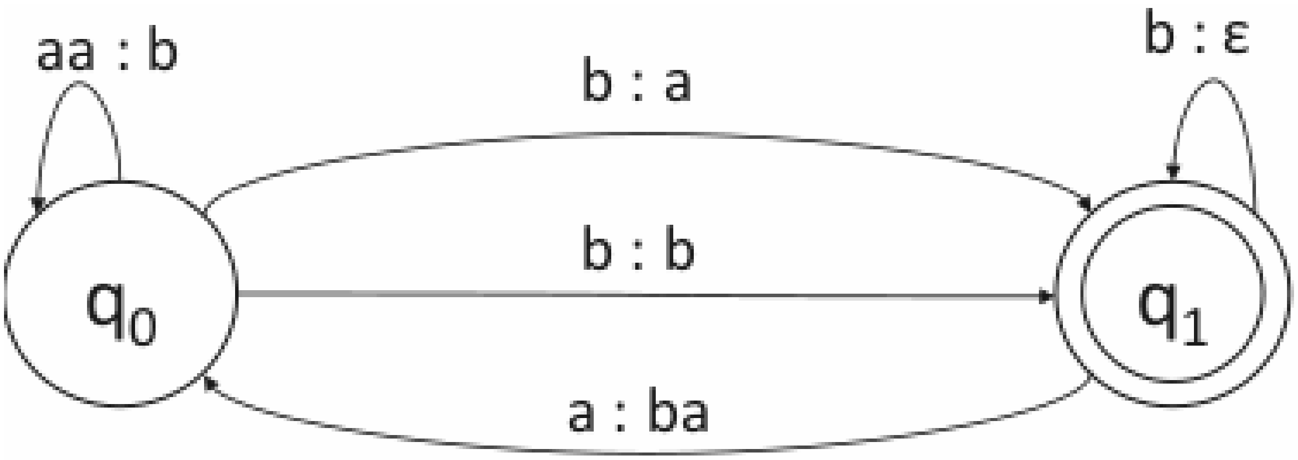
Many paths are possible…

```
Start with compute

    Computer -> computerize -> computerization
    Computation -> computational
    Computer -> computerize -> computerizable
    Compute -> computee
```

Finite-State Transducer

- A transducer maps between one representation and another.
- A finite-state transducer (FST) is a type of finite automaton which maps between two sets of symbols.



Definition

Q: a finite set of states

I,O: input and an output alphabets (which may include ε)

Σ: a finite alphabet of complex symbols i:o, $i \in I$ and $o \in O$

Q_0**:** the start state

F: a set of accept/final states ($F \subseteq Q$)

FST can be used as:

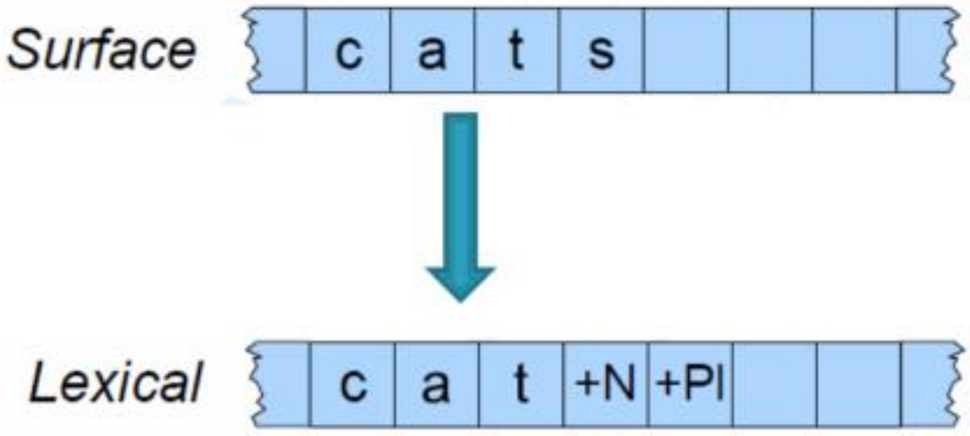
```
**Translators:** input one string from I, output another from O (or vice versa)
**Recognizers:** input a string from IxO
**Generator:** output a string from IxO
```

- Why do you want to recognize languages?
- Spell checkers
- Language identification
- Speech synthesis

FSTs and FSAs

- FSTs have a more general function than FSAs
  - An FSA defines a formal language by defining a set of strings
  - An FST defines a relation between sets of strings
- Another view: an FST is a machine that reads one string and generates another one.

We are interested in the transformation:

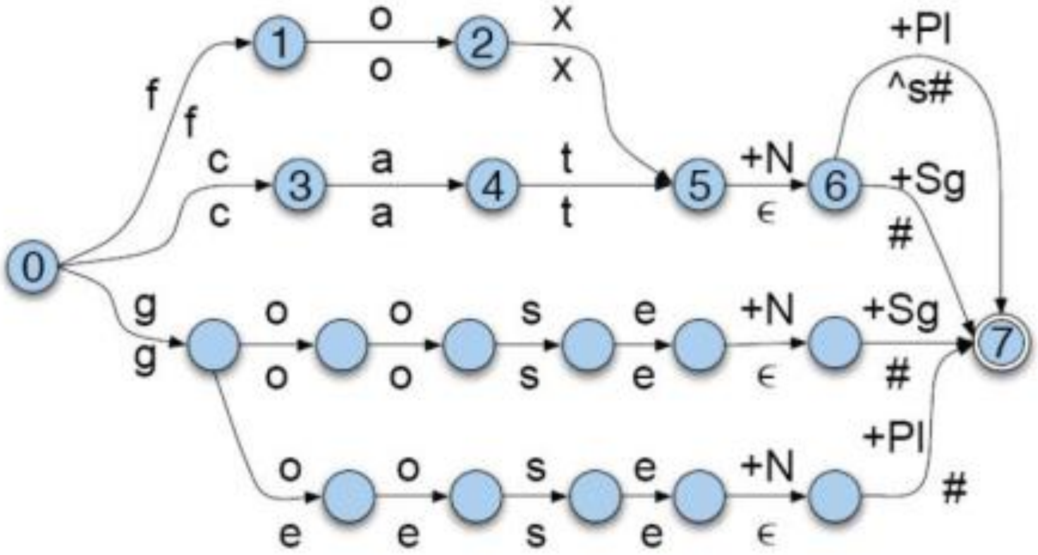


The s**urface level** represents the concatenation of letters which make up the actual spelling of the word

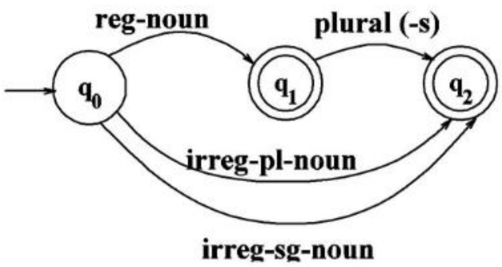
The **lexical level** represents a concatenation of morphemes making up a word

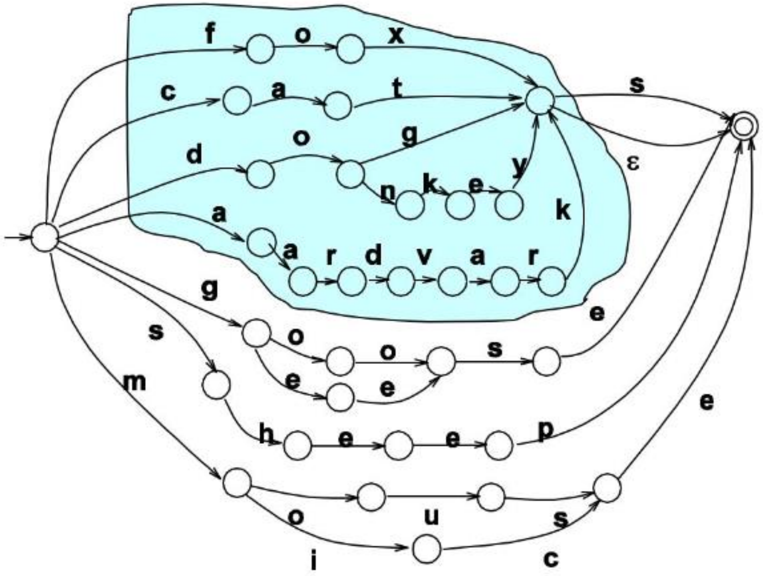
Example: Extracting the reg-noun, irreg-pl/sg-noun

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



Example: FSA for Portion of N Inflectional Morphology





```
Reg nouns ending in -s, -z, -sh, -ch, -x   -> es  (kiss, waltz, bush, rich, box)
Reg nouns ending -y preceded by a consonant change the -y to -i
```

Example: Small Fragment of V and N Derivational Morphology

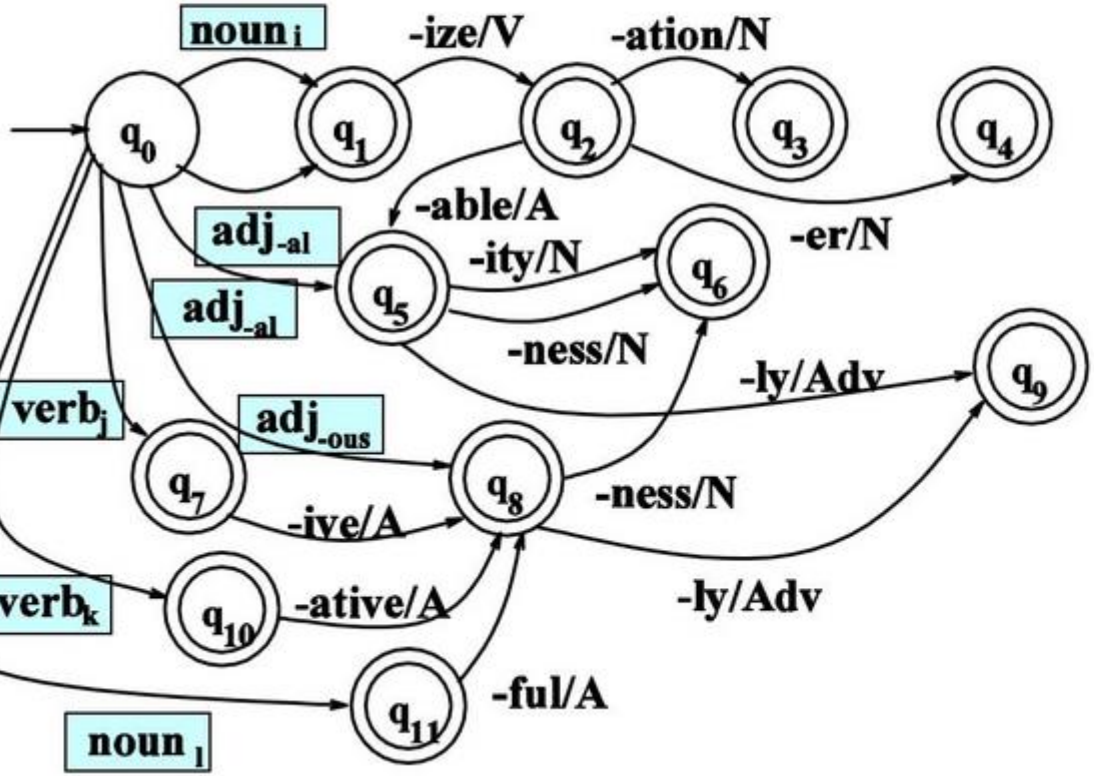
[noun$_i$] eg. hospital

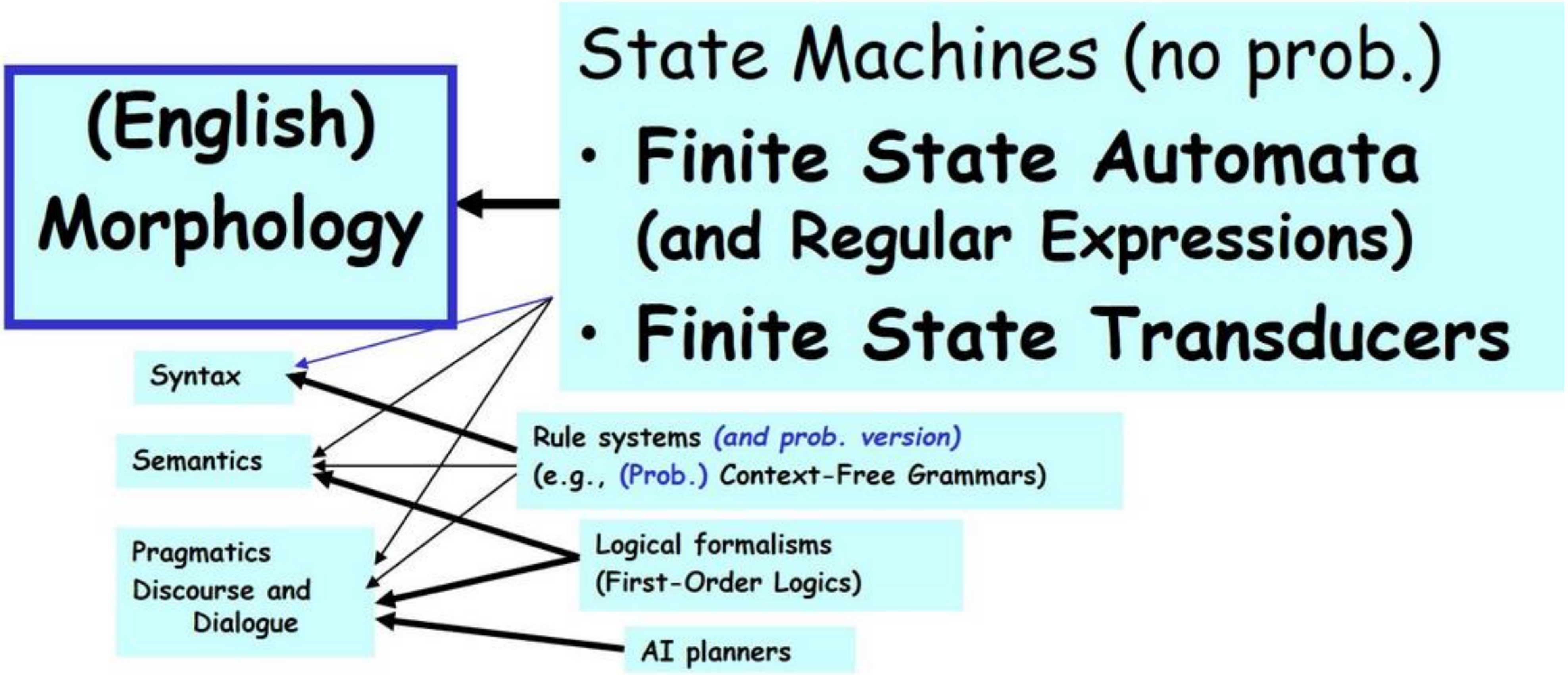
[adj$_{al}$] eg. formal

[adj$_{ous}$] eg. arduous

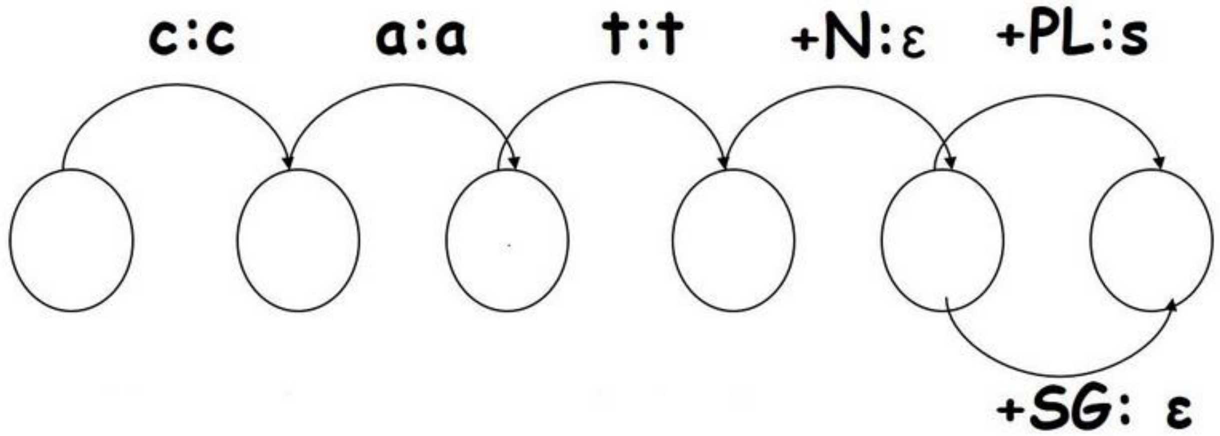
[verb$_j$] eg. speculate

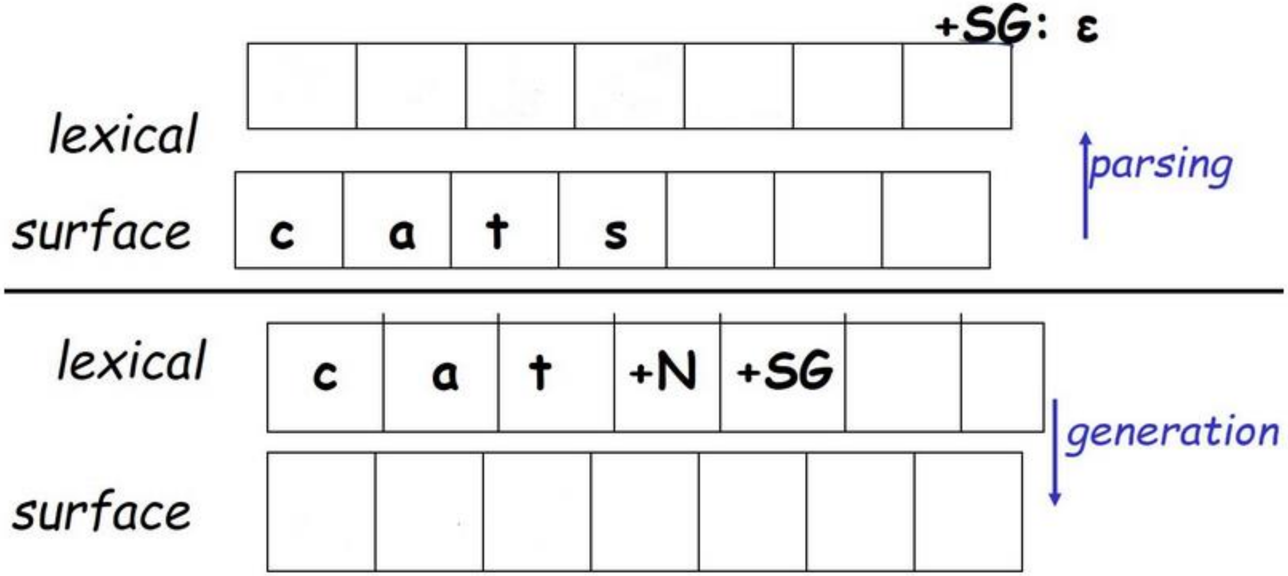
[verb$_k$] eg. conserve



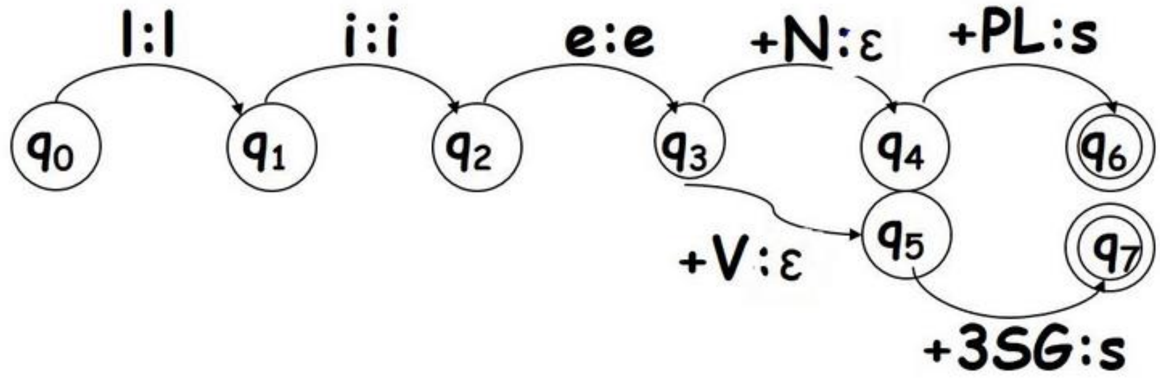

(English) Morphology
State Machines (no prob.)
• Finite State Automata (and Regular Expressions)
• Finite State Transducers
Syntax
Semantics
Pragmatics
Discourse and Dialogue
Rule systems (and prob. version)
(e.g., (Prob.) Context-Free Grammars)
Logical formalisms
(First-Order Logics)
AI planners

Example:

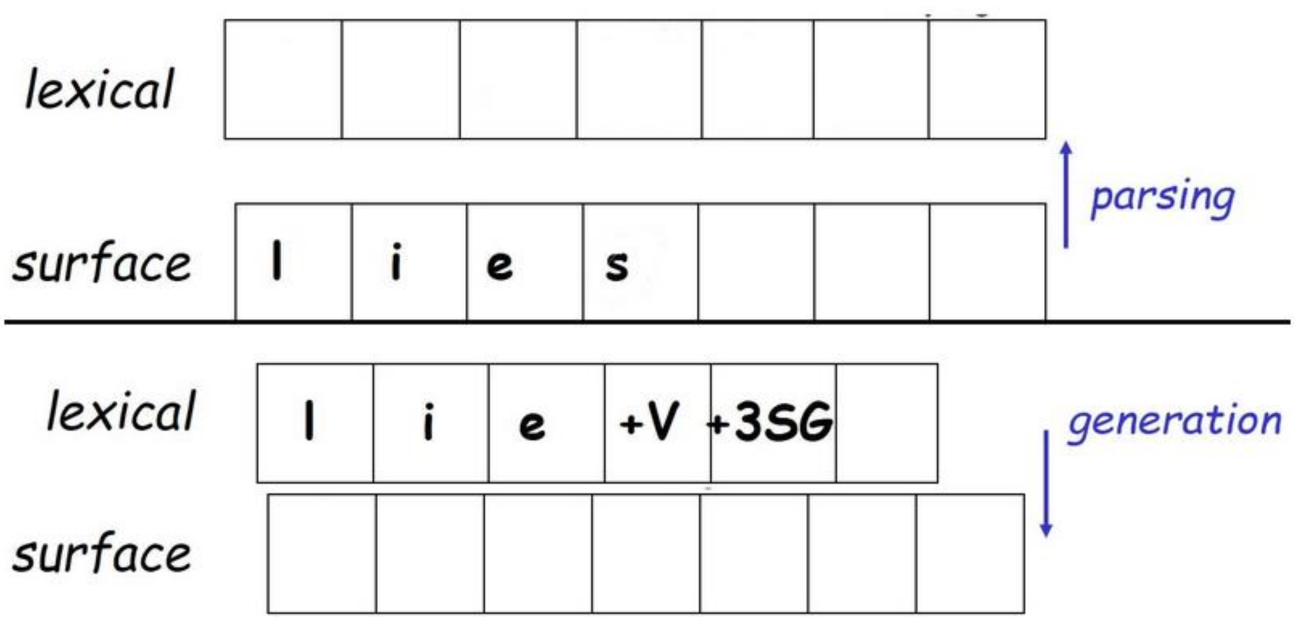

c:c
a:a
t:t
+N:ε
+PL:s
+SG: ε


+SG: ε
lexical
surface
c
a
t
s
parsing
lexical
c
a
t
+N
+SG
generation
surface

Example:



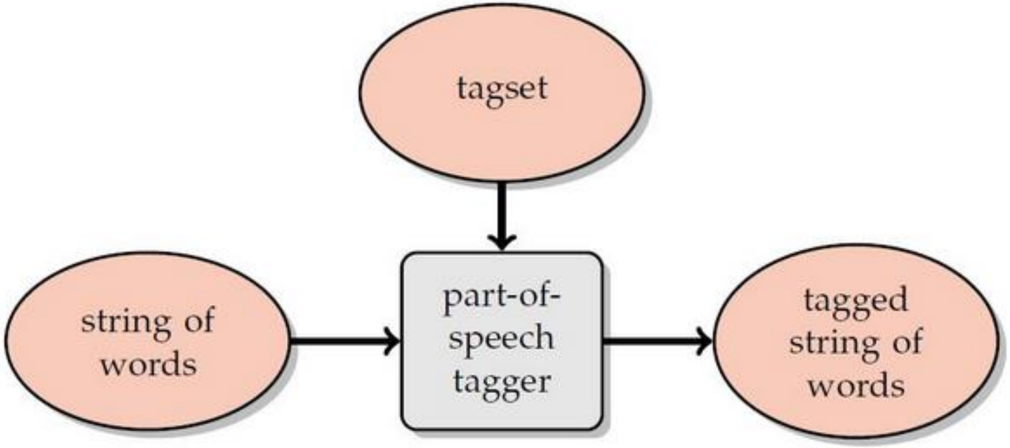
l:l
i:i
e:e
+N:ε
+PL:s
q0
q1
q2
q3
q4
q5
q6
q7
+V:ε
+3SG:s



lexical
surface
l
i
e
s
parsing
lexical
l
i
e
+V
+3SG
generation
surface

Part of Speech Tagging

Part-of-speech tagging is the process of labeling words with theappropriate part-of-speech.



```python
import nltk

tokens = nltk.word_tokenize("Can you please buy me an Arizona Ice Tea? It's $0.99.")

print("Tokens: ", tokens)
print("Parts of Speech: ", nltk.pos_tag(tokens))
```

```
Tokens:  ['Can', 'you', 'please', 'buy', 'me', 'an', 'Arizona', 'Ice', 'Tea', '?', 'It', "'s", '$', '0.99', '.']
Parts of Speech:  [('Can', 'MD'), ('you', 'PRP'), ('please', 'VB'), ('buy', 'VB'), ('me', 'PRP'), ('an', 'DT'), ('Ari
zona', 'NNP'), ('Ice', 'NNP'), ('Tea', 'NNP'), ('?', '.'), ('It', 'PRP'), ("'s", 'VBZ'), ('$', '$'), ('0.99', 'CD'),
('.', '.')]
```

- Rule-based part-of-speech tagging Two stage solution:

```
1. Morphological analysis and dictionary look-up to enumerate all possible POS for each word
2. Apply hand-written rules to remove inconsistent tags
```

- Stochastic part-of-speech tagging View part-of-speech tagging as a sequence classification task:

  given a sequence of words w_1^n

  determine a corresponding sequence of classes $\hat{t}_1^n$

$$\hat{t}_1^n = argmax_{t_1^n} P(t_1^n | w_1^n)$$

$$\hat{t}_1^n \approx argmax_{t_1^n} \prod_i^n P(w_i | t_i) P(t_i | t_{i-1})$$

Tag transition probabilities

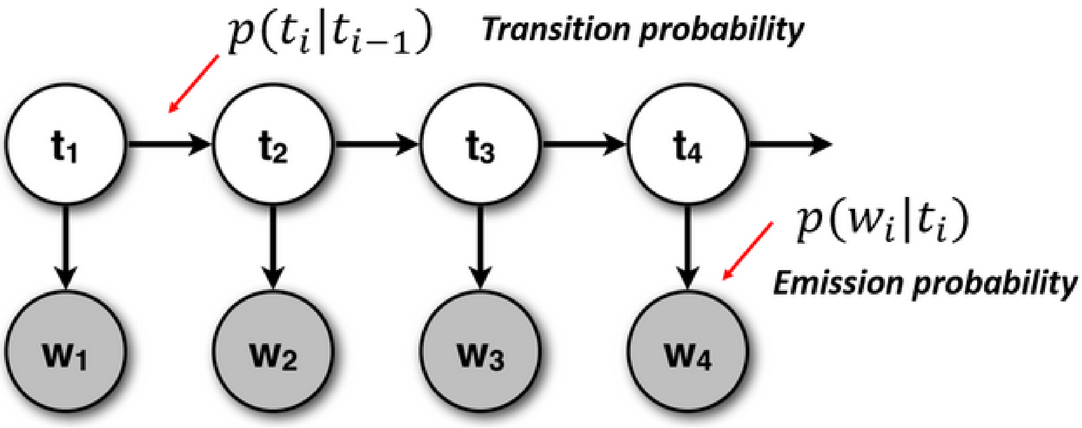
Based on a training corpus of previously tagged text, the MLE can be computed from the counts of observed tags:

$$P(t_i|t_{i-1}) = \frac{c(t_{i-1}, t_i)}{c(t_{i-1})}$$

Word likelihoods

Computed from relative frequencies in the same way:

$$P(w_j|t_i) = \frac{c(t_i, w_j)}{c(t_i)}$$



HMM

A hidden Markov model lets us handle both:

- observed events (like the words in a sentence) and
- I hidden events (like part-of-speech tags.

$Q = q_1 q_2 \cdots q_n$: a set of N states

$A = a_{11} a_{12} \cdots a_{n1} \cdots a_{nn}$: a transition probability matrix A, representing the probability of moving from state i to state j, such that $\sum_{j_1}^{n} a_{ij} = 1 \ \forall i$

$O = o_1 o_2 \cdots o_T$: a sequence of T observations, each one drawn from a vocabulary $V = v_1 v_2 \cdots v_T$

$B = b_i(o_t)$: A sequence of observation likelihoods, also called emission probabilities, each expressing the probability of an observation o_t being generated from a state i

q_0, q_F: a special start state and final state that are not associated with observations, together with transition probabilities $a_{01} a_{02} \cdots a_{0n}$ out of the start state and $a_{1F} a_{2F} \cdots a_{nF}$ into the final state.

Example 1 : What is the most likely sequence of tags t for the given sentence of words w?

- I want to race

- $\widehat{T} = \arg\max_{T \in t} P(T \mid W)$

Transition probabilities: $P(t_i \mid t_{i-1})$

	VB	TO	NN	PPSS
start	0.019	0.0043	0.041	0.067
VB	0.0038	0.0345	0.047	0.070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

Observation likelihoods: $P(w_i \mid t_i)$

	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

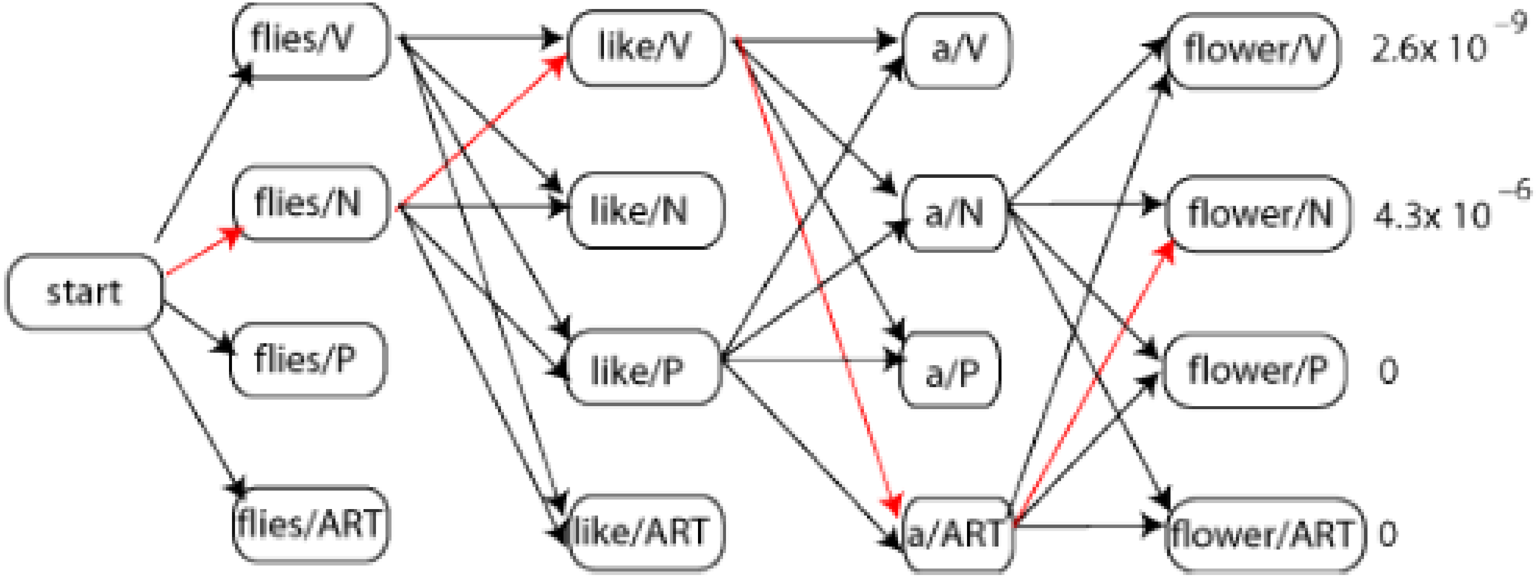
Example 2: What is the most likely sequence of tags t for the given sentence of words w?

Flies like a flower

Category	Count at i	Pair	Count at i,i+1	Bigram	Estimate
<start>	300	<start>,ART	213	Pr(Art|<start>)	.71
<start>	300	<start>,N	87	Pr(N|<start>)	.29
ART	558	ART,N	558	Pr(N|ART)	1
N	833	N,V	358	Pr(V|N)	.43
N	833	N,N	108	Pr(N|N)	.13
N	833	N,P	366	Pr(P|N)	.44
V	300	V,N	75	Pr(N|V)	.35
V	300	V,ART	194	Pr(ART|V)	.65
P	307	P,ART	226	Pr(ART|P)	.74
P	307	P,N	81	Pr(N|P)	.26

Pr(PP | start)	0.54		Pr(a | ART)	0.360
Pr(flies | N)	0.025		Pr(a | N)	0.001
Pr(flies | V)	0.076		Pr(flower | N)	0.063
Pr(like | V)	0.1		Pr(flower | V)	0.05
Pr(like | P)	0.068		Pr(birds | N)	0.076
Pr(like | N)	0.012			

The lexical generation probabilities



start
flies/V
flies/N
flies/P
flies/ART
like/V
like/N
like/P
like/ART
a/V
a/N
a/P
a/ART
flower/V
flower/N
flower/P
flower/ART
2.6x 10 −9
4.3x 10 −6
0
0