BBM 495
Regular Expressions
Finite State Automata and Morphology

LECTURER: BURCU CAN

2019-2020 SPRING

Today's contents

1. Regular expressions
2. Finite State Automata (FSA)
3. Finite State Transducer (FST)
4. Morphology

Regular expressions

n-gram models not good enough

- Want to model grammaticality
- A "training" sentence known to be grammatical:

- Resulting trigram model has to overgeneralize:
  - allows sentences with 0 verbs
    BOS **mouse traps** EOS
  - allows sentences with 2 or more verbs
    BOS **mouse traps catch mouse traps**
    catch mouse traps catch mouse traps EOS

Use of Regular Expressions in NLP

- Simple but powerful tools for large corpus analysis and 'shallow' processing
  - What word is most likely to begin a sentence?
  - What word is most likely to begin a question?
  - In your own email, are you more or less polite than the people you correspond with?

- With other unix tools, allow us to
  - Obtain word frequency and co-occurrence statistics
  - Build simple interactive applications (e.g., Eliza)

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Disjunctions

- Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges [A-Z]

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Negation in disjunction

- Negations [^Ss]
  - Carat means negation only when first in []

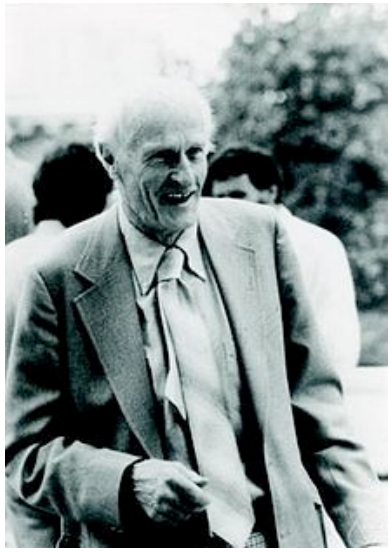
Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat b	Look up a^b now

More disjunctions

- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
groundhog|woodchuck	
yours|mine	yours mine
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	

Regular expressions: ? * + .



Stephen Kleene

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n		begin begun begun beg3n

Anchors: ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

  the

The recent attempt by the police to retain their current rates of pay has not gathered much favor with southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

  the

  [tT]he

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

  the

  [tT]he

*The recent attempt by the police to retain **the**ir current rates of pay has not ga**the**red much favor with the sou**the**rn factions.*

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

  the

  [tT]he

  \b[tT]he\b

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

  the

  [tT]he

  \b[tT]he\b

  (^|[^a-zA-Z])[tT]he[^a-zA-Z]

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A more complex example

- “any PC with more than *500 MHz* and *32 Gb* of disk space for less than *$1000*”

```
/$[0-9]+/

/$[0-9]+\.[0-9][0-9]/

/\b$[0-9]+(\.[0-9][0-9])?\b/

/\b[0-9]+ *(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/

/\b[0-9]+ *(Mb|[Mm]egabytes?)\b/

/\b[0-9]+(\.[0-9]+)? *(Gb|[Gg]igabytes?)\b/

/\b(Win95|Win98|WinNT|Windows *(NT|95|98|2000)?)\b/

/\b(Mac|Macintosh|Apple)\b/
```

Question-Answering: Eliza

- Men are all alike.
- IN WHAT WAY?
- They're always bugging us about something or other.
- CAN YOU THINK OF A SPECIFIC EXAMPLE?
- Well, my boyfriend made me come here.
- YOUR BOYFRIEND MADE YOU COME HERE
- He says I'm depressed much of the time.
- I AM SORRY TO HEAR YOU ARE DEPRESSED
- It's true. I am unhappy
- DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
- I need some help, that much seems certain.
- WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
- Perhaps I could learn to get along with my mother.
- TELL ME MORE ABOUT YOUR FAMILY
- My mother takes care of me.
- WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
- My father.
- YOUR FATHER
- You are like my father in some ways.

Eliza-style regular expressions

Step 1: replace first person with second person references

's/*regexp*/*replacement*/*flags*'

```
s/\bI('m| am)\b /YOU ARE/g
s/\bmy\b /YOUR/g
s/\bmine\b /YOURS/g
```

Step 2: use additional regular expressions to generate replies

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Step 3: use scores to rank possible transformations

Slide from Dorr/Monz

Common sets of characters

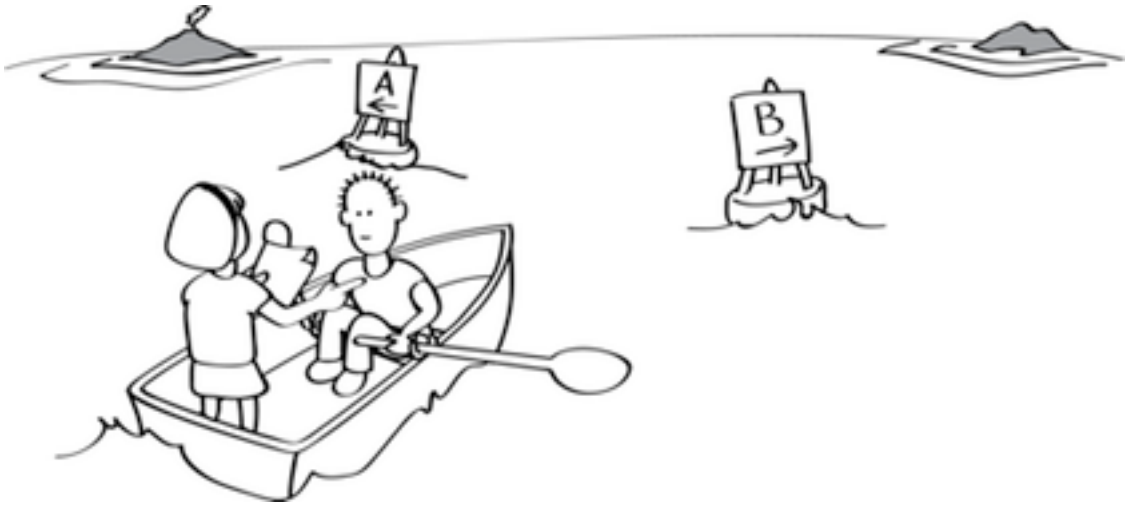
RE	Expansion	Match	Example Patterns
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9␣]	any alphanumeric or space	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Operators for counting

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression

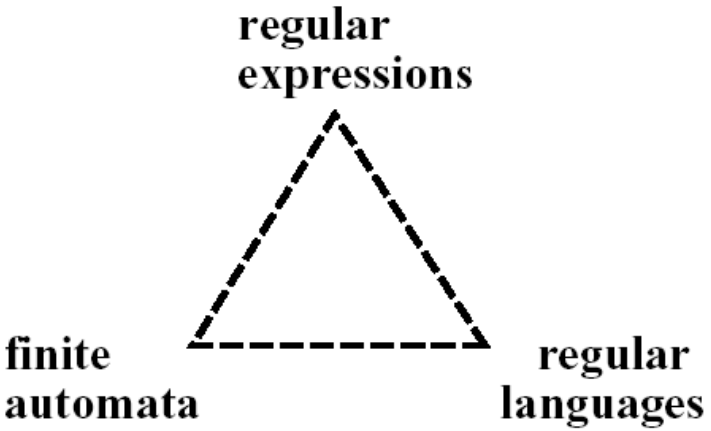
- In conclusion, regular expressions are very useful in capturing generalizations.

Finite state automata (FSA)

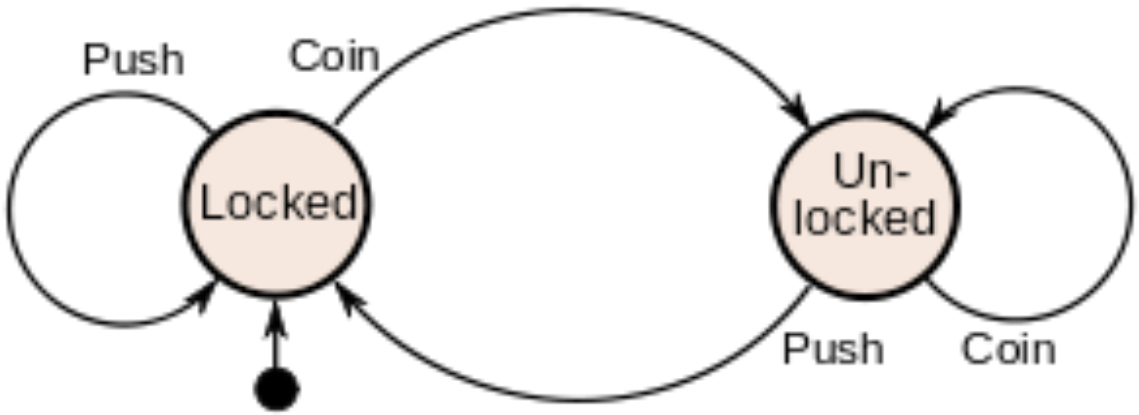


Finite state automata (FSA)

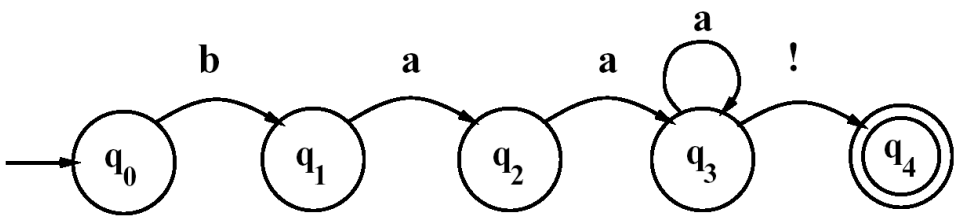
- A regular expression is one way of describing an FSA.
- A regular expression is one way of characterizing a particular kind of formal language called a regular language.

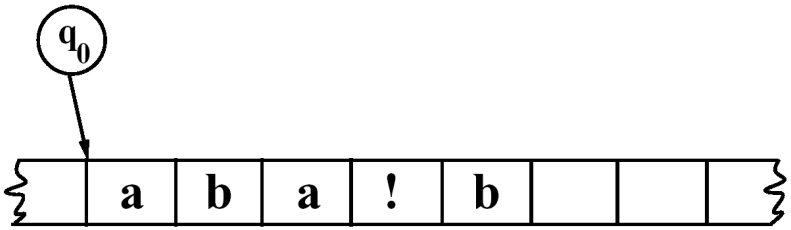


What is a FSA?



Using an FSA to recognize sheeptalk



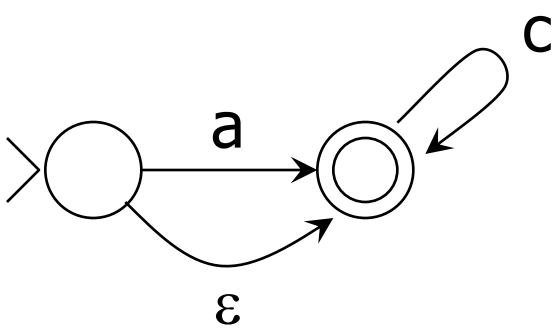


	Input		
State	b	a	!
0	1	0	0
1	0	2	0
2	0	3	0
3	0	3	4
4:	0	0	0

The transition-state table

- Automaton (finite automaton, finite-state automaton (FSA))
- State, start state, final state (accepting state)

Finite state acceptors (FSAs)



- Things you may know about FSAs:
  - Equivalence to regexps

Defines the language a? c*
= {a, ac, acc, accc, ε, c, cc, ccc, …}

Formal languages

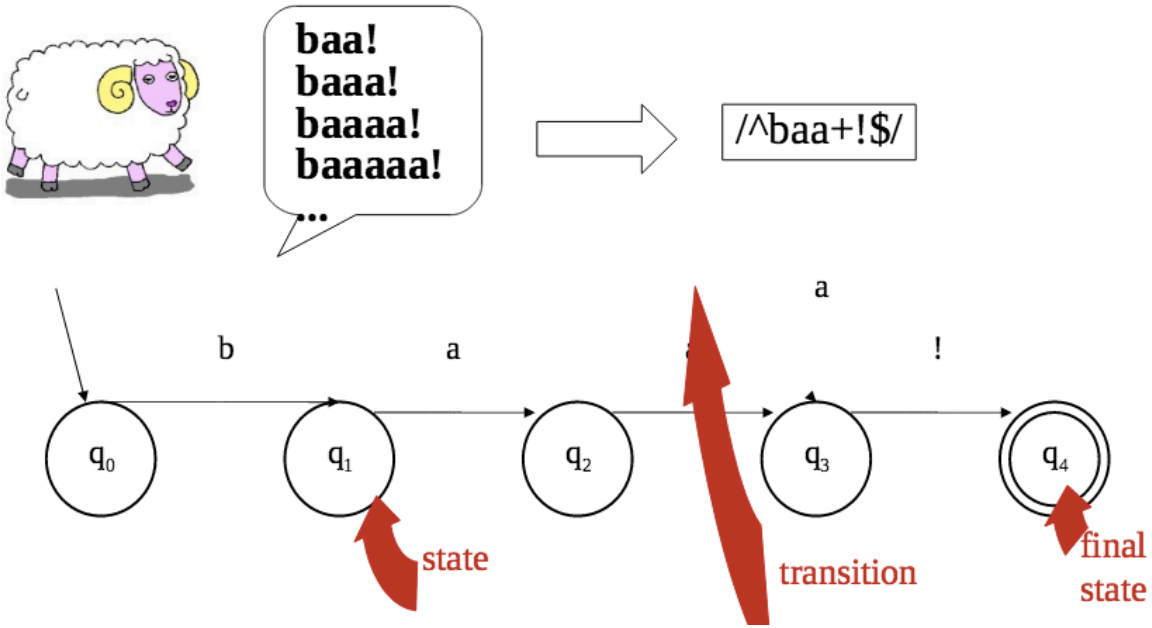
- An alphabet $\sum$ is a set of symbols:
  - e.g. $\sum = \{a, b, c\}$
- A string w is a sequence of symbols, e.g. $w = abcb$
- The Kleene closure $\sum^*$ is the infinite set of all strings that can be generated from $\sum$.
  $\sum^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, bac, ...\}$
- A language $L \subseteq \sum^*$ over $\sum$ is also a set of strings (but finite).

Automata and languages

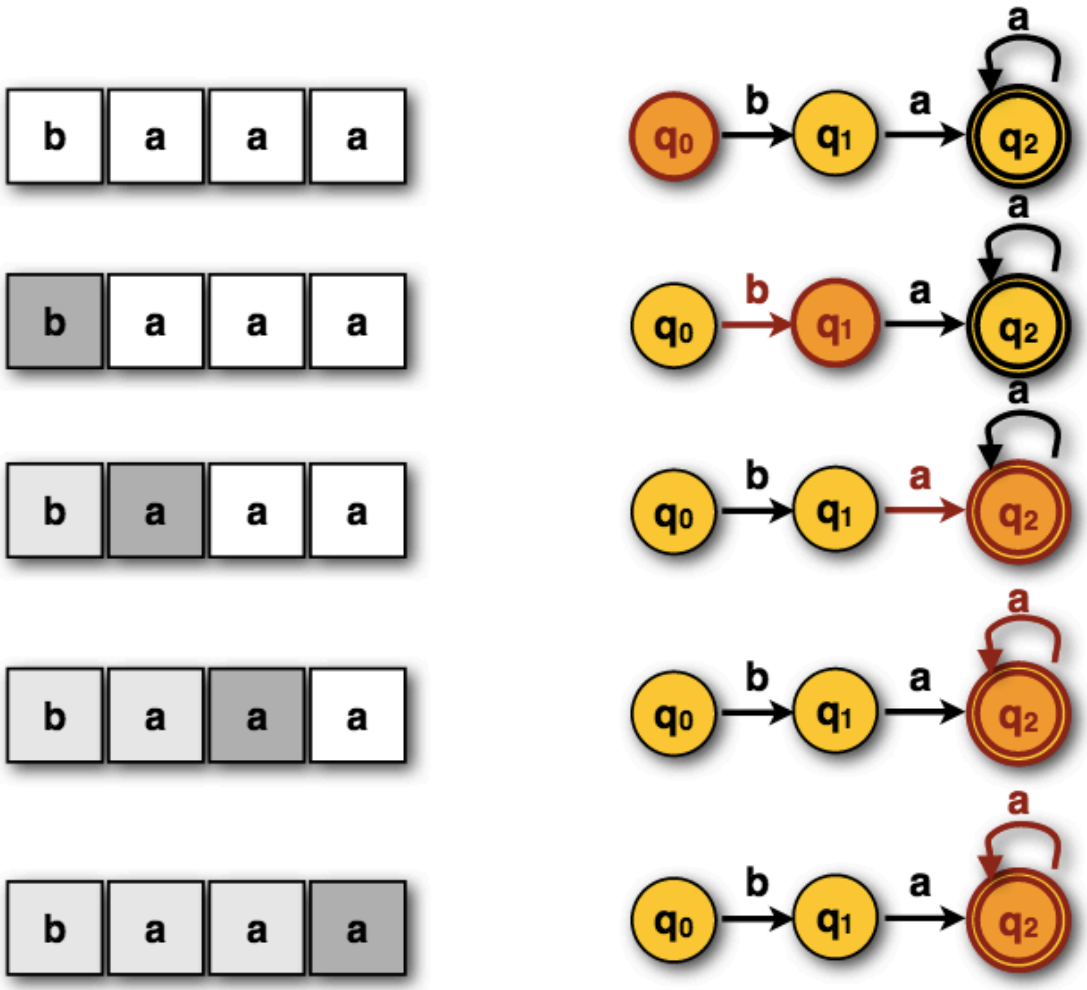
- An automaton is an abstract model of a computer which reads an input string, and changes its internal state depending on the current input symbol. It can either accept or reject the input string.
- Every automaton defines a language (the set of strings it accepts).
- Finite-state automata define regular languages.

Finite-state automata

- FSA is a 5-tuple consisting of
  - Q: set of states {q0,q1,q2,q3,q4}
  - Σ: an alphabet of symbols {a,b,!}
  - q0: a start state
  - F: a set of final states in Q {q4}
  - δ(q,i): a transition function mapping Q x Σ to Q

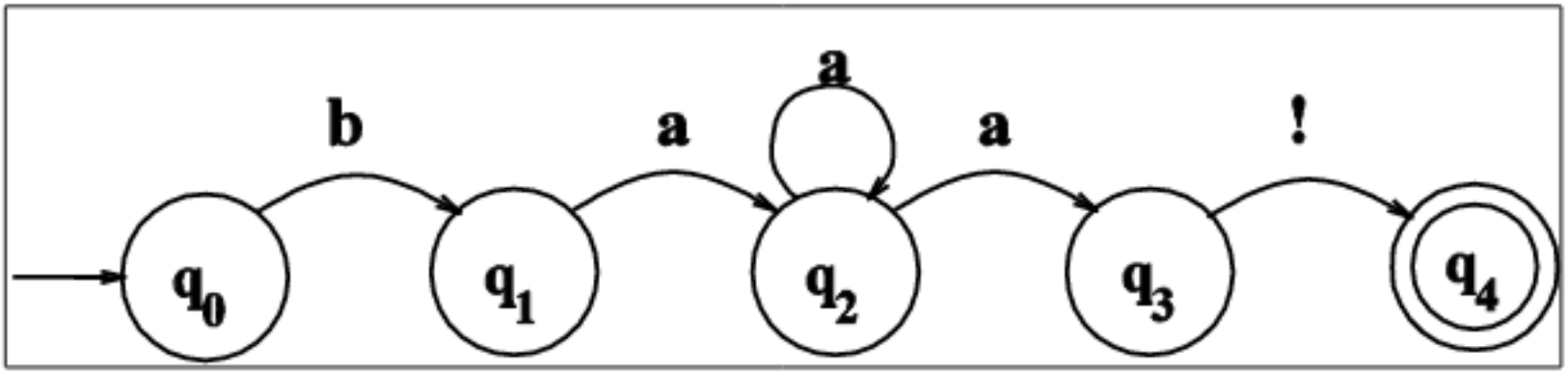


Finite-state automata (FSA's)



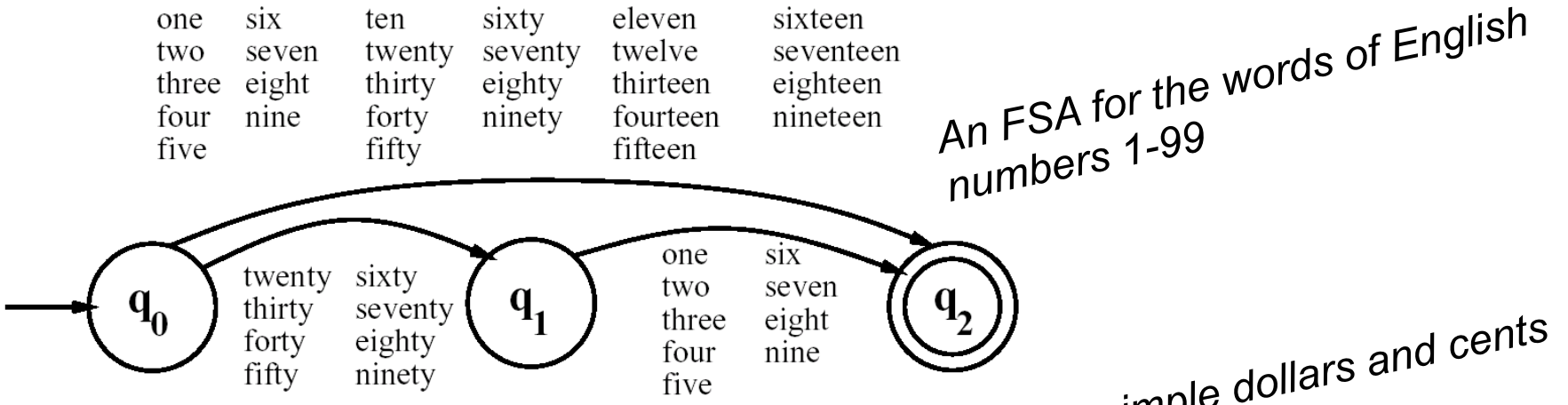
Sheep FSA

- We can say the following things about this machine:
  - It has 5 states
  - At least b,a, and ! are in its alphabet
  - q_0 is the start state
  - q_4 is an accept state
  - It has 5 transitions



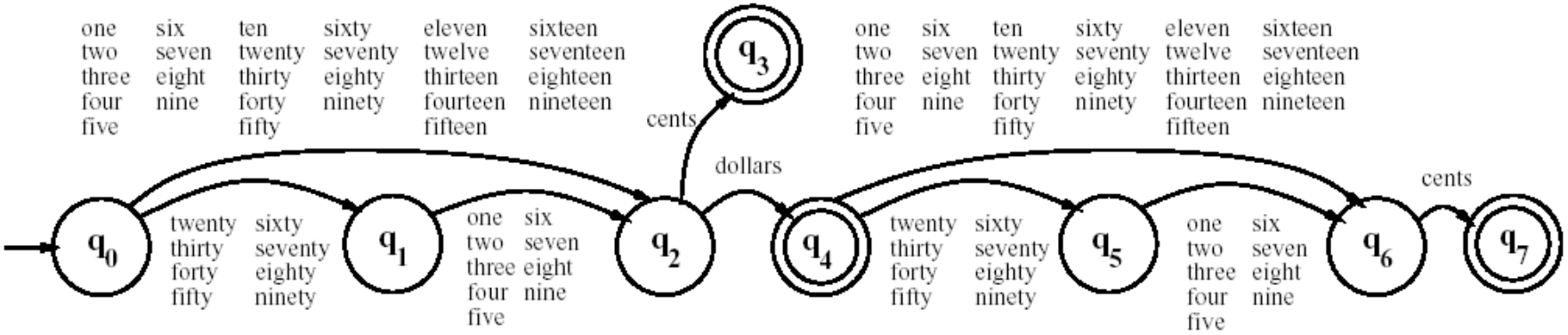
- Note: There are other machines that correspond to this language!

Another example



An FSA for the words of English numbers 1-99

FSA for the simple dollars and cents

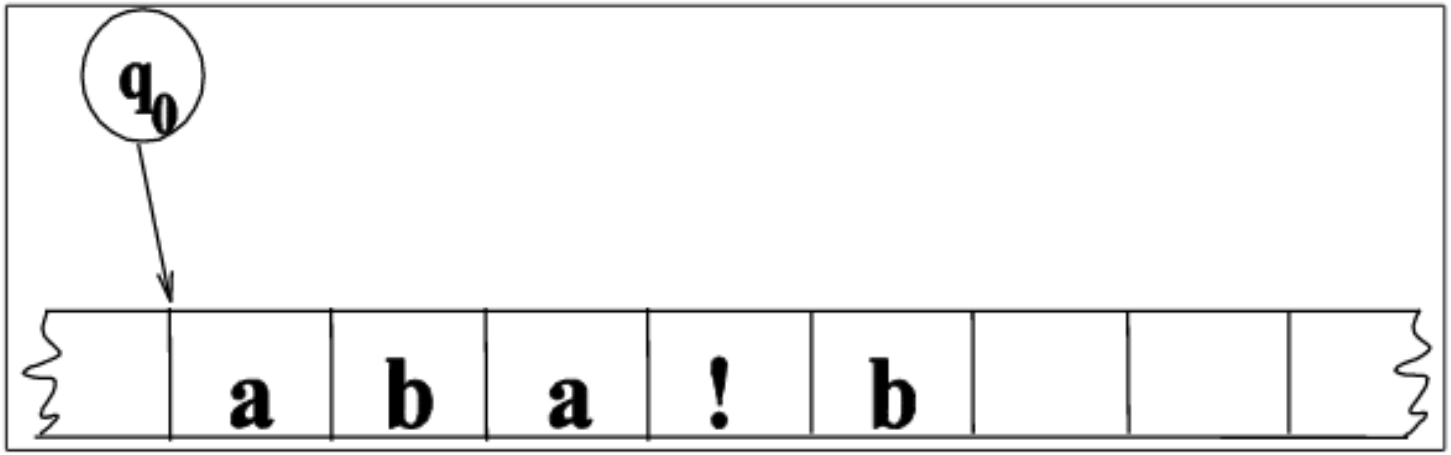


Recognition

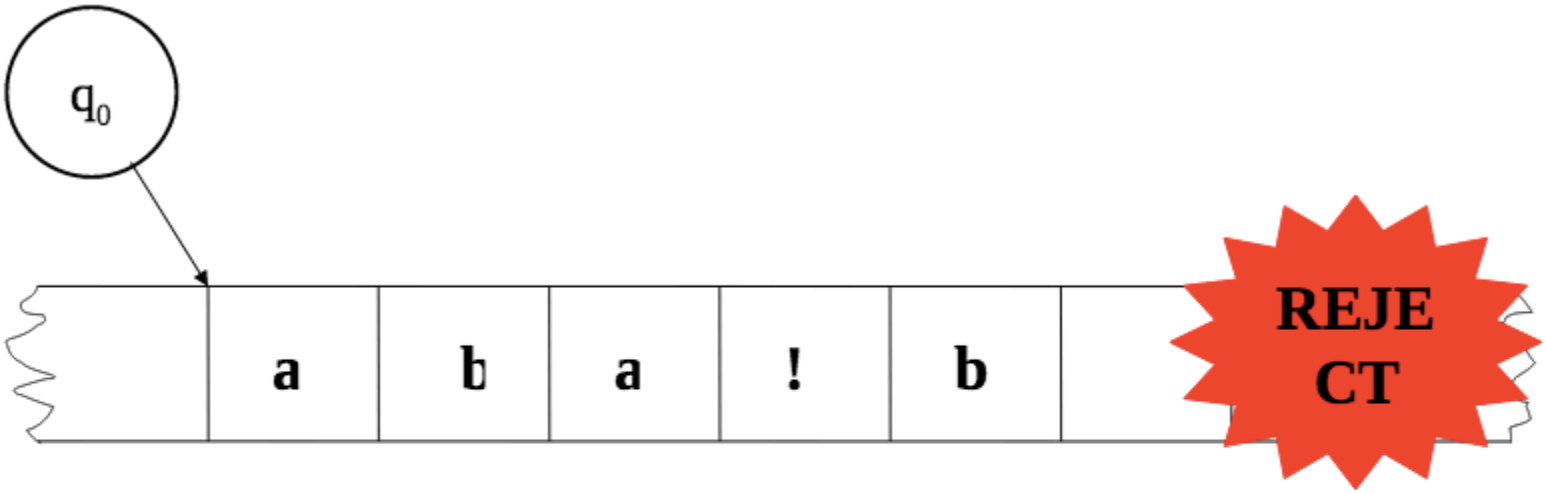
- Recognition is the process of determining if a string should be accepted by a machine.
- =
- Or … it's the process of determining if a string is in the language we're defining with the machine.
- =
- Or … it's the process of determining if a regular expression matches a string.
-

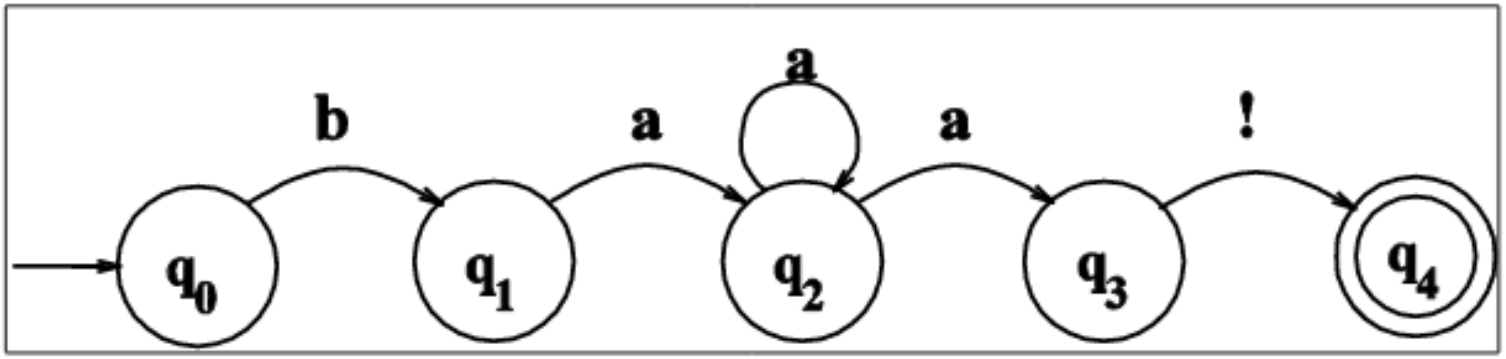
Recognition

- It is traditionally depicted by a tape (Turing's idea).

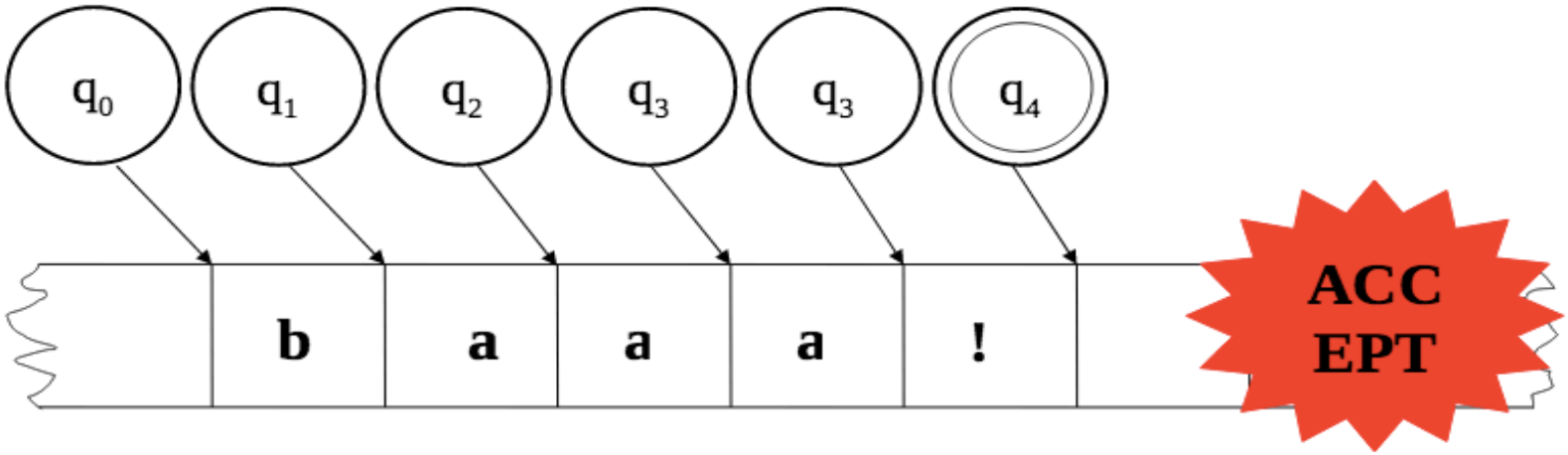


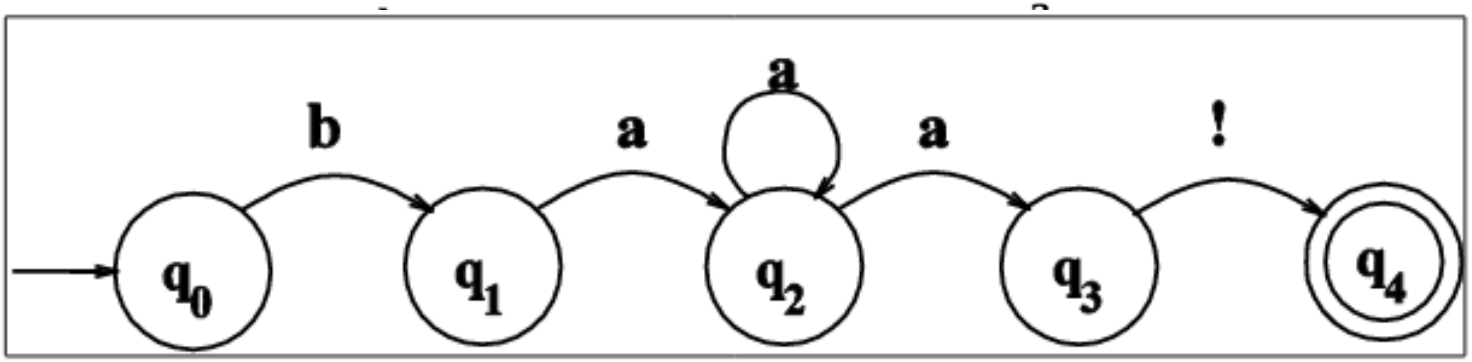
Recognition





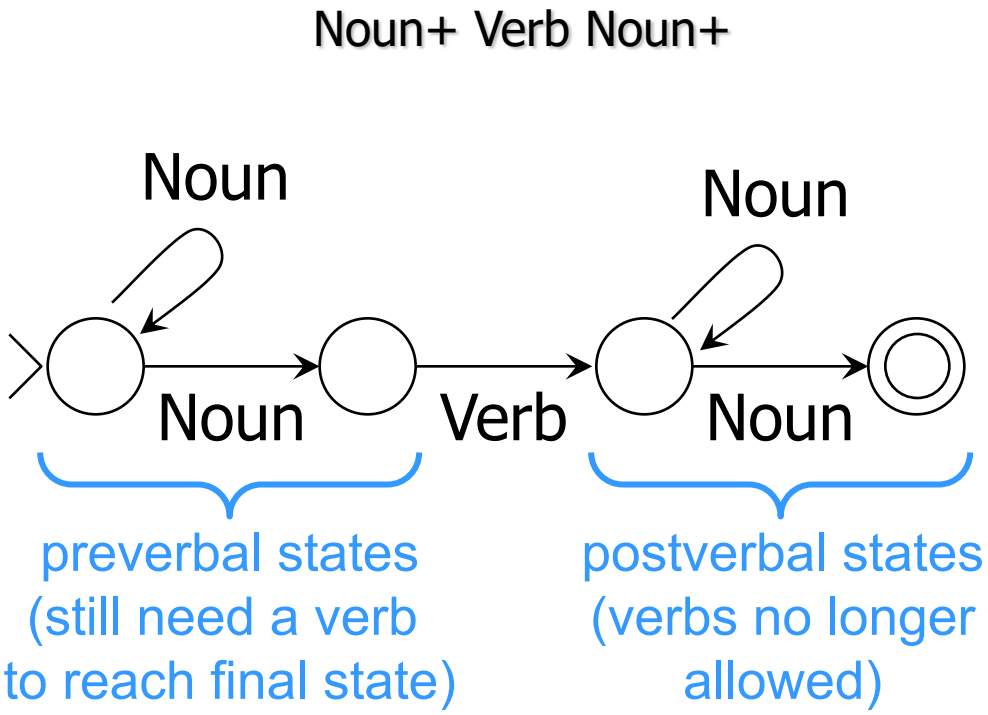
Recognition





Finite-state models can "get it"

- Want to model grammaticality
  BOS **mouse traps catch mouse traps** EOS
- Finite-state can capture the generalization here:



Allows arbitrarily long NPs (just keep looping around for another Noun modifier).

Still, never forgets whether it's preverbal or postverbal!

(Unlike 50-gram model)

FSTs and Morphology

What is morphology?

- Morphology is the study of the inner structure of words.
- Words are made up of smaller meaningful units called morphemes.

- We can divide the morphemes into two classes:
  - Stems: book
  - Affixes: ing

- Some words misbehave (i.e. irregular words)
  - Mouse/mice, go/went, see/saw

Why is Morphology Important to the Lexicon?

Full listing versus Minimal Redundancy

- true, truer, truest, truly, untrue, truth, truthful, truthfully, untruthfully, untruthfulness
- Untruthfulness = un- + true + -th + -ful + -ness
- These morphemes appear to be productive
- By representing knowledge about the internal structure of words and the rules of word formation, we can save room and search time.

Need to do Morphological Segmentation

Morphological Segmentation (or Stemming)

- Taking a surface input and breaking it down into its morphemes
- foxes breaks down into the morphemes fox (noun stem) and –es (plural suffix)
- rewrites breaks down into re- (prefix) and write (stem) and –s (suffix)

Why parse words?

- For spell-checking
  - Is muncheble a legal word?
- To identify a word's part-of-speech (pos)
  - For **sentence parsing**, for **machine translation**, …
- To identify a word's stem
  - For **information retrieval**

Two Broad Classes of Morphology

- Inflectional Morphology
  - Combination of stem and morpheme resulting in word of same class
  - Usually fills a syntactic feature such as agreement
  - E.g., plural –s, past tense -ed

- Derivational Morphology
  - Combination of stem and morpheme usually results in a word of a different class
  - Meaning of the new word may be hard to predict
  - E.g., +ation in words such as computerization

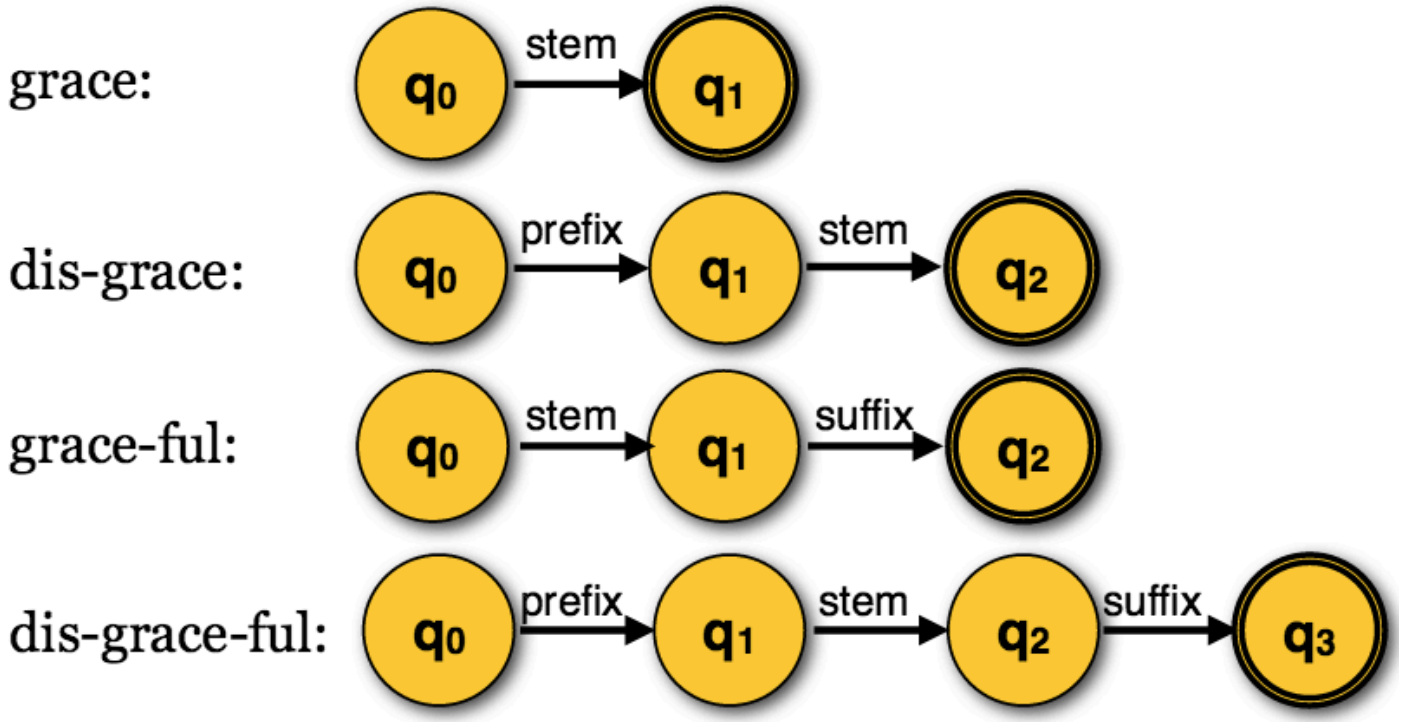
What do we need in order to build a morphological parser?

- Lexicon: stems and affixes (w/ corresponding pos)
- Morphotactics of the language: model of how morphemes can be affixed to a stem. E.g., plural morpheme follows noun in English
- Orthographic rules: spelling modifications that occur when affixation occurs
  - in → il in context of l (in- + legal)

Most morphological phenomena can be described with regular expressions – so finite state technologies are often used to represent these processes.

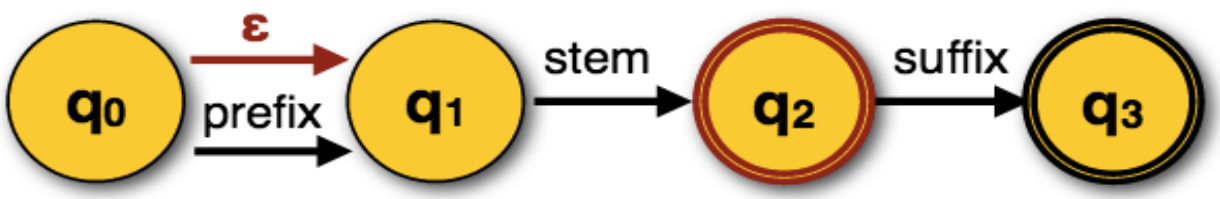
FSA and Morphology

- An important use of FSA's is for morphology, the study of word parts.

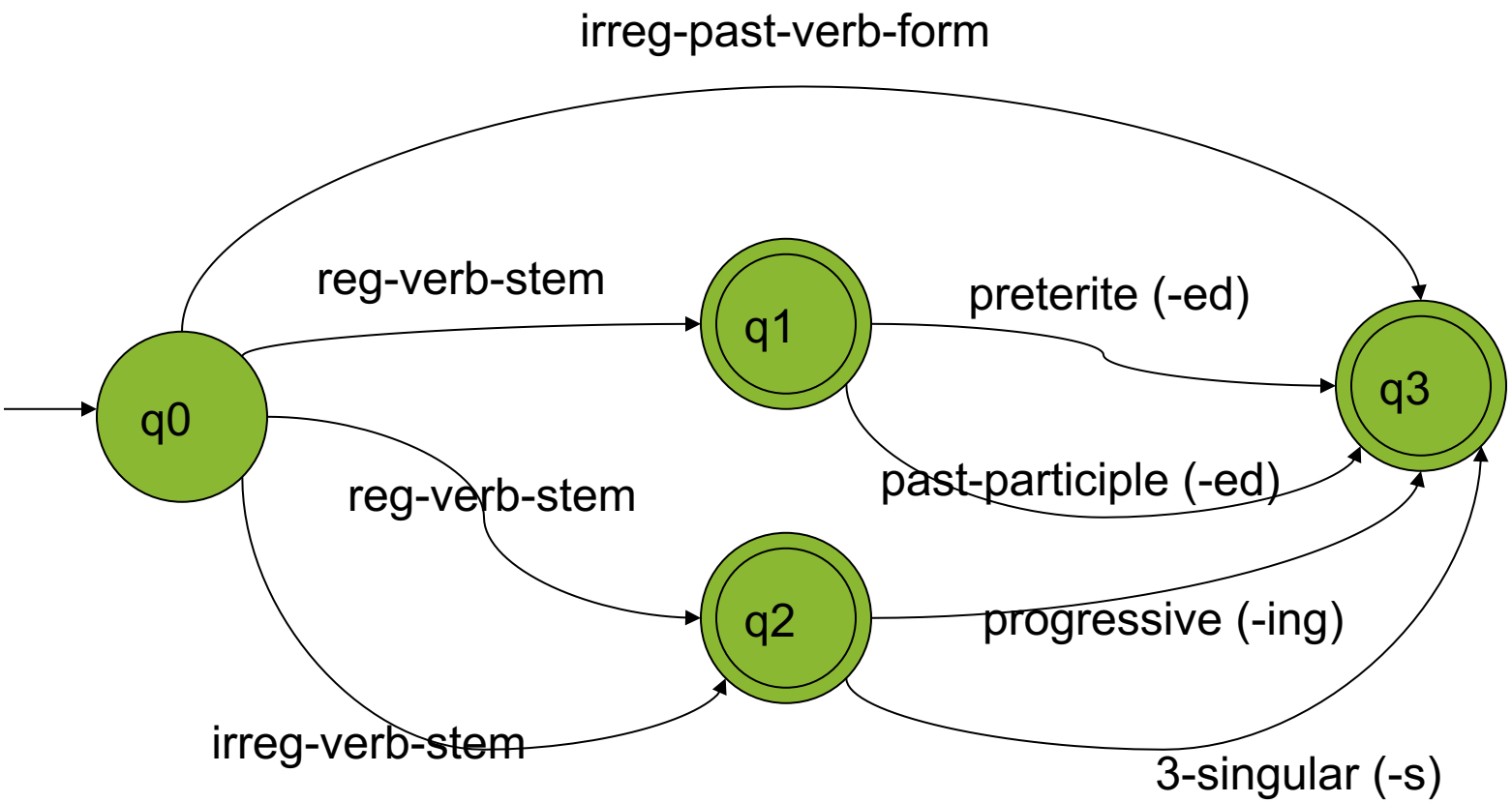


Union: Merging Automata

grace,
dis-grace,
grace-ful,
dis-grace-ful

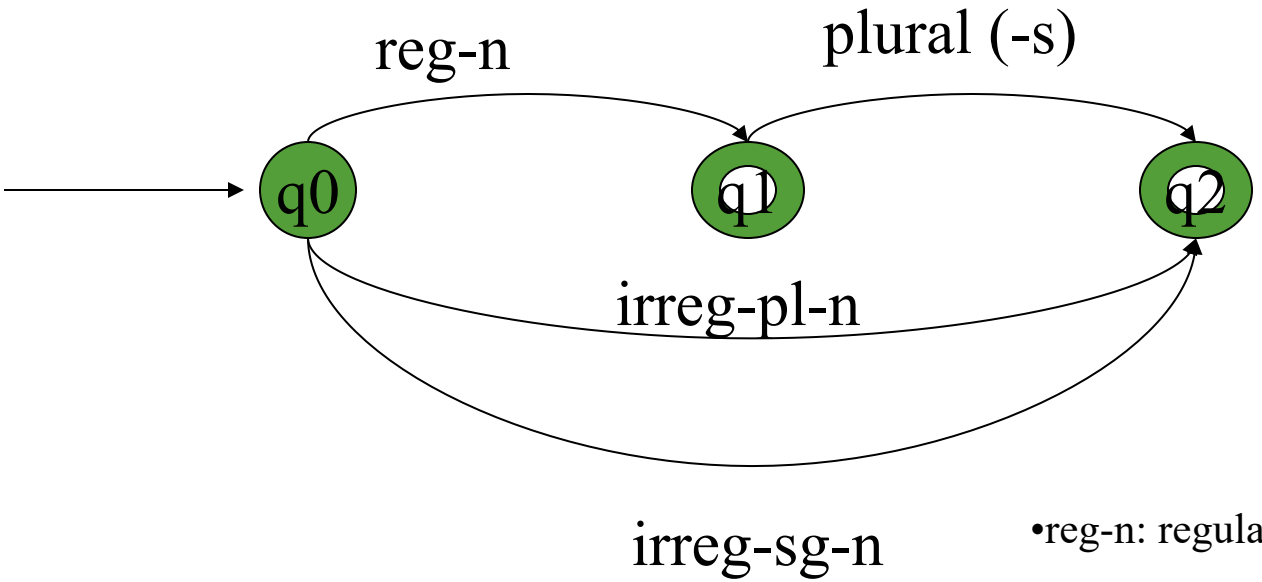


Finite-State Automaton for Inflection of English Verbs



Morphotactic Models

- English nominal inflection



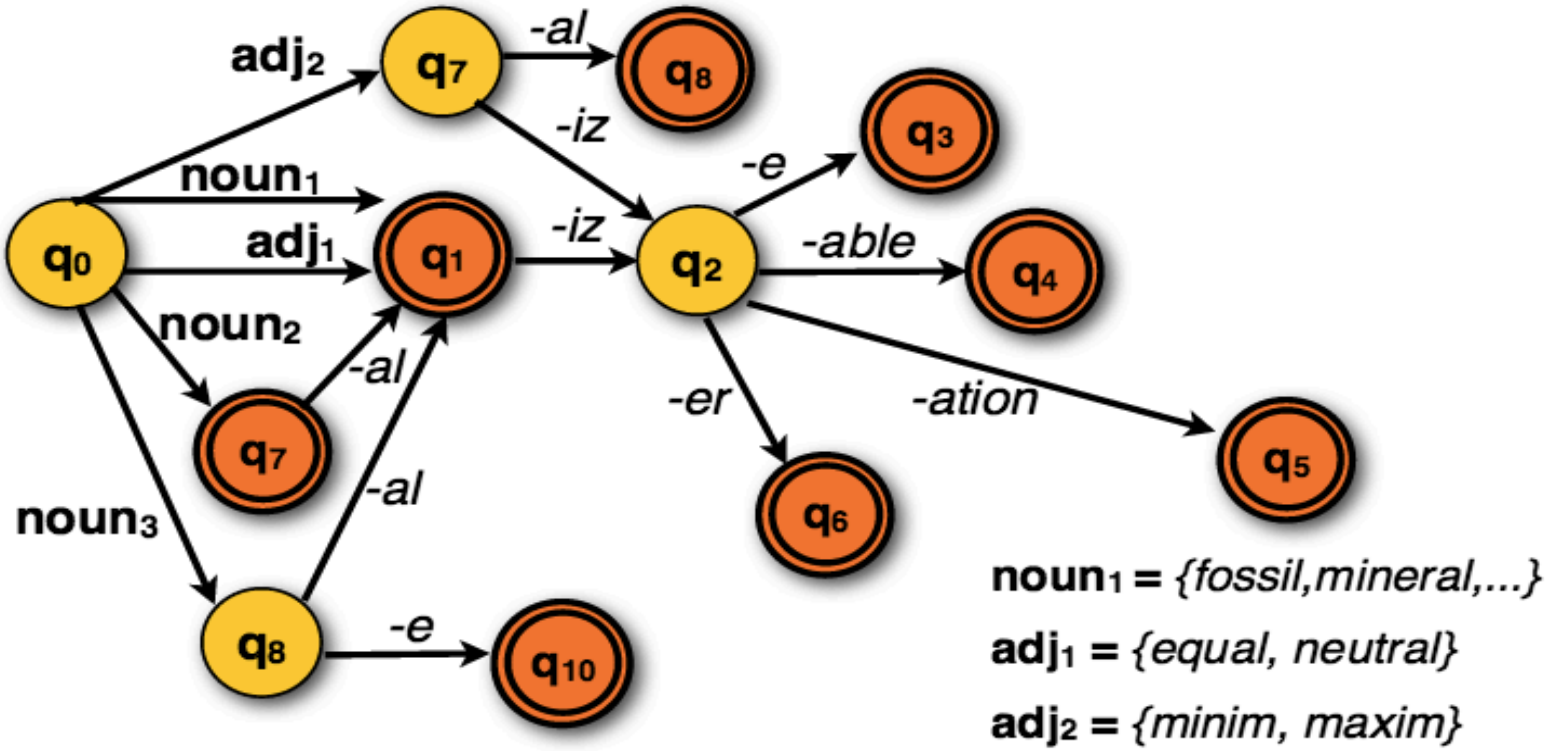
•reg-n: regular noun

•irreg-pl-n: irregular plural noun

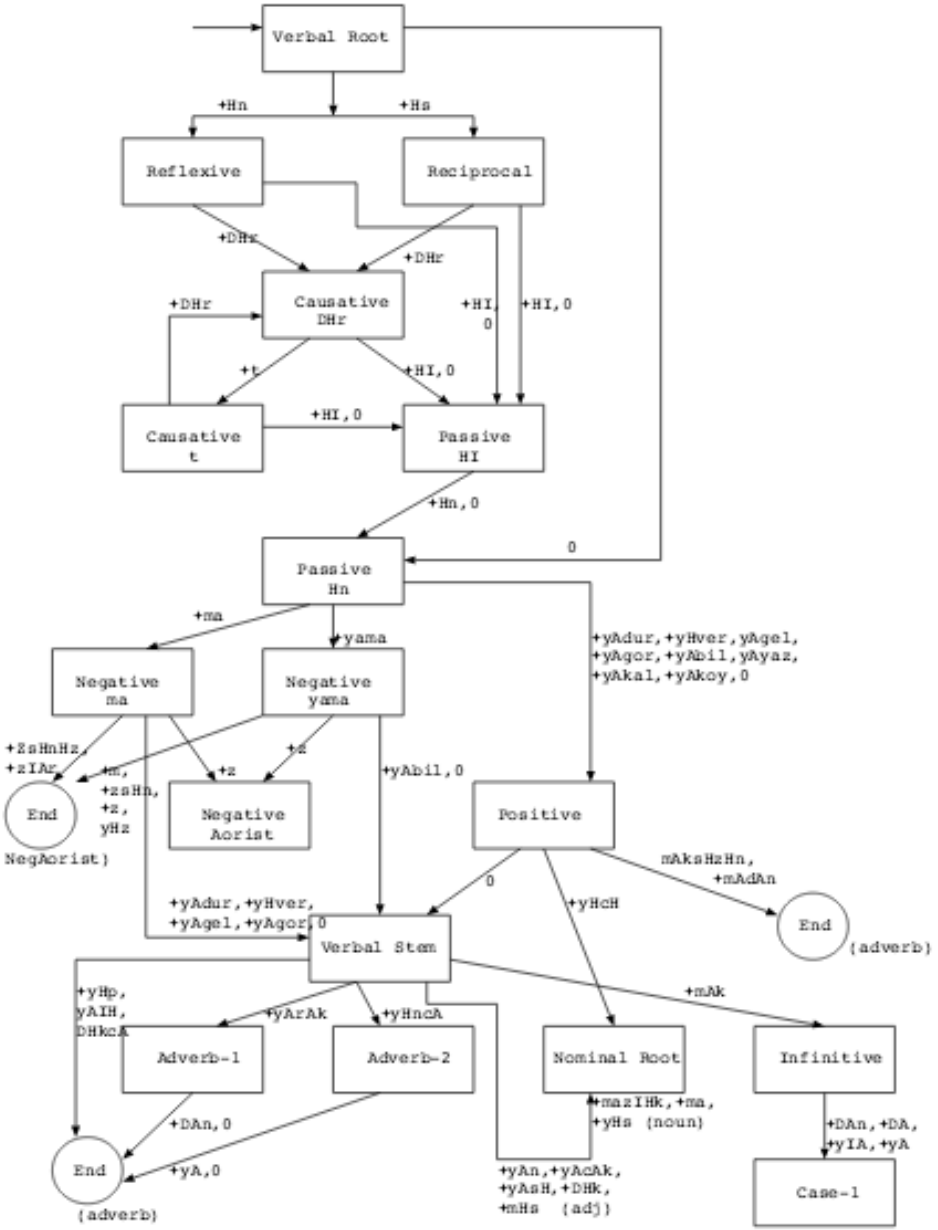
•irreg-sg-n: irregular singular noun

•Input words: cats, goose, geese

FSA for Derivational Morphology



FSA for Turkish verbs



Parsing/Generation vs. Recognition

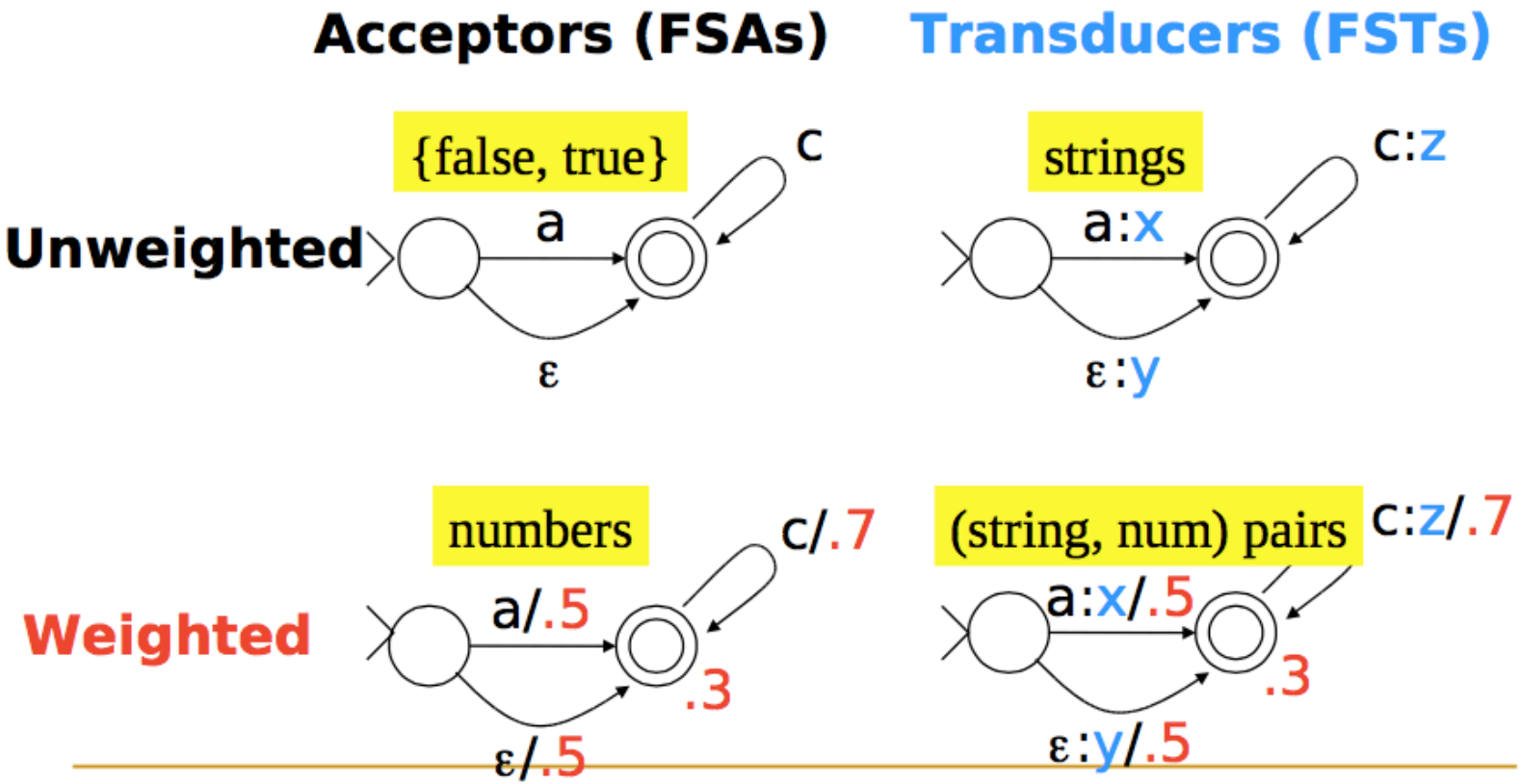
- Recognition is usually not quite what we need.
  - Usually if we find some string in the language we need to find the structure in it (parsing)
  - Or we have some structure and we want to produce a surface form (production/generation)
- Example
  - From "`cats`" to "`cat +N +PL`"

Finite State Transducers (FSTs)

- The simple story
  - Add another tape
  - Add extra symbols to the transitions

Example 1: One tape reads “cats”, the other tape writes “cat +N +PL” .
Example 2: One tape reads “kitaplar”, the other tape writes “kitap +N +PL” .

FSA vs FST


Acceptors (FSAs)
Transducers (FSTs)
Unweighted
{false, true}
c
a
ε
strings
c:z
a:x
ε:y
Weighted
numbers
c/.7
a/.5
.3
ε/.5
(string, num) pairs
c:z/.7
a:x/.5
.3
ε:y/.5

FSA vs FST

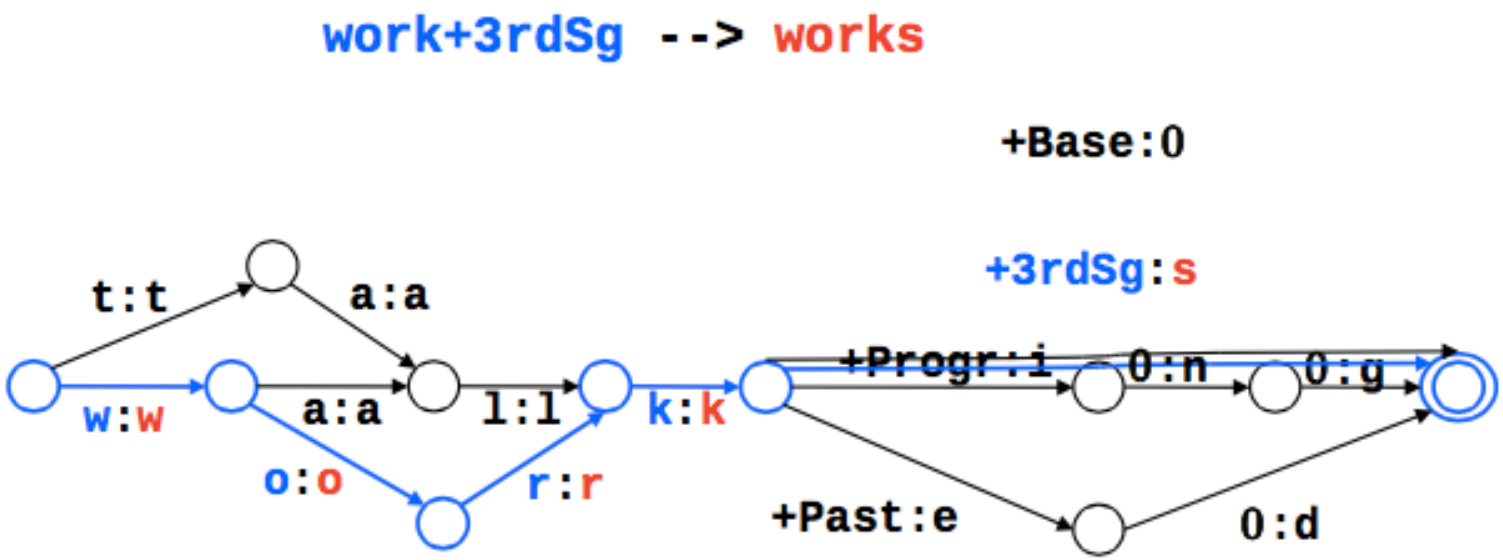
	Acceptors (FSAs)	Transducers (FSTs)
	{false, true}	strings
Unweighted	Grammatical?	Markup Correction Translation
	numbers	(string, num) pairs
Weighted	How grammatical? Better, how likely?	Good markups Good corrections Good translations

Two challenges

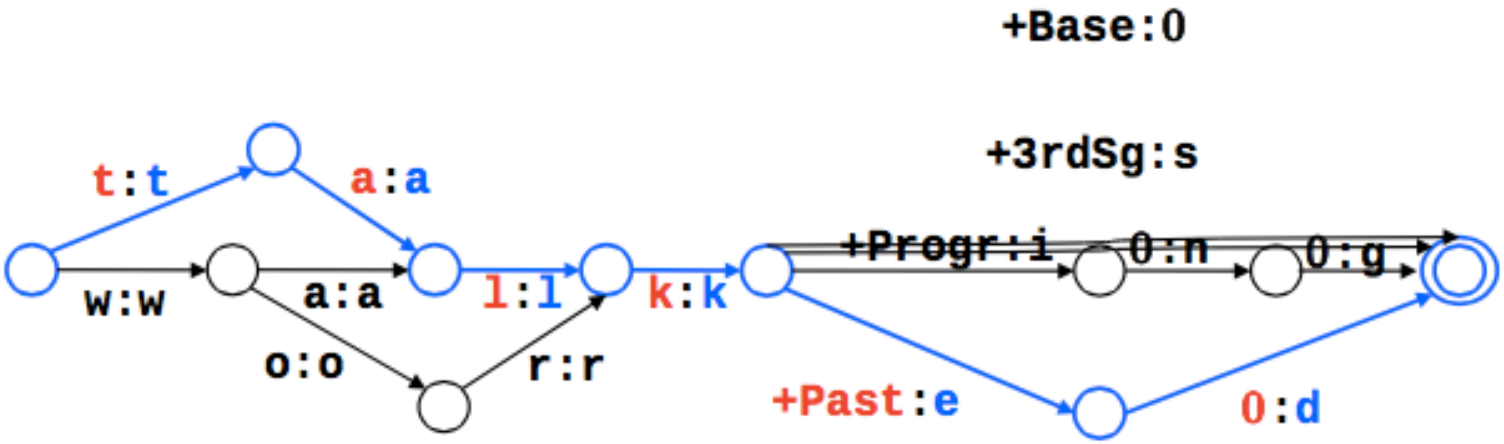
- Morphotactics
  - The order of the elements is important
    - Oku-yor-lar-dı is Turkish, oku-du-yor-lar is not Turkish
    - Respect-ive-ly is English, respect-ly-ive is not English

- Phonological alternations
  - The shape of an element may change.
    - Kitap-ı becomes kitab<u>ı</u> in Turkish
    - Die-ing becomes dying in English

Generation

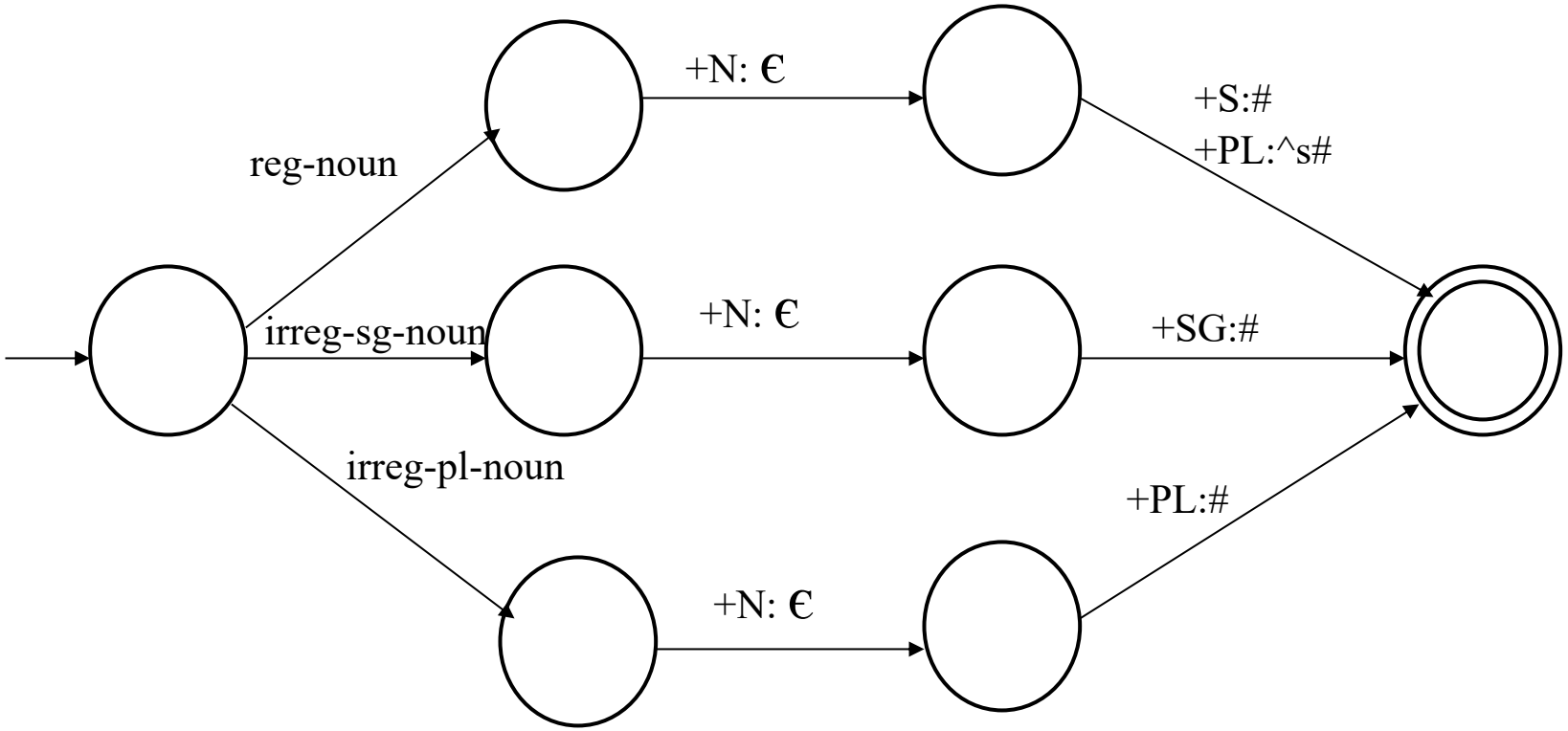

work+3rdSg --> works
+Base:0
+3rdSg:s
t:t
a:a
w:w
a:a
l:l
k:k
o:o
r:r
+Progr:i
0:n
0:g
+Past:e
0:d

Parsing/Analysis



talked --> talk+Past

An FST for Simple English Nominals



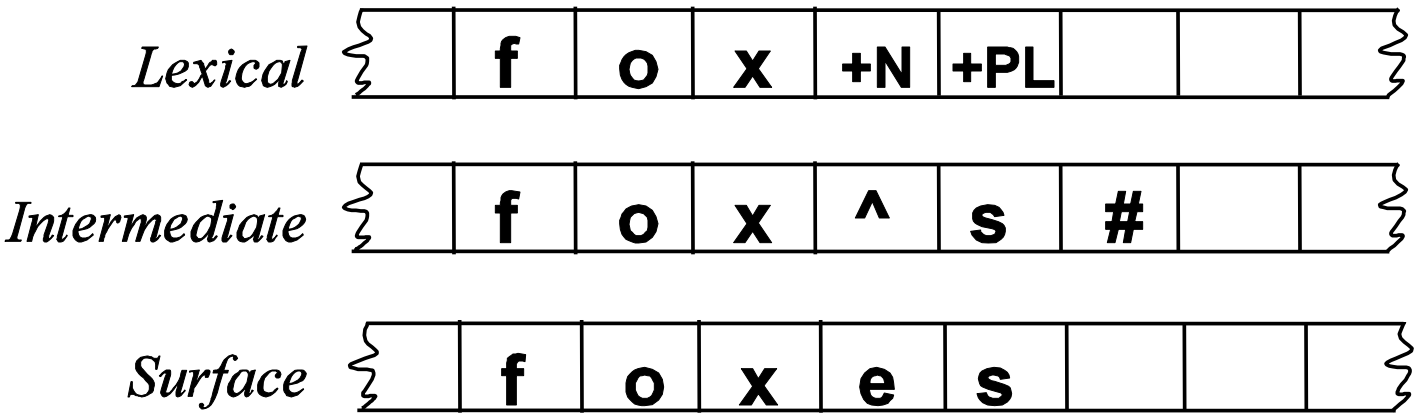
The Gory Details

- Of course, its not as easy as
  - `"cat +N +PL" <-> "cats"`
- Or even dealing with the irregulars geese, mice and oxen
- But there are also a whole host of spelling/pronunciation changes that go along with inflectional changes
  - E.g., fox +PL -> foxes

Multi-Tape Machines

- To deal with these complications, we will add more tapes and use the output of one tape machine as the input to the next
- So to handle irregular spelling changes we'll add intermediate tapes with intermediate symbols

Multi-Level Tape Machines



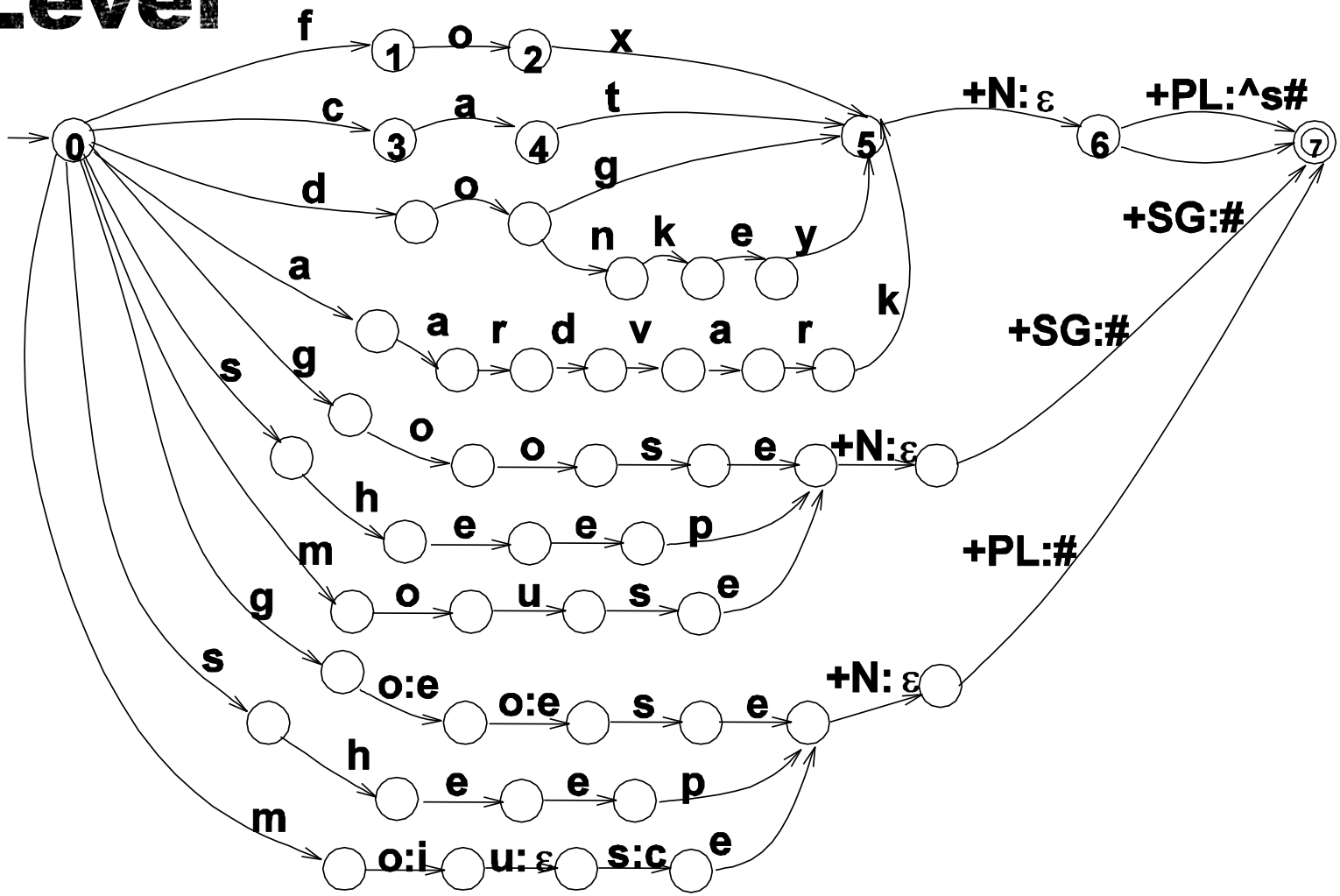
- We use one machine to transduce between the lexical and the intermediate level, and another to handle the spelling changes to the surface tape

Orthographic Rules and FSTs

- Define additional FSTs to implement rules such as consonant doubling (beg → begging), 'e' deletion (make → making), 'e' insertion (watch → watches), etc.

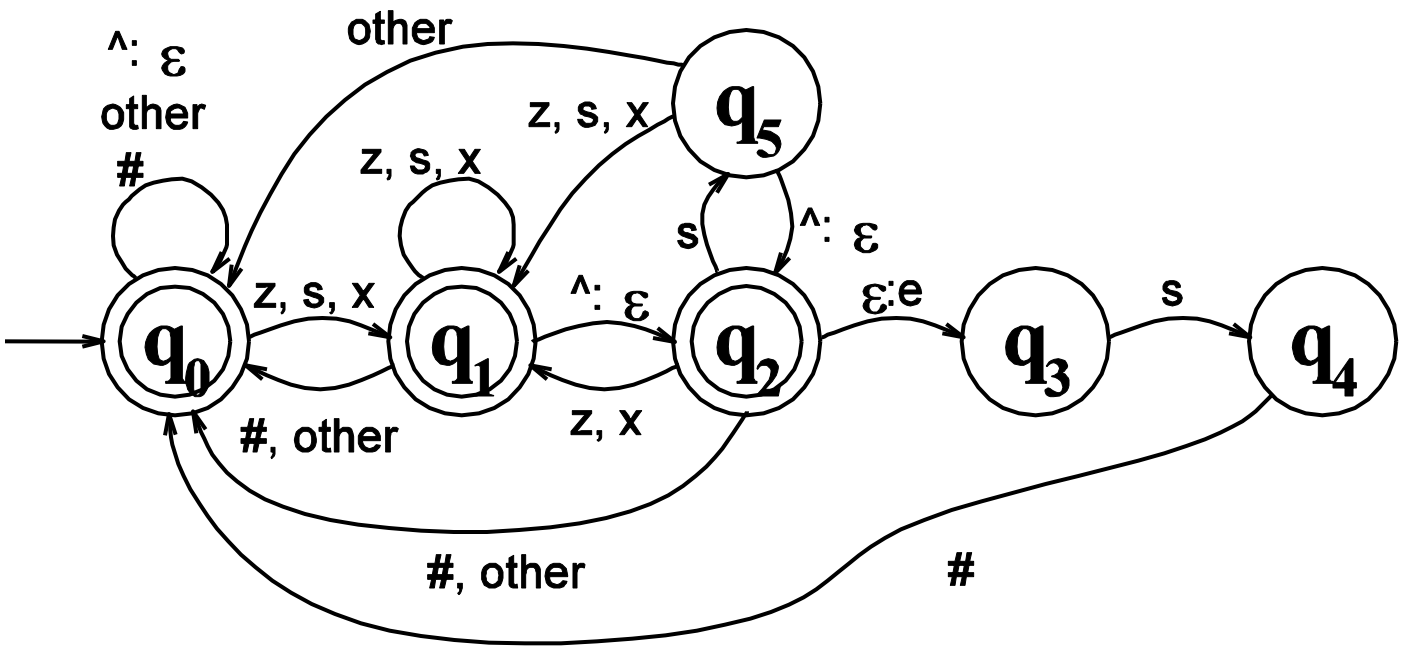
Lexical	f	o	x	+N	+PL	
Intermediate	f	o	x	^	s	#
Surface	f	o	x	e	s	

Lexical to Intermediate Level



Intermediate to Surface

- The add an “e” rule as in `fox^s# <-> foxes`



Note

- A key feature of this machine is that it doesn't do anything to inputs to which it doesn't apply.
- Meaning that they are written out unchanged to the output tape.

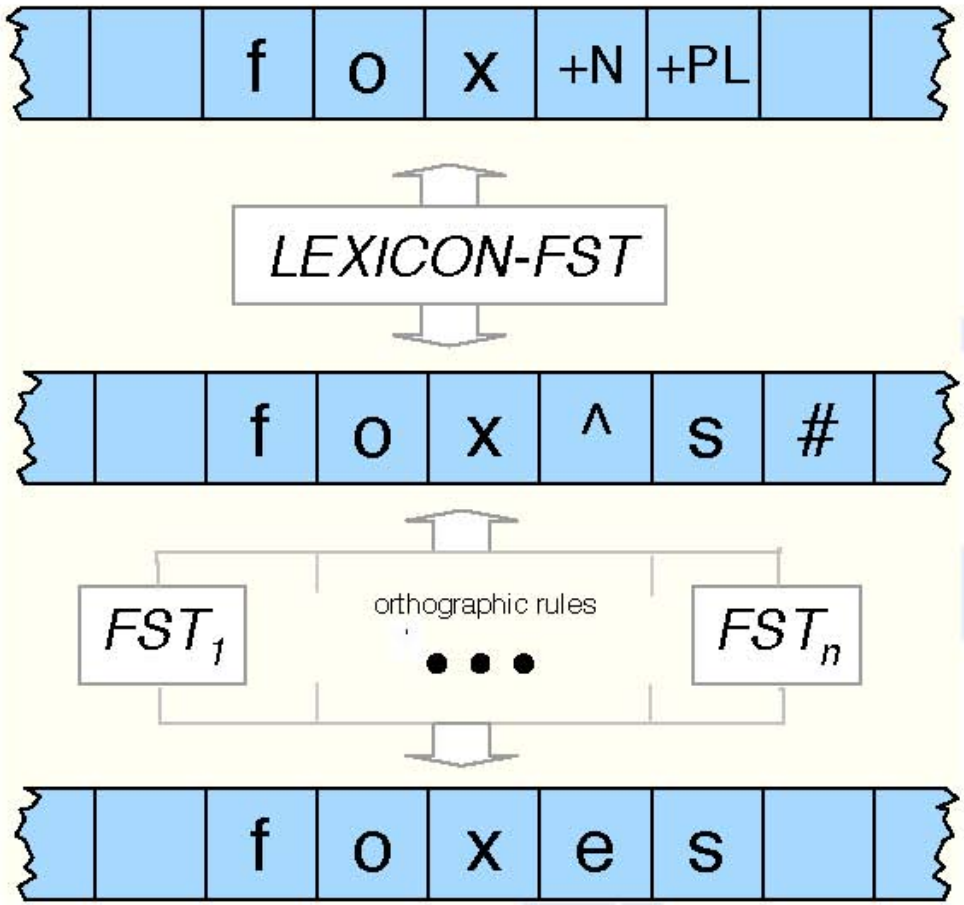


Figure 3.19 Generating or parsing with FST lexicon and rules

References

- Julia Hockenmaier, Finite-state methods for morphology slides
- Jurafsky and Karttunen, Finite-state automata, morphology and tokenization slides
- İlyas Çiçekli, Morphology slides
- Jurafsky, Basic Text Processing slides