CMSC 723: Computational Linguistics I — Session #3

Finite-State Morphology

Jimmy Lin
The iSchool
University of Maryland

Wednesday, September 16, 2009

Today's Agenda

- Computational tools
  - Regular expressions
  - Finite-state automata (deterministic vs. non-deterministic)
  - Finite-state transducers
- Overview of morphological processes
- Computational morphology with finite-state methods

Regular Expressions

- A metalanguage for specifying simple classes of strings
  - Very useful in searching and matching text strings
- Everyone does it!
  - Implementations in the shell, Perl, Java, Python, …

Regular Expressions

- Basic regular expressions

  /happy/ → happy

  /[abcd]/ → a, b, c, d

  /[a-d]/ → a, b, c, d

  /[^a-d]/ → e, f, g, … z

  /[Tt]he/ → the, The

  /(dog|cat)/ → dog, cat

- Special metacharacters

  /colou?r/ → color, colour

  /oo*h!/ → oh!, ooh!, oooh!, …

  /oo+h!/ → ooh!, oooh!, ooooh!, …

  /beg.n/ → began, begin, begun, begbn, …

NLP* with Regular Expressions

Transcript with Eliza, simulation of a Rogerian psychotherapist
(Weizenbaum, 1966)

User: Men are all alike

ELIZA: IN WHAT WAY

User: They're always bugging us about something or other

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE?

User: Well, my boyfriend made me come here

ELIZA: YOUR BOYFRIEND MADE YOU COME HERE

User: He says I'm depressed much of the time

ELIZA: I AM SORRY TO HEAR THAT YOU ARE DEPRESSED

How did it work?

- .* all .*

  → IN WHAT WAY

- .* always .*

  → CAN YOU THINK OF A SPECIFIC EXAMPLE

- .* I’m (depressed|sad) .*

  → I AM SORRY TO HEAR YOU ARE \1

- .* I’m (depressed|sad) .*

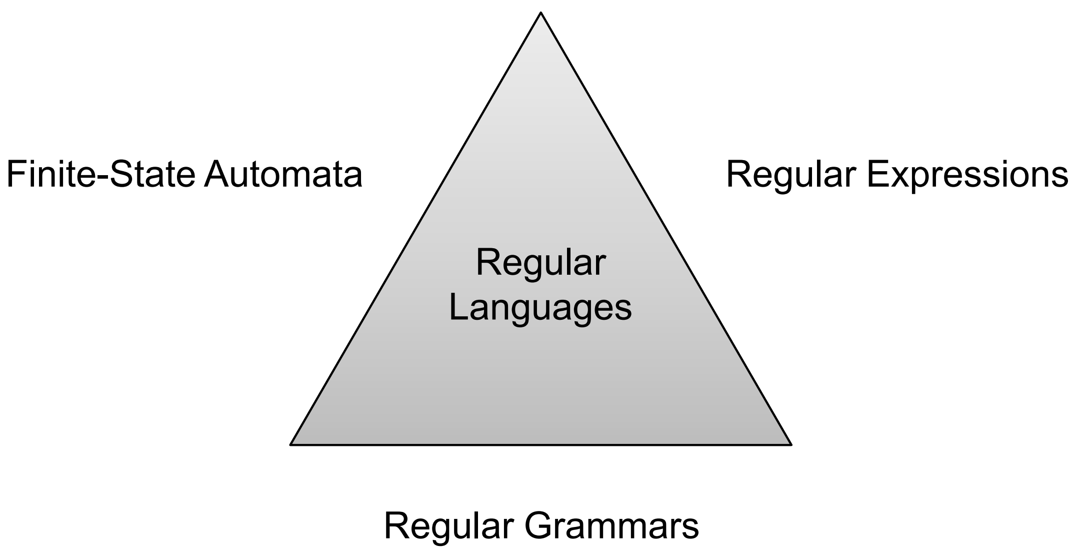
  → WHY DO YOU THINK YOU ARE \1?

Aside…

- What is intelligence?
- What does Eliza tell us about intelligence?

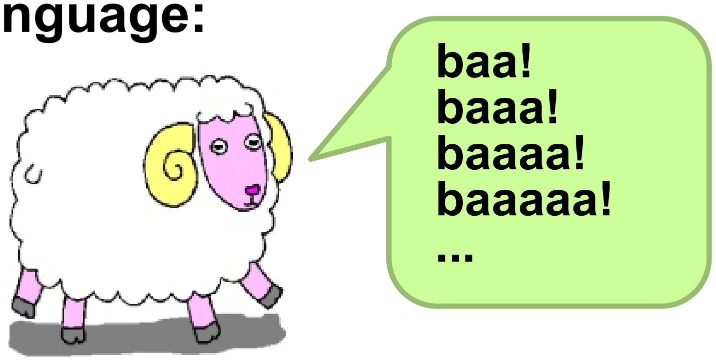
Equivalence Relations

- We can say the following
  - Regular expressions describe a regular language
  - Regular expressions can be implemented by finite-state automata
  - Regular languages can be generated by regular grammars
- So what?



Sheeptalk!

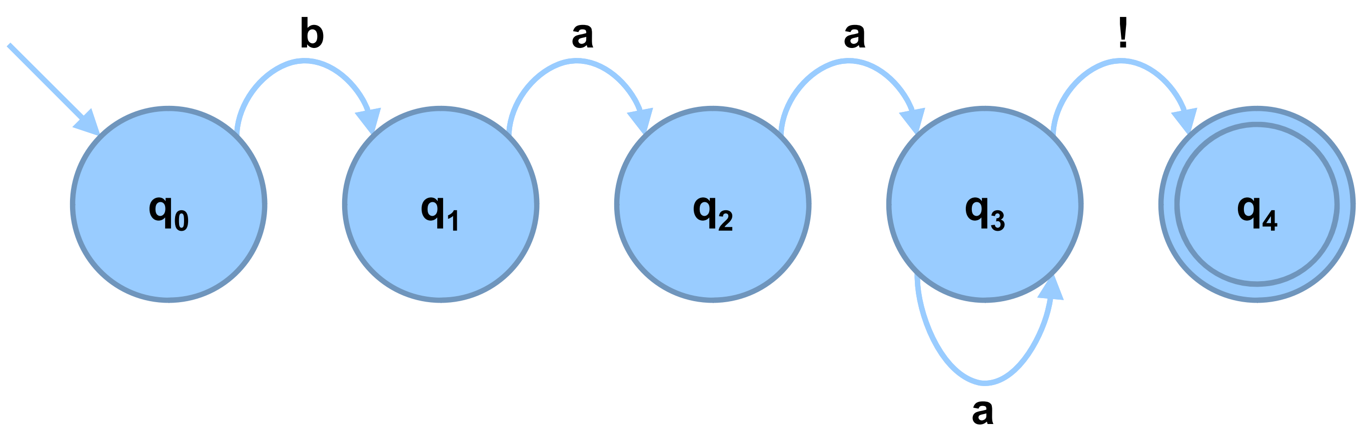
Language:



Regular Expression:

/baa+!/

Finite-State Automaton:

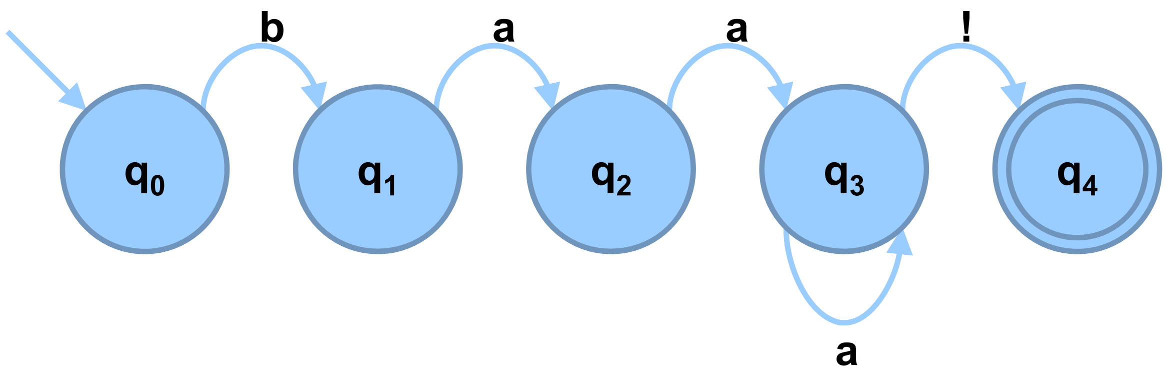


Finite-State Automata

- What are they?
- What do they do?
- How do they work?

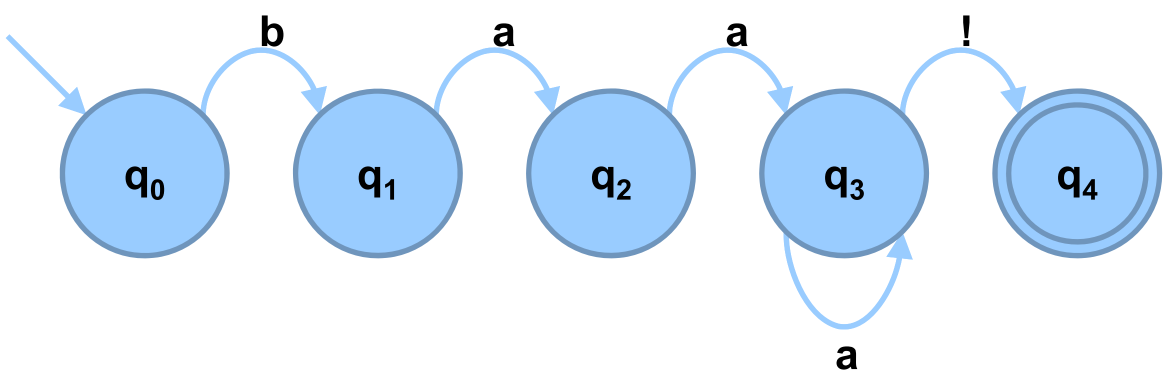
FSA: What are they?

- Q: a finite set of N states
  - Q = {q_0, q_1, q_2, q_3, q_4}
  - The start state: q_0
  - The set of final states: F = {q_4}
- Σ: a finite input alphabet of symbols
  - Σ = {*a*, *b*, *!*}
- δ(*q*,*i*): transition function
  - Given state *q* and input symbol *i*, return new state *q'*
  - $\delta(q_3, !) \rightarrow q_4$



FSA: State Transition Table

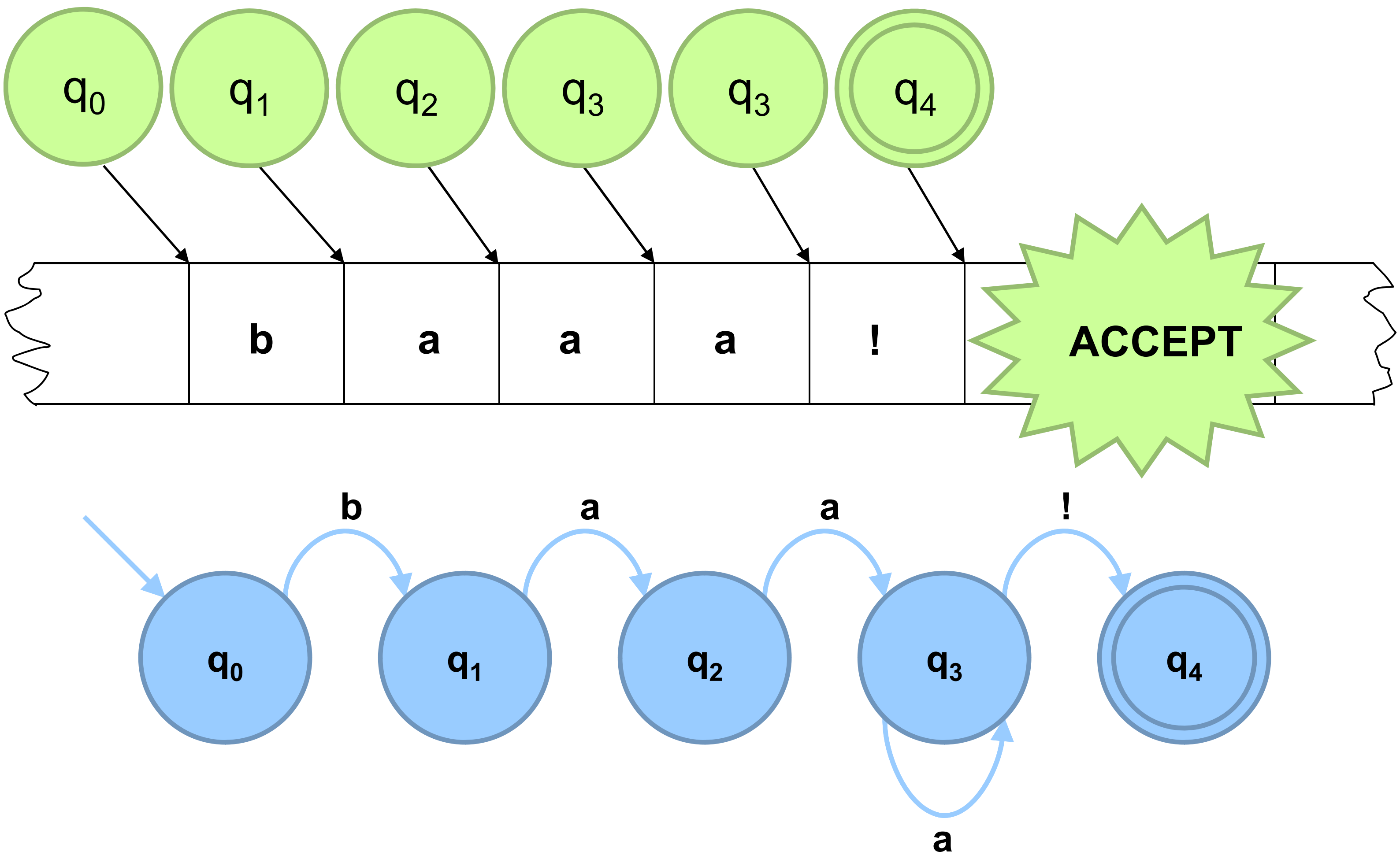
	Input		
State	**b**	**a**	**!**
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4	∅	∅	∅



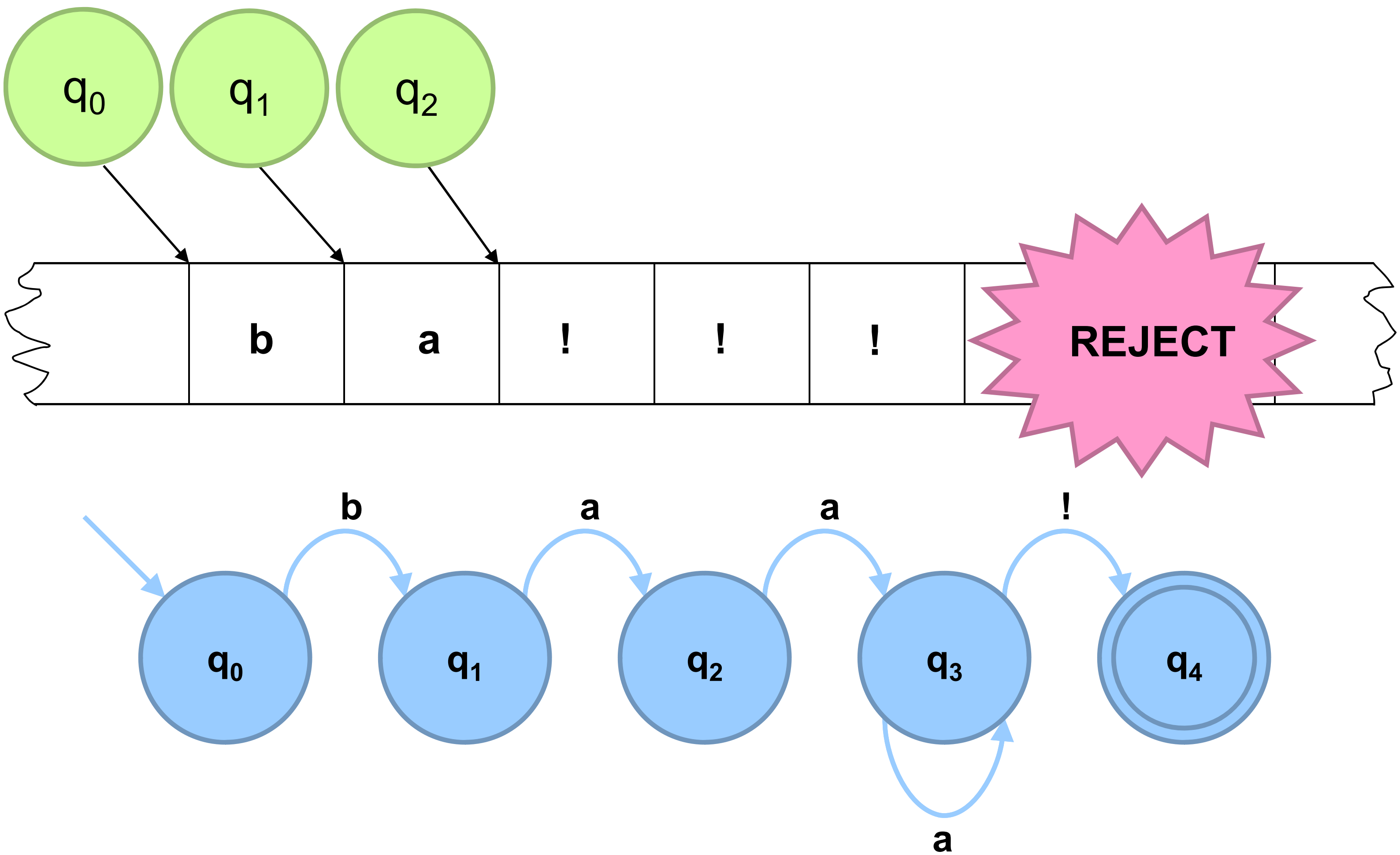
FSA: What do they do?

- Given a string, a FSA either rejects or accepts it
  - ba! → reject
  - baa! → accept
  - baaaz! → reject
  - baaaa! → accept
  - baaaaaa! → accept
  - baa → reject
  - moooo → reject
- What does this have to do with NLP?
  - Think grammaticality!

FSA: How do they work?



FSA: How do they work?



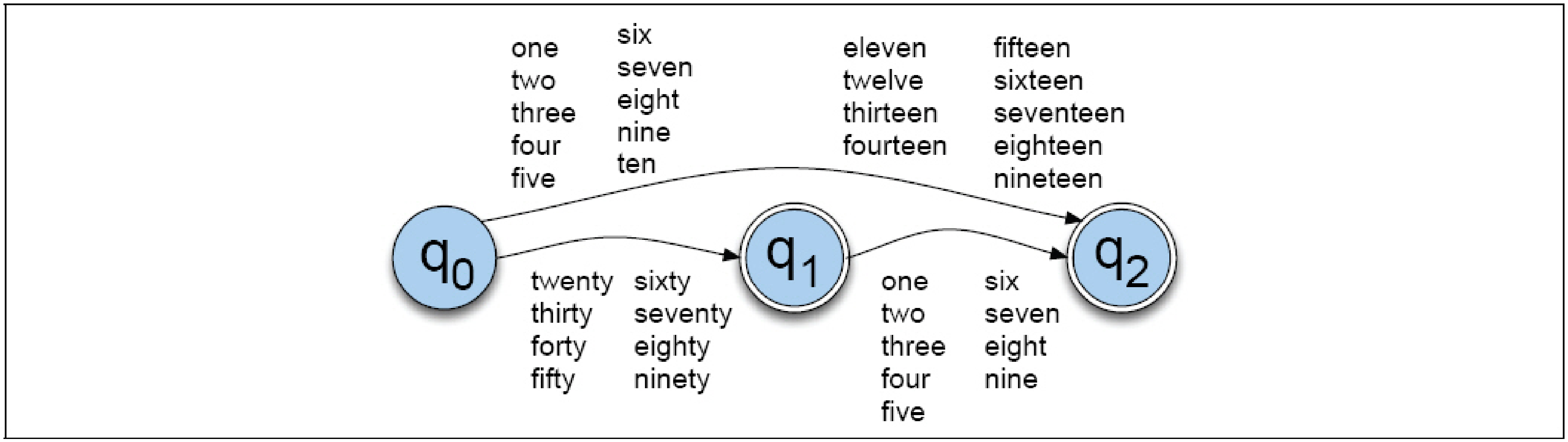
D-RECOGNIZE

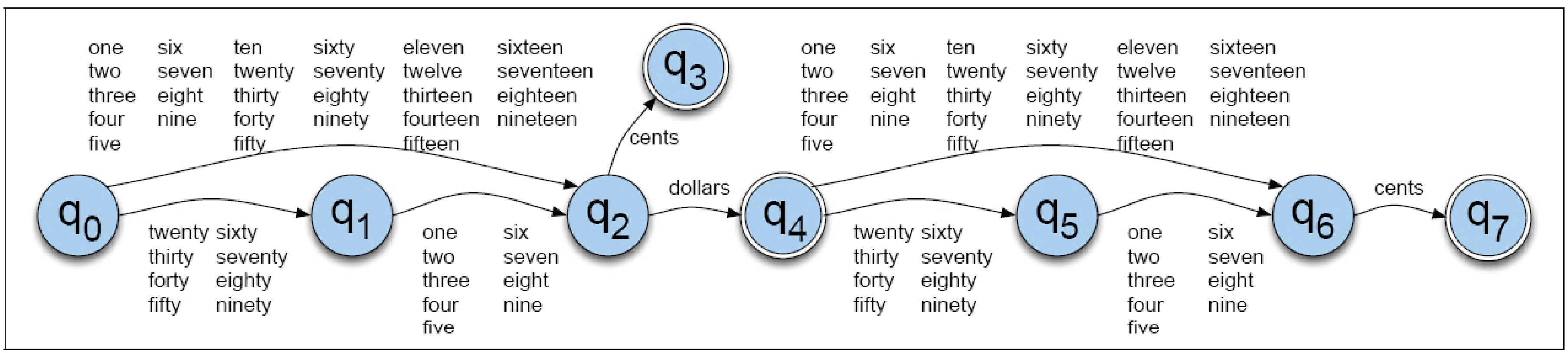
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached then
      if current-state is an accept state then
        return accept
      else
        return reject
    elsif transition-table[current-state,tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
  end
```

Accept or Generate?

- Formal languages are sets of strings
  - Strings composed of symbols drawn from a finite alphabet
- Finite-state automata define formal languages
  - Without having to enumerate all the strings in the language
- Two views of FSAs:
  - Acceptors that can tell you if a string is in the language
  - Generators to produce all and only the strings in the language

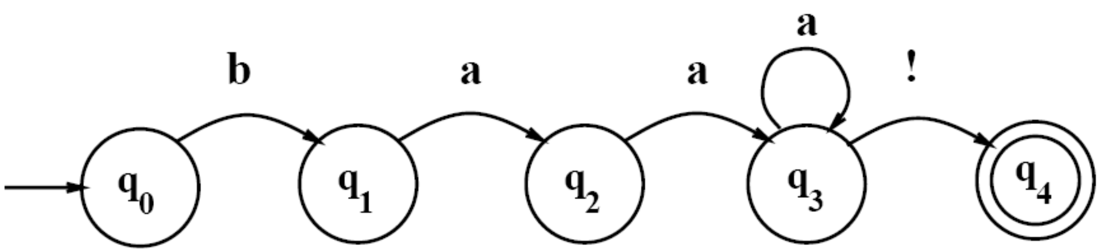
Simple NLP with FSAs

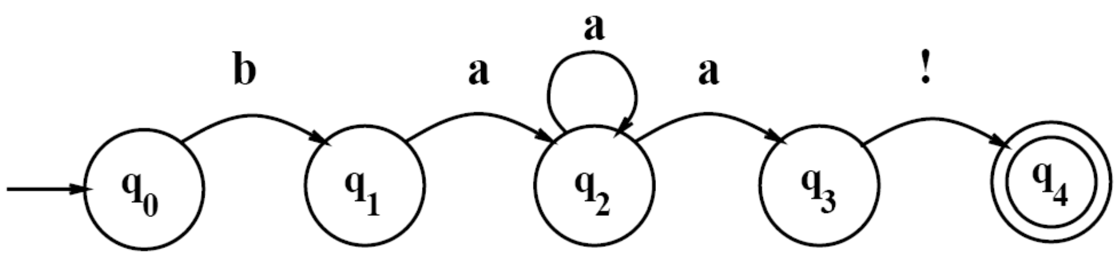




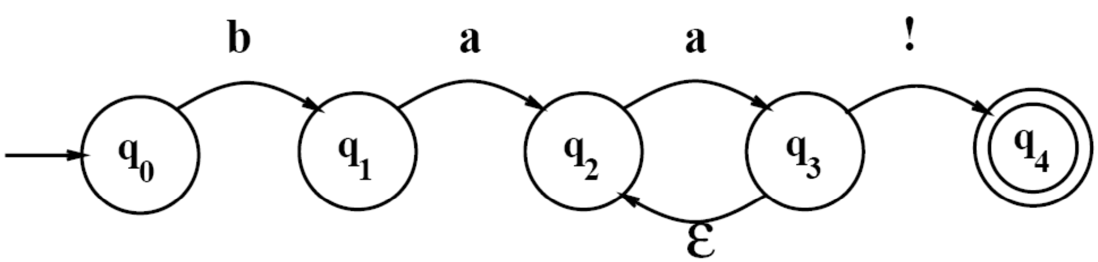
Introducing Non-Determinism

- Deterministic vs. Non-deterministic FSAs





- Epsilon (ε) transitions



Using NFSAs to Accept Strings

- What does it mean?
  - Accept: there exist at least one path (need not be all paths)
  - Reject: no paths exist
- General approaches:
  - Backup: add markers at choice points, then possibly revisit unexplored arcs at marked choice point
  - Look-ahead: look ahead in input to provide clues
  - Parallelism: look at alternatives in parallel
- Recognition with NFSAs as search through state space
  - Agenda holds (state, tape position) pairs

ND-RECOGNIZE

function ND-RECOGNIZE(*tape, machine*) **returns** accept or reject

```
agenda ← {(Initial state of machine, beginning of tape)}
current-search-state ← NEXT(agenda)
loop
  if ACCEPT-STATE?(current-search-state) returns true then
     return accept
  else
     agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
  if agenda is empty then
     return reject
  else
     current-search-state ← NEXT(agenda)
end
```

ND-RECOGNIZE

function GENERATE-NEW-STATES(*current-state*) **returns** a set of search-states

current-node ← the node the current search-state is in
index ← the point on the tape the current search-state is looking at
return a list of search states from transition table as follows:
(*transition-table[current-node,ε], index*)
∪
(*transition-table[current-node, tape[index]], index + 1*)

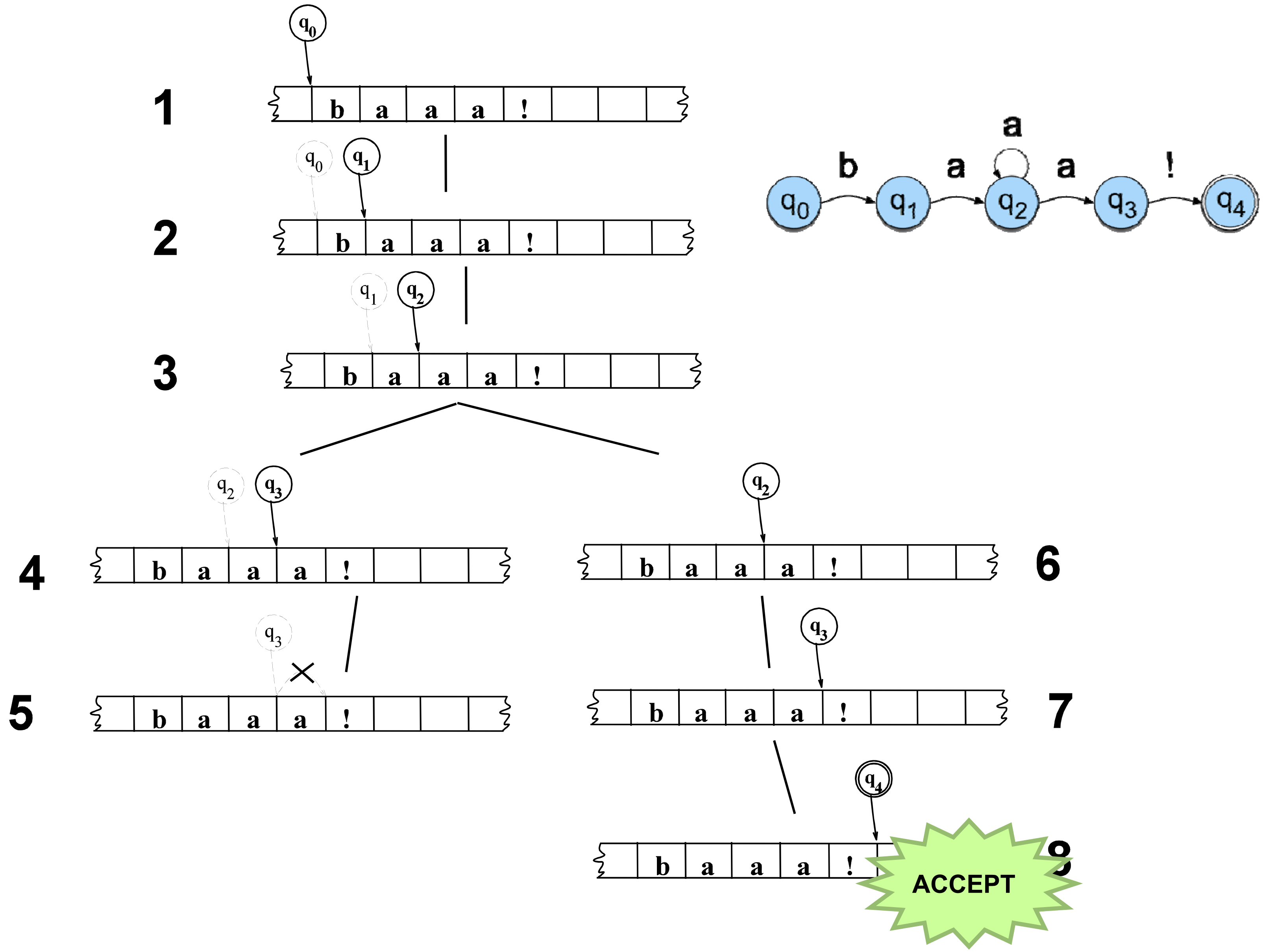
function ACCEPT-STATE?(*search-state*) **returns** true or false

current-node ← the node search-state is in
index ← the point on the tape search-state is looking at
if *index* is at the end of the tape **and** *current-node* is an accept state of machine
then
return true
else
return false

State Orderings

- Stack (LIFO): depth-first
- Queue (FIFO): breadth-first

ND-RECOGNIZE: Example



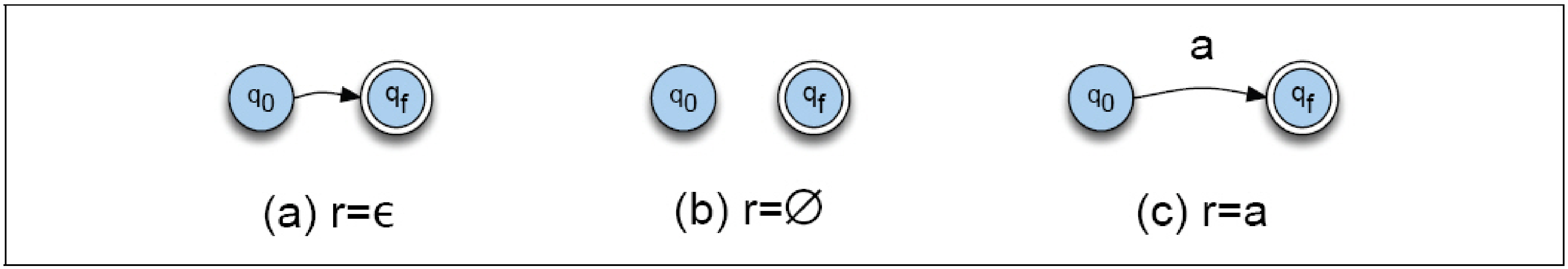
What's the point?

- NFSAs and DFSAs are equivalent
  - For every NFSA, there is a equivalent DFSA (and vice versa)
- Equivalence between regular expressions and FSA
  - Easy to show with NFSAs
- Why use NFSAs?

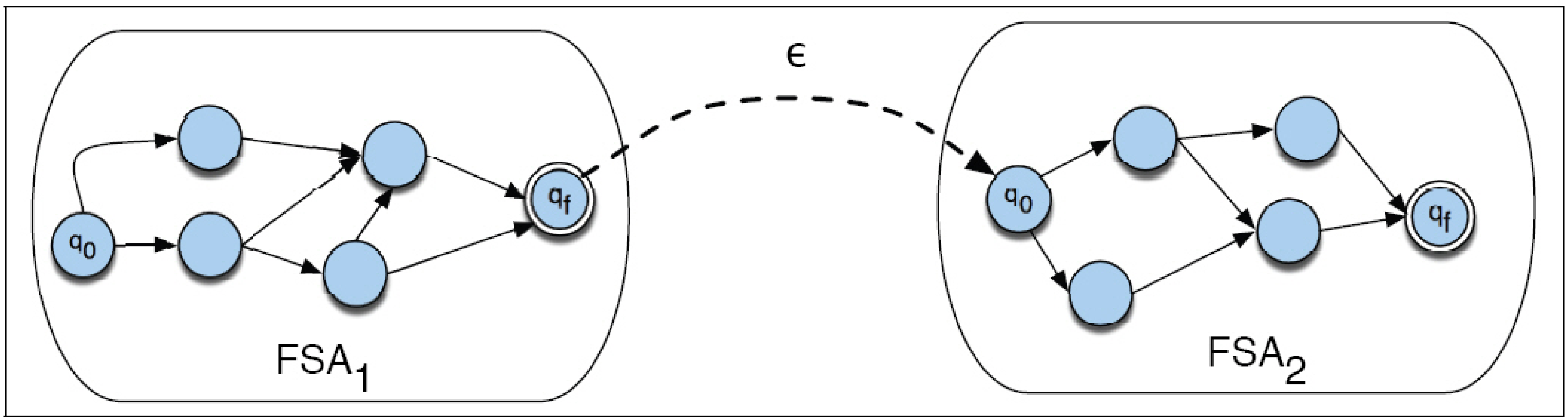
Regular Language: Definition

- $\varnothing$ is a regular language
- $\forall a \in \Sigma \cup \varepsilon$, $\{a\}$ is a regular language
- If L_1 and L_2 are regular languages, then so are:
  - $L_1 \cdot L_2 = \{x\, y \mid x \in L_1 ,\, y \in L_2 \}$, the *concatenation* of L_1 and L_2
  - $L_1 \cup L_2$, the *union* or *disjunction* of L_1 and L_2
  - $L_1{*}$, the *Kleene closure* of L_1

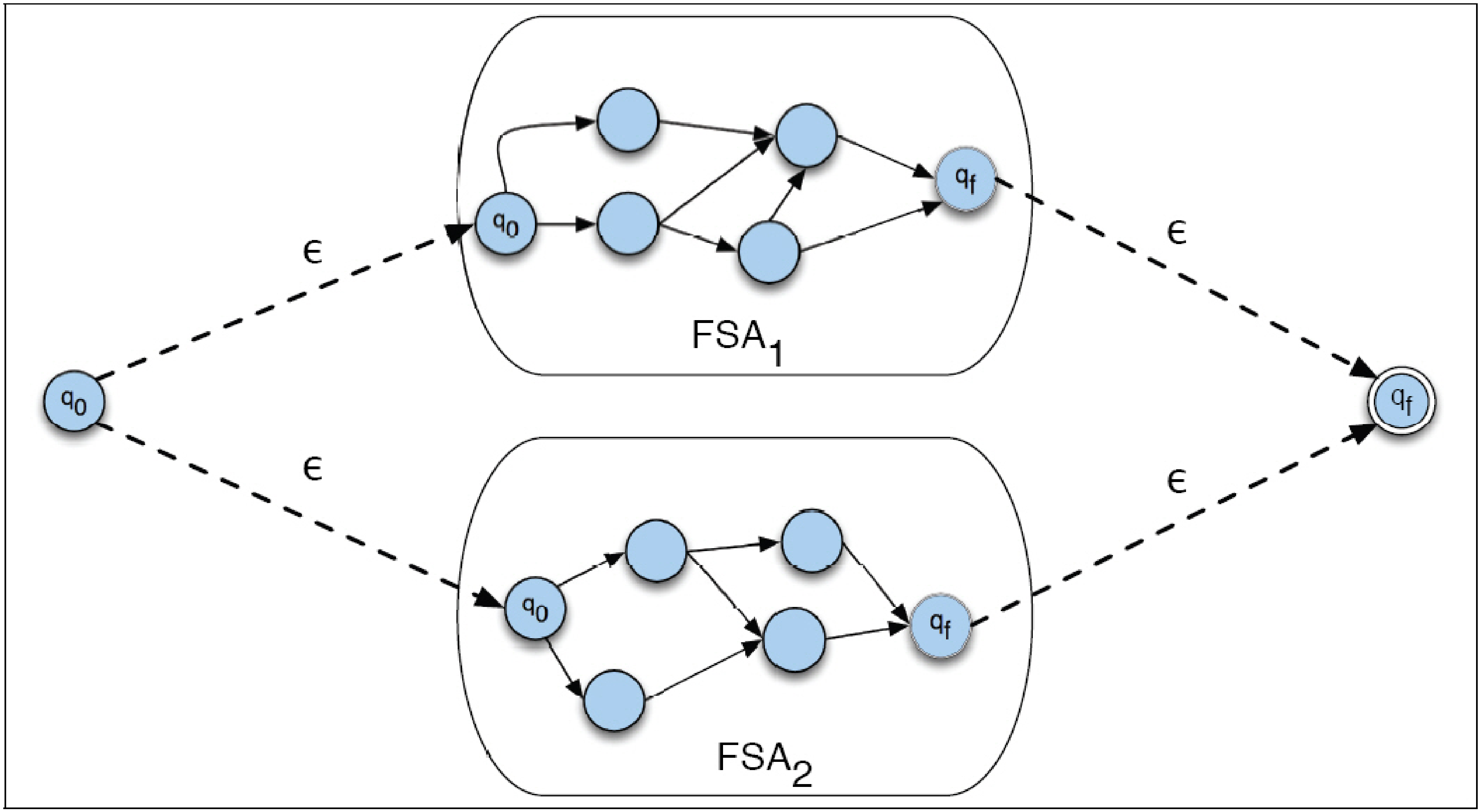
Regular Languages: Starting Points



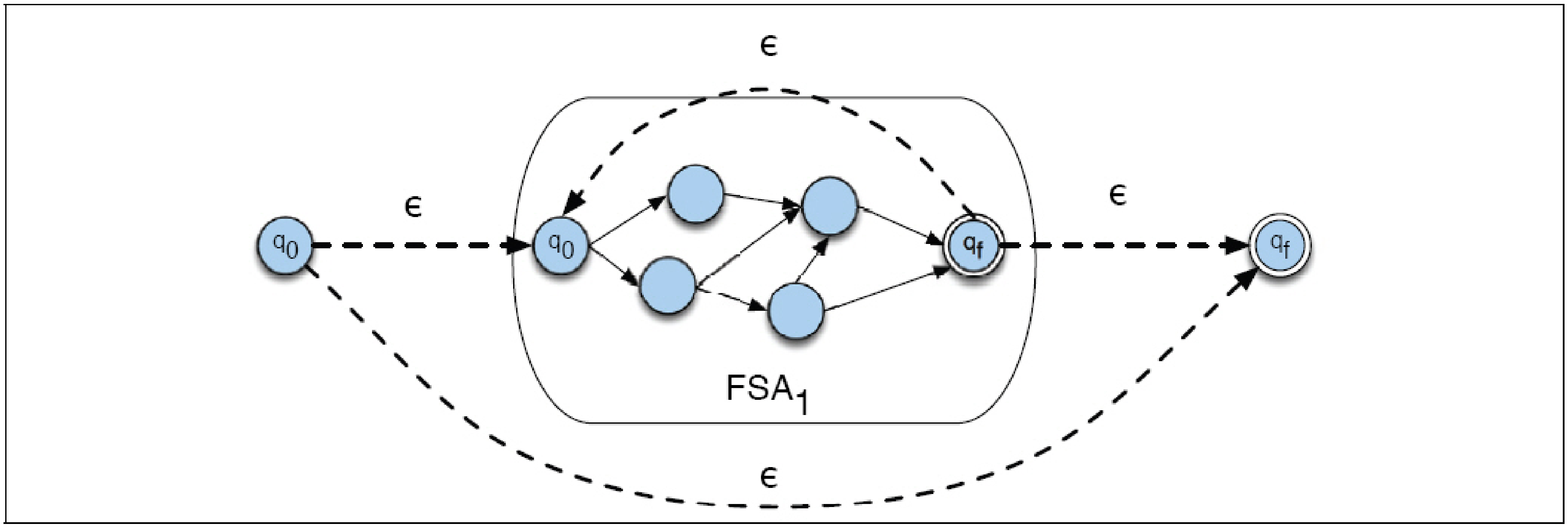
Regular Languages: Concatenation



Regular Languages: Disjunction

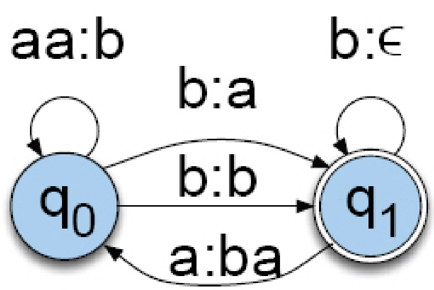


Regular Languages: Kleene Closure



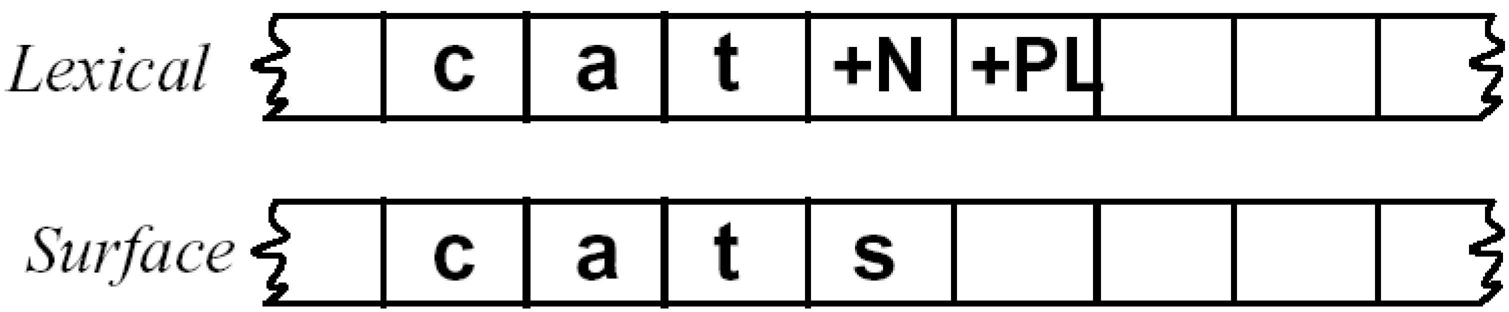
Finite-State Transducers (FSTs)

- A two-tape automaton that recognizes or generates pairs of strings
- Think of an FST as an FSA with two symbol strings on each arc
  - One symbol string from each tape



Four-fold view of FSTs

- As a recognizer
- As a generator
- As a translator
- As a set relater



Summary: Computational Tools

- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers

Computational Morphology

- Definitions and problems
  - What is morphology?
  - Topology of morphologies
- Computational morphology
  - Finite-state methods

Morphology

- Study of how words are constructed from smaller units of meaning
- Smallest unit of meaning = morpheme
  - fox has morpheme fox
  - cats has two morphemes cat and –s
  - Note: it is useful to distinguish morphemes from orthographic rules
- Two classes of morphemes:
  - Stems: supply the “main” meaning
  - Affixes: add “additional” meaning

Topology of Morphologies

- Concatenative vs. non-concatenative
- Derivational vs. inflectional
- Regular vs. irregular

Concatenative Morphology

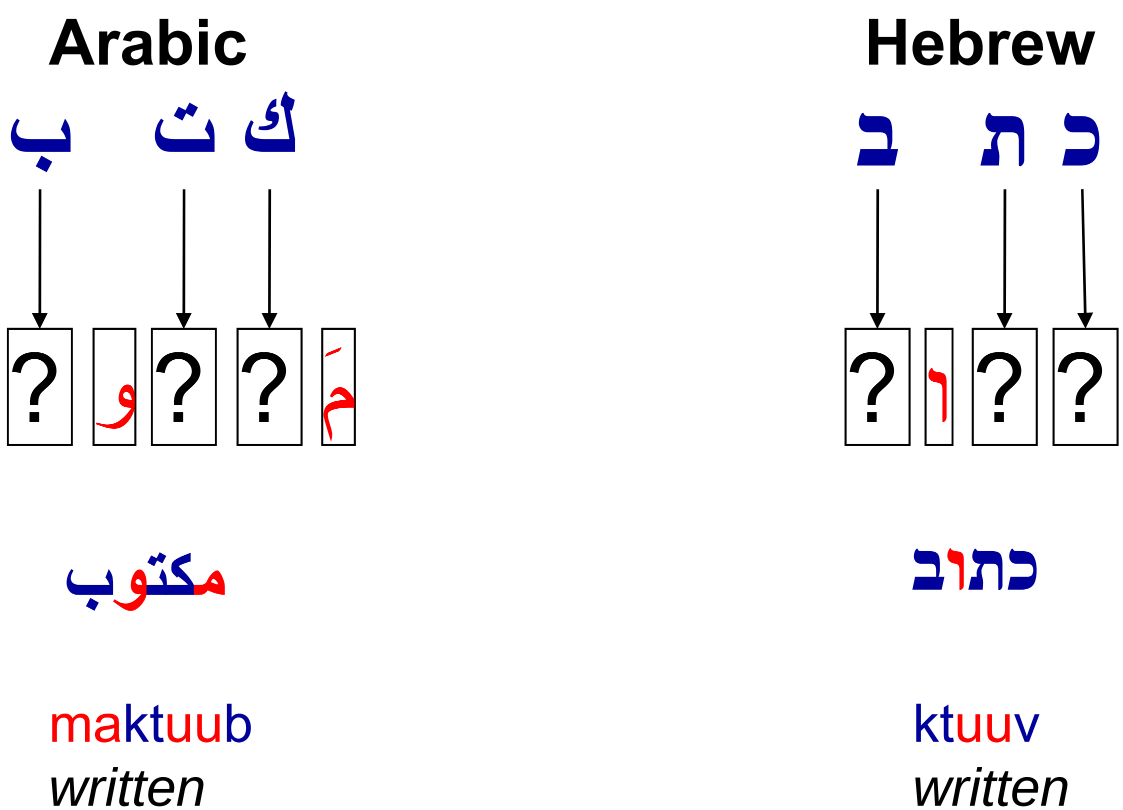
- Morpheme+Morpheme+Morpheme+…
- Stems (also called lemma, base form, root, lexeme):
  - hope+ing → hoping
  - hop+ing → hopping
- Affixes:
  - Prefixes: Antidis**establish**mentarianism
  - Suffixes: Antidis**establish**mentarianism
- Agglutinative languages (e.g., Turkish)
  - uygarlaştıramadıklarımızdanmışsınızcasına → uygar+laş+tır+ama+dık+lar+ımız+dan+mış+sınız+casına
  - Meaning: *behaving as if you are among those whom we could not cause to become civilized*

Non-Concatenative Morphology

- Infixes (e.g., Tagalog)
  - hingi (borrow)
  - humingi (borrower)
- Circumfixes (e.g., German)
  - sagen (say)
  - gesagt (said)
- Reduplication (e.g., Motu, spoken in Papua New Guinea)
  - mahuta (to sleep)
  - mahutamahuta (to sleep constantly)
  - mamahuta (to sleep, plural)

Templatic Morphologies

- Common in Semitic languages
- Roots and patterns



Derivational Morphology

- Stem + morpheme →
  - Word with different meaning or different part of speech
  - Exact meaning difficult to predict
- Nominalization in English:
  - -ation: computerization, characterization
  - -ee: appointee, advisee
  - -er: killer, helper
- Adjective formation in English:
  - -al: computational, derivational
  - -less: clueless, helpless
  - -able: teachable, computable

Inflectional Morphology

- Stem + morpheme →
  - Word with same part of speech as the stem
- Adds: tense, number, person,…
- Plural morpheme for English noun
  - cat+s
  - dog+s
- Progressive form in English verbs
  - walk+ing
  - rain+ing

Noun Inflections in English

- Regular
  - cat/cats
  - dog/dogs
- Irregular
  - mouse/mice
  - ox/oxen
  - goose/geese

Verb Inflections in English

Morphological Class	Regularly Inflected Verbs			
stem	walk	merge	try	map
-*s* form	walks	merges	tries	maps
-*ing* participle	walking	merging	trying	mapping
Past form or -*ed* participle	walked	merged	tried	mapped

Morphological Class	Irregularly Inflected Verbs		
stem	eat	catch	cut
-*s* form	eats	catches	cuts
-*ing* participle	eating	catching	cutting
preterite	ate	caught	cut
past participle	eaten	caught	cut

Verb Inflections in Spanish

	Present Indicative	Imperfect Indicative	Future	Preterite	Present Subjunctive	Conditional	Imperfect Subjunctive	Future Subjunctive
1SG	amo	amaba	amaré	amé	ame	amaría	amara	amare
2SG	amas	amabas	amarás	amaste	ames	amarías	amaras	amares
3SG	ama	amaba	amará	amó	ame	amaría	amara	amáreme
1PL	amamos	amábamos	amaremos	amamos	amemos	amaríamos	amáramos	amáremos
2PL	amáis	amabais	amaréis	amasteis	améis	amaríais	amarais	amareis
3PL	aman	amaban	amarán	amaron	amen	amarían	amaran	amaren

Morphological Parsing

- Computationally decompose input forms into component morphemes
- Components needed:
  - A lexicon (stems and affixes)
  - A model of how stems and affixes combine
  - Orthographic rules

Morphological Parsing: Examples

WORD	**STEM (+FEATURES)***
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
ducks	(duck +N +PL) or (duck +V +3SG)
merging	merge +V +PRES-PART
caught	(catch +V +PAST-PART) or (catch +V +PAST)

Different Approaches

- Lexicon only
- Rules only
- Lexicon and rules
  - finite-state automata
  - finite-state transducers

Lexicon-only

- Simply enumerate all surface forms and analyses
- So what's the problem?
- When might this be useful?

```
acclaim          acclaim $N$
acclaim          acclaim $V+0$
acclaimed        acclaim $V+ed$
acclaimed        acclaim $V+en$
acclaiming       acclaim $V+ing$
acclaims         acclaim $N+s$
acclaims         acclaim $V+s$
acclamation      acclamation $N$
acclamations     acclamation $N+s$
acclimate        acclimate   $V+0$
acclimated       acclimate   $V+ed$
acclimated       acclimate   $V+en$
acclimates       acclimate   $V+s$
acclimating      acclimate   $V+ing$
```

Rule-only: Porter Stemmer

- Cascading set of rules
  - ational → ate (e.g., reational → relate)
  - ing → ε (e.g., walking → walk)
  - sses → ss (e.g., grasses → grass)
  - …
- Examples
  - cities → citi
  - city→ citi
  - generalizations
    → generalization
    → generalize
    → general
    → gener

Porter Stemmer: What's the Problem?

- Errors…

Errors of Commission		Errors of Omission	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity

- Why is it still useful?

Lexicon + Rules

- FSA: for recognition
  - Recognize all grammatical input and only grammatical input
- FST: for analysis
  - If grammatical, analyze surface form into component morphemes
  - Otherwise, declare input ungrammatical

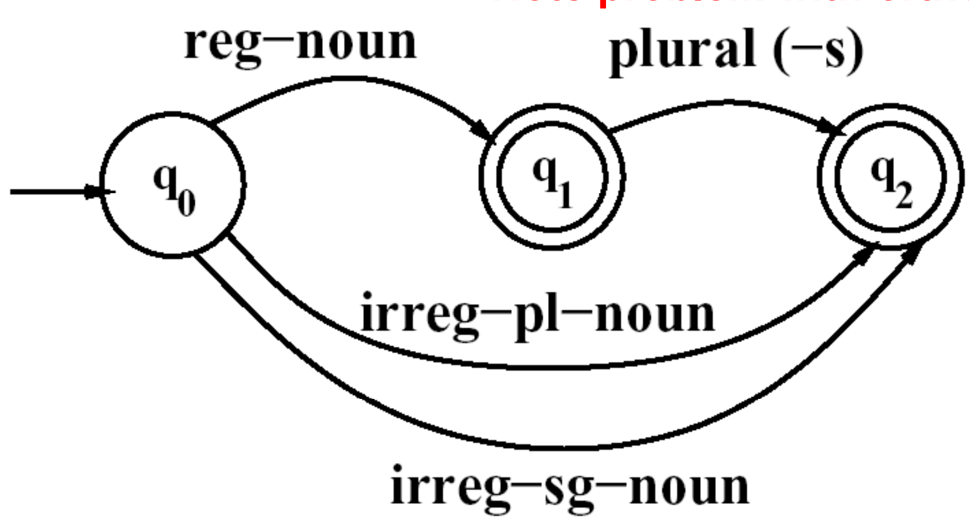
FSA: English Noun Morphology

Lexicon

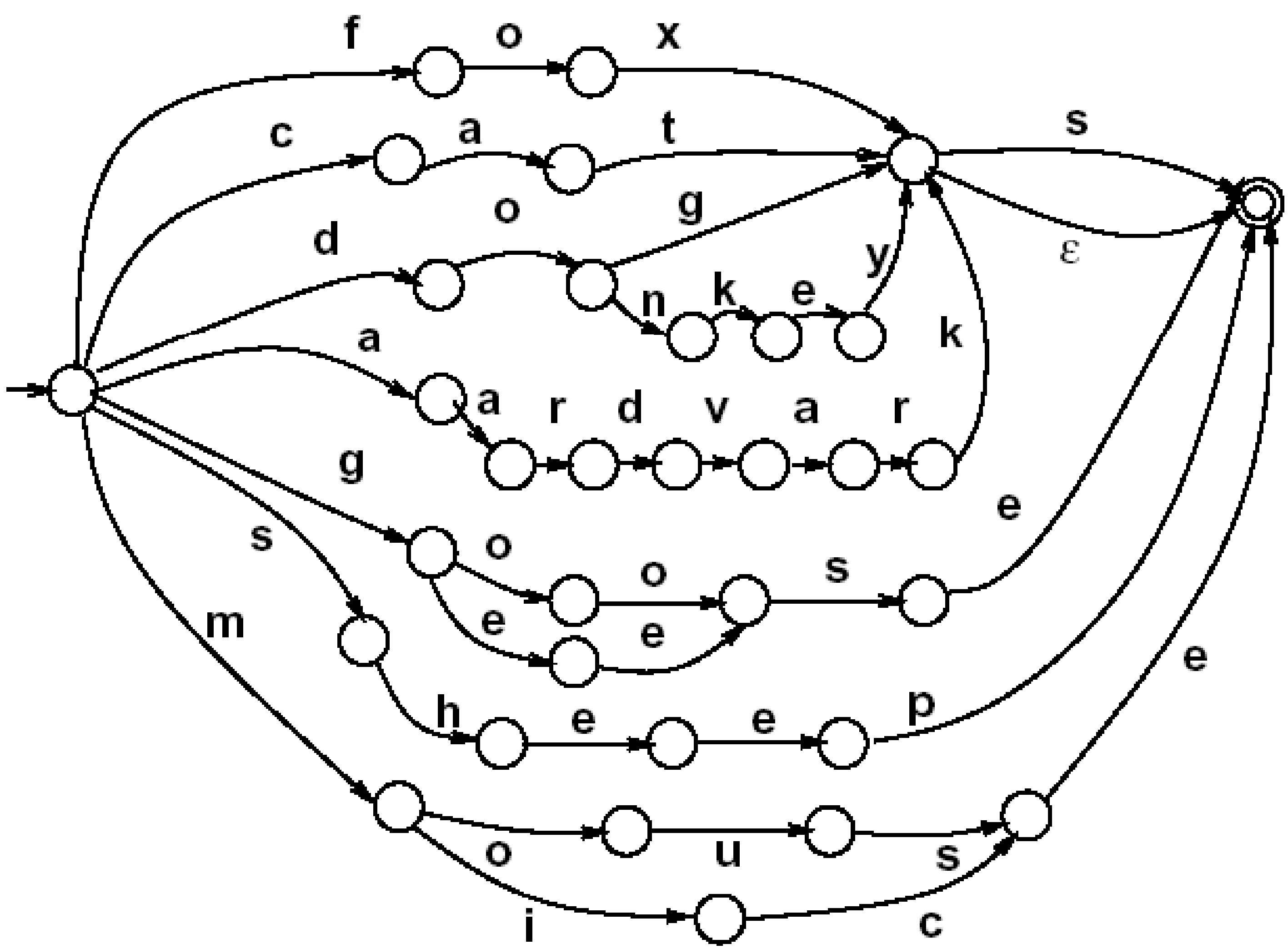
reg-noun	irreg-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	

Rule

Note problem with orthography!



FSA: English Noun Morphology

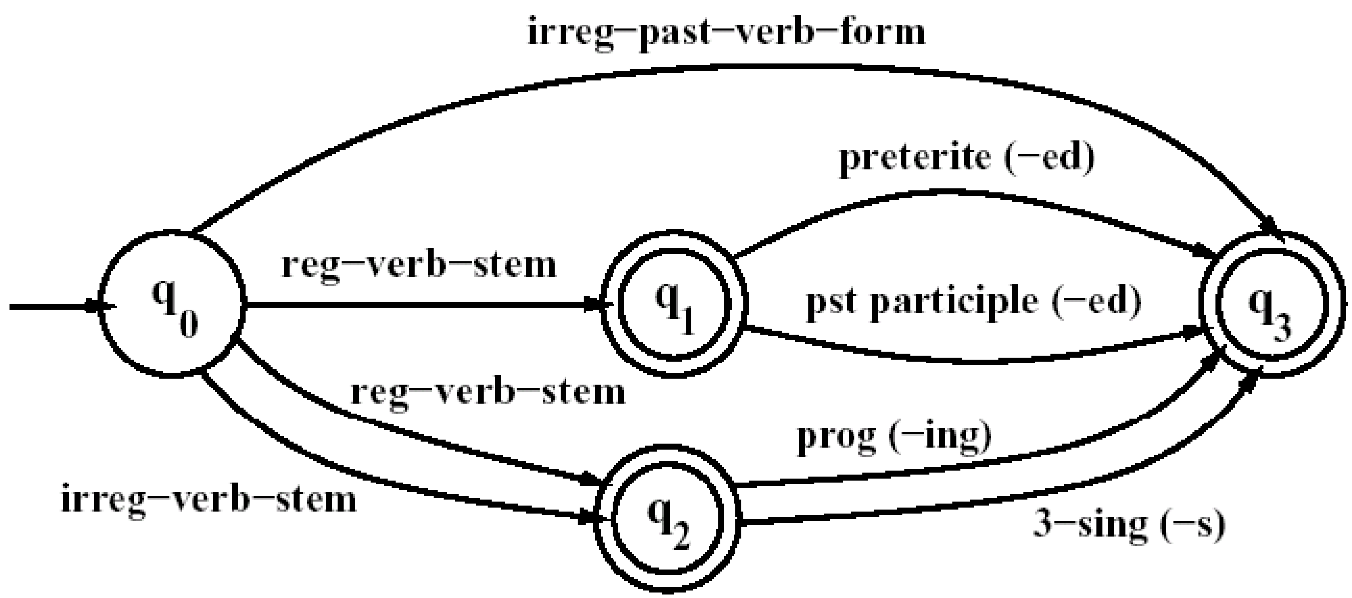


FSA: English Verb Morphology

Lexicon

reg-verb-stem	irreg-verb-stem	irreg-past-verb	past	past-part	pres-part	3sg
walk fry talk impeach	cut speak spoken sing sang	caught ate eaten	-ed	-ed	-ing	-s

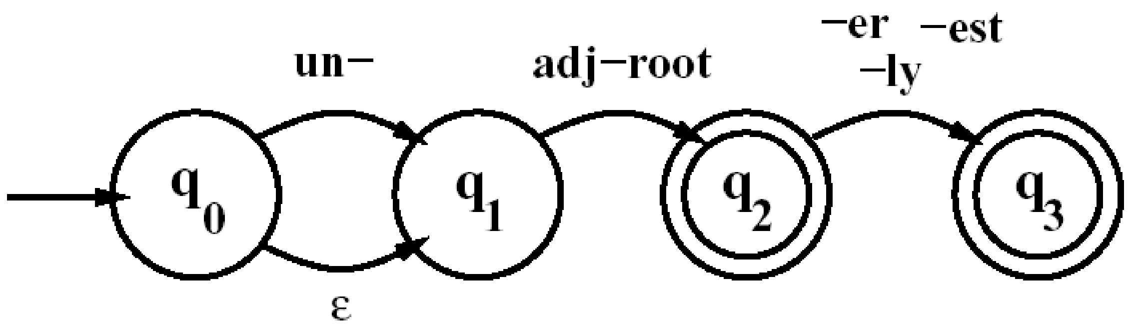
Rule

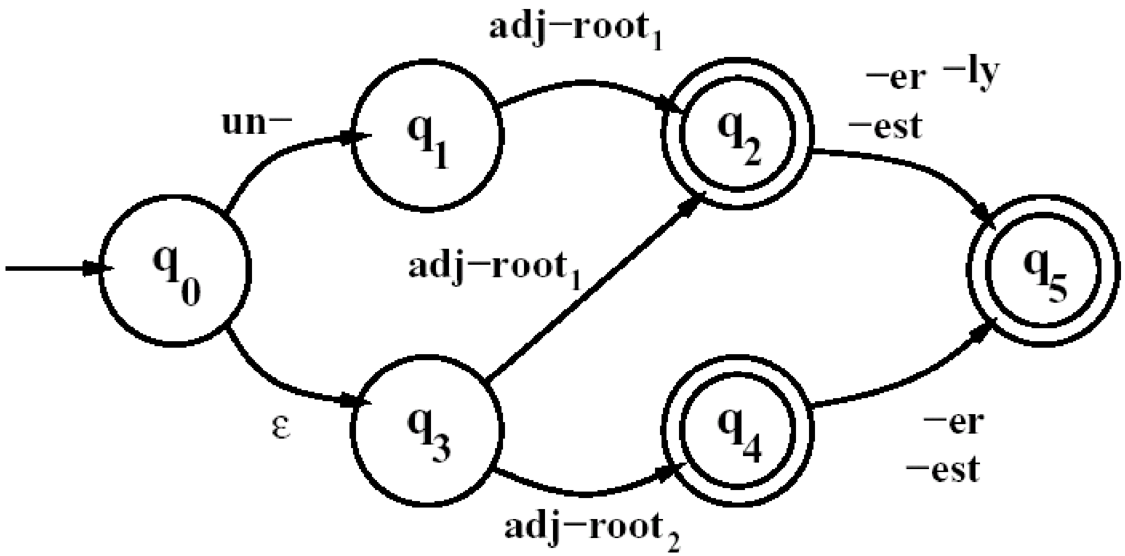


FSA: English Adjectival Morphology

- Examples:
  - big, bigger, biggest
  - smaller, smaller, smallest
  - happy, happier, happiest, happily
  - unhappy, unhappier, unhappiest, unhappily
- Morphemes:
  - Roots: big, small, happy, etc.
  - Affixes: un-, -er, -est, -ly

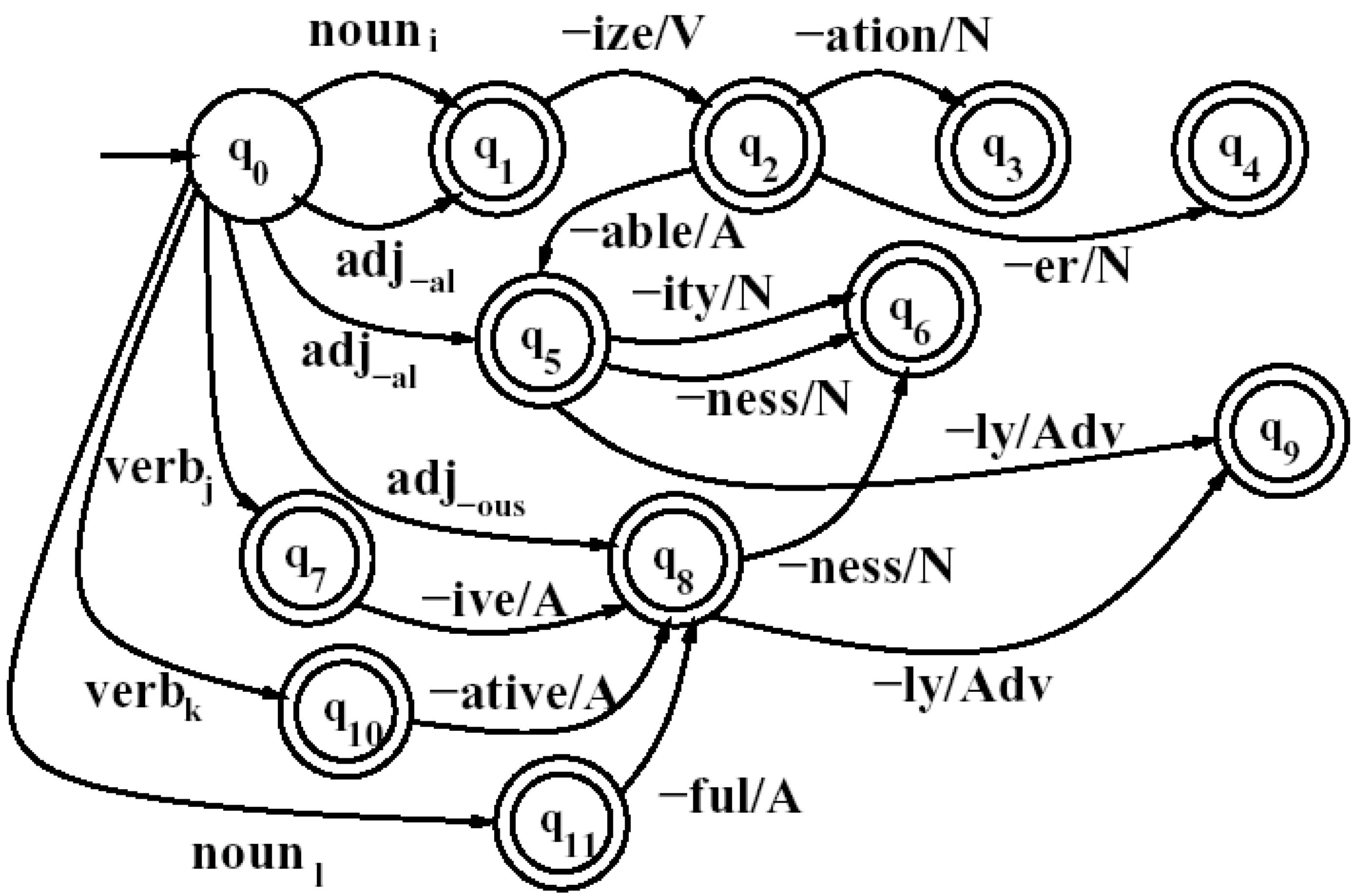
FSA: English Adjectival Morphology





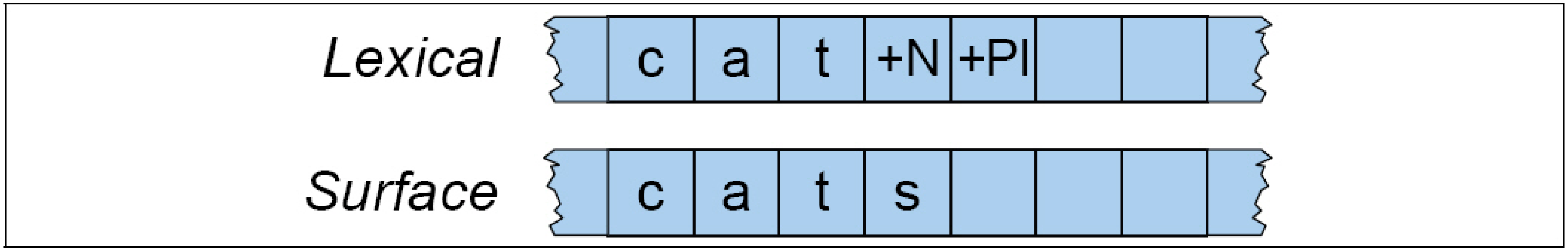
adj-root$_1$: {happy, real, …}
adj-root$_2$: {big, small, …}

FSA: Derivational Morphology



Morphological Parsing with FSTs

- Limitation of FSA:
  - Accepts or rejects an input… but doesn't actually provide an analysis
- Use FSTs instead!
  - One tape contains the input, the other tape as the analysis
  - What if both tapes contain symbols?
  - What if only one tape contains symbols?

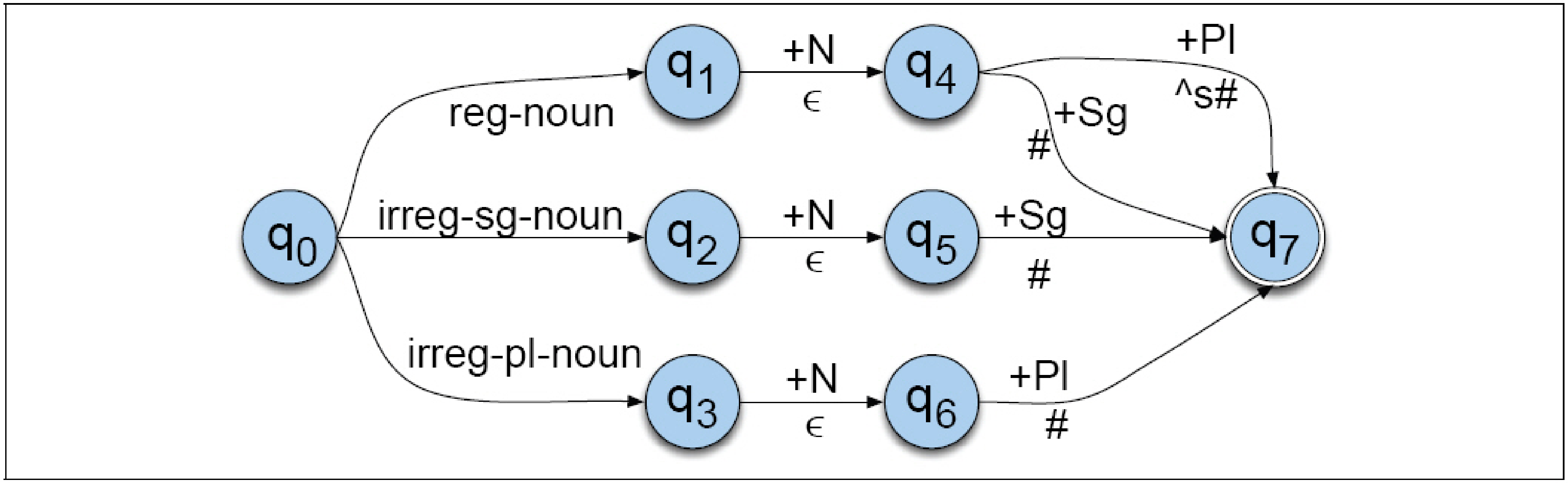


Terminology

- Transducer alphabet (pairs of symbols):
  - a:b = *a* on the upper tape, *b* on the lower tape
  - a:ε = *a* on the upper tape, nothing on the lower tape
  - If a:a, write a for shorthand
- Special symbols
  - # = word boundary
  - ^ = morpheme boundary
  - (For now, think of these as mapping to ε)

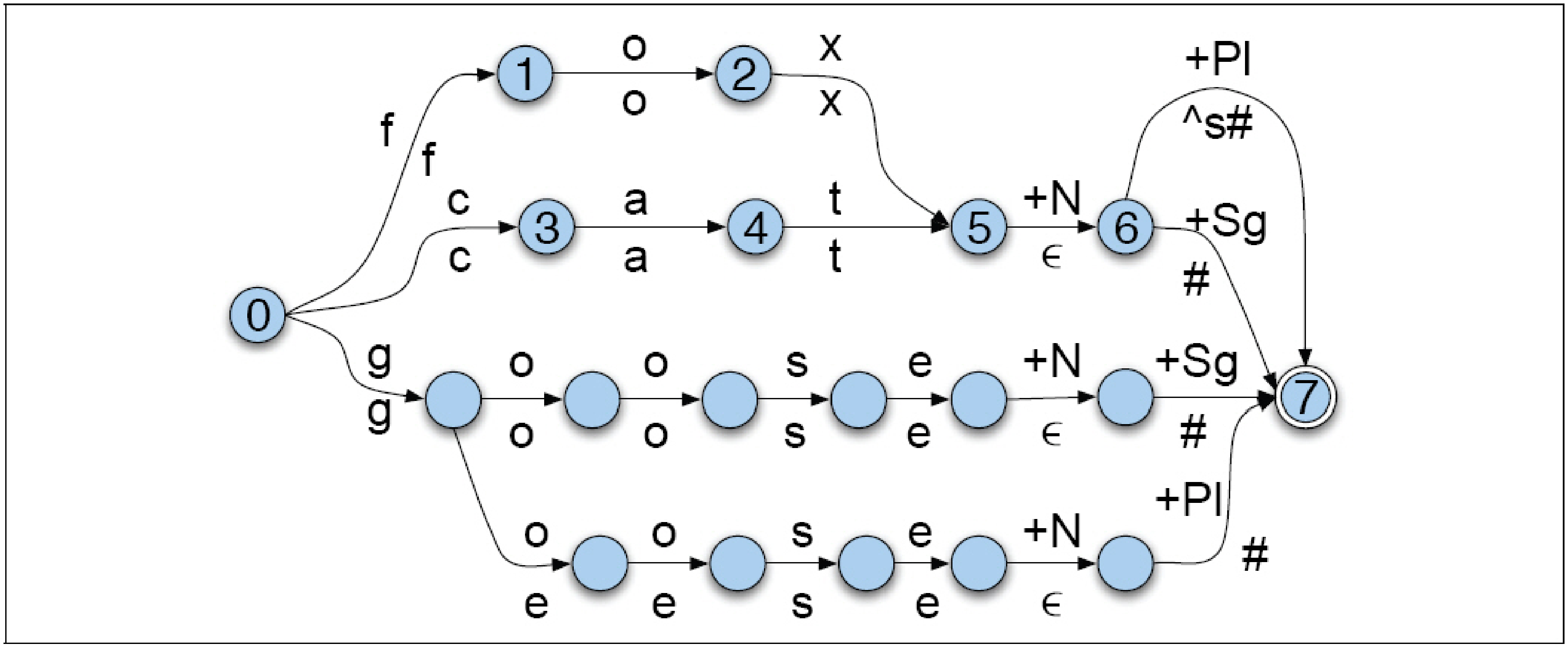
FST for English Nouns

- First try:

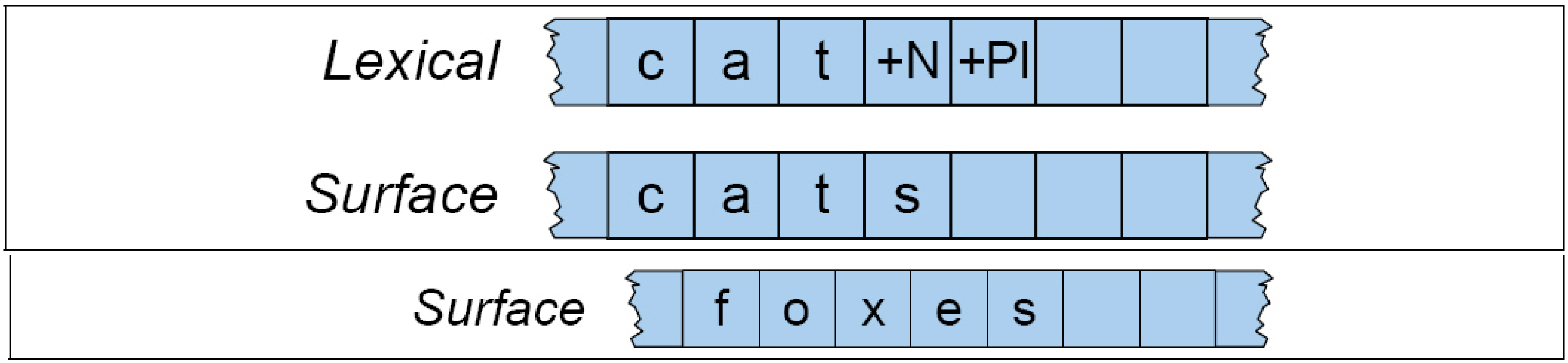


- What’s the problem here?

FST for English Nouns

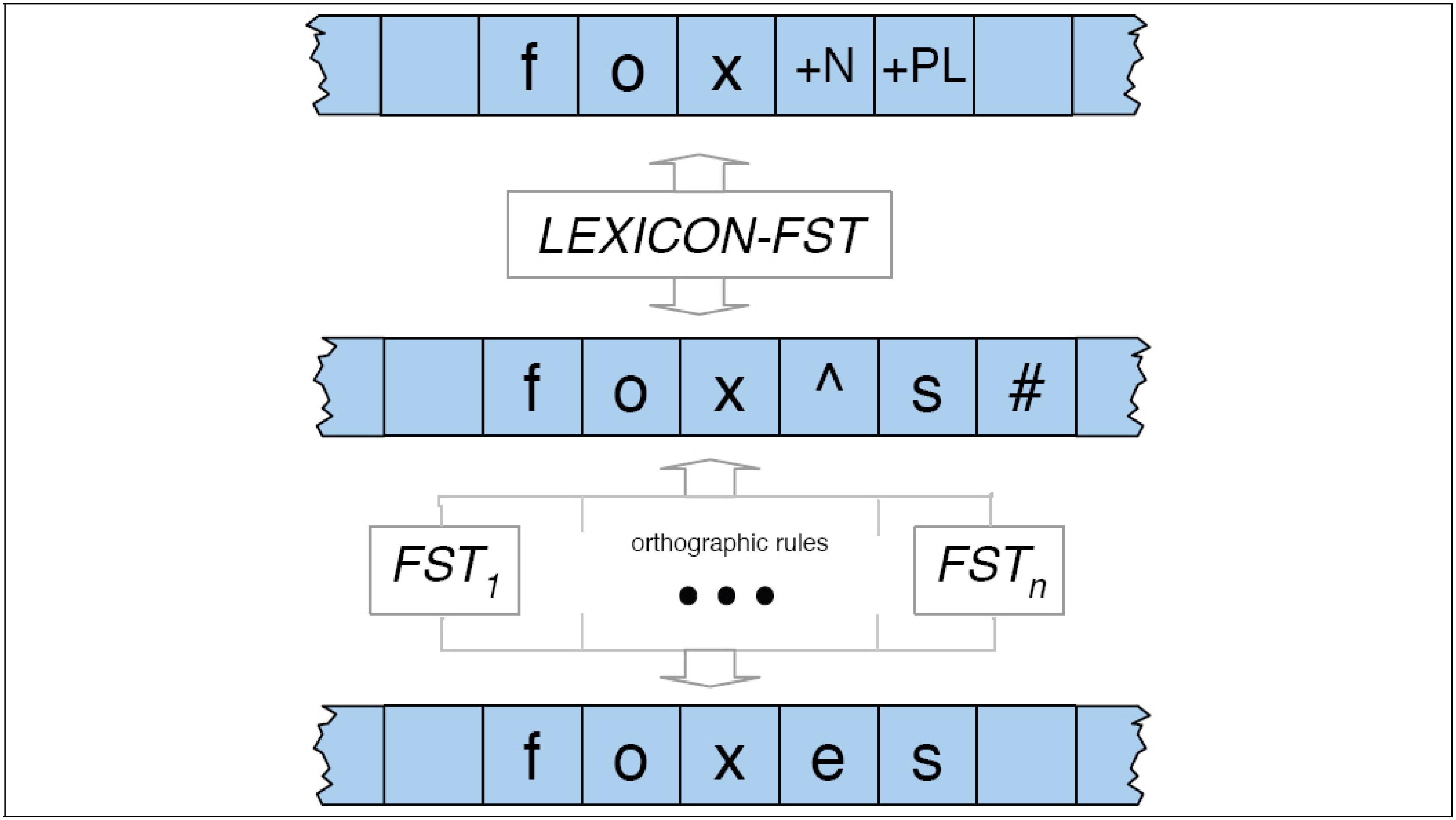


Handling Orthography



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

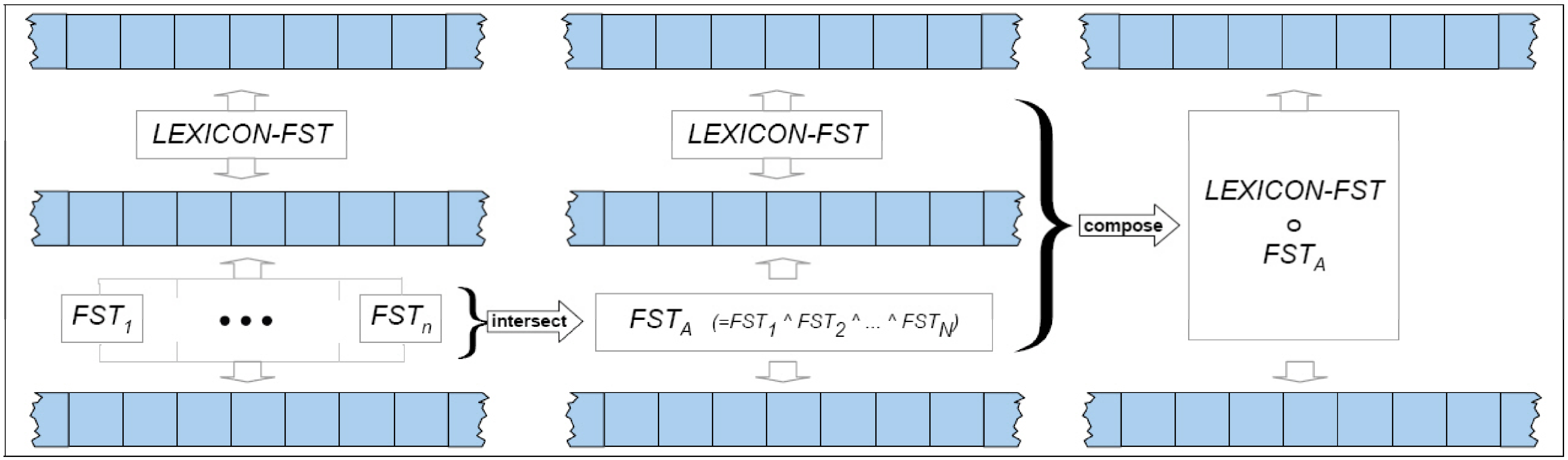
Complete Morphological Parser



FSTs and Ambiguity

- unionizable
  - **union** +ize +able
  - un+ **ion** +ize +able
- assess
  - **assess** +V
  - **ass** +N +essN

Optimizations



Practical NLP Applications

- In practice, it is almost never necessary to write FSTs by hand…
- Typically, one writes rules:
  - Chomsky and Halle Notation: $a \rightarrow b \ / \ c\_\_d$
    = rewrite a as b when occurs between c and d
  - E-Insertion rule

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

- Rule → FST compiler handles the rest…

What we covered today…

- Computational tools
  - Regular expressions
  - Finite-state automata (deterministic vs. non-deterministic)
  - Finite-state transducers
- Overview of morphological processes
- Computational morphology with finite-state methods
- One final question: is morphology actually finite state?