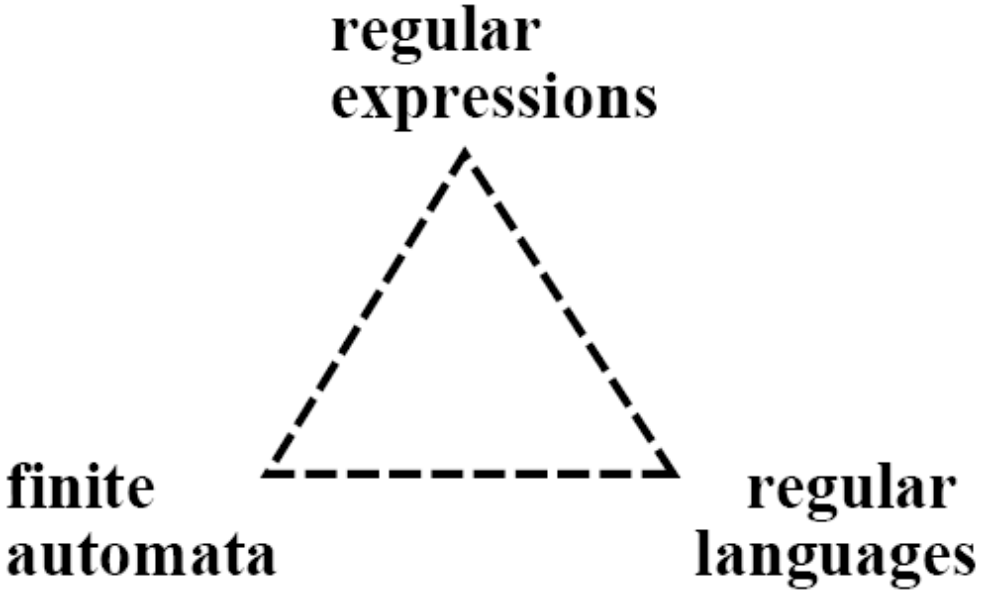
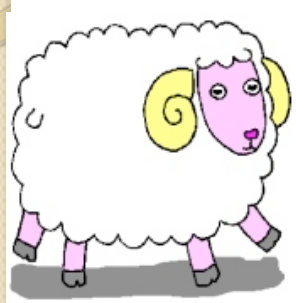
Regular Expressions and Finite State Automata

Finite-state automata

- Finite-state automata (FSA)
- Regular languages
- Regular expressions

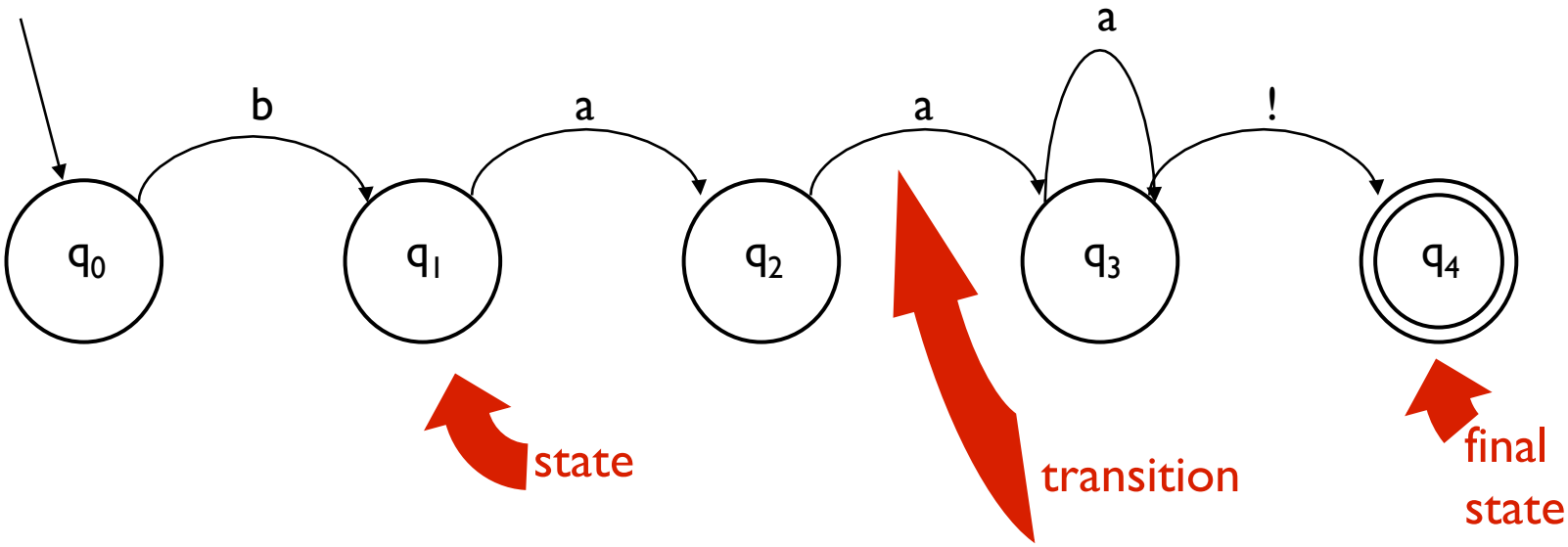


Finite-state Automata (Machines)

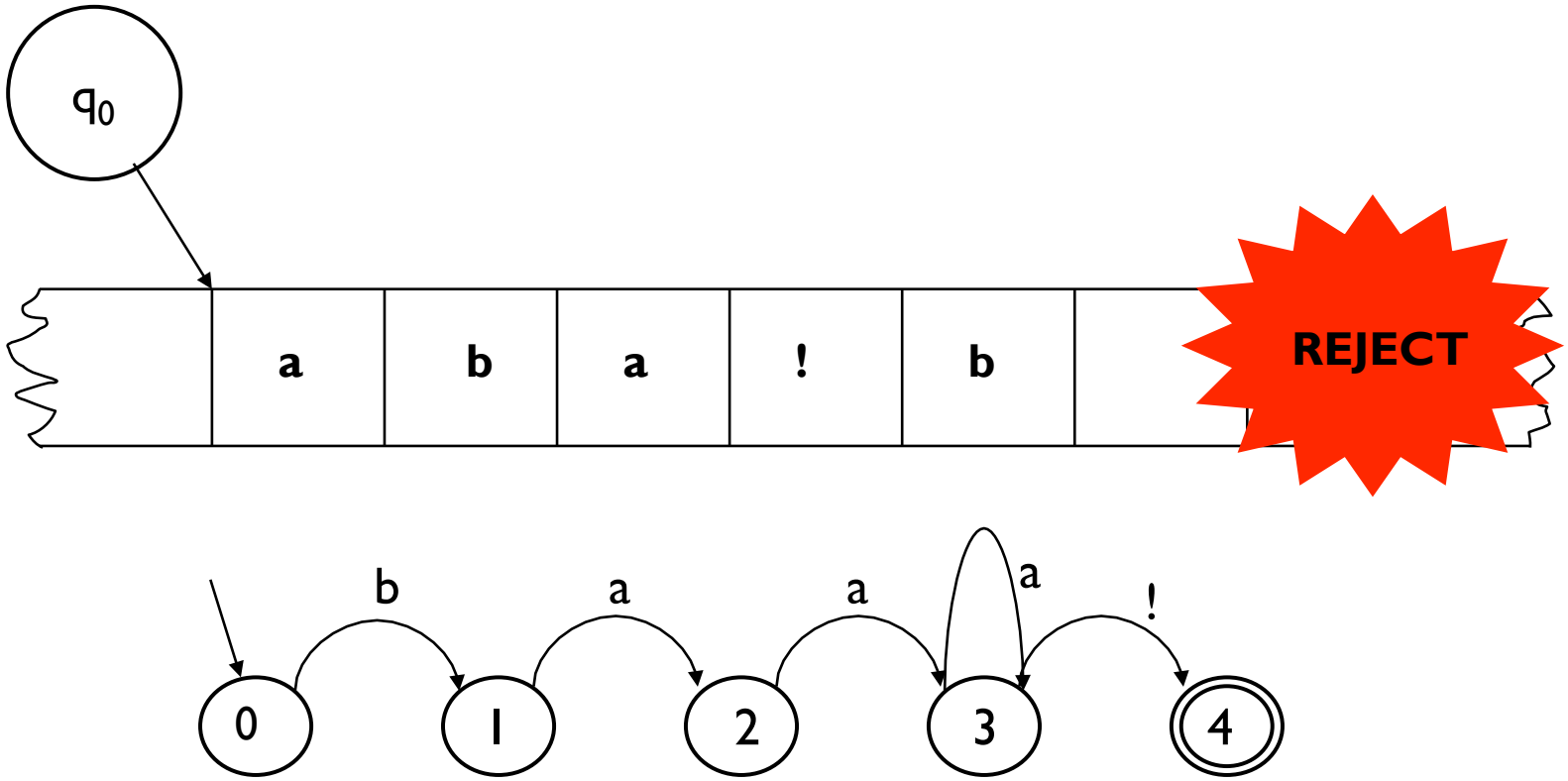


baa!
baaa!
baaaa!
baaaaa!
...

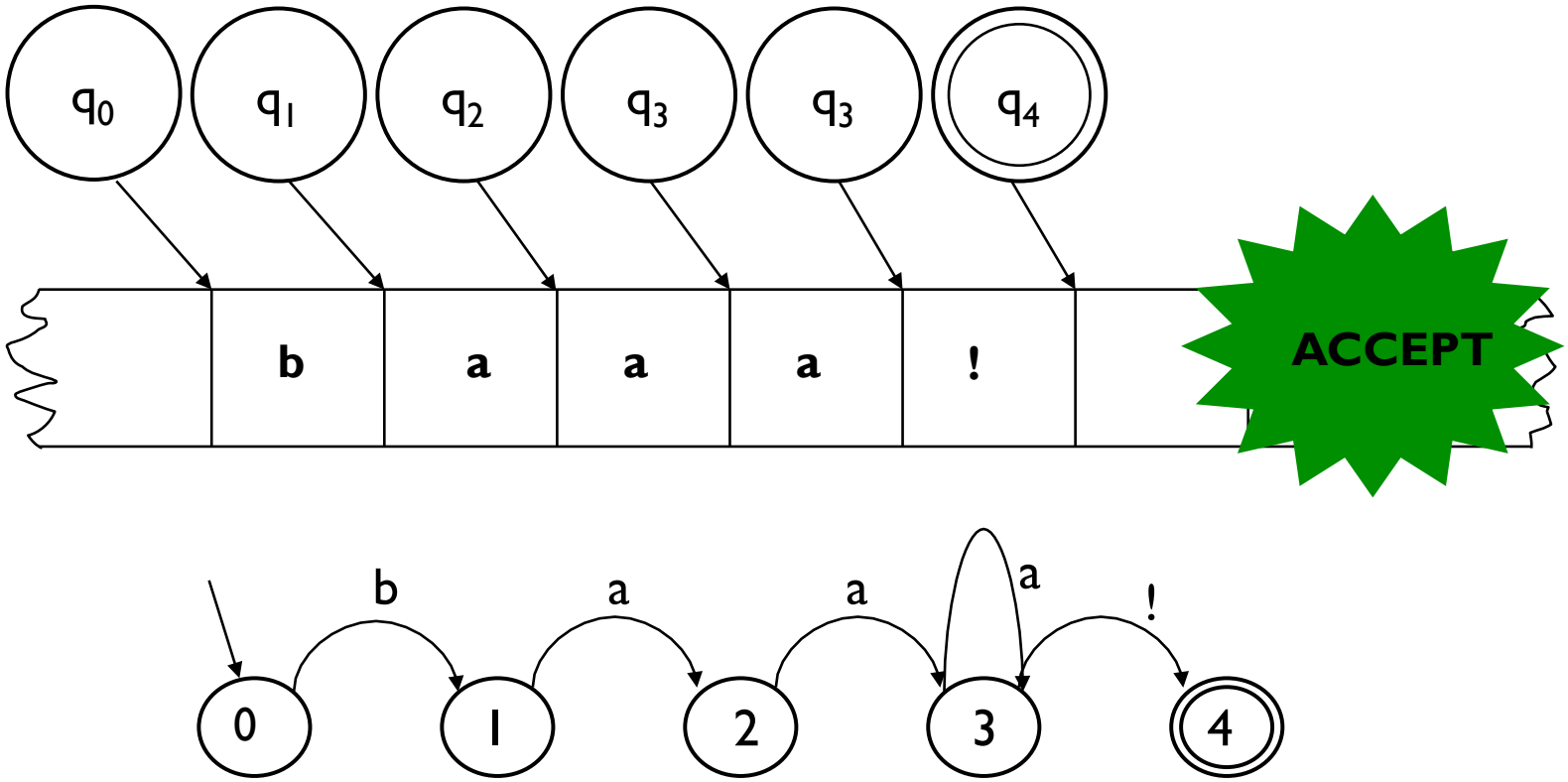
/baa+!/



Input Tape



Input Tape

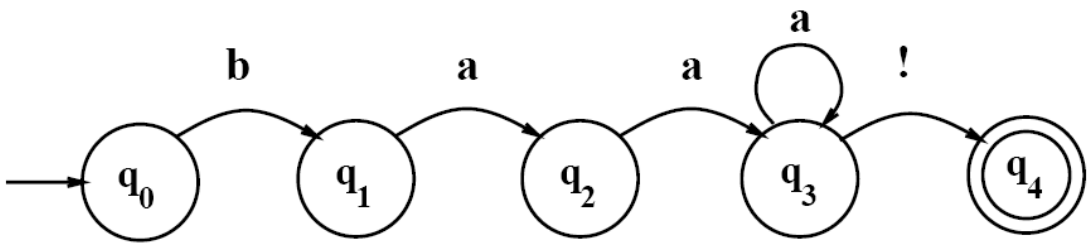


Finite-state Automata

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

State-transition table

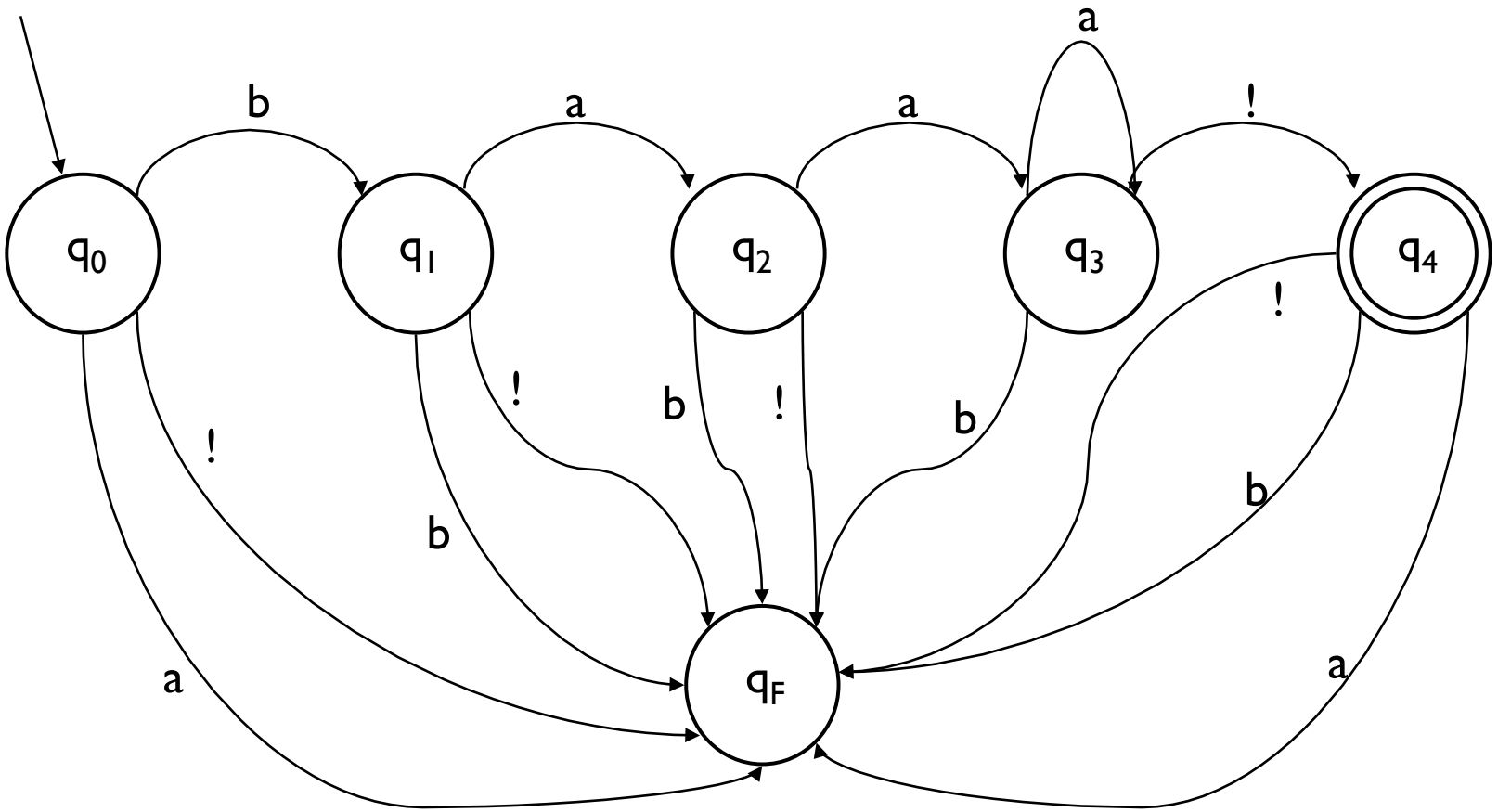
	Input		
State	b	a	!
0	1	ϕ	ϕ
1	ϕ	2	ϕ
2	ϕ	3	ϕ
3	ϕ	3	4
4:	ϕ	ϕ	ϕ



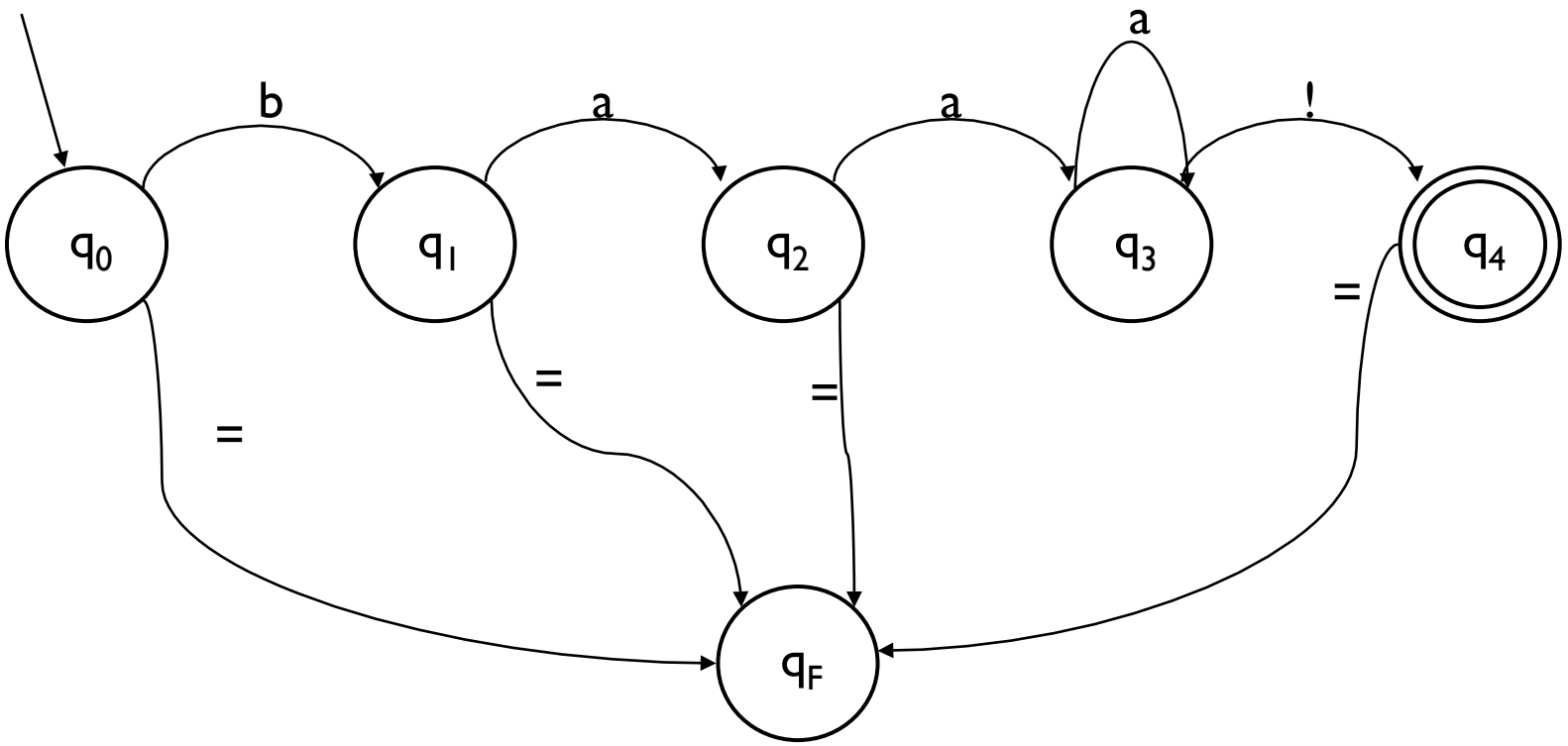
D-RECOGNIZE

```
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
```

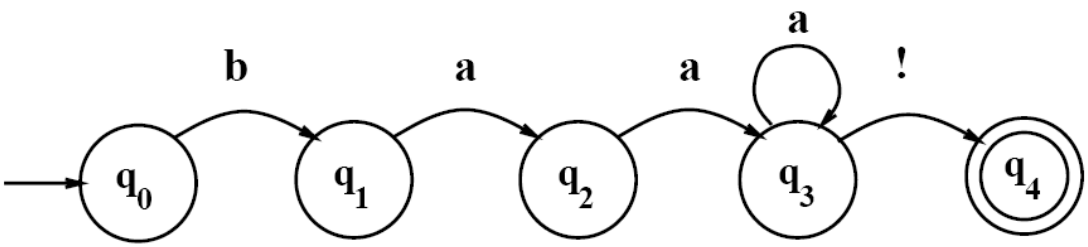
Adding a failing state



Adding an "all else" arc

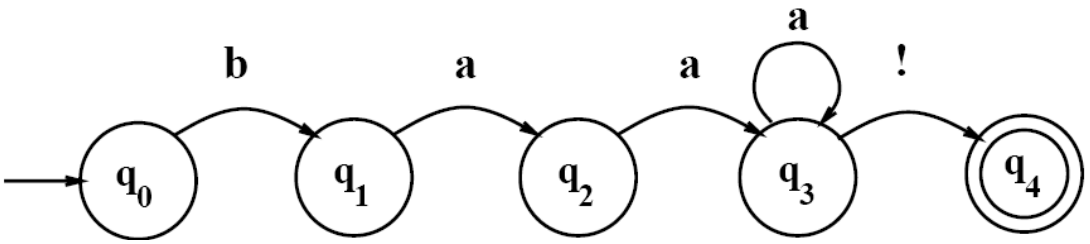


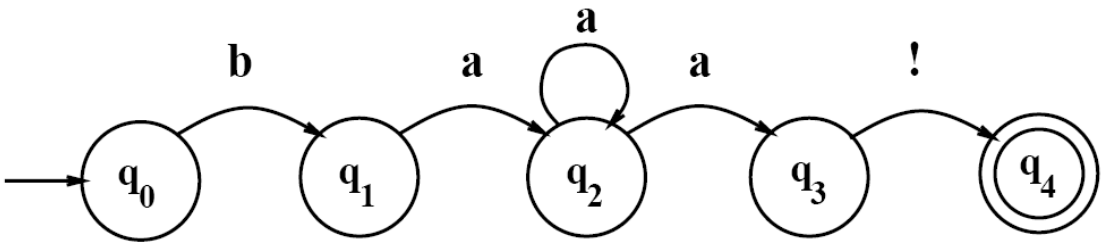
Recognize or generate



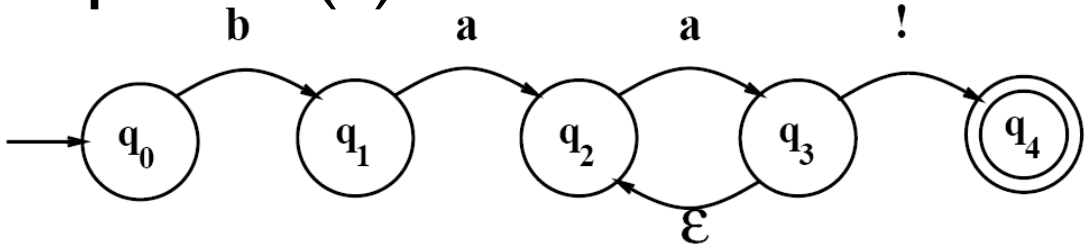
Languages and Automata

- Deterministic vs. Non-deterministic FSAs





- ★Epsilon (ε) transitions

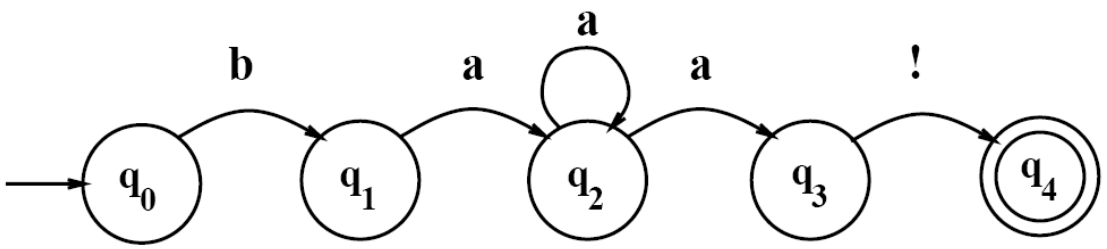


Using NFSAs to accept strings

- Backup: add markers at choice points, then possibly revisit unexplored arcs at marked choice point.
- Look-ahead: look ahead in input
- Parallelism: look at alternatives in parallel

Using NFSAs

	Input			
State	b	a	!	ε
0	1	ϕ	ϕ	ϕ
1	ϕ	2	ϕ	ϕ
2	ϕ	2,3	ϕ	ϕ
3	ϕ	ϕ	4	ϕ
④	ϕ	ϕ	ϕ	ϕ



More about NFSAs

- Equivalence of DFSAs and NFSAs
  - For every NFSA, there is an equivalent DFSA.
- Recognition is a search

Recognition using NFSAs

```
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
```

Recognition using NFSAs

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

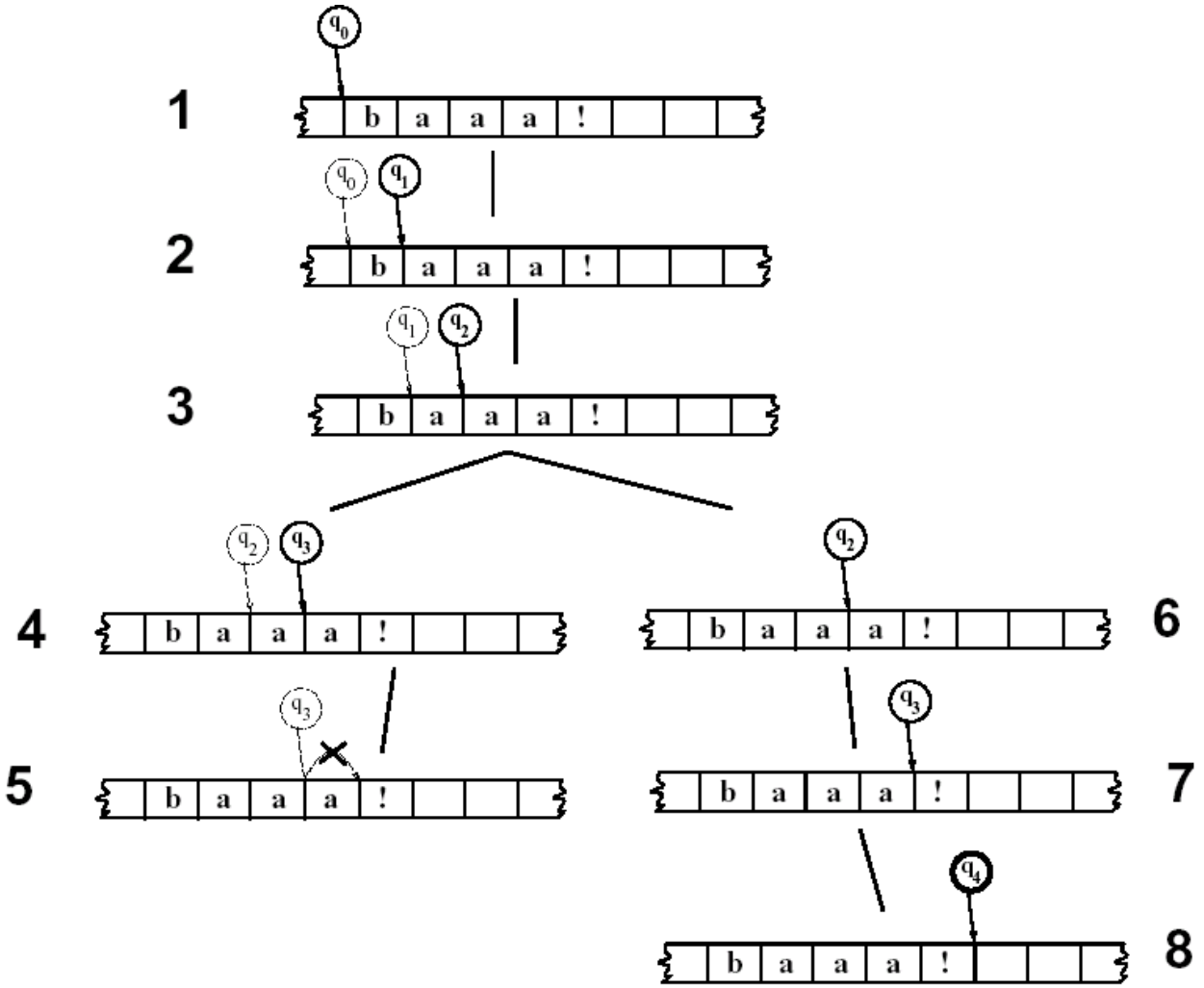
current-node ← the node the current search-state is in
index ← the point on the tape the current search-state is looking at
return a list of search states from transition table as follows:
(*transition-table[current-node,ε], index*)
∪
(*transition-table[current-node, tape[index]], index + 1*)

Recognition using NFSAs

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

NFSA Recognition of "baaa!"



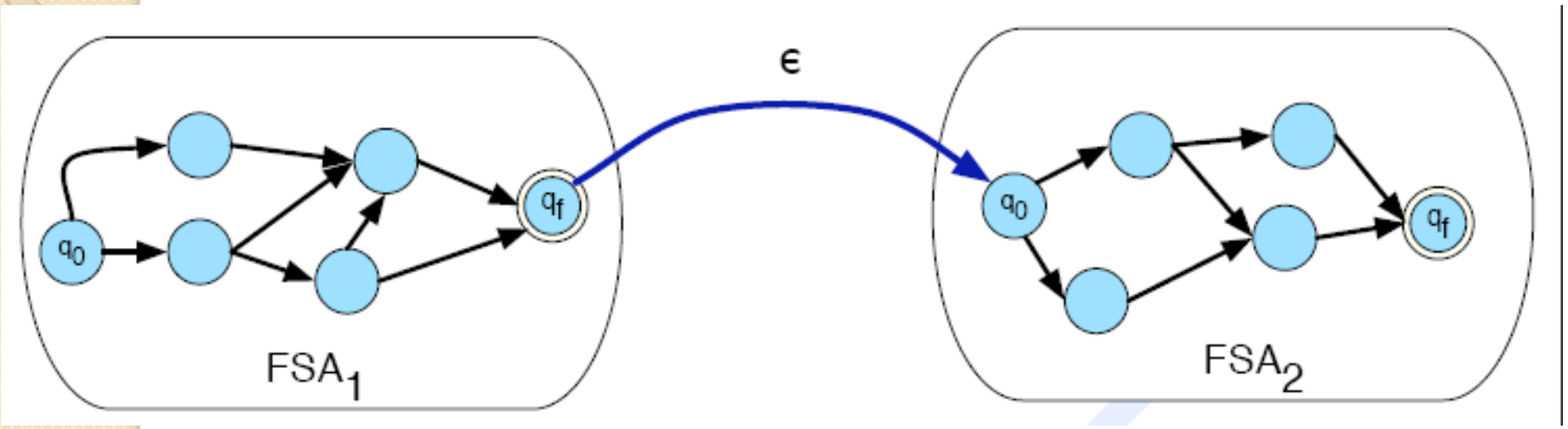
Regular language

- ϕ is a regular language
- $\forall a \in \Sigma \cup \varepsilon$, {a} is a regular language
- If L1 and L2 are regular languages, then so are:
  - $L1 \cdot L2 = \{xy \mid x \in L1, y \in L2\}$, the concatenation of L1 and L2
  - $L1 \cup L2$, the union or disjunction of L1 and L2
  - $L1^*$, the Kleene closure of L1

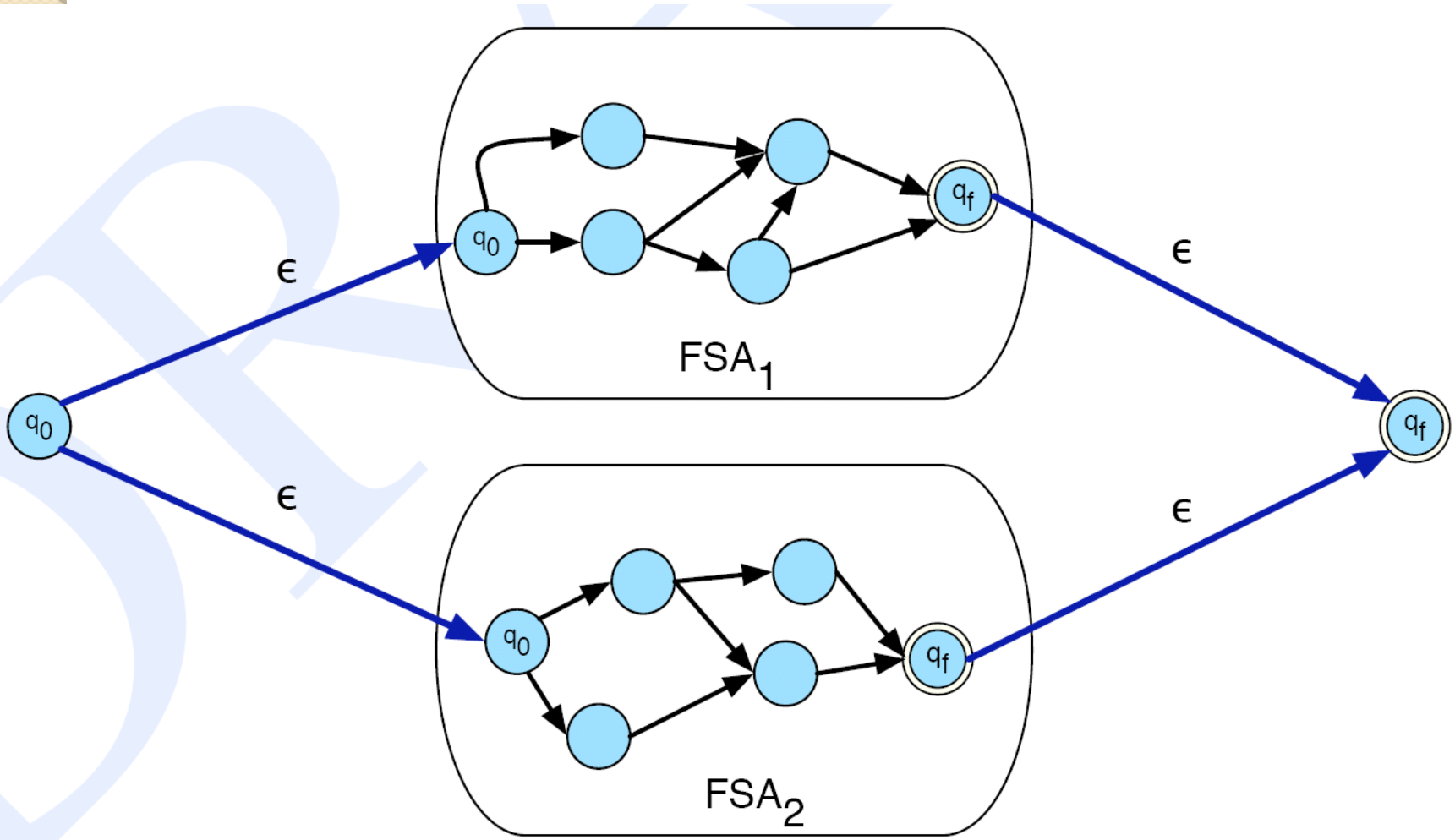
Regular languages

- Regular languages are characterized by FSAs
- Regular languages are closed under concatenation, Kleene closure, union.

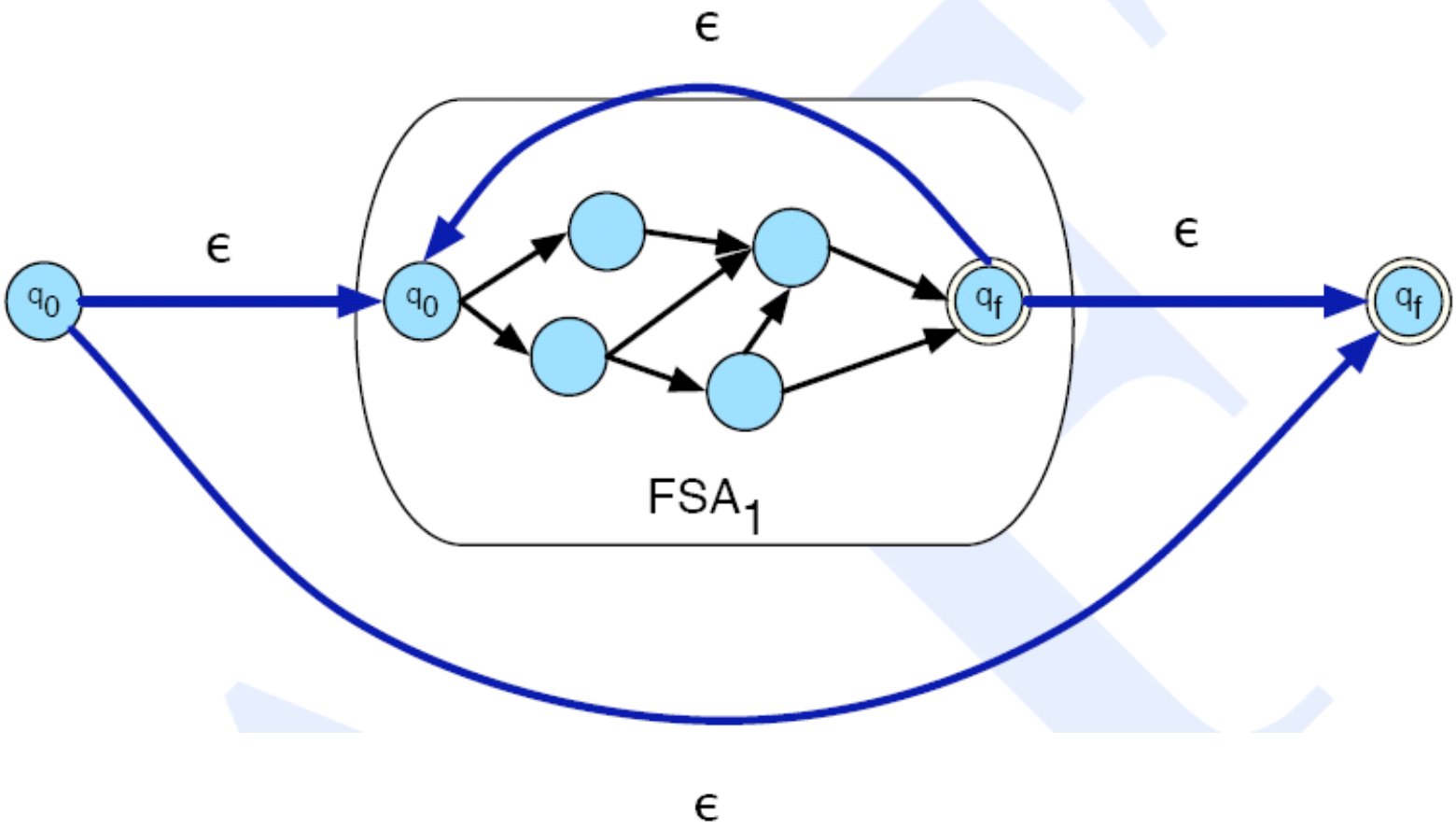
Concatenation



Union

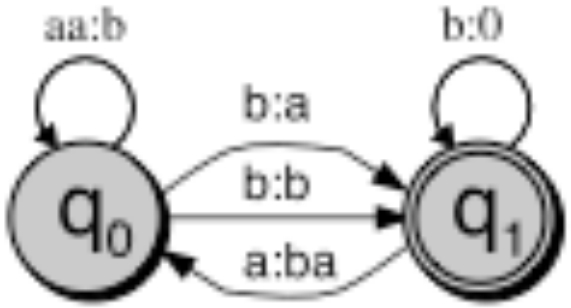


Kleene Closure



Finite state transducers

- An automaton that maps between two sets of symbols
- 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

Digression: Composition (for FSTs)

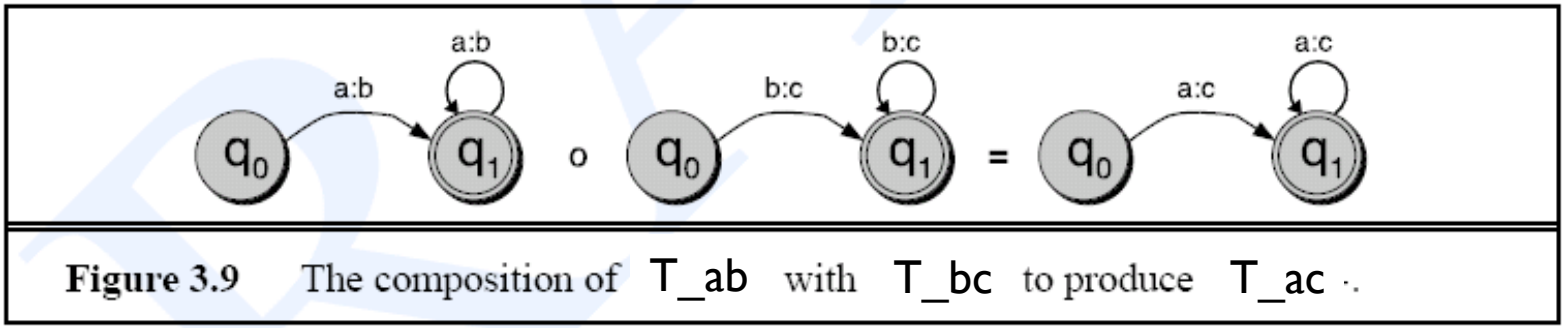


Figure 3.9 The composition of T_ab with T_bc to produce T_ac .

In assignment 1: You will work with FST composition

Similar to concatenation of FSA's, but two internal conversions compiled into more efficient single transition a:c

FST's have input/output pair on arcs instead of a single read-only input

Compose: Performs transitive closure to get single arc with input symbol of first automaton and output symbol of last automaton

Composition Example

```
from soundexutils import compose
output = compose(S, f1, f2, f3)
```

The above function call computes `f1` ∘ `f2` ∘ `f3`. Obviously, it will raise an error if one or more of the input transducers produce no output.

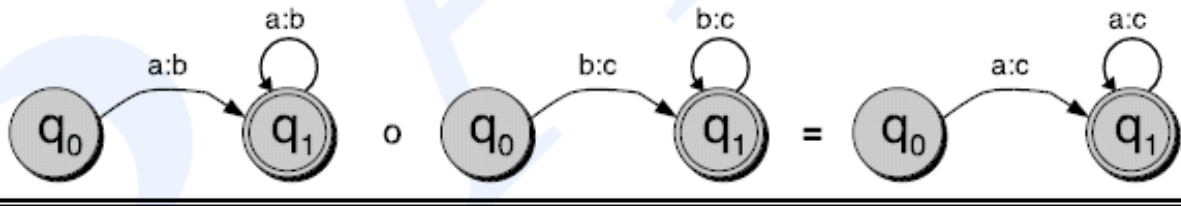


Figure 3.9 The composition of T_ab with T_bc to produce T_ac.

```
from soundexutils import compose
output = compose('a', T_ab, T_bc)
    ➔'c'
```

Morphology

- Definitions and Problems
  - What is Morphology?
  - Topology of Morphologies
- Approaches to Computational Morphology
  - Lexicons and Rules
  - Computational Morphology Approaches

Morphology

- Study of the way words are built up from smaller meaning bearing units of language
- Smallest meaning bearing units are called morphemes

  fox has morpheme *fox*

  cats has two morphemes *cat* and *–s*
- Two classes of morphemes:
  - Stems: supplies the main meaning
  - Affixes: add additional meaning

Concatenative morphology

- *Morpheme+Morpheme+Morpheme+…*
- Stems: also called lemma, base form, root, lexeme
  - hope+ing → hoping hop → hopping
- Affixes
  - Prefixes: Antidisestablishmentarianism
  - Suffixes: Antidisestablishmentarianism
- Agglutinative Languages
  - uygarlaştıramadıklarımızdanmışsınızcasına
  - uygar+laş+tır+ama+dık+lar+ımız+dan+mış+sınız+casına
  - *Behaving as if you are among those whom we could not cause to become civilized*

Non-concatenative morphology

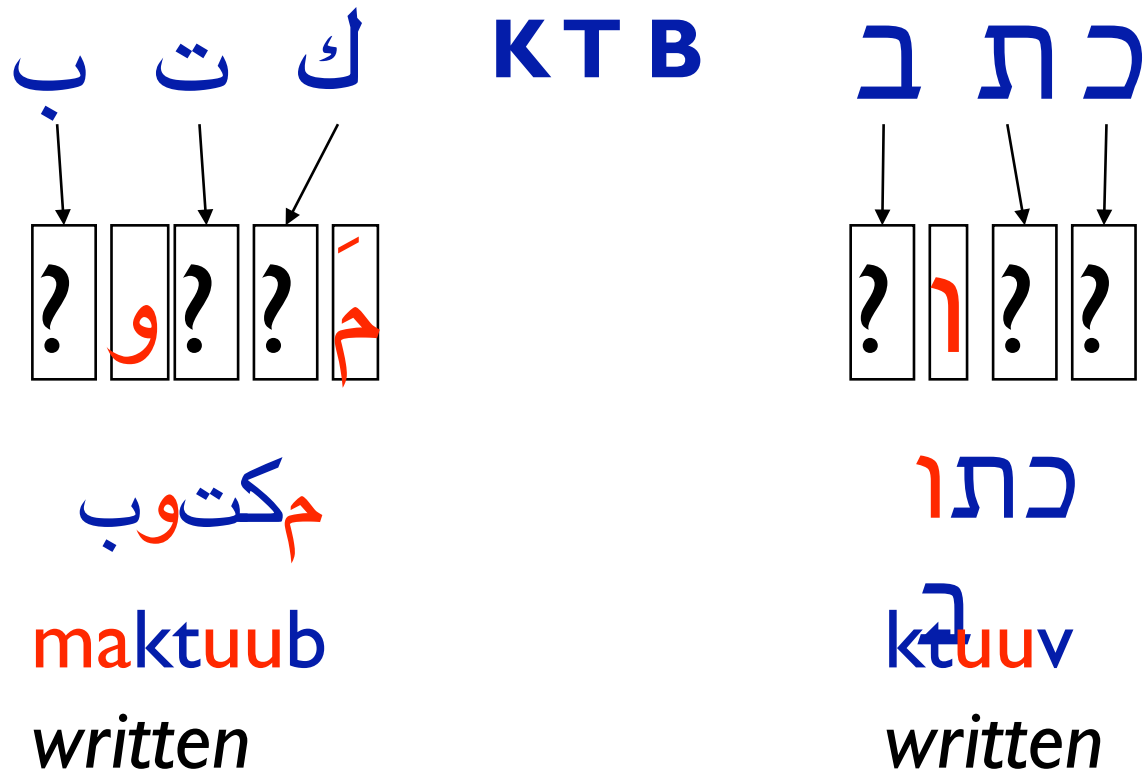
- Affixes continued:
  - Infixes: hingi (*borrow*) – humingi (*borrower*) in Tagalog
  - Circumfixes: sagen (*say*) – gesagt (*said*) in German

Topology of morphologies

- Concatenative *vs.* non-concatenative (infix, circumfix, templatic)
- Derivational *vs.* inflectional
- Regular *vs.* irregular

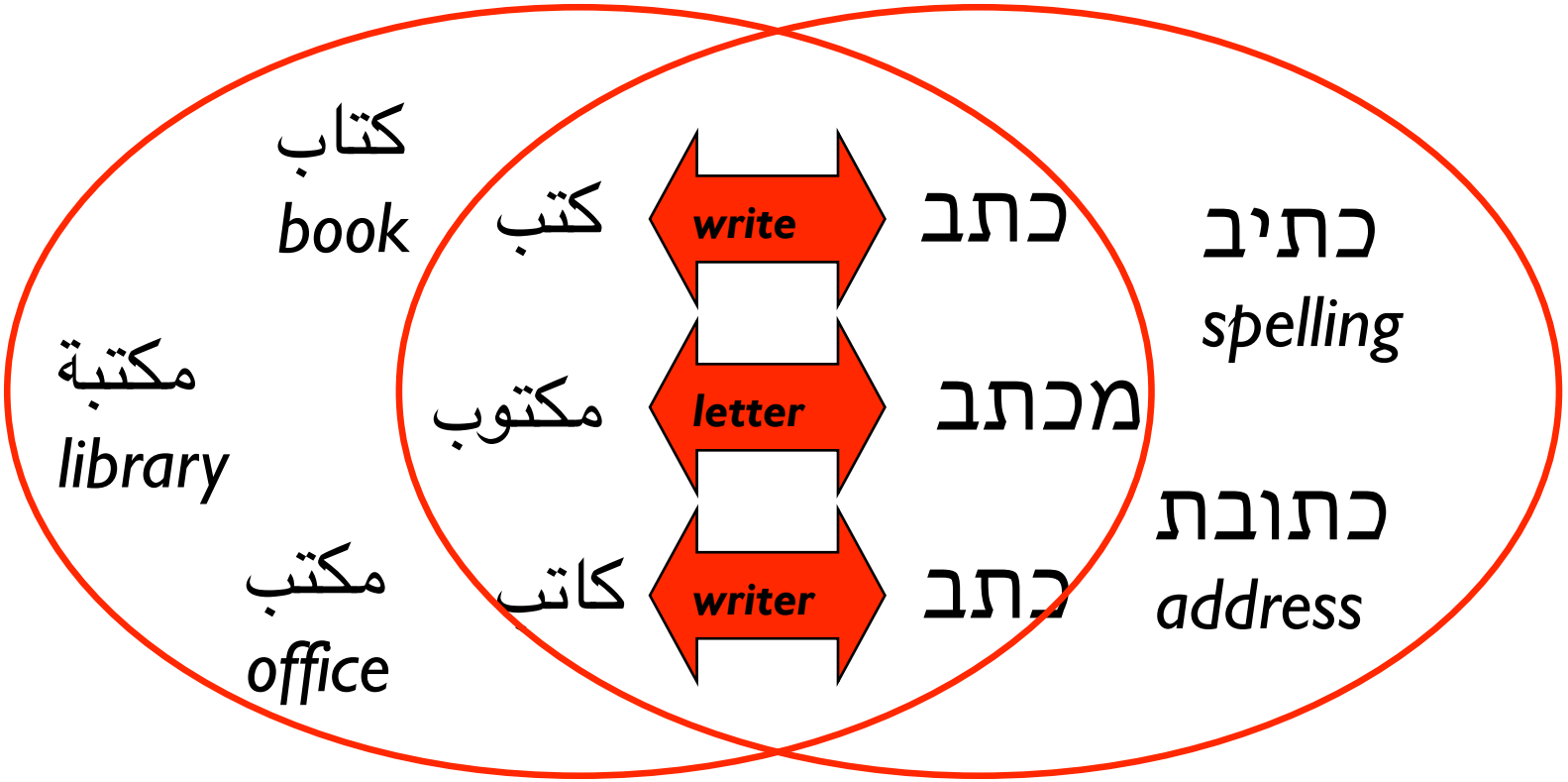
Templatic Morphology

- Roots and Patterns



Templatic Morphology: Root Meaning

- KTB: *writing* "stuff"



Derivational morphology

- Stem + morpheme -> word with part of speech different from the stem
- Nominalization: computerization, appointee, killer, fuzziness
- Formation of adjectives: computational, clueless, embraceable
- CatVar: Categorial Variation Database

  http://clipdemos.umiacs.umd.edu/catvar/

Inflectional morphology

- Stem + morpheme -> word with same part of speech as the stem
- Adds: tense, number, person, mood, aspect
- Five verb forms in English
- Other languages have (lots more)

Nouns and verbs (in English)

- Nouns have simple inflectional morphology
  - cat
  - cat+s, cat+'s
- Verbs have more complex morphology

Regulars and Irregulars

- Nouns
  - Cat/Cats
  - Mouse/Mice, Ox, Oxen, Goose, Geese
- Verbs
  - Walk/Walked
  - Go/Went, Fly/Flew

Regular (English) Verbs

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

Irregular (English) Verbs

Morphological Form Classes	**Irregularly Inflected Verbs**		
Stem	eat	catch	cut
-s form	eats	catches	cuts
-ing form	eating	catching	cutting
Past form	ate	caught	cut
-ed participle	eaten	caught	cut

“To love” in Spanish

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

Morphological parsing

Recognizing that a word (like *foxes*) breaks down into component morphemes (*fox* and *-es*) and building a structured representation.

Examples: morphological parsing

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)

Building a morphological parser

- Approaches
  - lexicon only
  - lexicon and rules
    - finite-state automata
    - finite-state transducers
  - rules only

Lexicon-only Morphology

- The lexicon lists all surface level and lexical level pairs
- Analysis/generation easy
- Very large for English
- What about Arabic or Turkish?

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 $V+ing$

Building a Morphological Parser

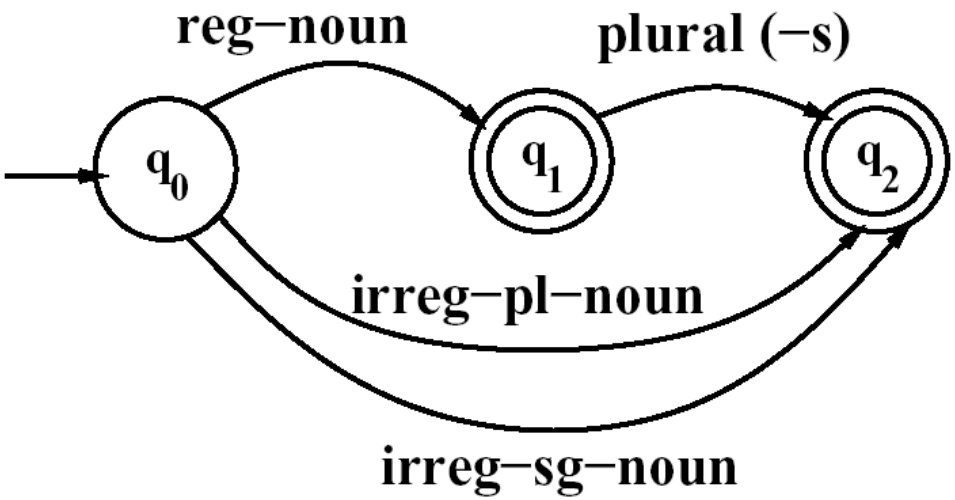
- Approaches
  - Lexicon only
  - Lexicon and rules
    - Finite-state automata
    - Finite-state transducers
  - Rules only

Lexicon and Rules: FSA Inflectional Noun Morphology

- English noun lexicon

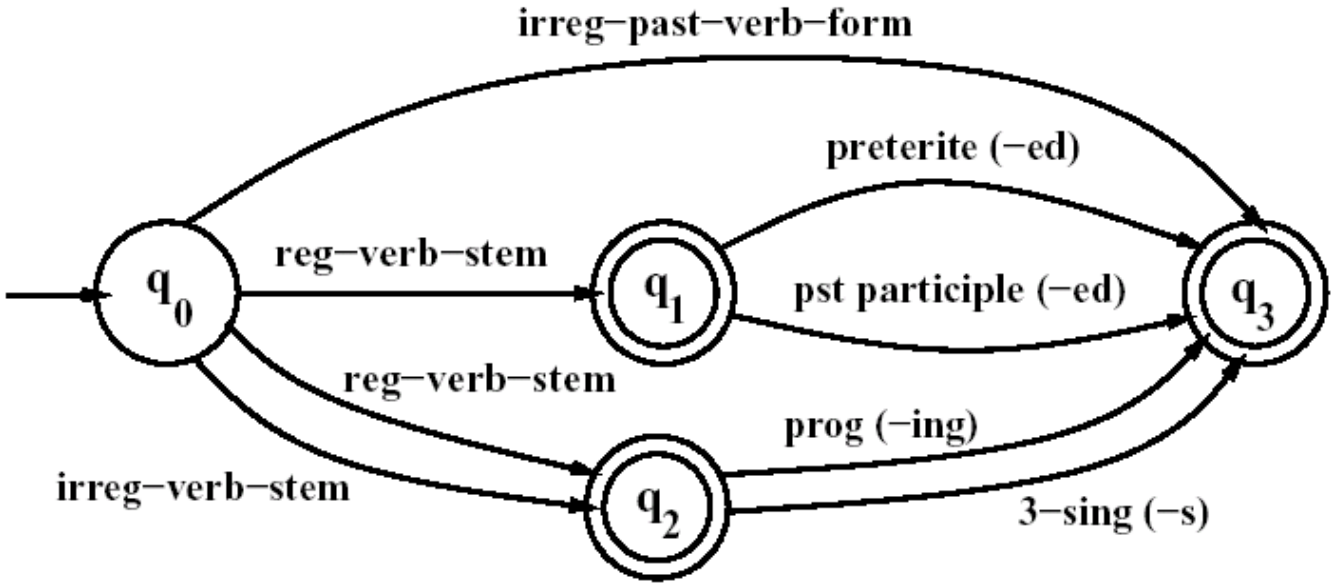
reg-noun	Irreg-pl-noun	Irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	

- English noun rule

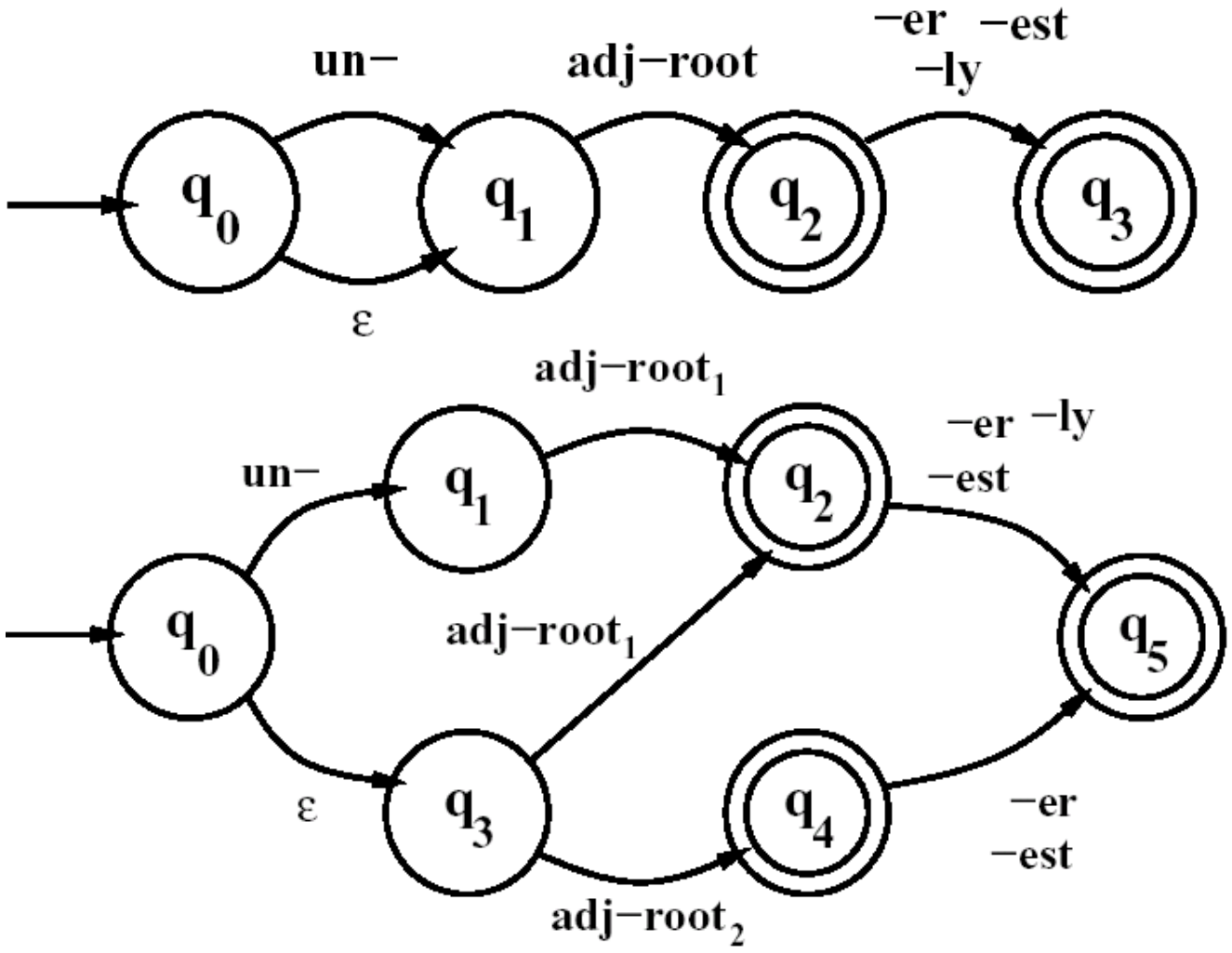


Lexicon and Rules: FSA English Verb Inflectional Morphology

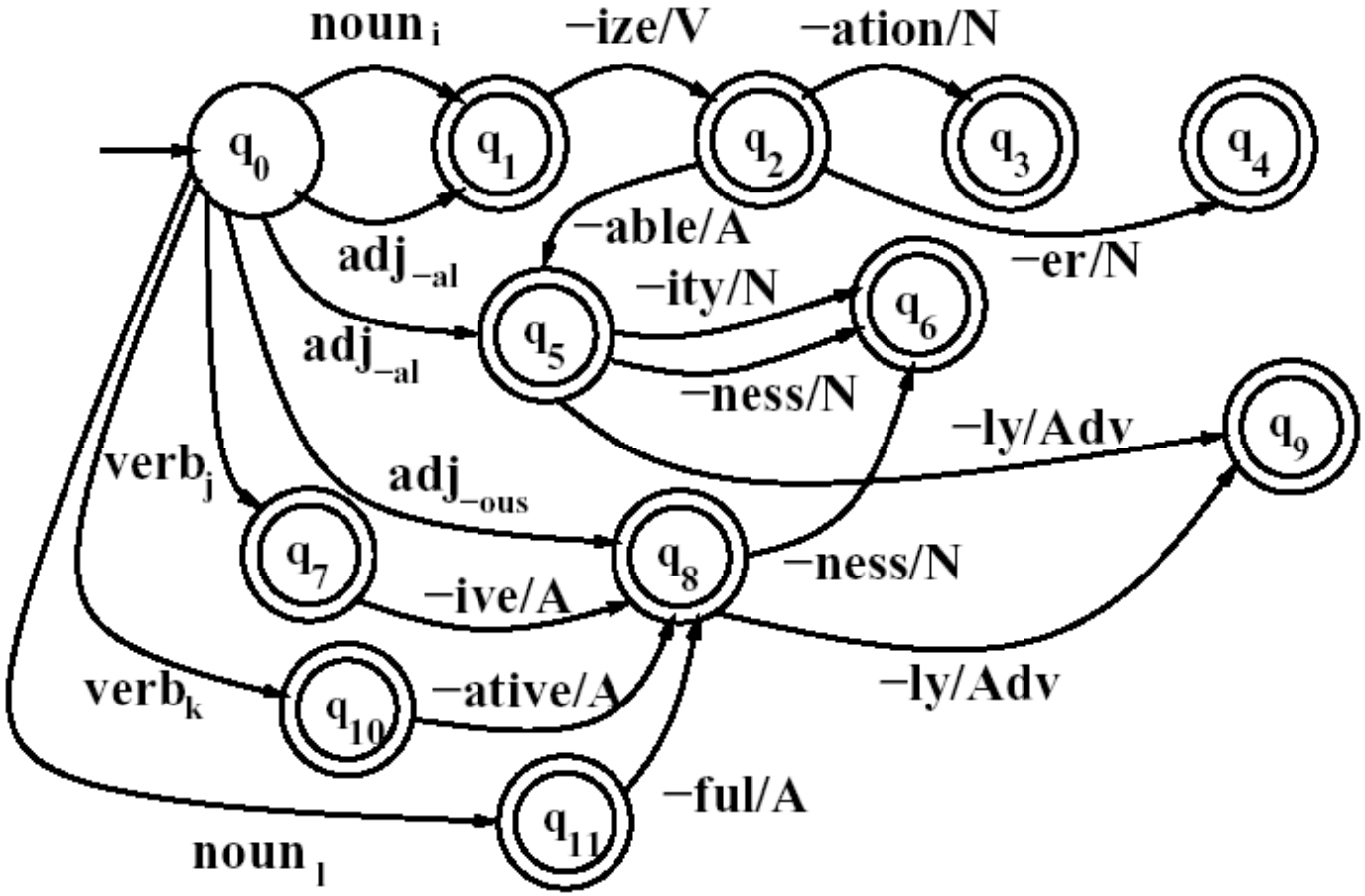
reg-verb-stem	irreg-verb-stem	irreg-past-verb	past	past-part	pres-part	3sg
walk fry talk impeach	cut speak spoken sing sang	caught ate eaten	-ed	-ed	-ing	-s



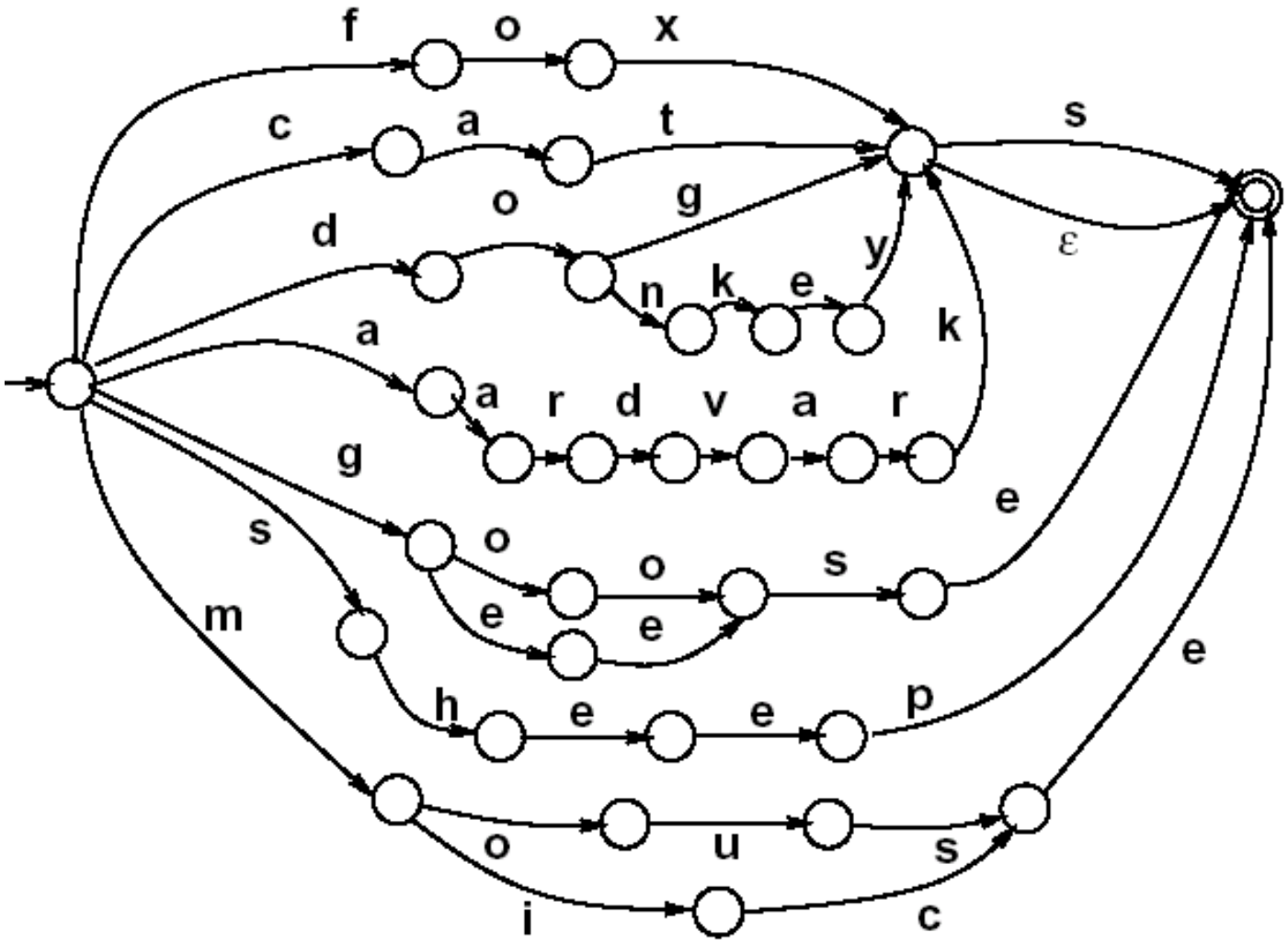
FSA for Derivational Morphology: Adjectival Formation



More Complex Derivational Morphology



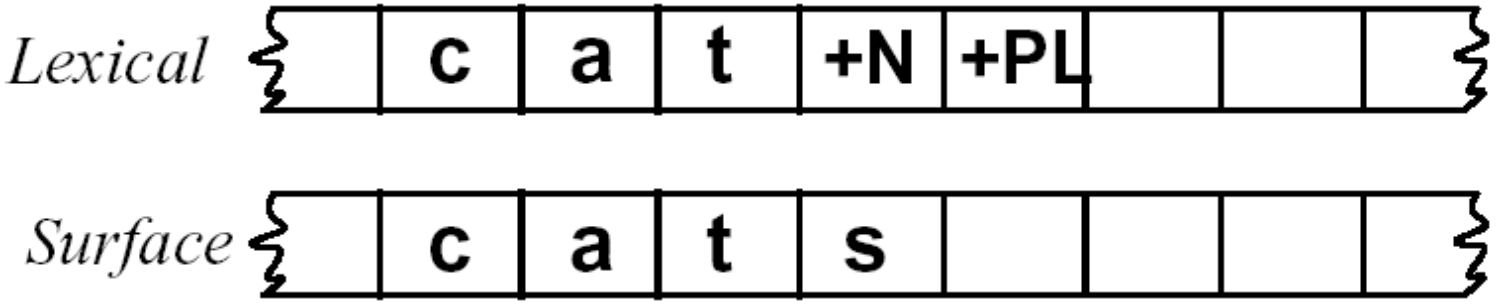
Using FSAs for Recognition: English Nouns and their Inflection



Morphological Parsing

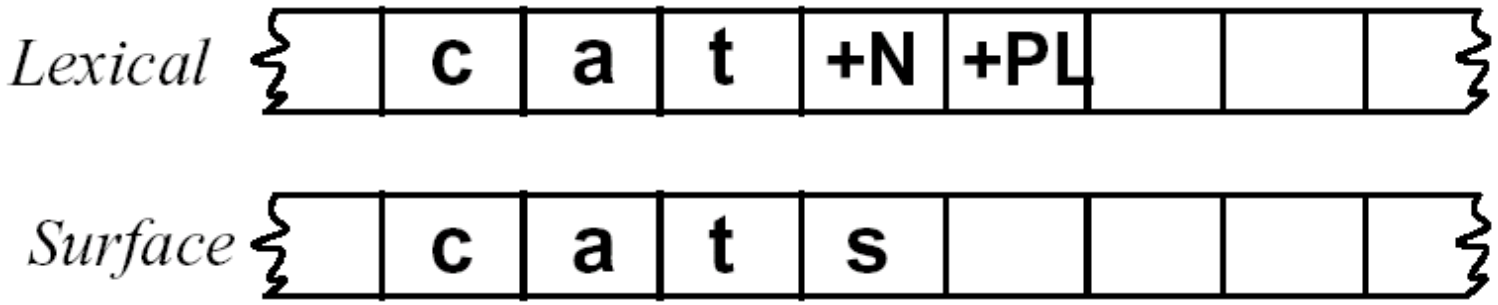
- Finite-state automata (FSA)
  - Recognizer

- Finite-state transducers (FST)
  - input-output pair

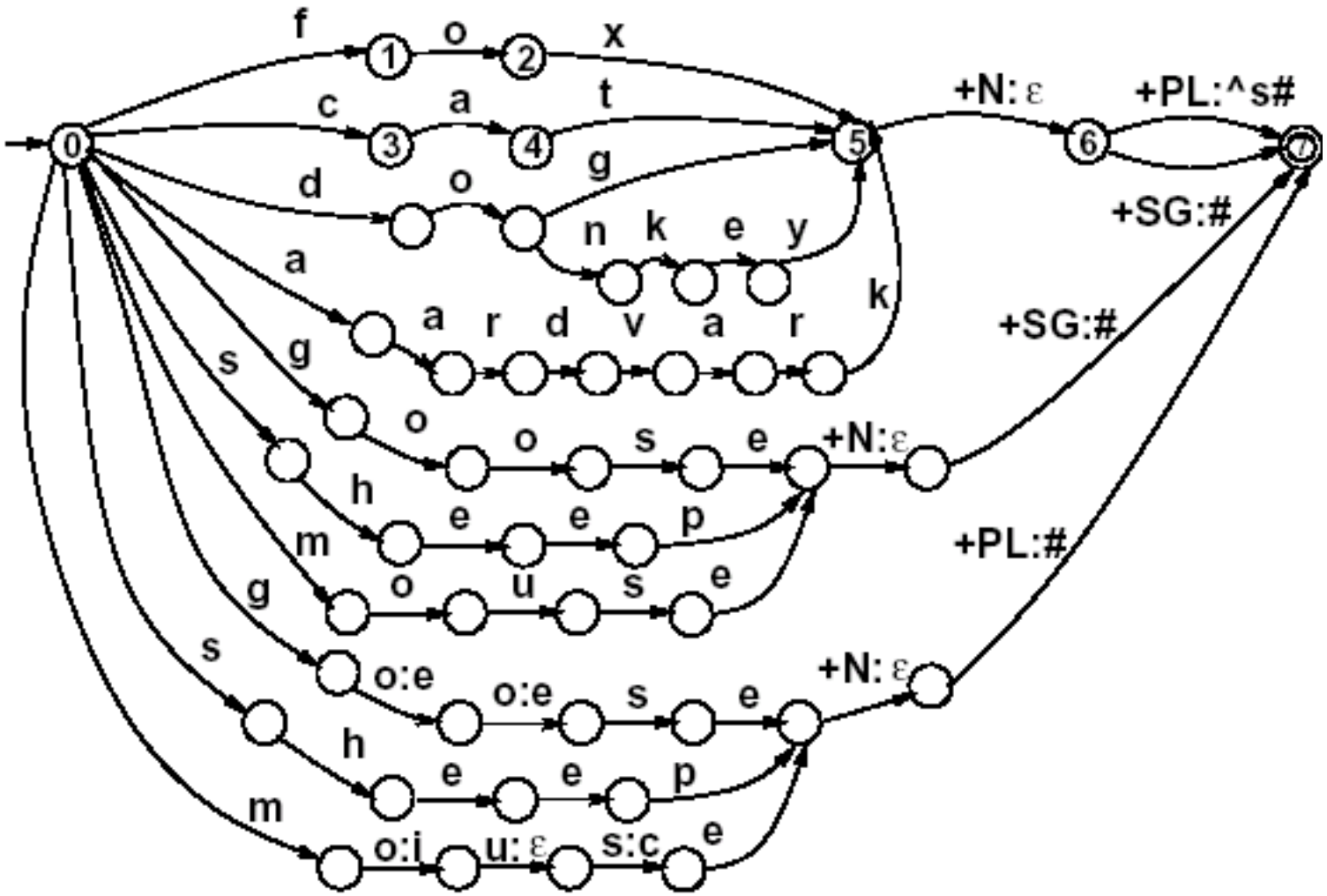


Terminology

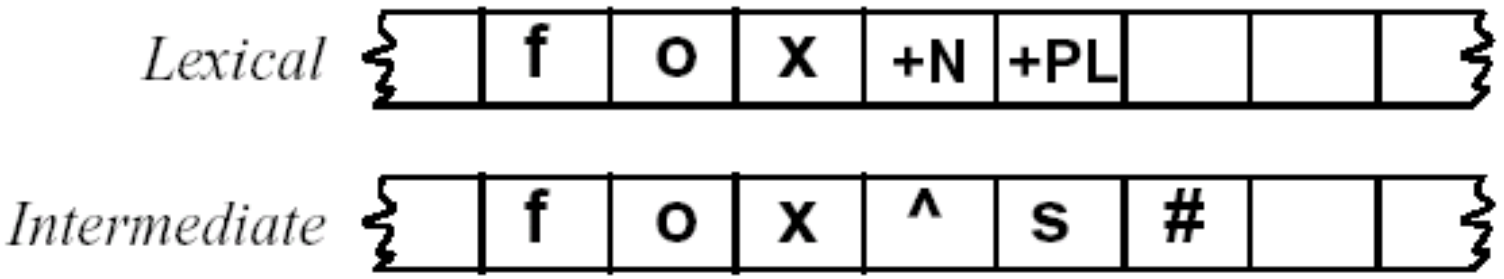
- Characters correspond to pairs, written a:b
- If "a:a", write "a" for shorthand
- # = word boundary
- ^ = morpheme boundary
- Other = "any feasible pair that is not in this transducer"


Lexical
c a t +N +PL
Surface
c a t s

Nominal Inflection FST

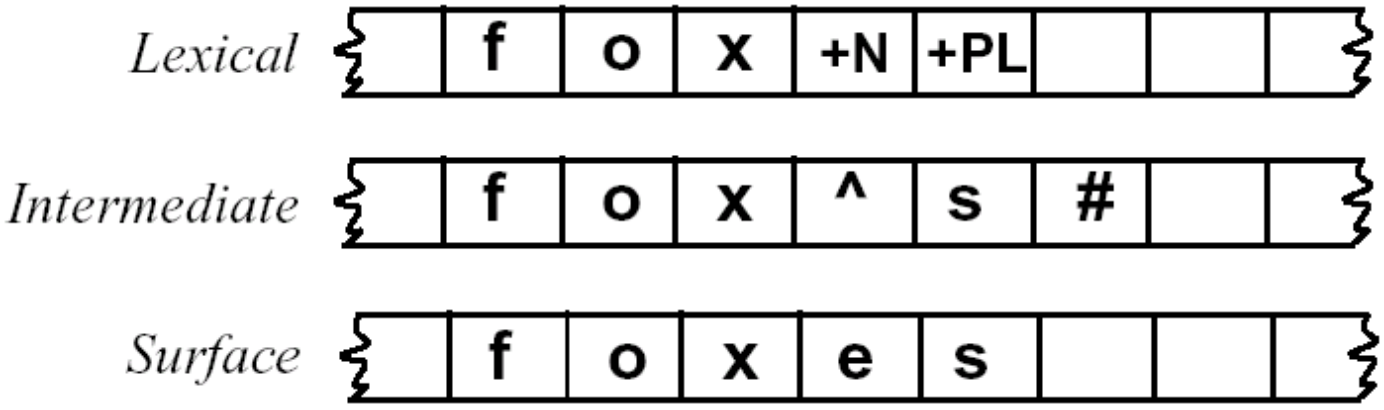


Lexical and Intermediate Tapes

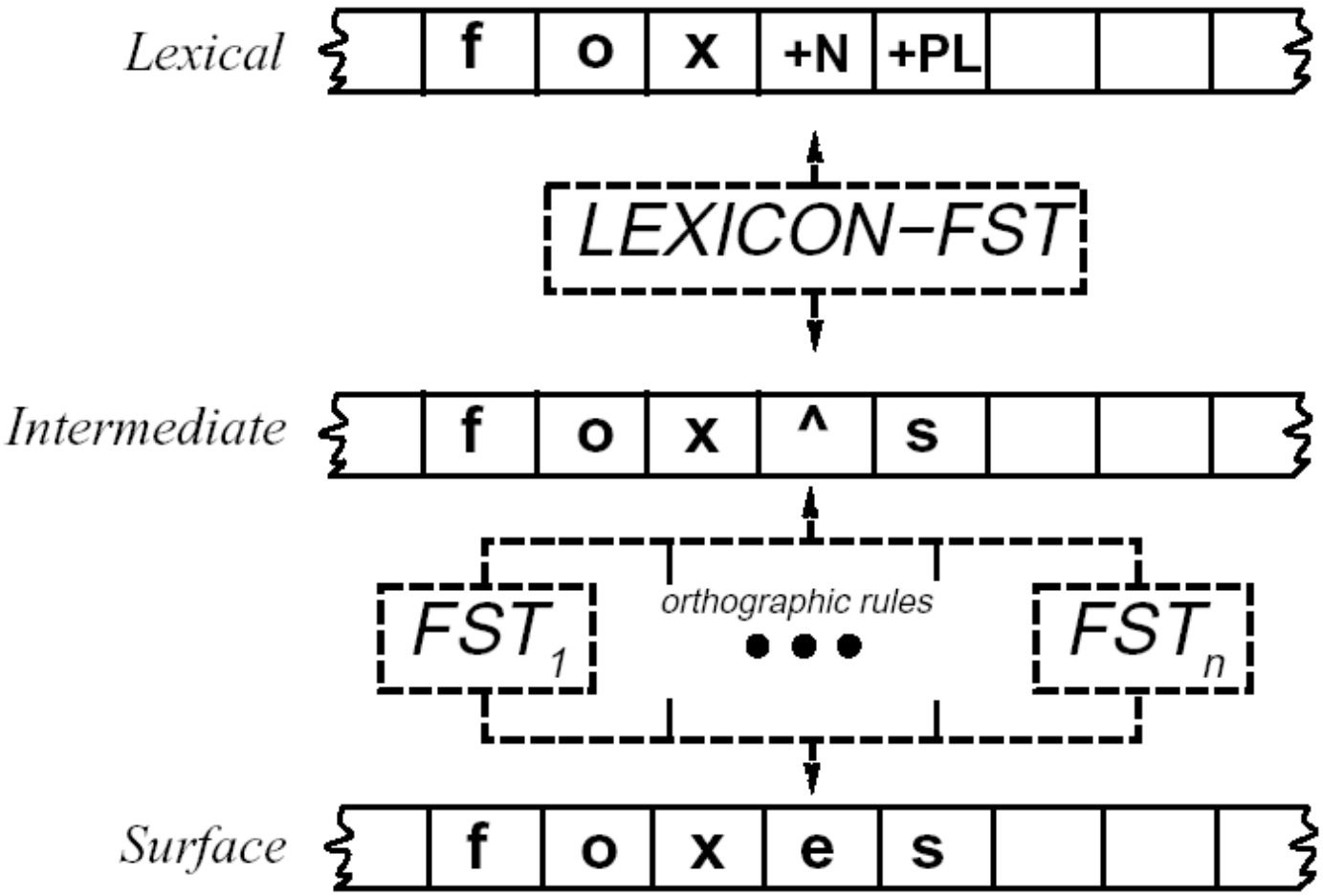


Spelling Rules

Name	Rule Description	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



Two-Level Morphology



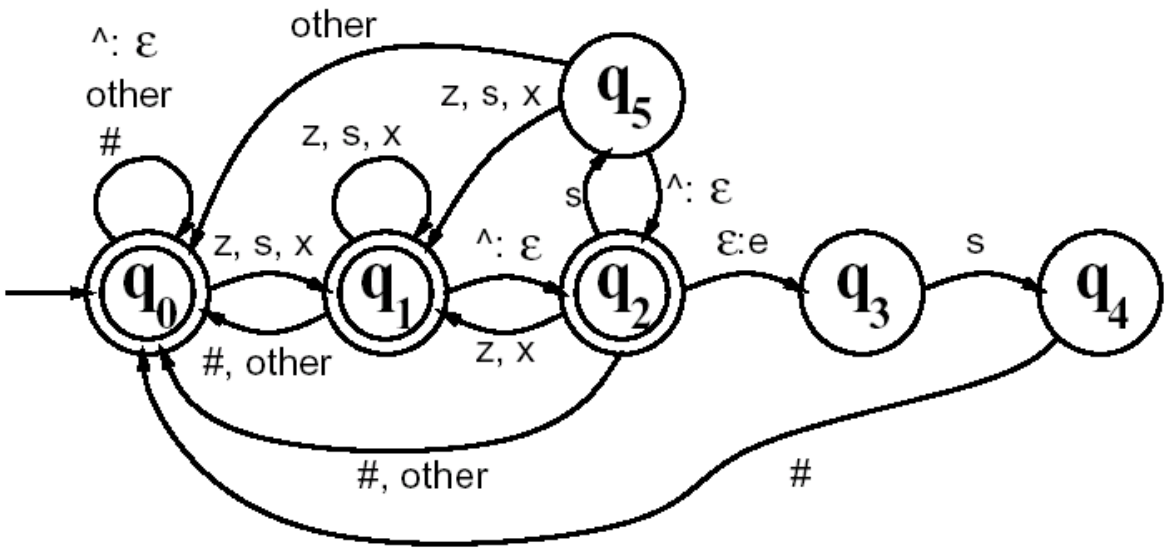
Chomsky and Halle Notation

- Rules to move from intermediate level to surface level
- a → b / c__d
  - Rewrite a as b when it occurs between c and d

Chomsky and Halle Notation

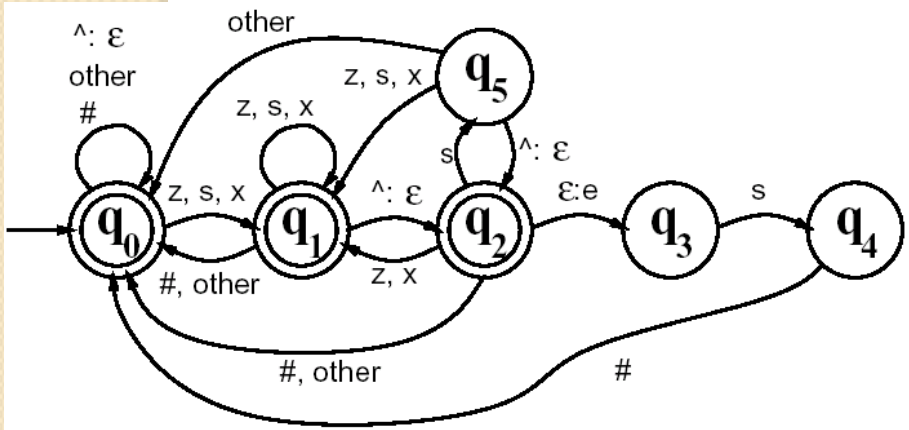
$$\varepsilon \rightarrow e \,/ \left\{ \begin{array}{c} x \\ s \\ z \end{array} \right\} \text{ ^ \_\_ s \#}$$

Intermediate-to-Surface Transducer



State Transition Table

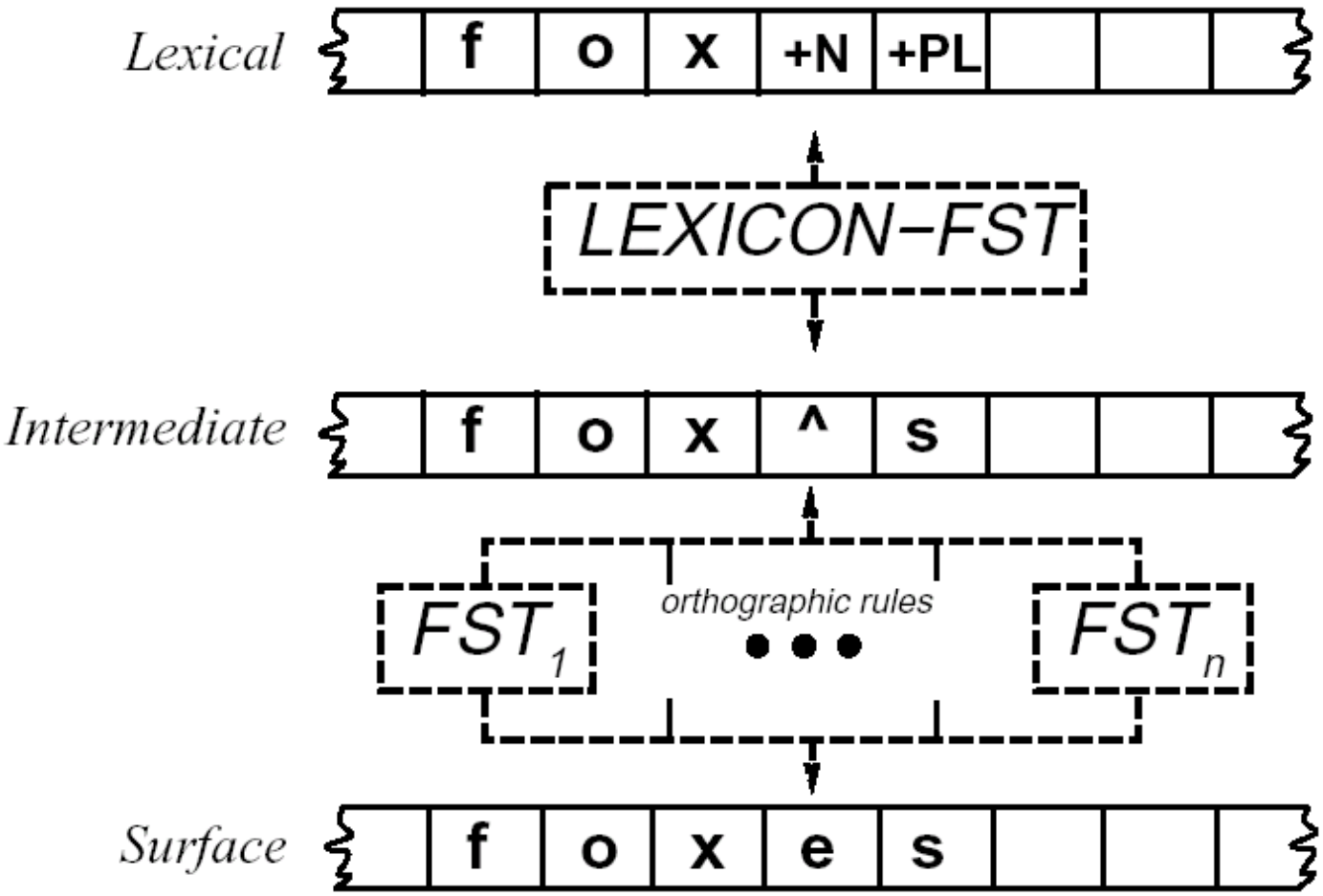
State\Input	s : s	x : x	z : z	^:ε	ε : e	#	other
q_0:	1	1	1	0	-	0	0
q_1:	1	1	1	2	-	0	0
q_2:	5	1	1	0	3	0	0
q_3	4	-	-	-	-	-	-
q_4	-	-	-	-	-	0	-
q_5	1	1	1	2	-	-	0



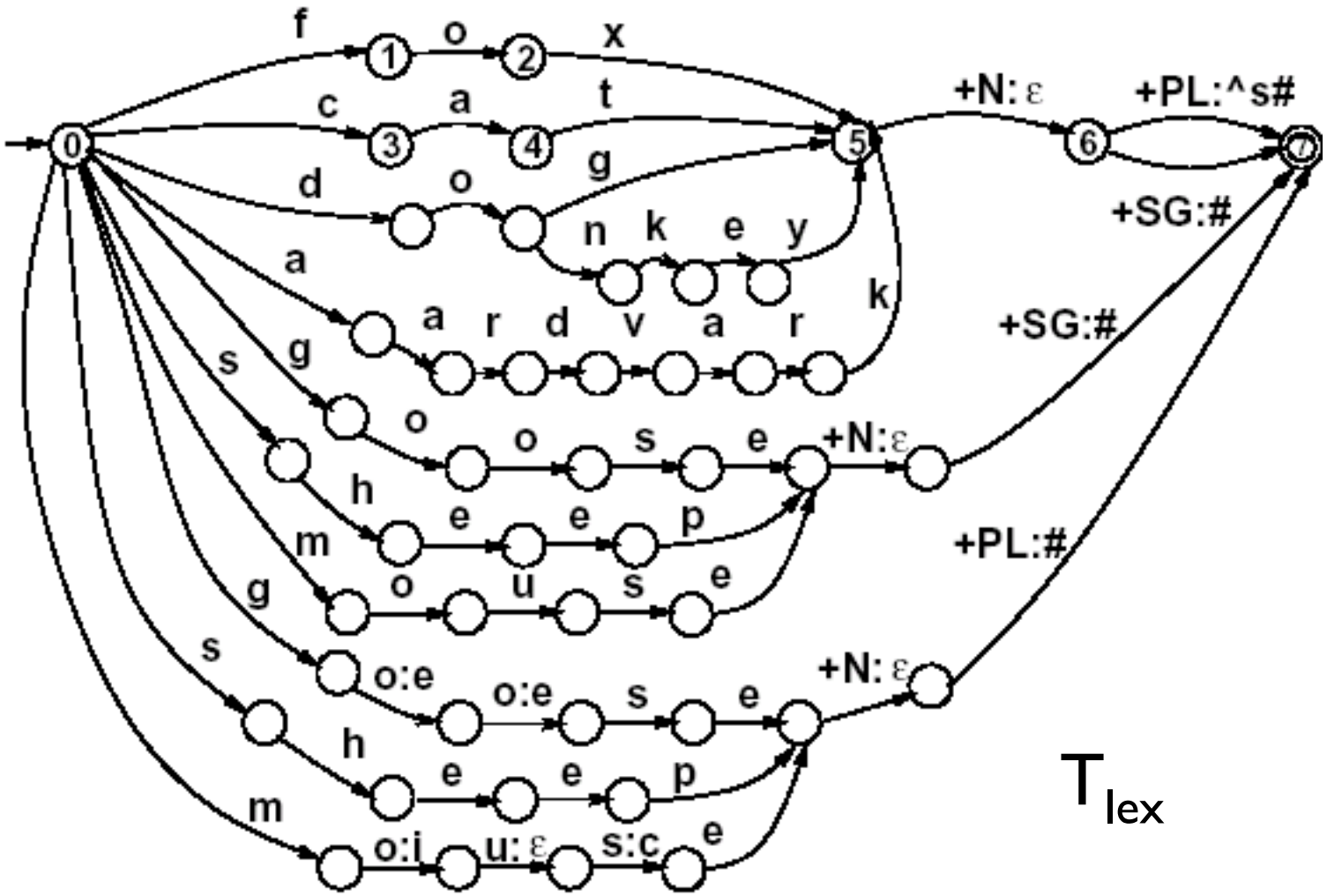
How would we represent this in NLTK?

```
from nltk_contrib.fst import fst
f = fst.FST('pluralize')
f.add_state('q0')
f.initial_state = 'q0'
f.set_final('q0')
f.add_state('q1')
f.add_arc('q0', 'q1', 'z', 'z')
```

Two-Level Morphology

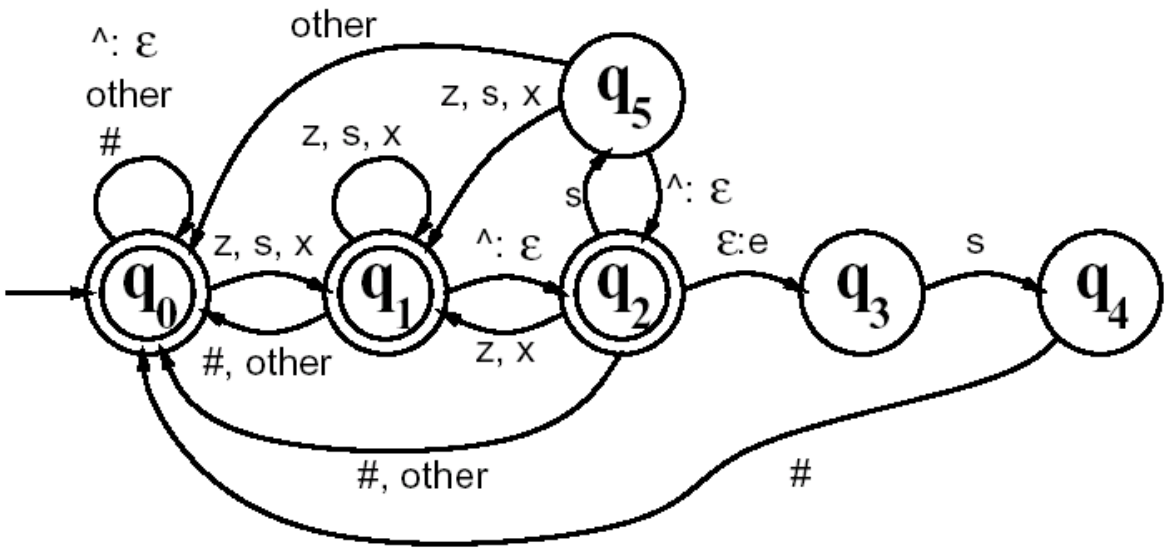


Nominal Inflection FST



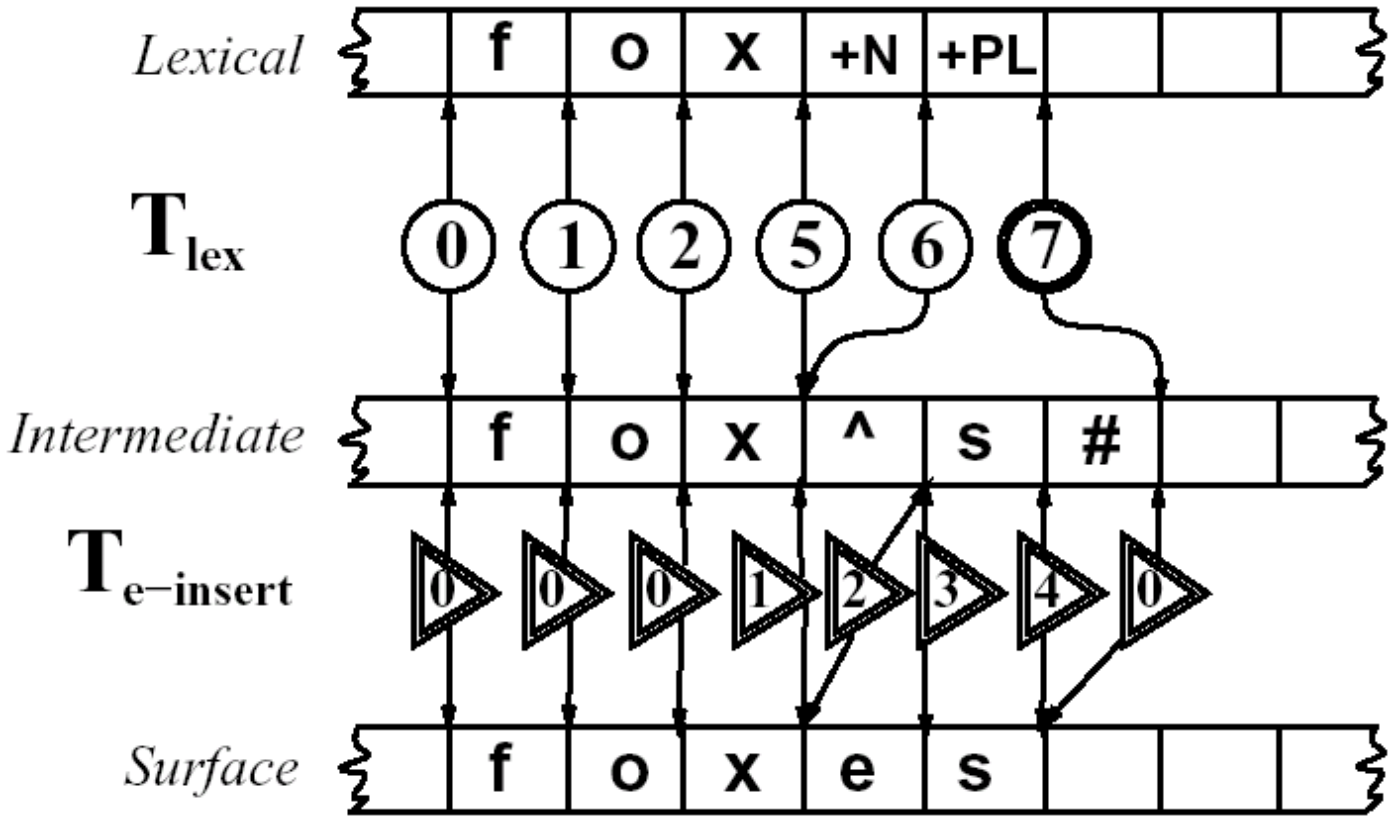
T_{lex}

Intermediate-to-Surface Transducer



$T_{\text{e-insert}}$

Sample Run


Lexical
f o x +N +PL
T_{lex}
0 1 2 5 6 7
Intermediate
f o x ^ s #
$T_{e-insert}$
0 0 0 1 2 3 4 0
Surface
f o x e s

FSTs and ambiguity

- Parse Example 1: unionizable
  - union +ize +able
  - un+ ion +ize +able
- Parse Example 2: assess
  - assessV
  - assN +essN
- Parse Example 3: tender
  - tenderAJ
  - tenNum+dAJ+erCMP

What to do about Global Ambiguity?

- Accept first successful structure
- Run parser through all possible paths
- Bias the search in some manner

Building a morphological parser

- Approaches
  - lexicon only
  - lexicon and rules
    - finite-state automata
    - finite-state transducers
  - rules only

Lexicon-Free Morphology: Porter Stemmer

- Lexicon-Free FST Approach
- By Martin Porter (1980)
  http://www.tartarus.org/%7Emartin/PorterStemmer/
- Here is one you can try online:
  http://www.utilitymill.com/utility/Porter_Stemming_Algorithm
- Cascade of substitutions given specific conditions

```
GENERALIZATIONS
GENERALIZATION
GENERALIZE
GENERAL
```