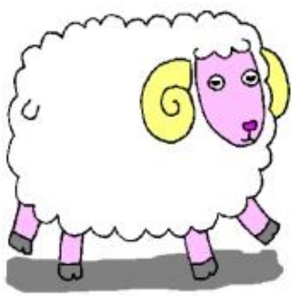
Finite-State Transducers

ANLP | 25 September 2017

slides from Marine Carpuat

Sheeptalk!

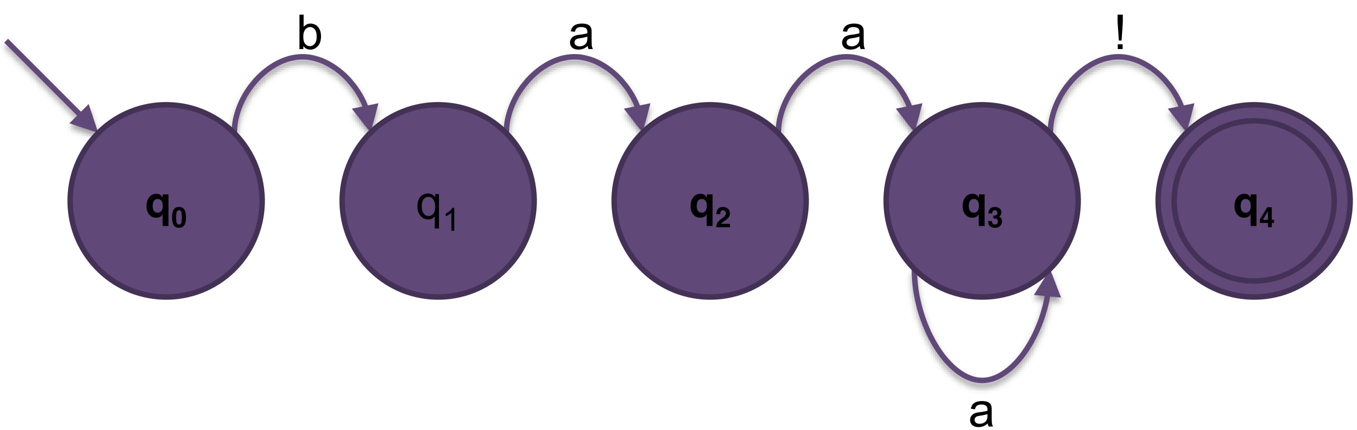
Language:



baa!
baaa!
baaaa!
baaaaa!
...

Regular Expression:
/baa+!/

Finite-State Automaton:



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

Exercise

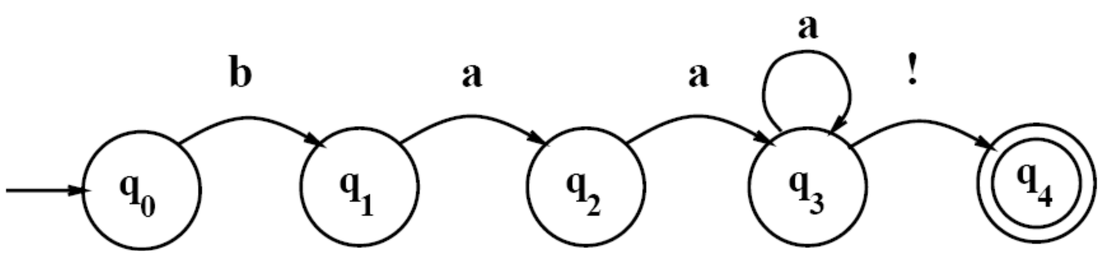
Define an FSA representing the language of all non-zero binary strings of even length

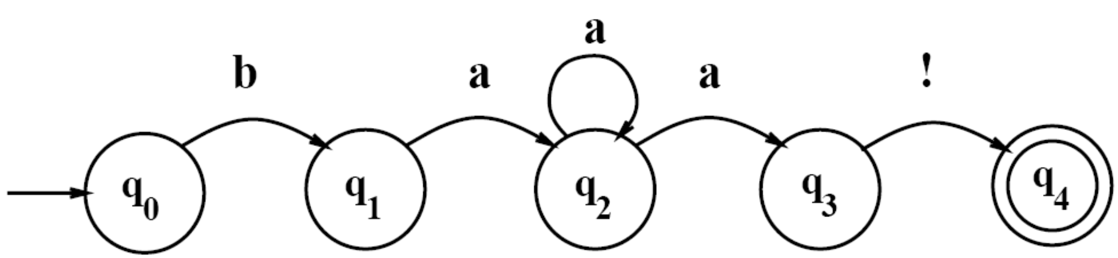
Exercise

Define an FSA representing the language of all non-zero binary strings of odd length

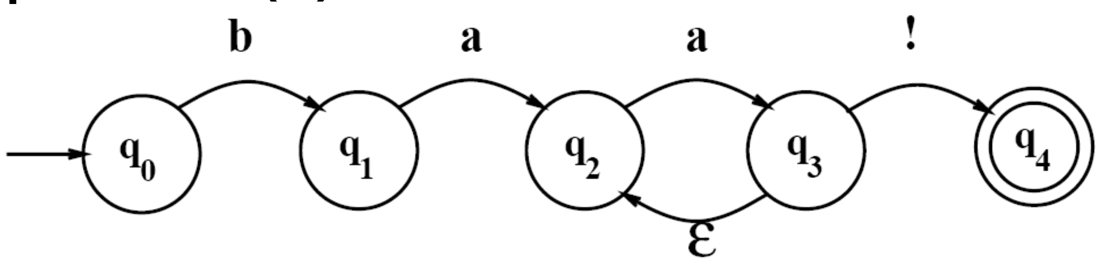
Introducing Non-Determinism

- 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
  - Explore paths in parallel
  - Recognition with NFSAs as search through state space

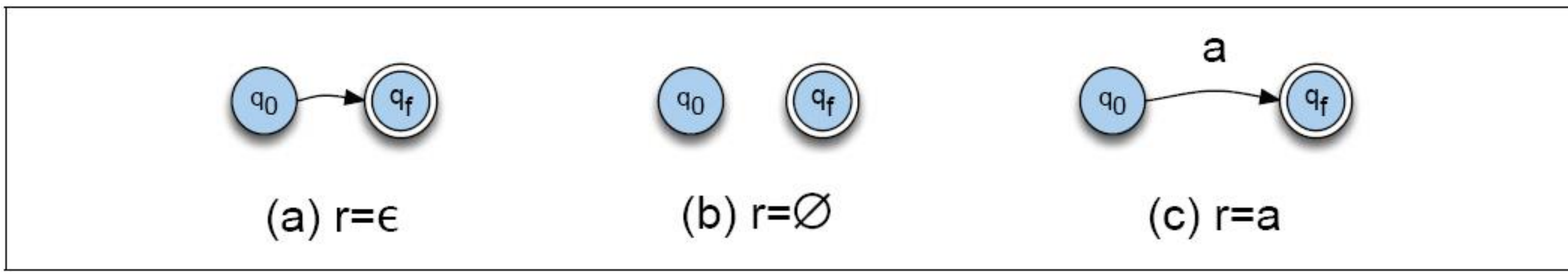
What’s the point?

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

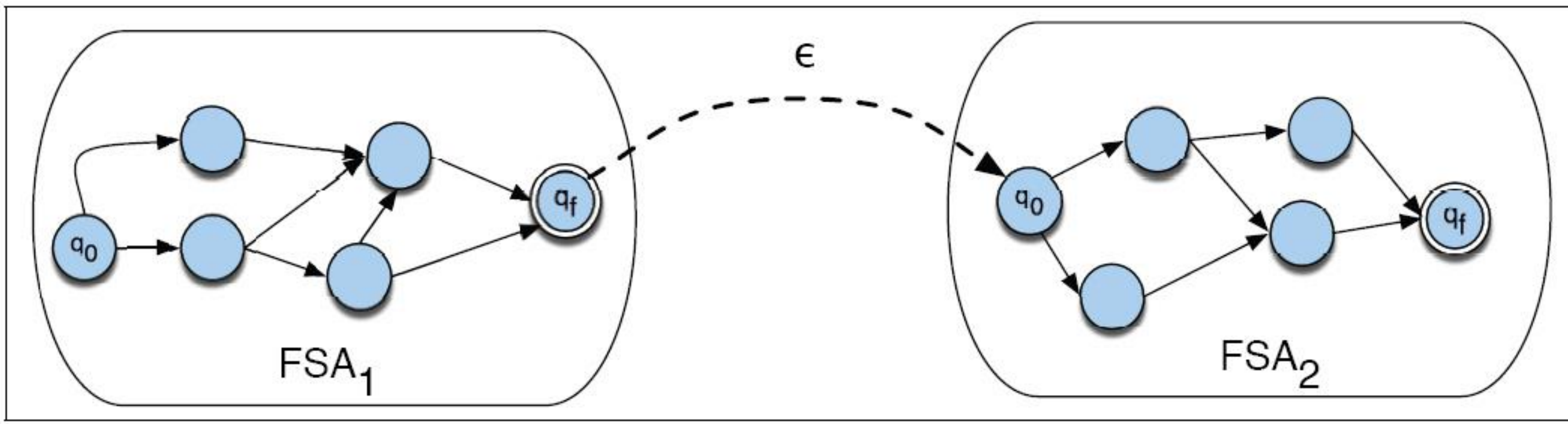
Regular Language: Definition

- $\varnothing$ 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

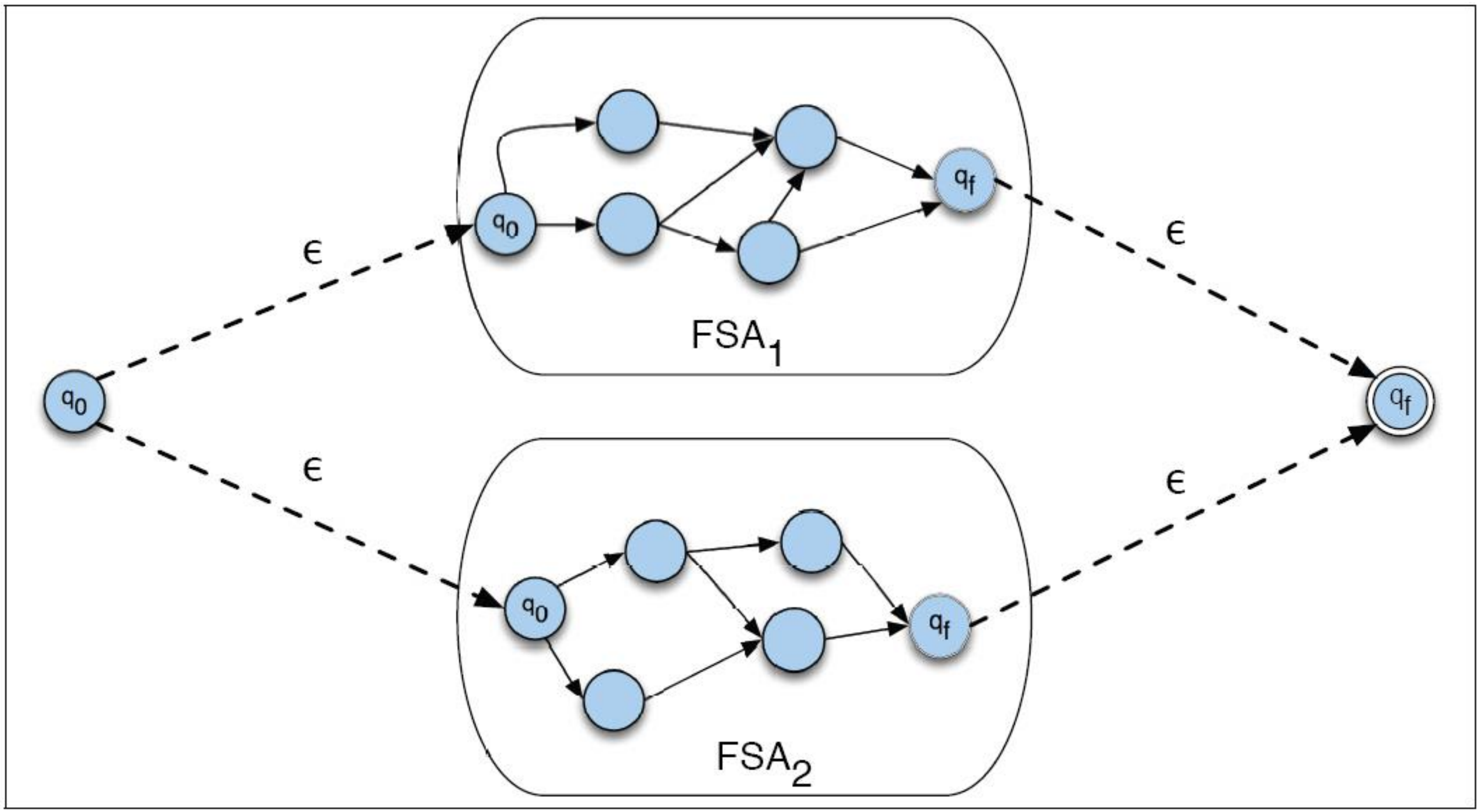
Regular Languages: Starting Points



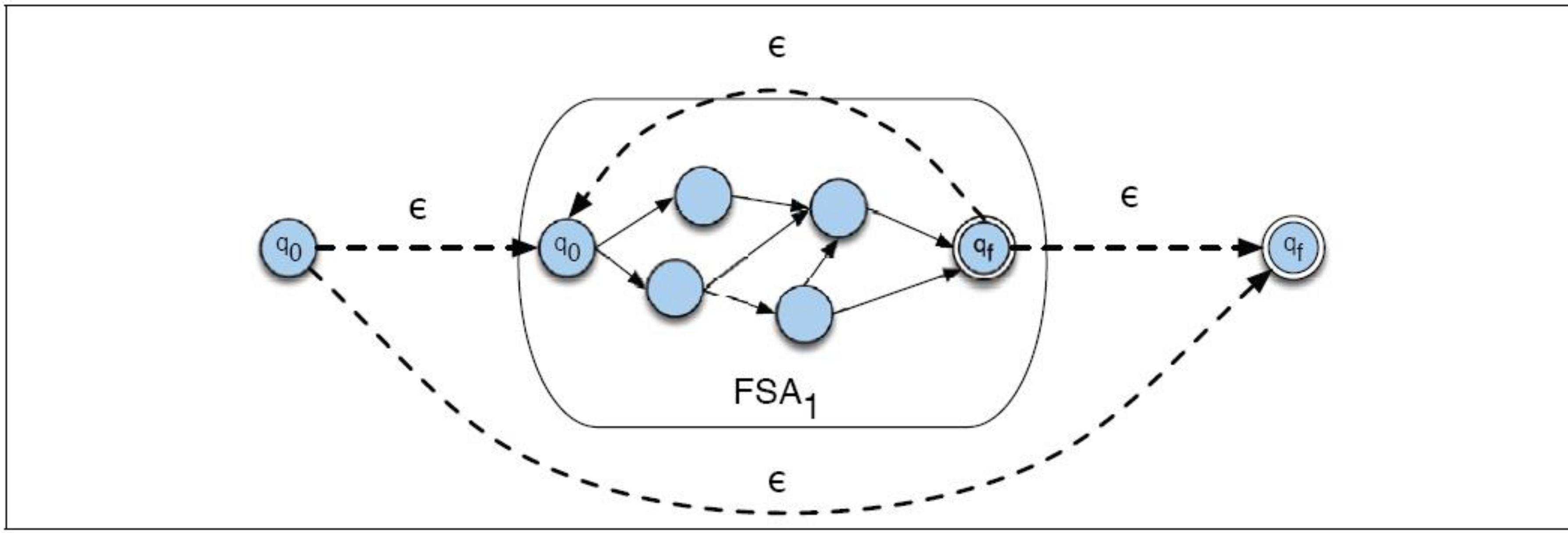
Regular Languages: Concatenation



Regular Languages: Disjunction

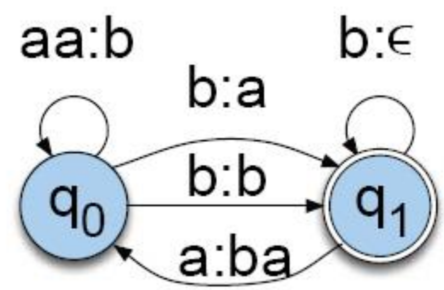


Regular Languages: Kleene Closure



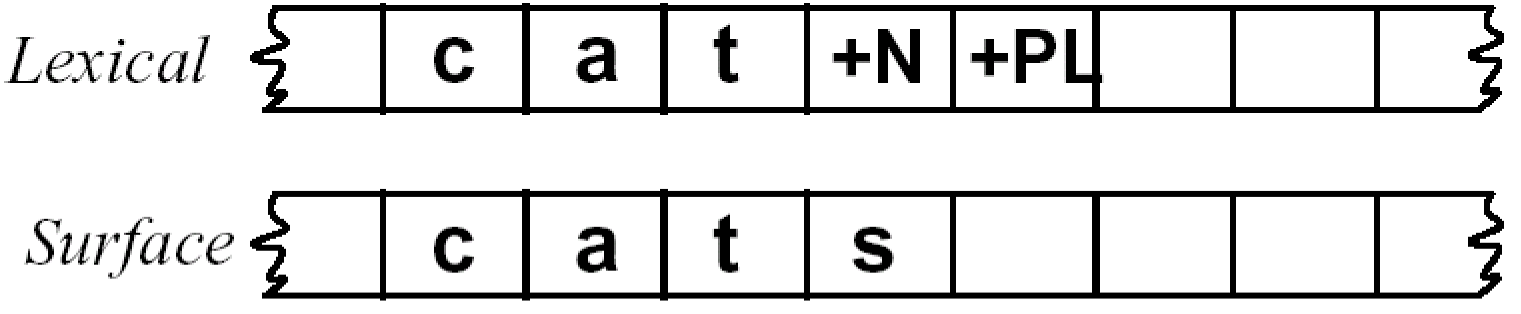
Finite-State Transducers (FSTs)

- 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



Four-fold view of FSTs

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



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

```
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

- Cascading set of rules
  - s → ε
  - ation → e
  - ize → ε
  - …
- Example
  - generalizations
    → generalization
    → generalize
    → general
  - organizations
    → organization
    → organize
    → organ

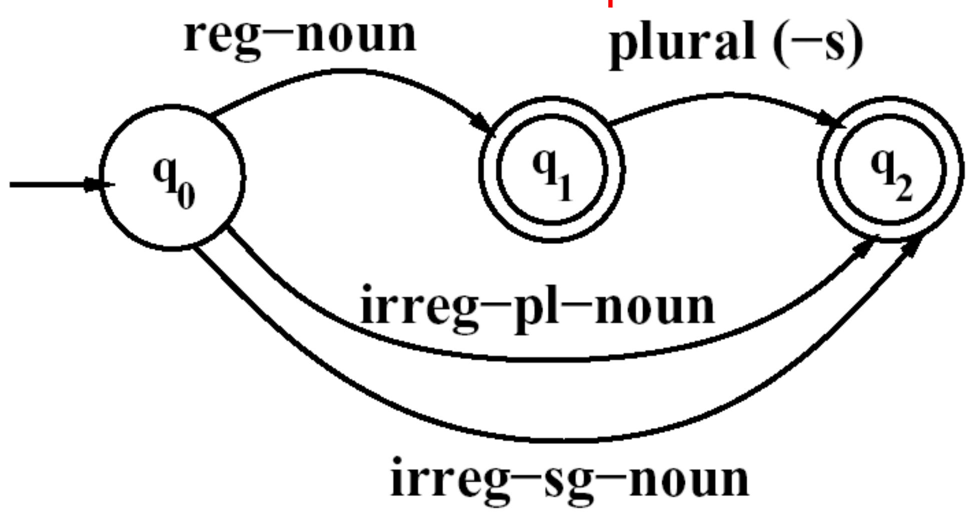
Lexicon + Rules

- 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

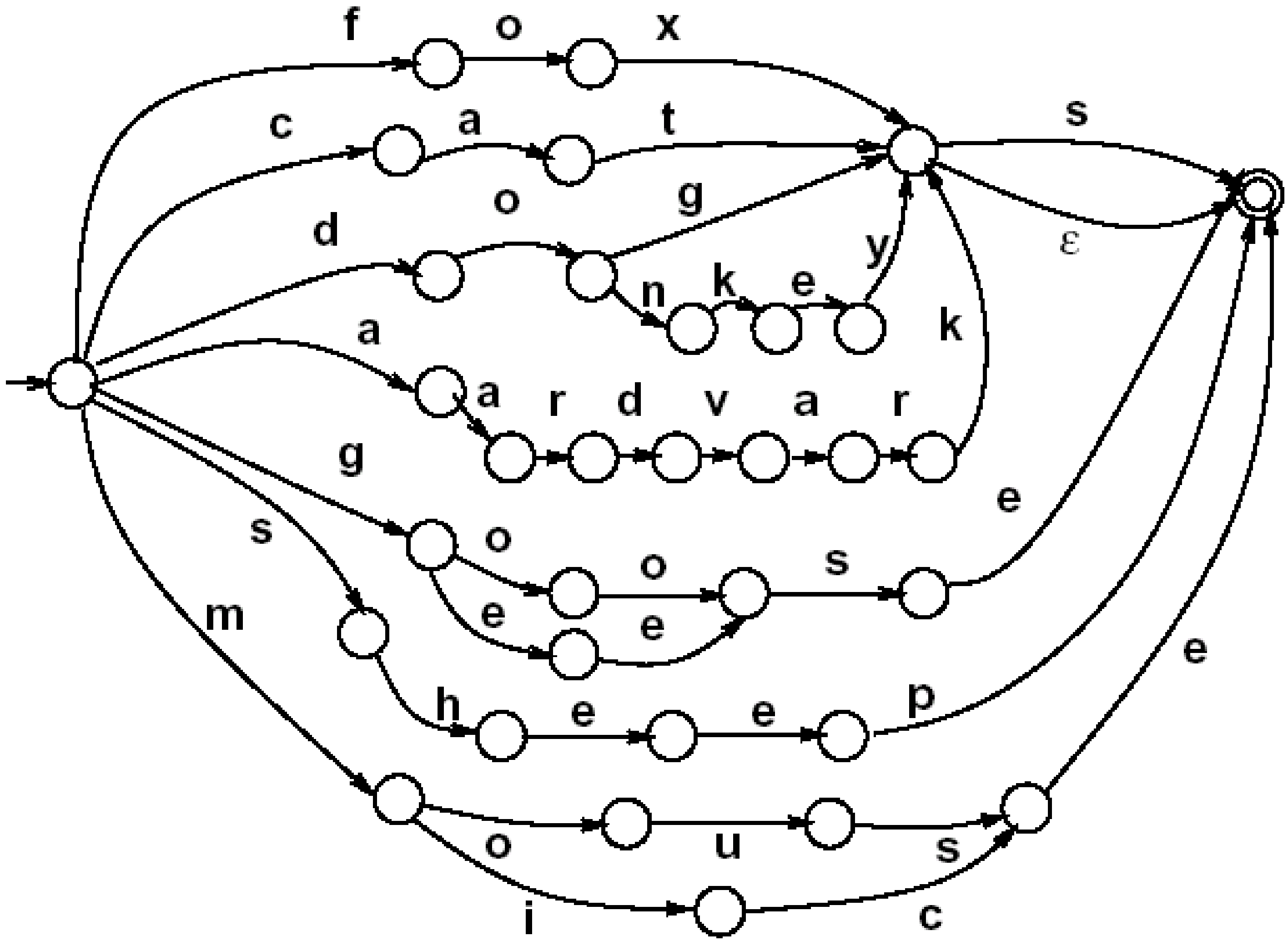
FSA: English Noun Morphology

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

Note problem with orthography!



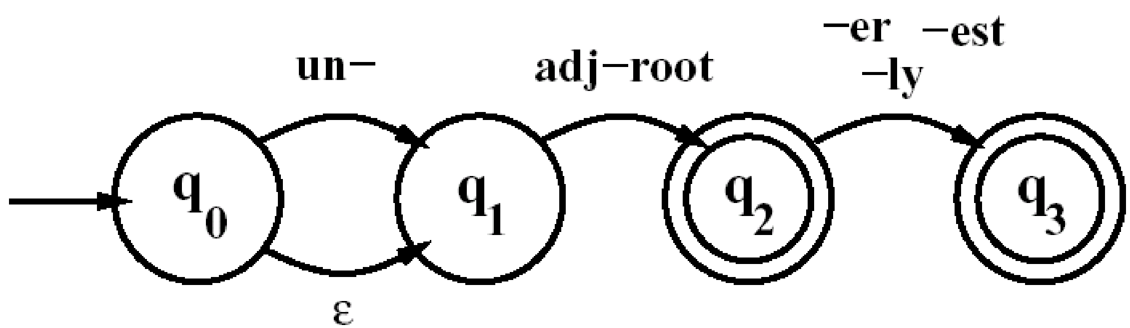
FSA: English Noun Morphology

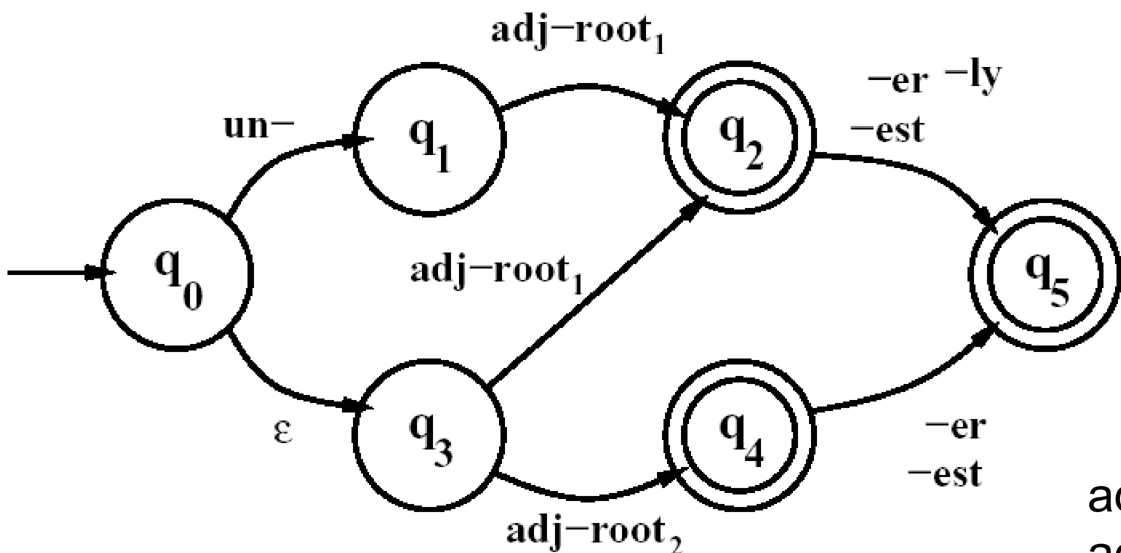


FSA: English Adjectival Morphology

- Examples:
  - big, bigger, biggest
  - small, 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

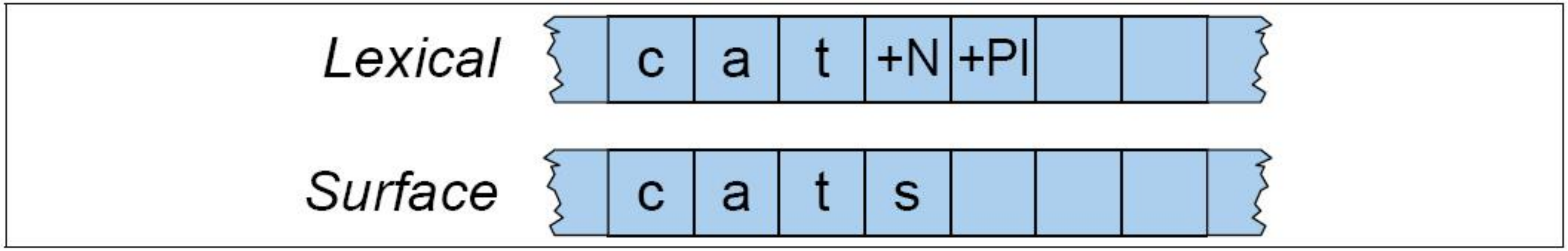




adj-root$_1$: {happy, real, …}
adj-root$_2$: {big, small, …}

Morphological Parsing with FSTs

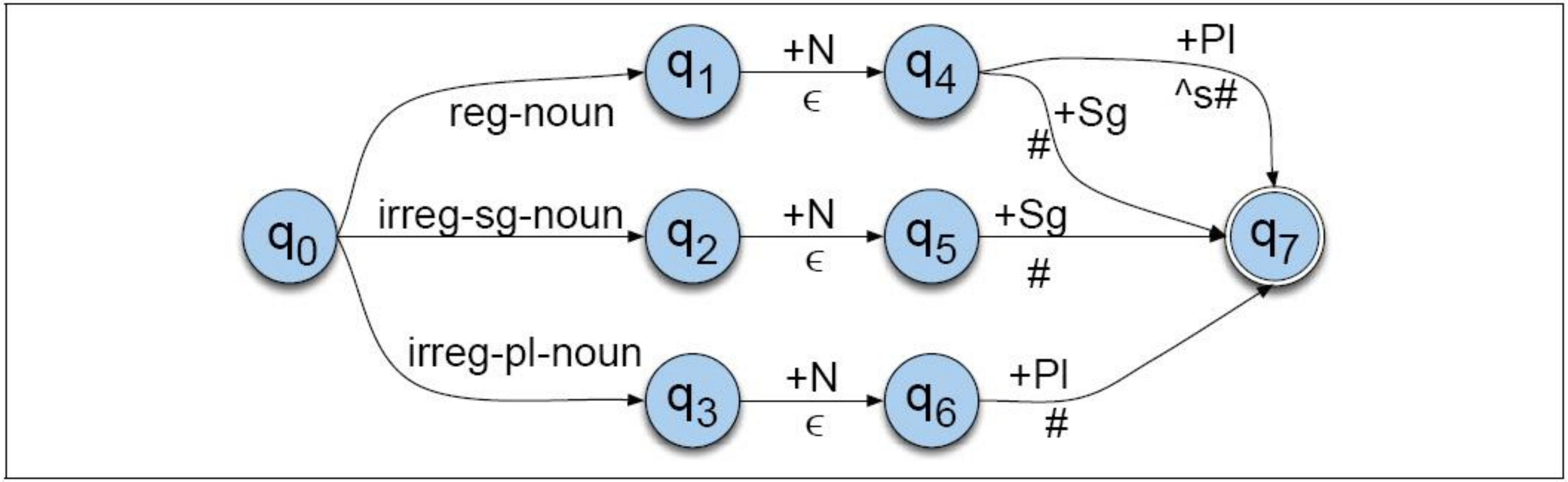
- Limitation of FSA:
  - Accepts or rejects an input… but doesn't actually provide an analysis
- Use FSTs instead!
  - One tape contains the input, the other tape as the analysis



Terminology

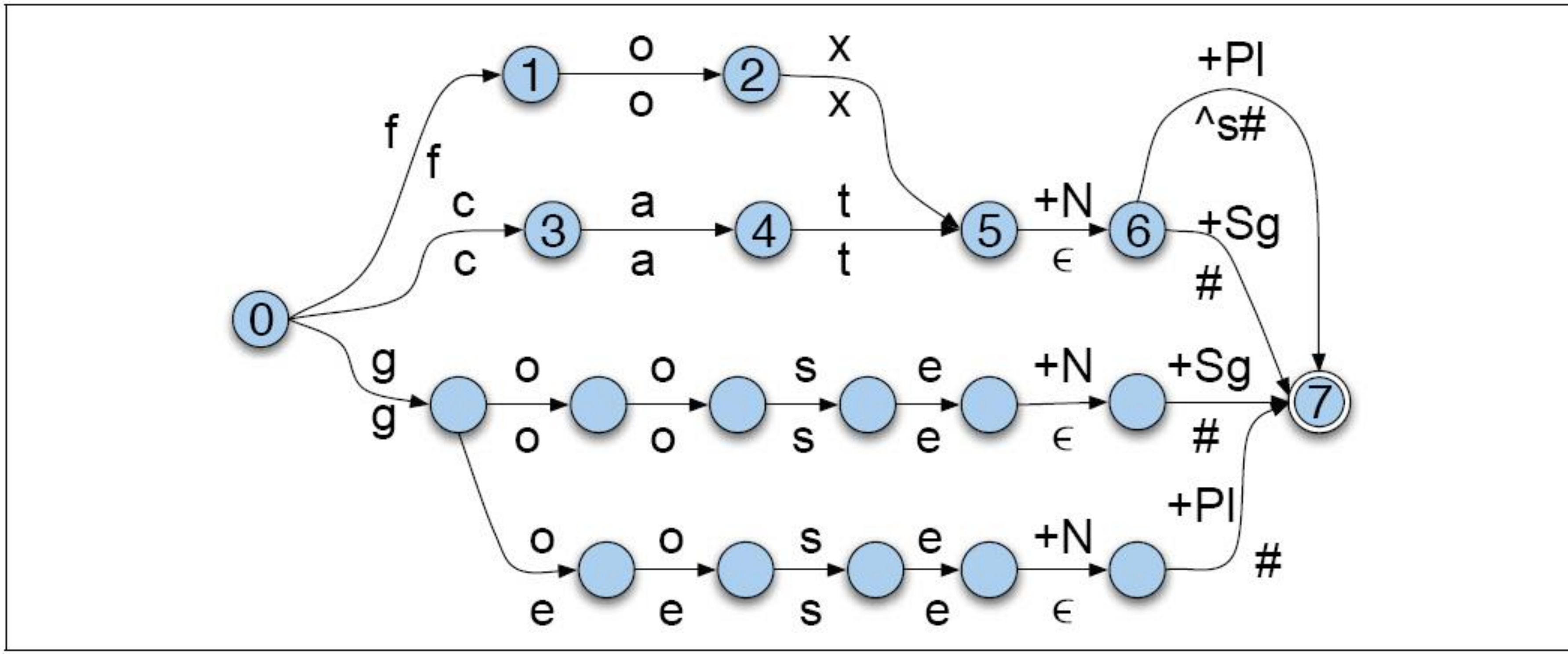
- Transducer alphabet (pairs of symbols):
  - a:b = *a* on the upper tape, *b* on the lower tape
  - a:ε = *a* on the upper tape, nothing on the lower tape
  - If a:a, write a for shorthand
- Special symbols
  - # = word boundary
  - ^ = morpheme boundary
  - (For now, think of these as mapping to ε)

FST for English Nouns

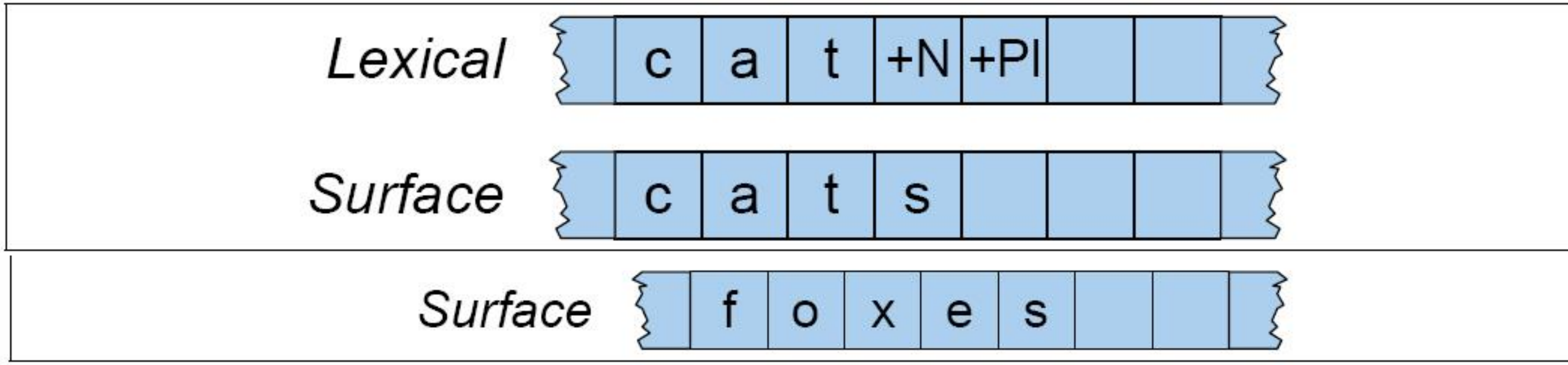


- What's the problem here?

FST for English Nouns

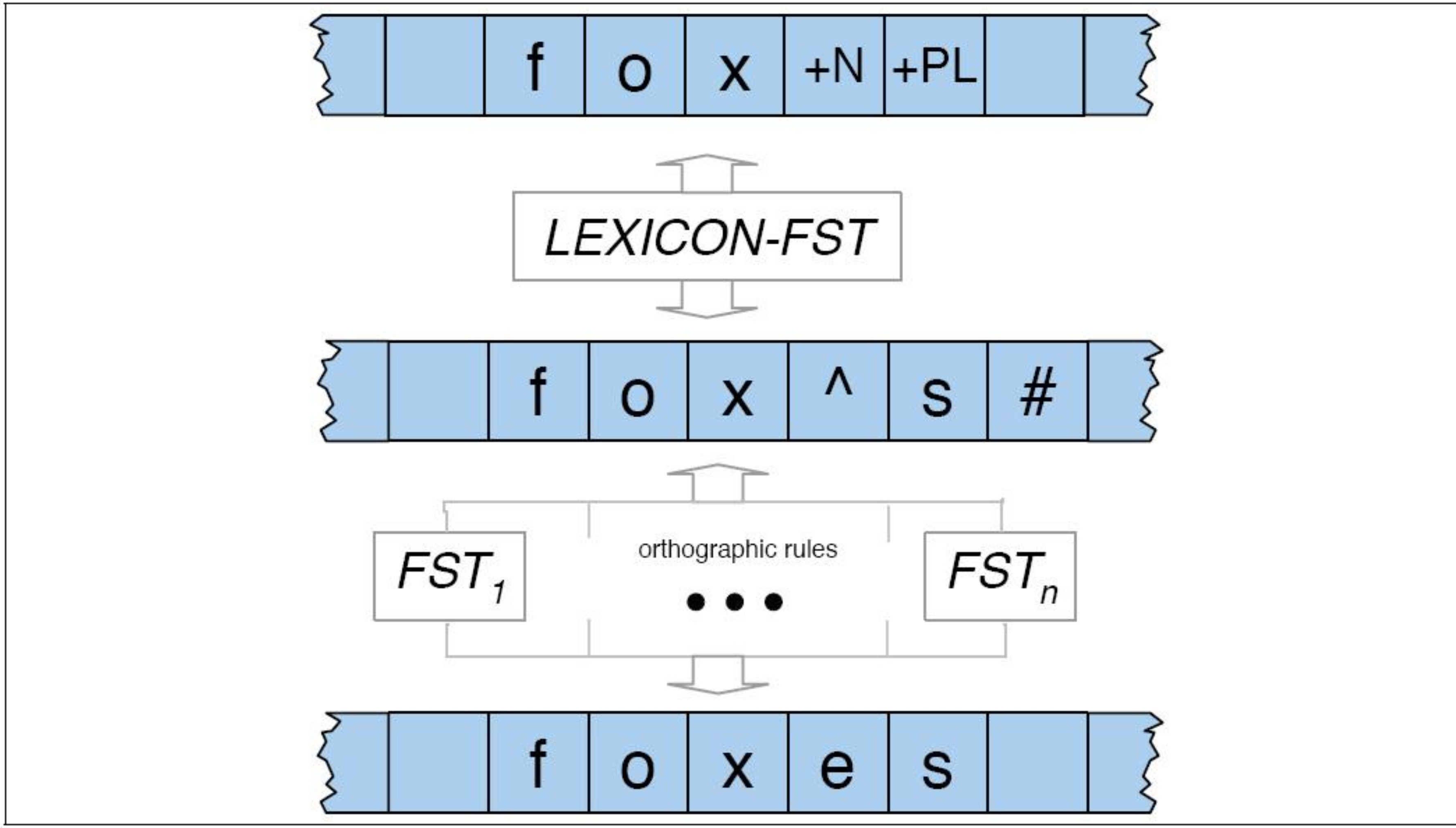


Handling Orthography



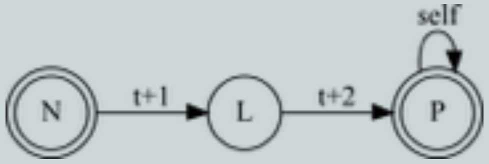
Name	Description of Rule	Example
Consonant doubling	1-letter consonant doubled before *-ing*/*-ed*	beg/begging
E deletion	silent e dropped before *-ing* and *-ed*	make/making
E insertion	e added after *-s*,*-z*,*-x*,*-ch*, *-sh* before *-s*	watch/watches
Y replacement	*-y* changes to *-ie* before *-s*, *-i* before *-ed*	try/tries
K insertion	verbs ending with *vowel* + *-c* add *-k*	panic/panicked

Complete Morphological Parser



Regular Relations

- A regular relation describes:
  - for every state change in a regular automaton
  - a **finite set** of possible outputs
- Regular relations are like bilingual dictionaries for two regular languages
  - They allow **inversion** (we can go from L2 <> L1)
  - Allow **composition** (L1 > L2, L2 > L3 → L1 > L3)



slide from Amir Zeldes

FSTs and Ambiguity

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

Practical NLP Applications

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

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

- Rule → FST compiler handles the rest…