Finite-state methods for morphology

Julia Hockenmaier

Today's lecture

What are words? How many words are there?

What is the structure of words?
(in English, Chinese, Arabic,…)

- Morphology: the area of linguistics that deals with this.

How can we identify the structure of words?

- We need to build a morphological analyzer (parser).
- We will use finite-state transducers for this task.

Finite-State Automata and Regular Languages

- (Review)

Morphology:
What is a word?

A Turkish word

uygarlaştıramadıklarımızdanmışsınızcasına

uygar_laş_tır_ama_dık_lar_ımız_dan_mış_sınız_casına

"as if you are among those whom we were not able to civilize (=cause to become civilized)"

uygar: *civilized*
_laş: *become*
_tır: *cause somebody to do something*
_ama: *not able*
_dık: past participle
_lar: plural
_ımız: 1st person plural possessive (our)
_dan: *among* (ablative case)
_mış: past
_sınız: 2nd person plural (you)
_casına: *as if* (forms an adverb from a verb)

K. Oflazer pc to J&M

Basic word classes (parts of speech)

Content words (open-class):

- Nouns: student, university, knowledge,...
- Verbs: write, learn, teach,...
- Adjectives: difficult, boring, hard, ...
- Adverbs: easily, repeatedly,...

Function words (closed-class):

- Prepositions: in, with, under,...
- Conjunctions: and, or,...
- Determiners: a, the, every,...

How many words are there?

The Unix command ***“wc -w”*** counts the words in a file.

```
> cat example.txt
This company isn't New York-based anymore
We moved to Chicago

> wc -w example.txt
     10 example.txt
```

“wc -w” uses blanks to identify words:

This$_1$ company$_2$ isn't$_3$ New$_4$ York-based$_5$ anymore$_6$
We$_7$ moved$_8$ to$_9$ Chicago$_{10}$

Words aren’t just defined by blanks

Problem 1: Compounding

“ice cream”, “website”, “web site”, “New York-based”

Problem 2: Other writing systems have no blanks

Chinese: 我开始写小说 =

我	开始	写	小说
I	*start(ed)*	*writing*	*novel(s)*

Problem 3: Clitics

English: “doesn’t” , “I’m” ,

Italian: “dirglielo” = dir + gli(e) + lo

tell + *him* + *it*

How many words are there?

Of course he wants to take the advanced course too.
He already took two beginners' courses.

This is a bad question. Did I mean:

How many word tokens are there?
(16 to 19, depending on how we count punctuation)

How many word types are there?
(i.e. How many different words are there?
Again, this depends on how you count, but it's usually much less than the number of tokens)

How many words are there?

Of course he wants to take the advanced course too.
He already took two beginners' courses.

The same (underlying) word can take different forms:
course/courses, take/took

We distinguish concrete word forms (take, taking) from abstract lemmas or dictionary forms (take)

Different words may be spelled/pronounced the same:
of course vs. advanced course
two vs. too

How many different words are there?

Inflection creates different forms of the same word:

Verbs: to be, being, I am, you are, he is, I was,

Nouns: one book, two books

Derivation creates different words from the same lemma:

grace ⇒ disgrace ⇒ disgraceful ⇒ disgracefully

Compounding combines two words into a new word:

cream ⇒ ice cream ⇒ ice cream cone ⇒ ice cream cone bakery

Word formation is productive:

New words are subject to all of these processes:

Google ⇒ Googler, to google, to ungoogle, to misgoogle, googlification, ungooglification, googlified, Google Maps, Google Maps service,...

Inflectional morphology in English

Verbs:

– Infinitive/present tense: walk, go

– 3rd person singular present tense (s-form): walks, goes

– Simple past: walked, went

– Past participle (ed-form): walked, gone

– Present participle (ing-form): walking, going

Nouns:

– Number: singular (book) vs. plural (books)

– Plural: books

– Possessive (~ genitive case): book's, books

– Personal pronouns inflect for person, number, gender, case:
I saw him; he saw me; you saw her; we saw them; they saw us.

Derivational morphology

Nominalization:

V + -ation: computerization
V+ -er: killer
Adj + -ness: fuzziness

Negation:

un-: undo, unseen, ...
mis-: mistake,...

Adjectivization:

V+ -able: doable
N + -al: national

Morphemes: stems, affixes

dis-grace-ful-ly
prefix-stem-suffix-suffix

Many word forms consist of a stem plus a number of affixes (*prefixes* or *suffixes*)

- *Infixes* are inserted inside the stem.
- *Circumfixes* (German <u>ge</u>seh<u>en</u>) surround the stem

Morphemes: the smallest (meaningful/grammatical) parts of words.

- *Stems* (grace) are often free morphemes.
  - Free morphemes can occur by themselves as words.
- *Affixes* (dis-, -ful, -ly) are usually bound morphemes**.**
  - Bound morphemes have to combine with others to form words.

Morphemes and morphs

There are many *irregular* word forms:

– Plural nouns add -*s* to singular: book-book**s**,
  but: box-box**es**, fly-fl**ies**, child-child**ren**

– Past tense verbs add -*ed* to infinitive: walk-walk**ed**,
  but: like-like**d**, leap-leap**t**

Morphemes are abstract categories

Examples: plural morpheme, past tense morpheme

The same morpheme (e.g. for plural nouns) can be realized as different surface forms (morphs):
-s/-es/-ren

Allomorphs: two different realizations (-s/-es/-ren)
of the same underlying morpheme (plural)

Morphological parsing and generation

Morphological parsing

disgracefully

dis	***grace***	***ful***	***ly***
prefix	*stem*	*suffix*	*suffix*
NEG	grace*+N*	*+ADJ*	*+ADV*

Morphological generation

Generate possible English words:

grace, graceful, gracefully
disgrace, disgraceful, disgracefully,
ungraceful, ungracefully,
undisgraceful, undisgracefully,…

Don’t generate impossible English words:

*gracelyful, *gracefuly, *disungracefully,…

Review:
Finite-State Automata and Regular Languages

Formal languages

An alphabet $\sum$ is a set of symbols:

e.g. $\sum = \{a, b, c\}$

A string ω is a sequence of symbols, e.g $\omega = abcb$.

The empty string ε consists of zero symbols.

The Kleene closure $\sum$* ('sigma star') is the (infinite) set of all strings that can be formed from $\sum$:

$\sum^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, \ldots\}$

A language $L \subseteq \sum^*$ over $\sum$ is also a set of strings.

Typically we only care about proper subsets of $\sum^*$ $(L \subset \Sigma)$.

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).

Different automata define different language classes:

- **Finite-state** automata define **regular** languages
- **Pushdown** automata define **context-free** languages
- **Turing machines** define **recursively enumerable** languages

Finite State Automata (FSAs)

A finite-state automaton $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, .., q_n\}$
- A finite alphabet Σ of input symbols (e.g. $\Sigma = \{a, b, c, ...\}$)
- A designated start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ:
  - The transition function for a deterministic (D)FSA: $Q \times \Sigma \rightarrow Q$

    $\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$

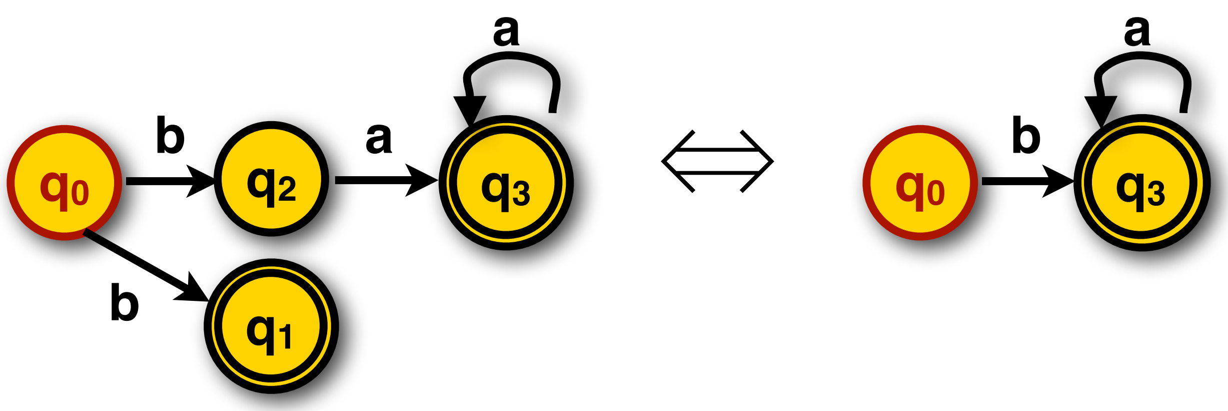
    If the current state is q and the current input is w, go to q'
  - The transition function for a nondeterministic (N)FSA: $Q \times \Sigma \rightarrow 2^Q$

    $\delta(q,w) = Q'$ for $q \in Q, Q' \subseteq Q, w \in \Sigma$

    If the current state is q and the current input is w, go to any $q' \in Q'$

Finite State Automata (FSAs)

Every NFA can be transformed into an equivalent DFA:



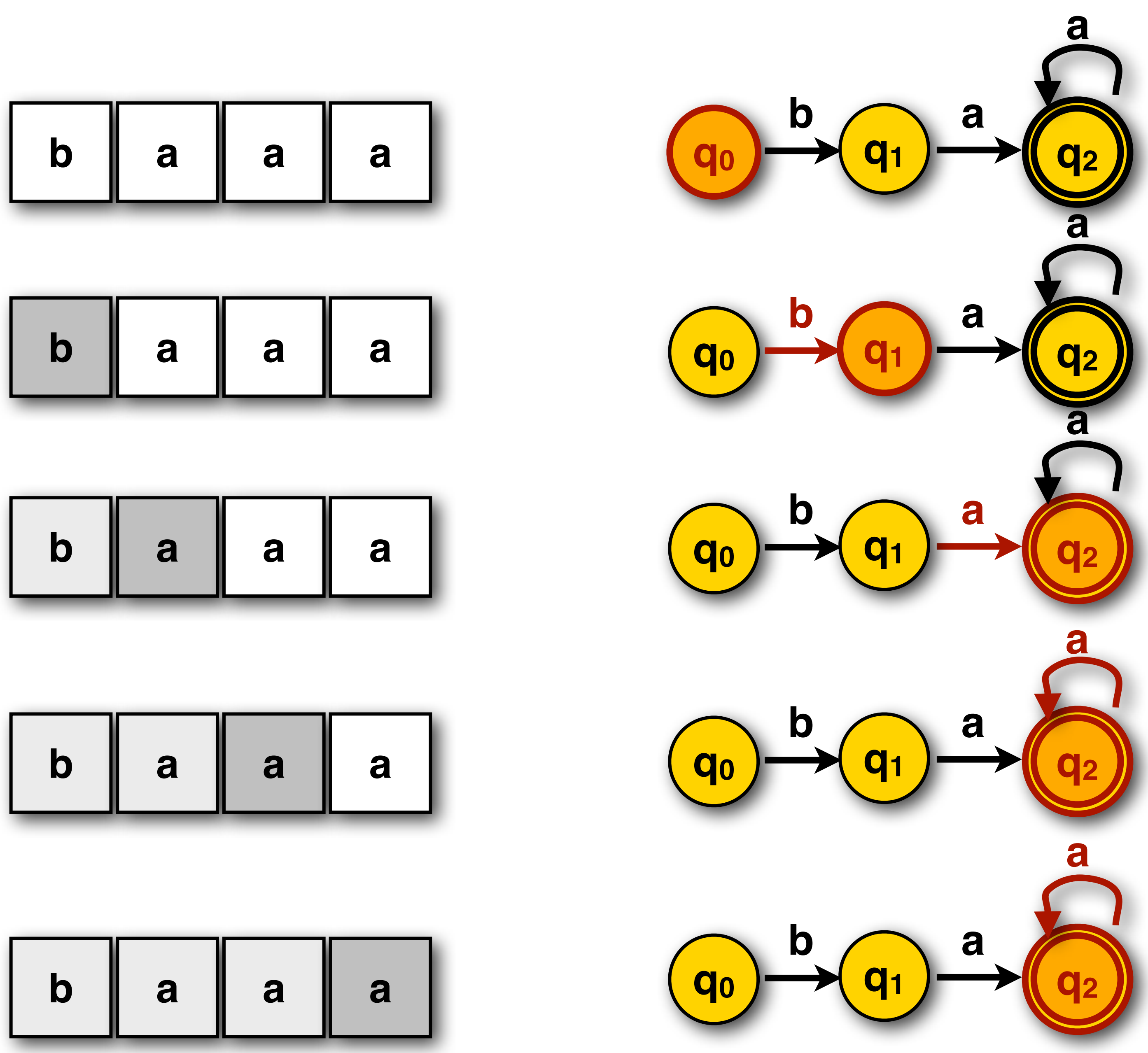
Recognition of a string *w* with a DFA is linear in the length of *w*

Finite-state automata define the class of regular languages

$L_1 = \{ a^n b^m \} = \{ab, aab, abb, aaab, abb, \ldots \}$ is a regular language,

$L_2 = \{ a^n b^n \} = \{ab, aabb, aaabbb, \ldots\}$ is not (it's context-free).

You cannot construct an FSA that accepts all the strings in L_2 and nothing else.


b a a a
q0 b q1 a q2 a
b a a a
q0 b q1 a q2 a
b a a a
q0 b q1 a q2 a
b a a a
q0 b q1 a q2 a
b a a a
q0 b q1 a q2 a

Regular Expressions

Simple patterns:

- **Standard characters** match themselves: *‘a’*, *‘1’*
- **Character classes**: *‘[abc]’*, *‘[0-9]’*, **negation:** *‘[^aeiou]’*
  (Predefined: \s (whitespace), \w (alphanumeric), etc.)
- **Any character** (except newline) is matched by ‘.’

Complex patterns: (e.g. ^[A-Z]([a-z])+\s)

- **Group:** ‘(…)’
- **Repetition:** 0 or more times: ‘*’, 1 or more times: ‘+’
- **Disjunction:** ‘...|...’
- **Beginning of line** ‘^’ and **end of line** ‘$’

Python: Regular Expressions

```
>>> import re                        %% Import re package
>>> ex = re.compile('a.c')           %% '…': reg.expression
>>> m = ex.search('ab')              %% Does 'ab' contain ex?
>>> print m                          %% No.
None
>>> m = ex.search('abc')             %% Does 'abc' contain ex?
>>> print m                          %% Yes.
<_sre.SRE_Match object at 0x70640>
```

http://docs.python.org/dev/howto/regex.html
http://docs.python.org/lib/module-re.html

Finite-state methods for morphology

Finite state automata for morphology

grace:

$$\begin{tikzcd} q_0 \arrow[r, "\text{stem}"] & q_1 \end{tikzcd}$$

dis-grace:

$$\begin{tikzcd} q_0 \arrow[r, "\text{prefix}"] & q_1 \arrow[r, "\text{stem}"] & q_2 \end{tikzcd}$$

grace-ful:

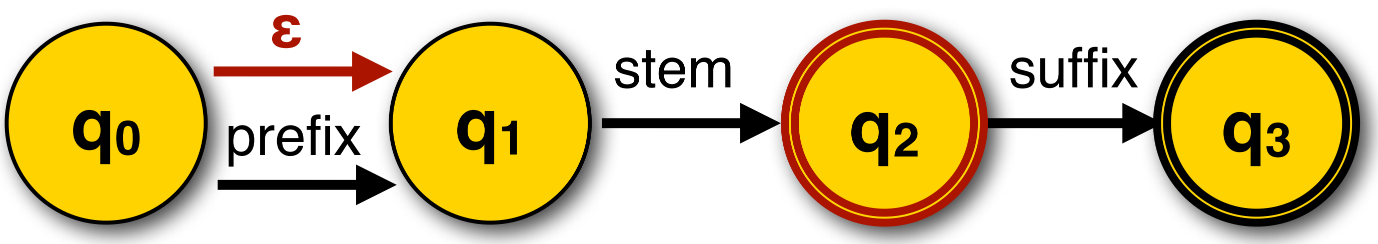
$$\begin{tikzcd} q_0 \arrow[r, "\text{stem}"] & q_1 \arrow[r, "\text{suffix}"] & q_2 \end{tikzcd}$$

dis-grace-ful:

$$\begin{tikzcd} q_0 \arrow[r, "\text{prefix}"] & q_1 \arrow[r, "\text{stem}"] & q_2 \arrow[r, "\text{suffix}"] & q_3 \end{tikzcd}$$

Union: merging automata

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

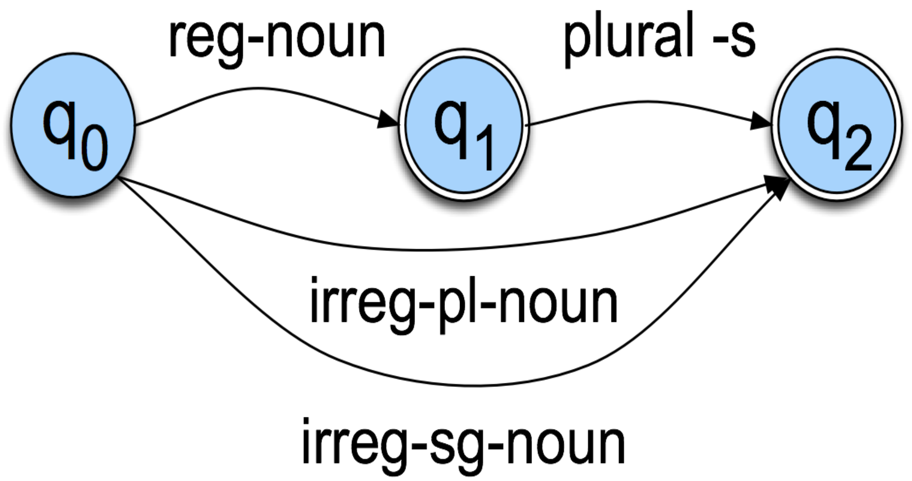


Stem changes

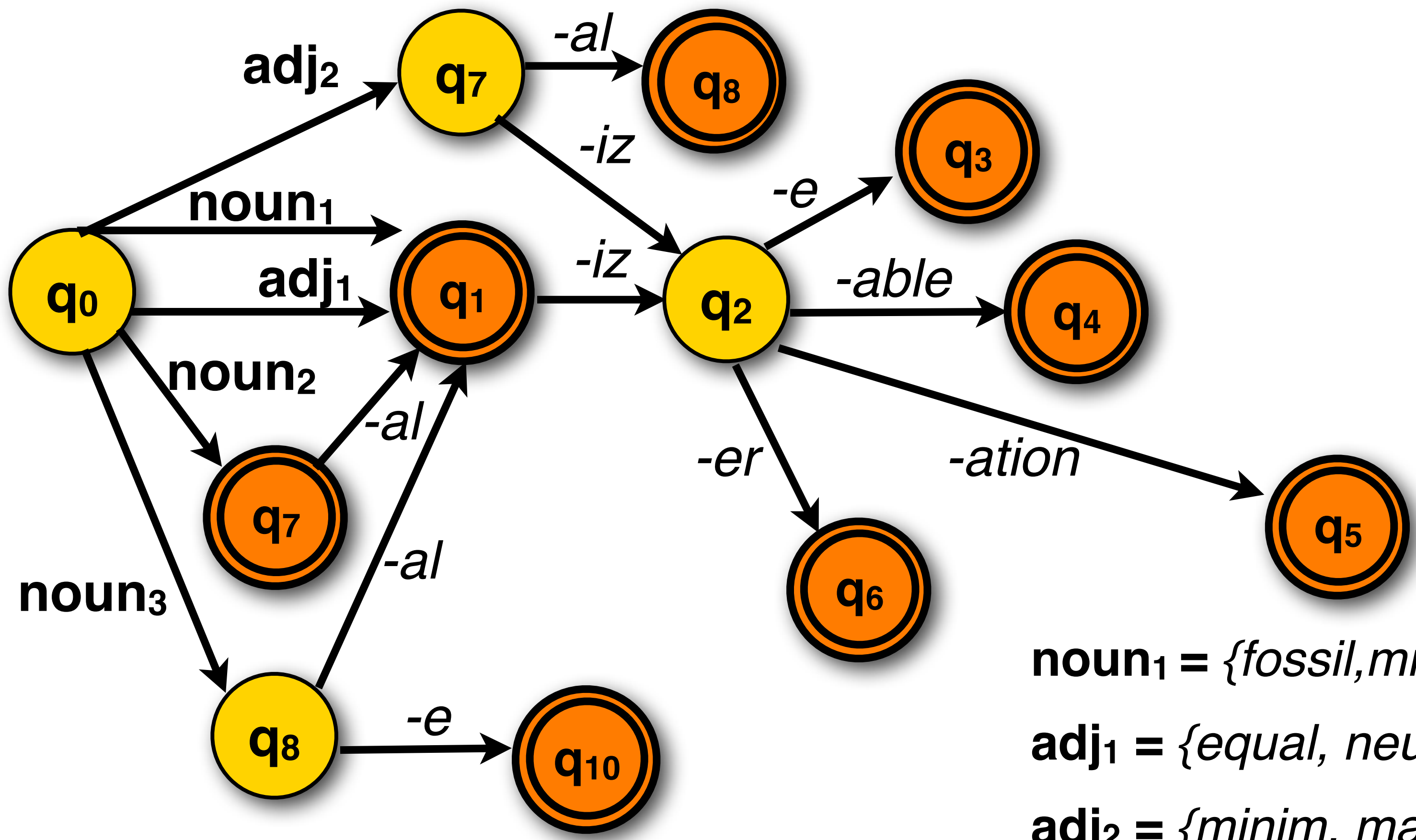
Some irregular words require stem changes:

Past tense verbs:
teach-taught, go-went, write-wrote

Plural nouns:
mouse-mice, foot-feet, wife-wives



FSAs for derivational morphology



noun$_1$ **=** *{fossil,mineral,...}*

adj$_1$ **=** *{equal, neutral}*

adj$_2$ **=** *{minim, maxim}*

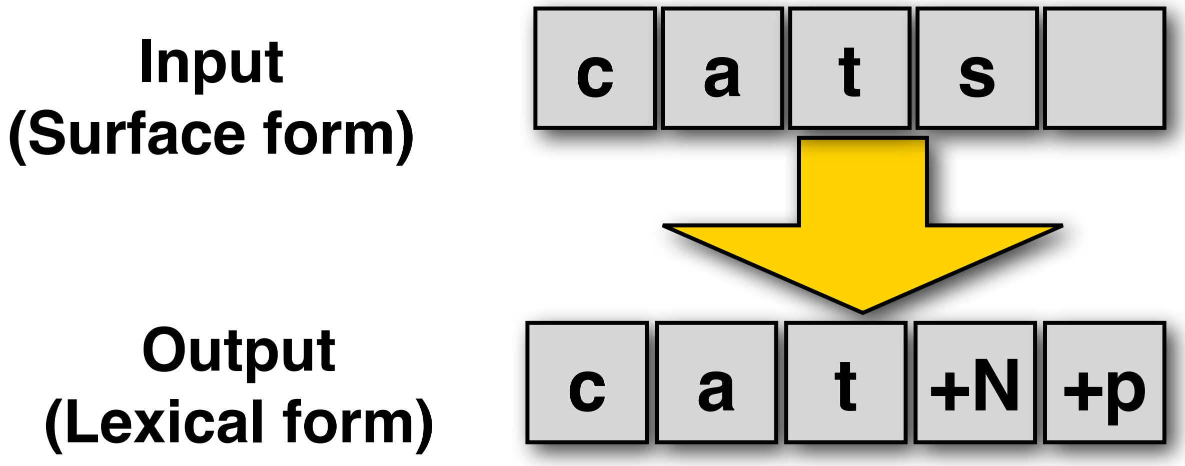
noun$_2$ **=** *{nation, form,…}*

noun$_3$ **=** *{natur, structur,…}*

Recognition vs. Analysis

FSAs can recognize (accept) a string, but they don't tell us its internal structure.

We need is a machine that maps (transduces) the input string into an output string that encodes its structure:



Finite-state transducers

A **finite-state transducer** $T = \langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle$ consists of:

- A finite **set of states** $Q = \{q_0, q_1, .., q_n\}$
- A finite alphabet Σ of **input symbols** (e.g. $\Sigma = \{a, b, c, ...\}$)
- A finite alphabet Δ of **output symbols** (e.g. $\Delta = \{+N, +pl, ...\}$)
- A designated **start state** $q_0 \in Q$
- A set of **final states** $F \subseteq Q$
- A **transition function** $\delta: Q \times \Sigma \rightarrow 2^Q$

  $\delta(q,w) = Q'$ *for* $q \in Q, Q' \subseteq Q, w \in \Sigma$
- **An output function** $\sigma: Q \times \Sigma \rightarrow \Delta^*$

  $\sigma(q,w) = \omega$ for $q \in Q, w \in \Sigma, \omega \in \Delta^*$

  If the current state is *q* and the current input is w, write ω.

Finite-state transducers

An FST $T = L_{in} \times L_{out}$ defines a relation between two regular languages L_{in} and L_{out}:

$$\begin{tikzcd}
L_{in} = \{ & \textbf{cat,} & \textbf{cats,} & \textbf{fox,} & \textbf{foxes, ...}\} \\
L_{out} = \{ & \mathit{cat{+}N{+}sg,} \arrow[u] & \mathit{cat{+}N{+}pl,} \arrow[u] & \mathit{fox{+}N{+}sg,} \arrow[u] & \mathit{fox{+}N{+}PL} \ldots\} \arrow[u]
\end{tikzcd}$$

T = { <**cat**, *cat+N+sg*>,
<**cats**, *cat+N+pl*>,
<**fox**, *fox+N+sg*>,
<**foxes**, *fox+N+pl*> }

Some FST operations

Inversion T^{-1}:

The inversion (T^{-1}) of a transducer switches input and output labels.

This can be used to switch from parsing words to generating words.

Composition ($T \circ T'$): *(Cascade)*

Two transducers $T = L_1 \times L_2$ and $T' = L_2 \times L_3$ can be composed into a third transducer $T'' = L_1 \times L_3$.

Sometimes intermediate representations are useful

English spelling rules

English spelling (orthography) is funny:

The underlying morphemes (*plural-s*, etc.) can have different orthographic surface realizations (-s, -es)

Spelling changes at morpheme boundaries:

– E-insertion: fox +s = fox**e**s

– E-deletion: mak**e** +ing = making

Intermediate representations

English plural -s: cat ⇒ cats   dog ⇒ dogs
but: fox ⇒ foxes,   bus ⇒ buses   buzz ⇒ buzzes

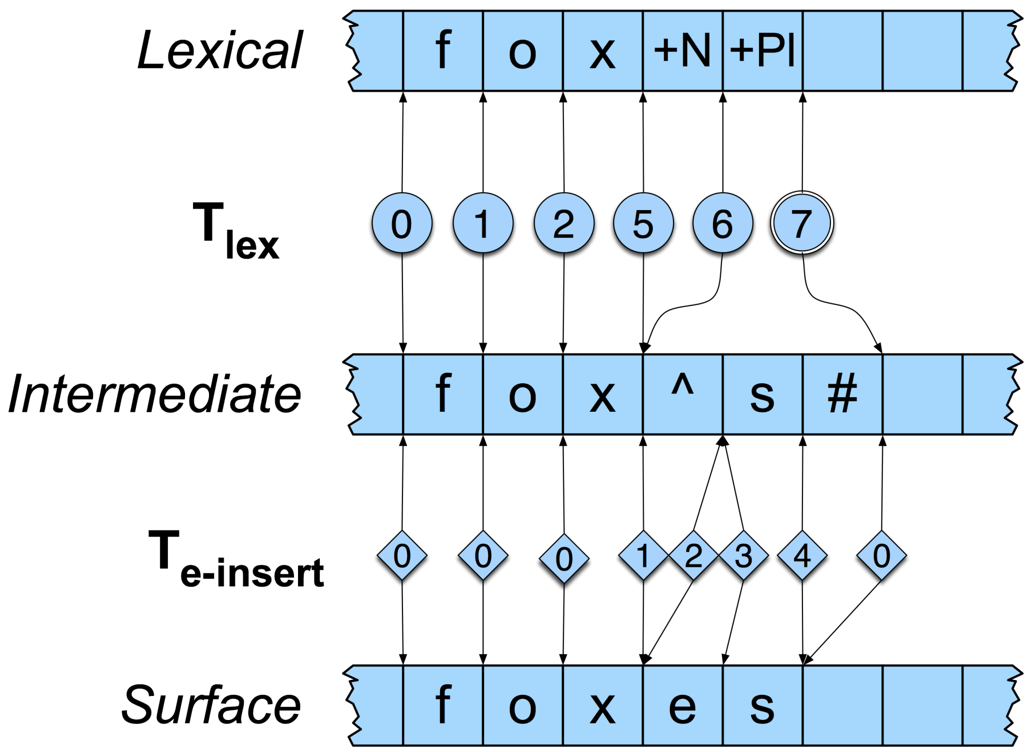
We define an intermediate representation which captures morpheme boundaries (^) and word boundaries (#):

Lexicon:	***cat+N+PL***	***fox+N+PL***
⇒ *Intermediate representation:*	***cat^s#***	***fox^s#***
⇒ *Surface string:*	***cats***	***foxes***

Intermediate-to-Surface Spelling Rule:

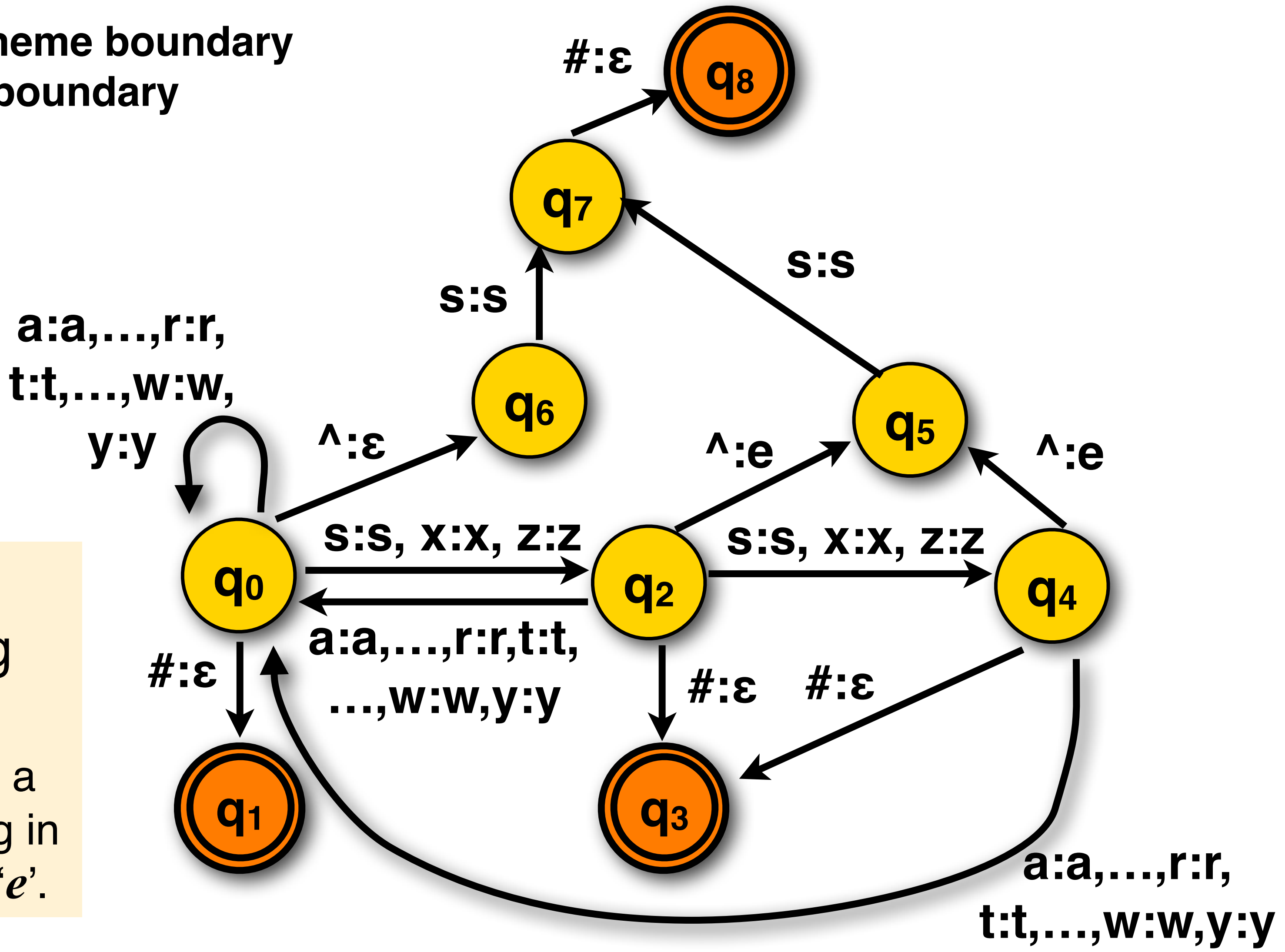
If plural '***s***' follows a morpheme ending in '***x***','***z***' or '***s***', insert '***e***'.

FST composition/cascade:



FST: intermediate to surface level

^ = morpheme boundary
# = word boundary



Intermediate-to-Surface Spelling Rule:

If plural ‘*s*’ follows a morpheme ending in ‘*x*’,‘*z*’ or ‘*s*’, insert ‘*e*’.

Dealing with ambiguity

$book$: $book$ $+N$ $+sg$ or $book$ $+V$?

Generating words is generally unambiguous, but analyzing words often requires disambiguation.

Efficiency problem:
Not every nondeterministic FST can be translated into a deterministic one!

A computer therapist?

Computer: TELL ME MORE ABOUT YOUR FAMILY
Human: My mother takes care of me.
Computer: WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
Human: My father.
Computer: YOUR FATHER
Human: You are like my father in some ways.
Computer: WHAT RESEMBLANCE DO YOU SEE
Human: You are not very aggressive but I think you don't want me to notice that.
Computer: WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
Human: You don't argue with me.
Computer: WHY DO YOU THINK I DON'T ARGUE WITH YOU
Human: You are afraid of me.

Weizenbaum (1966), ELIZA.

ELIZA as a FST cascade

Human: *You don't argue with me.*
Computer: *WHY DO YOU THINK I DON'T ARGUE WITH YOU*

1. Replace **you** with ***I*** and ***me*** with ***you:***
   I don't argue with you.
2. Replace ***<...>*** with ***Why do you think <...>:***
   Why do you think I don't argue with you.

What about compounds?

Compounds have hierarchical structure:

(((ice cream) cone) bakery)
not (ice ((cream cone) bakery))

((computer science) (graduate student))
not (computer ((science graduate) student))

We will need context-free grammars to capture this underlying structure.