CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 2: Finite-State Methods and Tokenization

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

DRES accommodations

If you need any disability related accommodations, talk to DRES (http://disability.illinois.edu, disability@illinois.edu, phone 333-4603)

- If you are concerned you have a disability-related condition that is impacting your academic progress, there are academic screening appointments available on campus that can help diagnosis a previously undiagnosed disability by visiting the DRES website and selecting "Sign-Up for an Academic Screening" at the bottom of the page."

Come and talk to me as well, especially once you have a letter of accommodation from DRES.

- Do this early enough so that we can take your requirements into account for exams and assignments.

Today's lecture: all about words!

Let's start simple….:

- What is a word?
- How many words are there (in English)?
- Do words have structure?

Later in the semester we'll ask harder questions:

- What is the meaning of words?
- How do we represent the meaning of words?

Why do we need to worry about these questions when developing NLP systems?

Dealing with words

Basic word classes in English (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,...

Pronouns: I, you, ..., me, my, mine,.., who, which, what, ...

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)

Also: upper vs. lower case: Of vs. of, etc.

Different words may be spelled the same:

course: of *course* or advanced *course*

How many words are there?

How large is the vocabulary of English
(or any other language)?

Vocabulary size = nr of distinct word types

Google N-gram corpus: 1 trillion tokens,
13 million word types that appear 40+ times

If you count words in text, you will find that…

…a few words (mostly closed-class) are very frequent
(the, be, to, of, and, a, in, that,...)

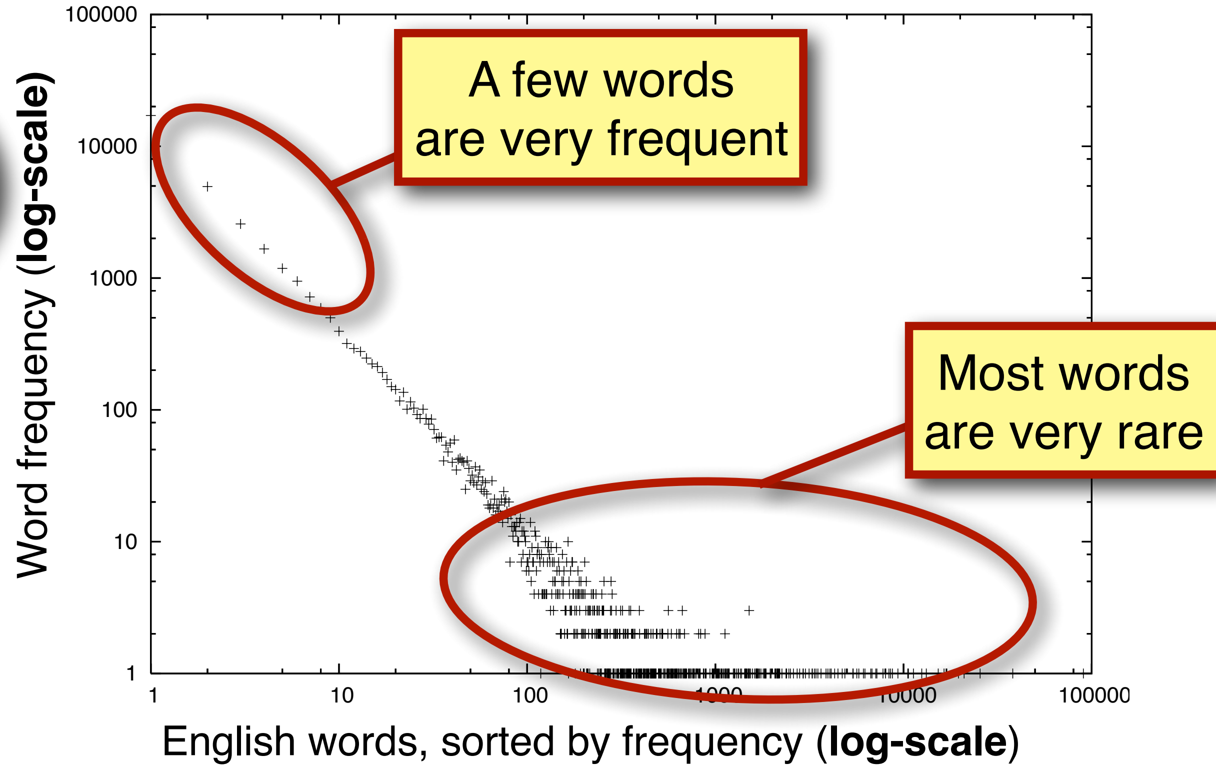
… most words (all open class) are very rare.

… even if you've read a lot of text, you will keep finding words you haven't seen before.

Zipf's law: the long tail

How many words occur once, twice, 100 times, 1000 times?

the r-th most common word w_r has $P(w_r) \propto 1/r$



$w_1 = the$, $w_2 = to$, …., $w_{5346} = computer$, ...

In natural language:

- A small number of events (e.g. words) occur with high frequency
- A large number of events occur with very low frequency

Implications of Zipf's Law for NLP

The good:

Any text will contain a number of words that are very **common**.

We have seen these words often enough that we know (almost) everything about them. These words will help us get at the structure (and possibly meaning) of this text.

The bad:

Any text will contain a number of words that are **rare**.

We know something about these words, but haven't seen them often enough to know everything about them. They may occur with a meaning or a part of speech we haven't seen before.

The ugly:

Any text will contain a number of words that are **unknown** to us.

We have *never* seen them before, but we still need to get at the structure (and meaning) of these texts.

Dealing with the bad and the ugly

Our systems need to be able to **generalize** from what they have seen to unseen events.

There are two (complementary) approaches to generalization:

— **Linguistics** provides us with insights about the rules and structures in language that we can exploit in the (symbolic) representations we use

E.g.: a finite set of grammar rules is enough to describe an infinite language

— **Machine Learning/Statistics** allows us to learn models (and/or representations) from real data that often work well empirically on unseen data

E.g. most statistical or neural NLP

How do we represent words?

Option 1: Words are **atomic symbols**

Can't capture syntactic/semantic relations between words

— Each (surface) word form is its own symbol

— Map different forms of a word to the same symbol

- **Lemmatization**: map each word to its lemma (esp. in English, the lemma is still a word in the language, but lemmatized text is no longer grammatical)
- **Stemming**: remove endings that differ among word forms (no guarantee that the resulting symbol is an actual word)
- **Normalization:** map all variants of the same word (form) to the same canonical variant (e.g. lowercase everything, normalize spellings, perhaps spell-check)

How do we represent words?

Option 2: Represent the **structure** of each word

“books” => “book N pl” (or “book V 3rd sg”)

This requires a **morphological analyzer** (more later today)

The output is often a lemma plus morphological information

This is particularly useful for highly inflected languages (less so for English or Chinese)

How do we represent unknown words?

Systems that use machine learning may need to have a unique representation of each word.

Option 1: the **UNK** token

Replace all rare words (in your training data) with an UNK token (for Unknown word).

Replace *all* unknown words that you come across after training (including rare training words) with the same UNK token

Option 2: **substring-based** representations

Represent (rare and unknown) words as sequences of characters or substrings

- Byte Pair Encoding: learn which character sequences are common in the vocabulary of your language

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

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 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:

Common nouns inflect for number:

singular (book) vs. plural (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 in English

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*)

Exceptions: *Infixes* are inserted inside the stem

Circumfixes (German *ge*seh*en*) 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

The *same information* (plural, past tense, …) is often expressed in *different ways* in the same language.

One way may be more common than others,
and exceptions may depend on specific words:

- Most plural nouns: add *-s* to singular: book-book**s**, but: box-box**<u>es</u>**, fly-fl**<u>ies</u>**, child-child**<u>ren</u>**
- Most past tense verbs add *-ed* to infinitive: walk-walk**ed**, but: like-like**<u>d</u>**, leap-leap**<u>t</u>**

Such exceptions are called *irregular* word forms

Linguists say that there is one underlying morpheme (e.g. for plural nouns) that is "realized" as different "surface" forms (morphs) (e.g. -s/-es/-ren)

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

Side note: “Surface”?

This terminology comes from Chomskyan Transformational Grammar.

- Dominant early approach in theoretical linguistics, superseded by other approaches (“minimalism”).
- Not computational, but has some historical influence on computational linguistics (e.g. Penn Treebank)

“Surface” = standard English (Chinese, Hindi, etc.).

“Surface string” = a written sequence of characters or words

vs. “Deep”/“Underlying” structure/representation:

A more abstract representation.

Might be the same for different sentences/words with the same meaning.

Morphological parsing and generation

Morphological parsing

disgracefully

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

Morphological generation

We cannot enumerate all possible English words, but we would like to capture the rules that define whether a string *could* be an English word or not.

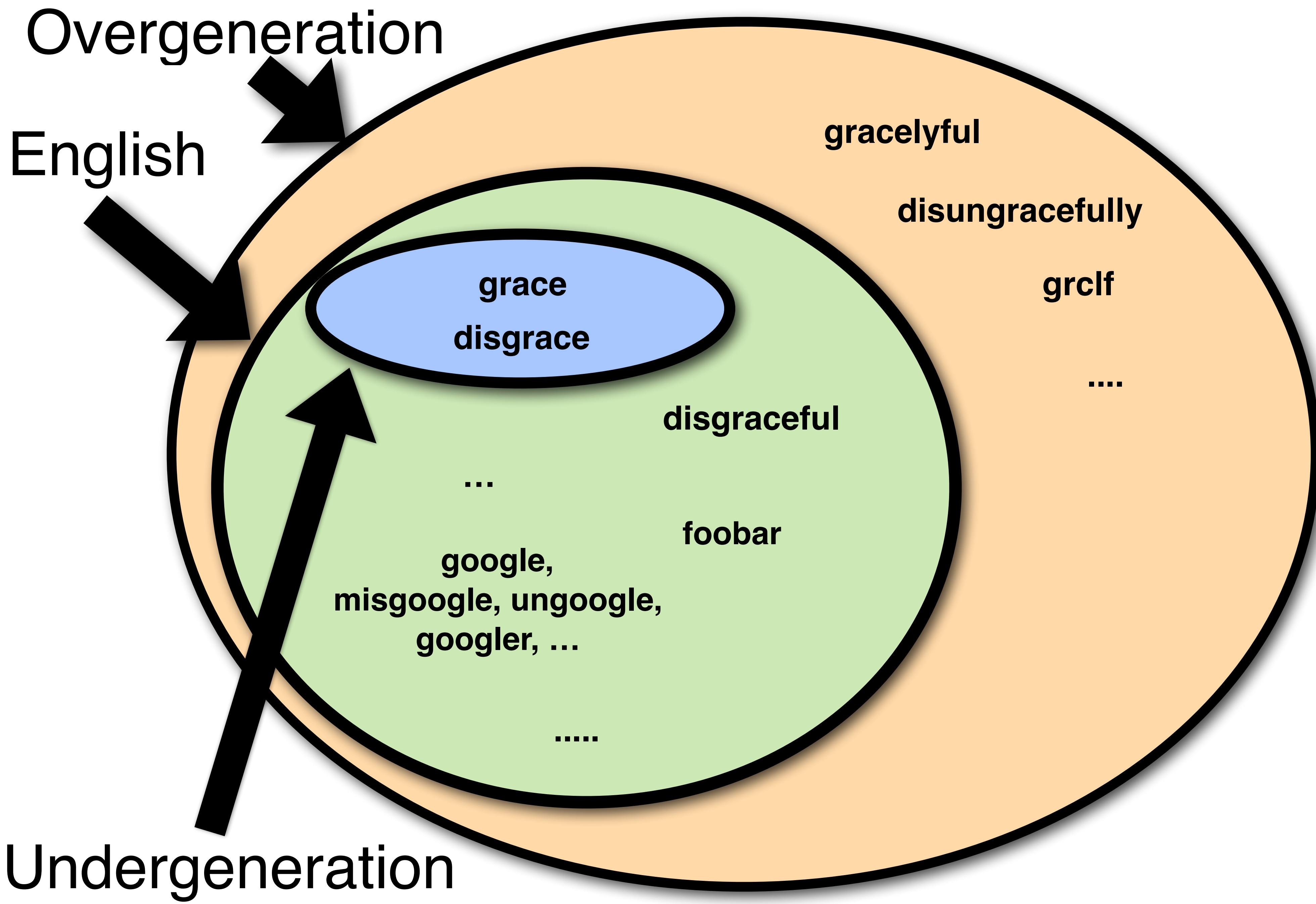
That is, we want a procedure that can generate (or accept) possible English words…

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

without generating/accepting impossible English words

*gracelyful, *gracefuly, *disungracefully,…

NB: * is linguists' shorthand for "this is ungrammatical"


Overgeneration
English
Undergeneration
gracelyful
disungracefully
grclf
....
grace
disgrace
disgraceful
...
foobar
google,
misgoogle, ungoogle,
googler, ...
.....

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 $\sum$:

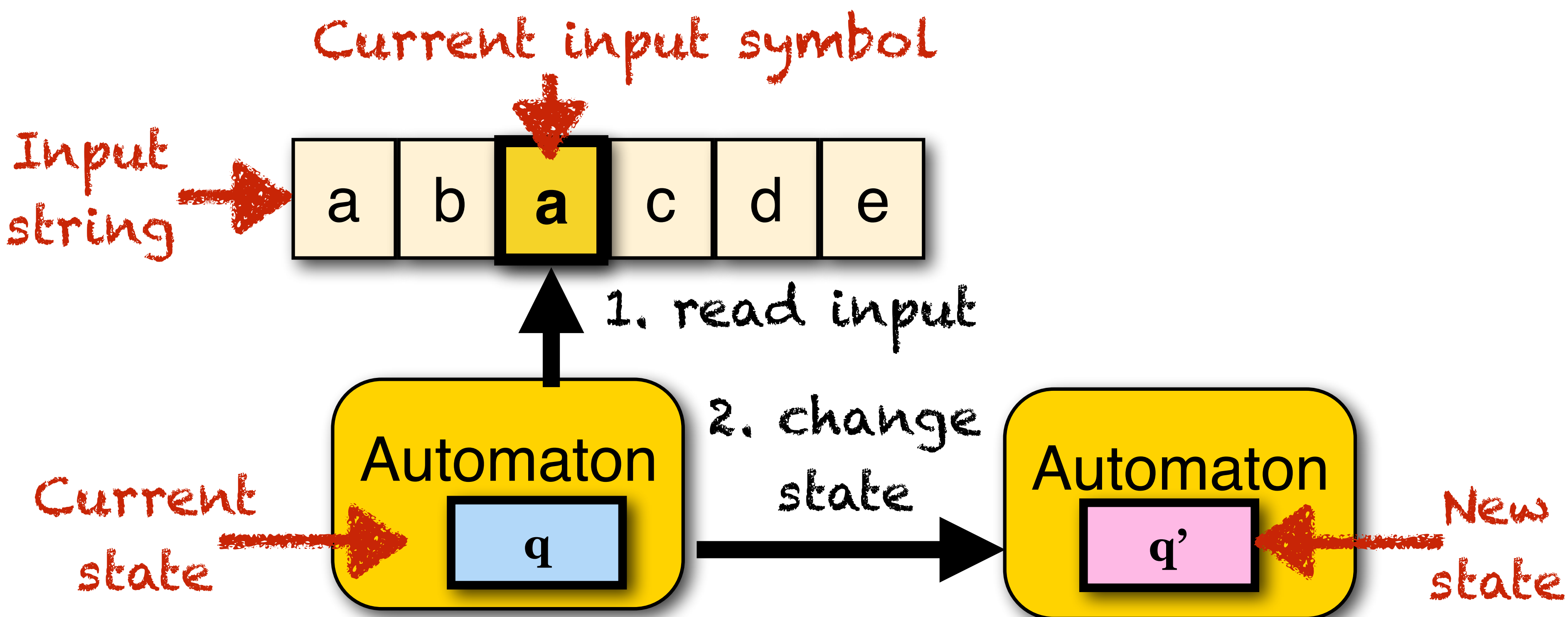
$\sum^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, ...\}$

A language $\mathrm{L} \subseteq \sum^*$ 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.
It *reads* an input string symbol by symbol.
It *changes* its internal state depending on
the current input symbol and its current internal state.

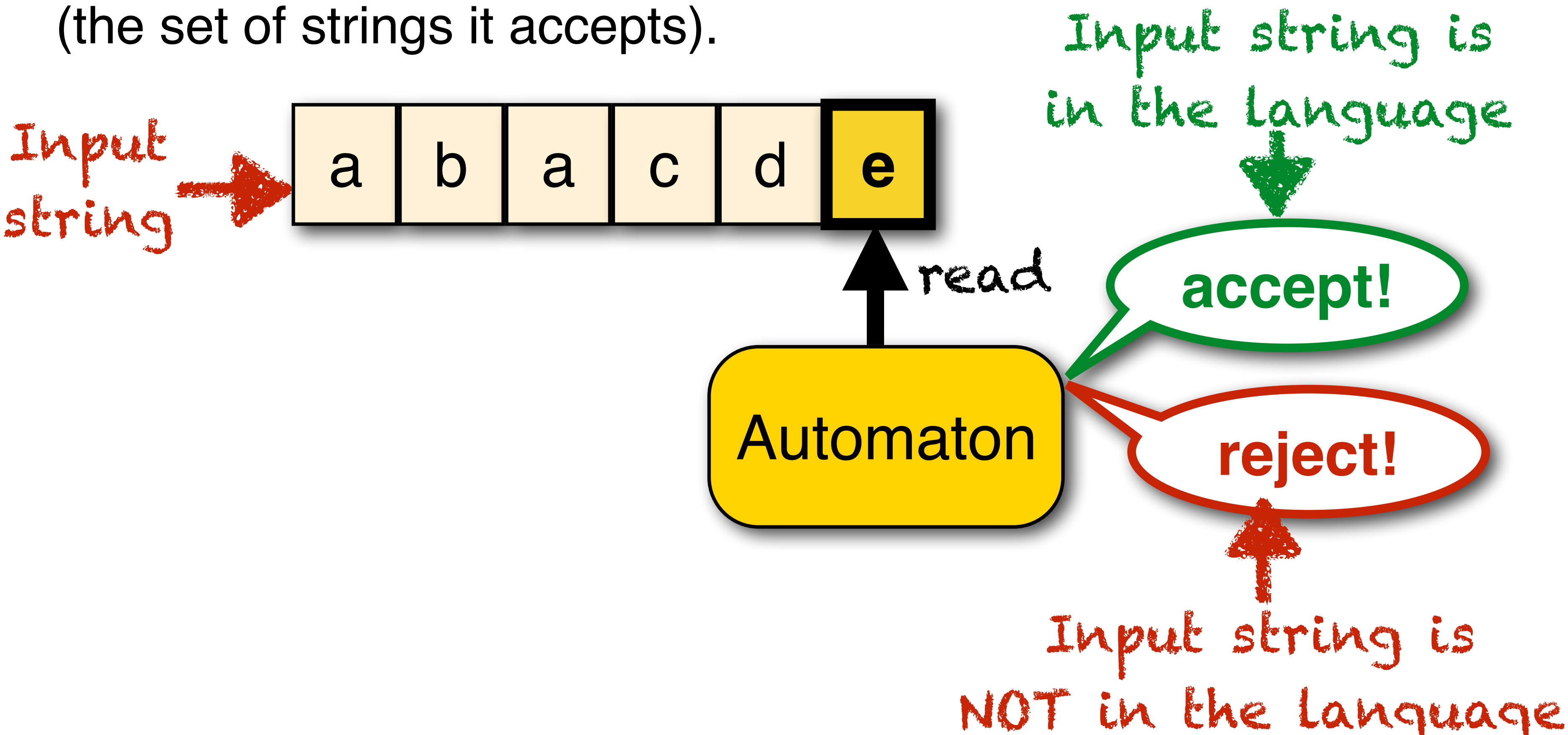


Automata and languages

The automaton either accepts or rejects the input string.

Every automaton defines a language

(the set of strings it accepts).



Automata and languages

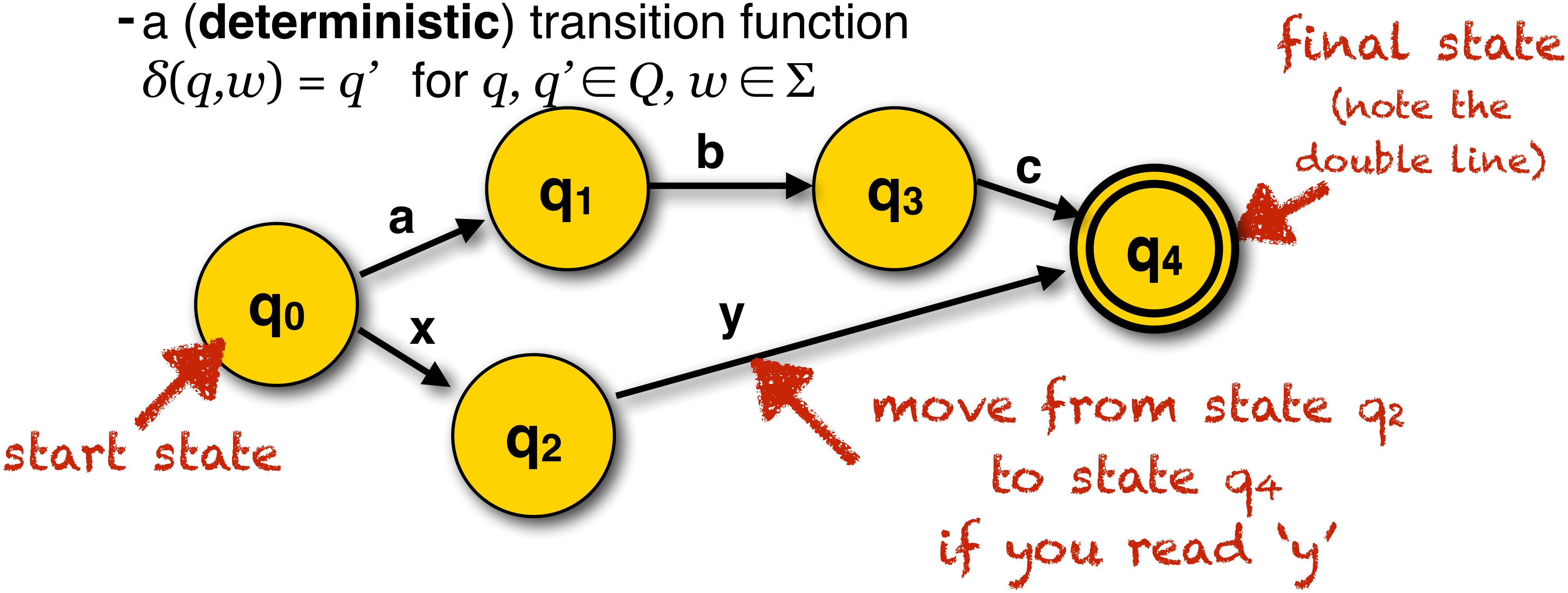
Different types of automata define
different language classes:

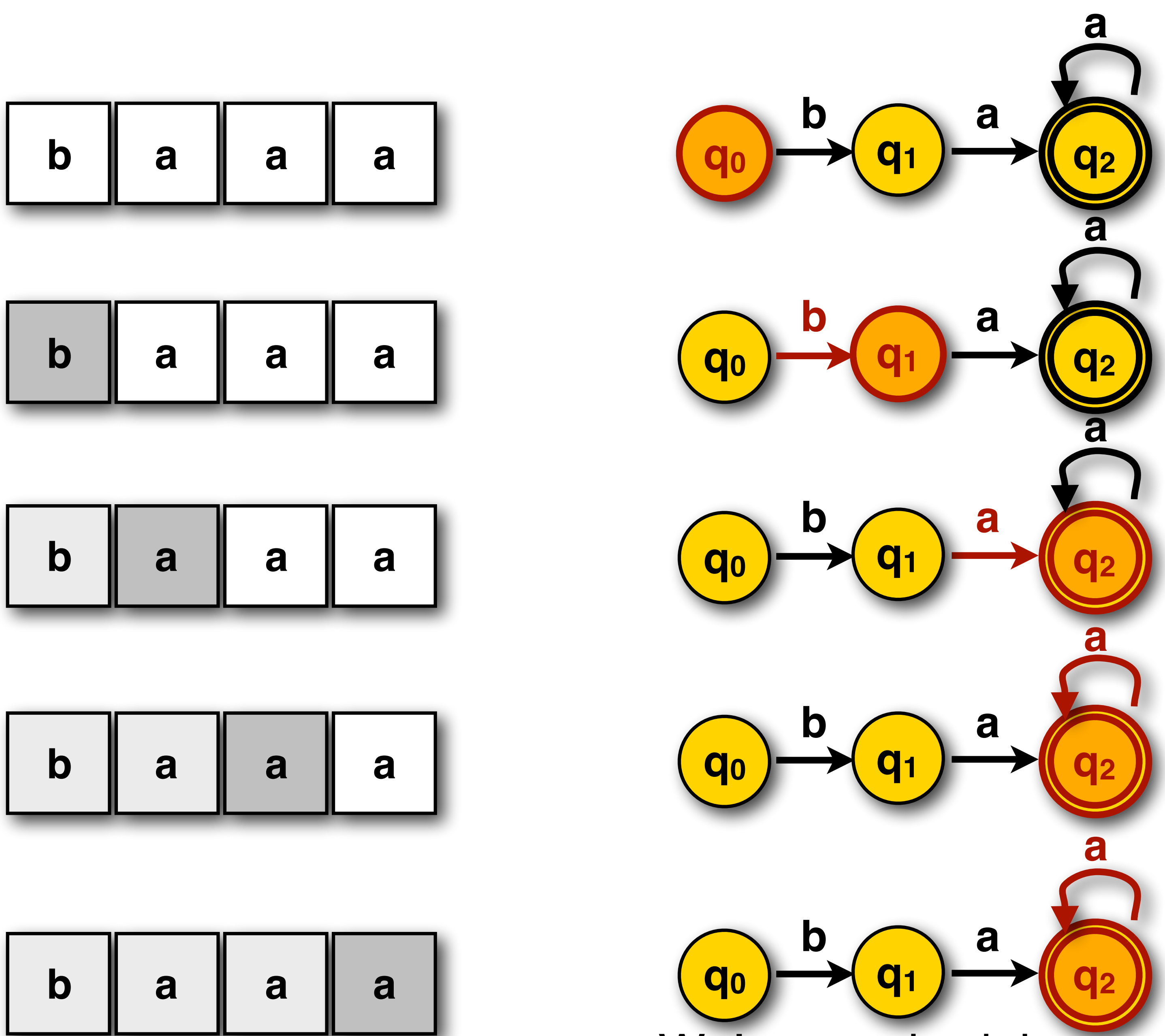
- **Finite-state** automata define **regular** languages
- **Pushdown** automata define **context-free** languages
- **Turing machines** define **recursively enumerable** languages

Finite-state automata

A (deterministic) finite-state automaton (FSA) consists of:

- a **finite set of states** $Q = \{q_0 \ldots q_N\}$, including **a start state** q_0 and one (or more) **final (=accepting) states** (say, q_N)
- a (**deterministic**) transition function $\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$



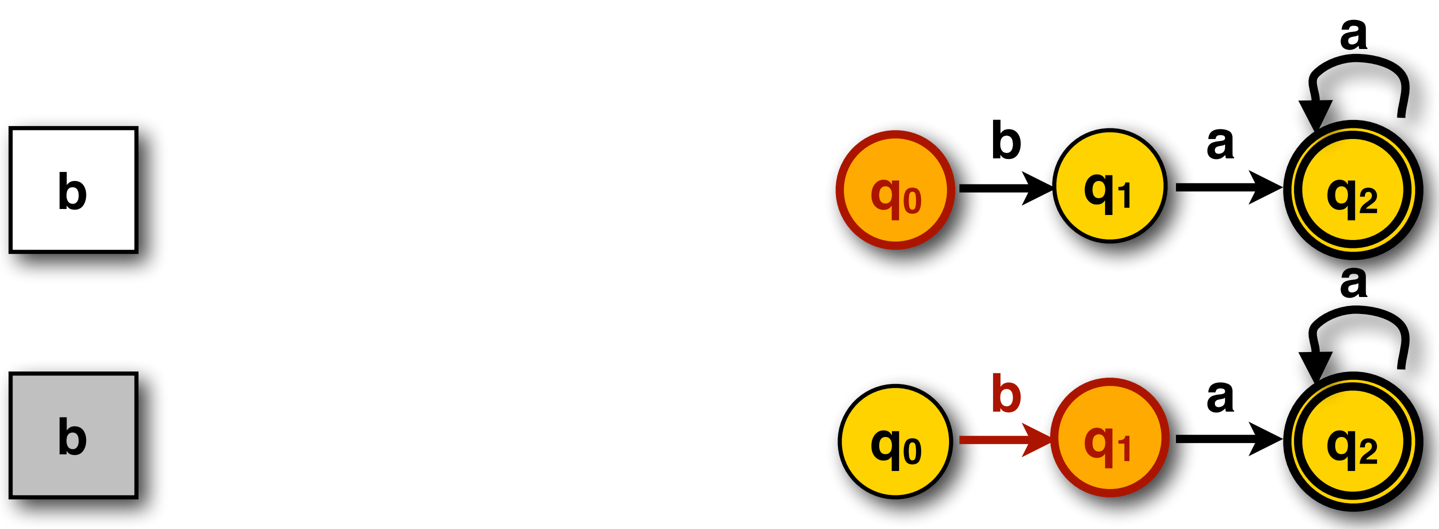


Start in q_0

Accept!

We've reached the end of the string, and are in an accepting state.

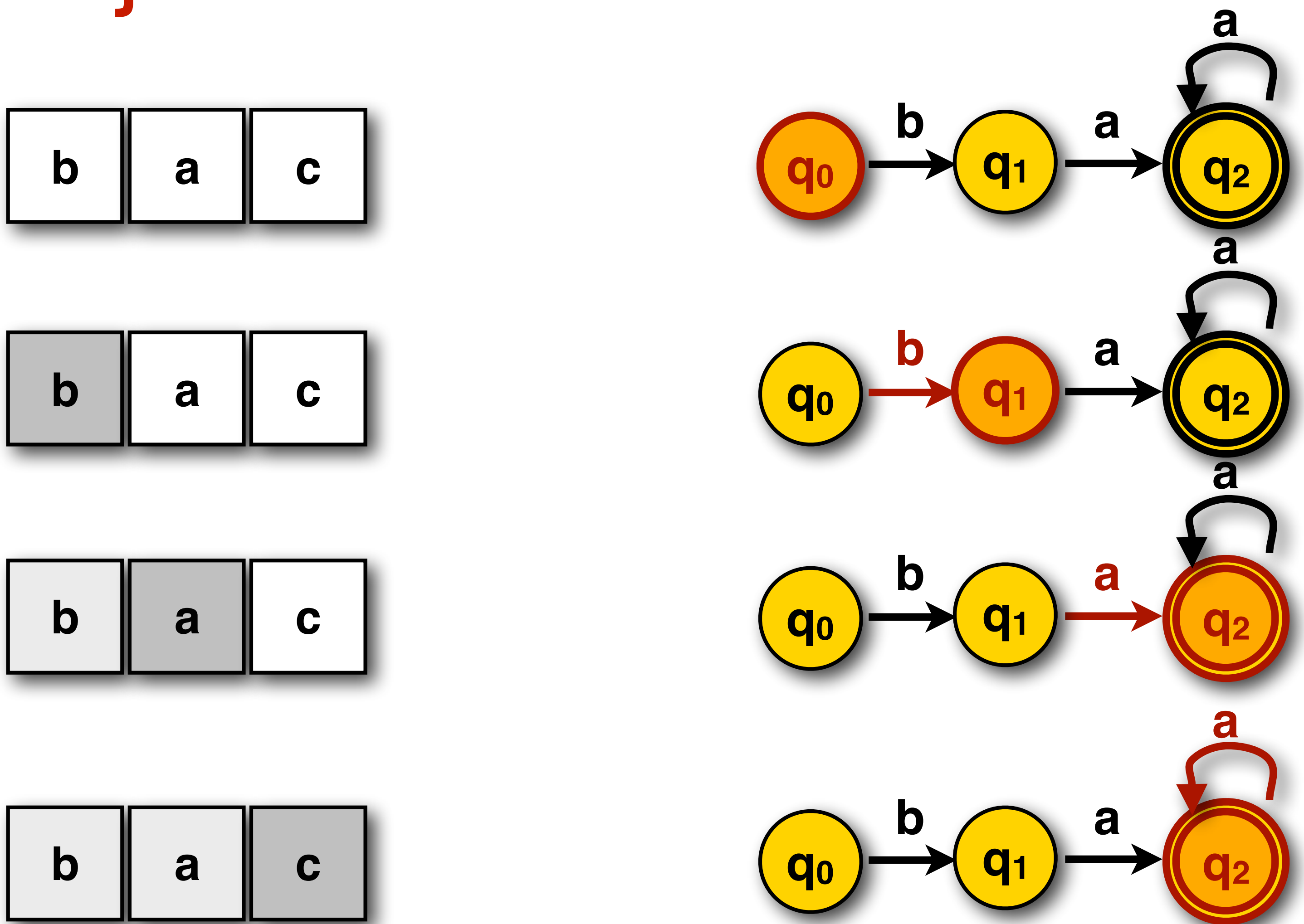
Rejection: Automaton does not end up in accepting state



Start in q_0

Reject!
(q_1 is not a final state)

Rejection: Transition not defined



Start in q_0

Reject! (There is no transition labeled ‘c’)

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 $q, q' \in Q, w \in \Sigma$

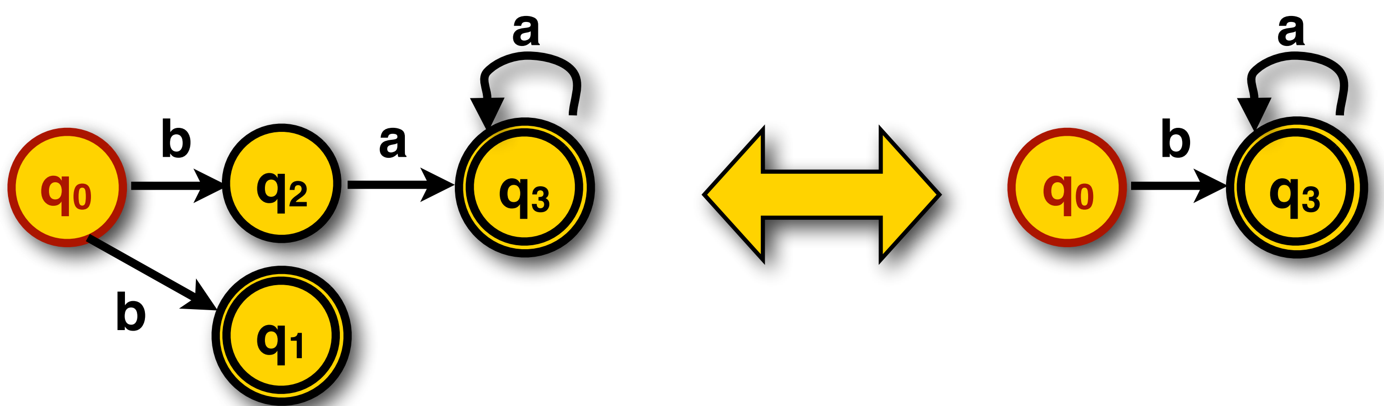
    If the current state is q and the current input is w, go to q'
  - The transition function for a nondeterministic (N)FSA: $Q \times \Sigma \rightarrow 2^Q$

    $\delta(q,w) = Q'$ 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.

Regular Expressions

Regular expressions can also be used to define a regular language.

Simple patterns:

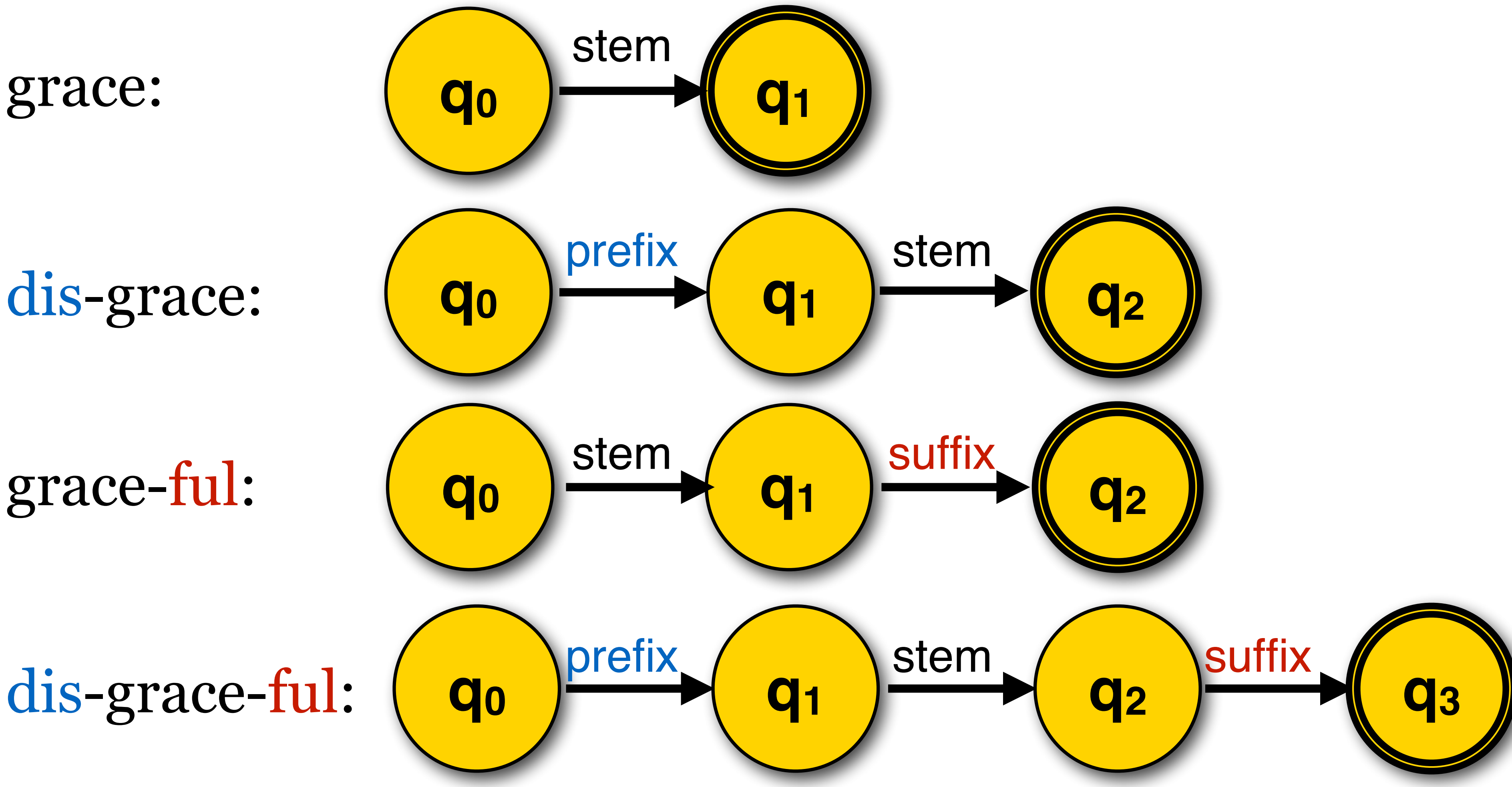
- **Standard characters** match themselves: *'a'*, '1'
- **Character classes**: '[*abc*]', '[0-9]', **negation:** '[^*aeiou*]'
  (Predefined: *s* (whitespace), *w* (alphanumeric), etc.)
- **Any character** (except newline) is matched by '.'

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** '$'

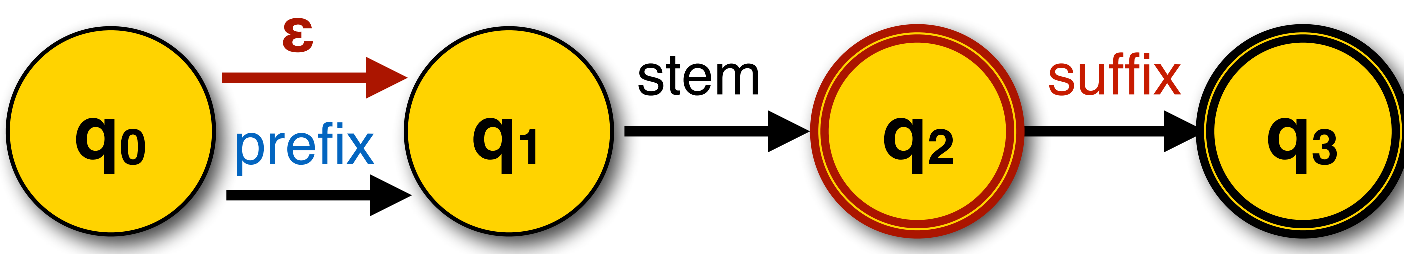
Finite-state methods for morphology

Finite state automata for morphology

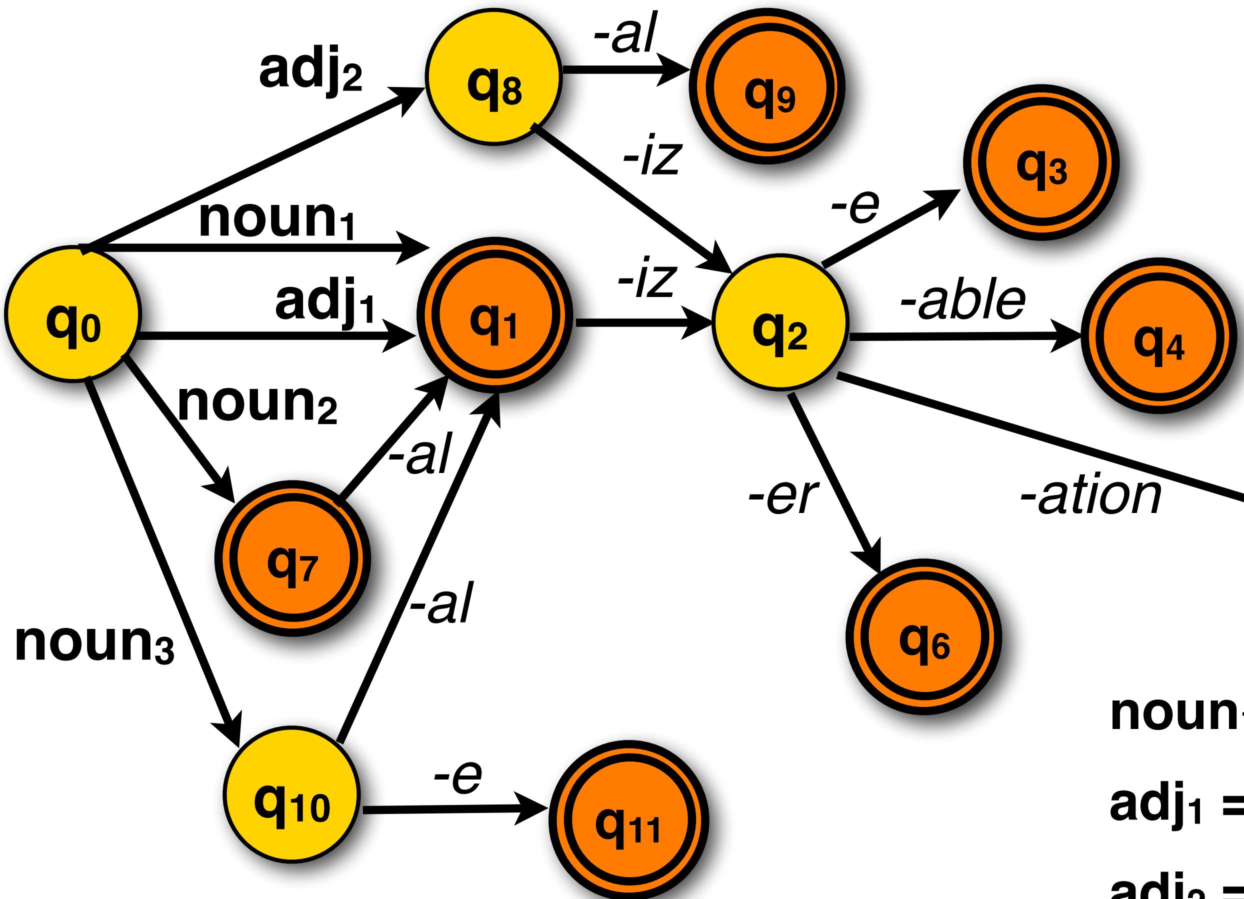


Union: merging automata

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



FSAs for derivational morphology



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

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

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

noun$_2$ = *{nation, form,…}*

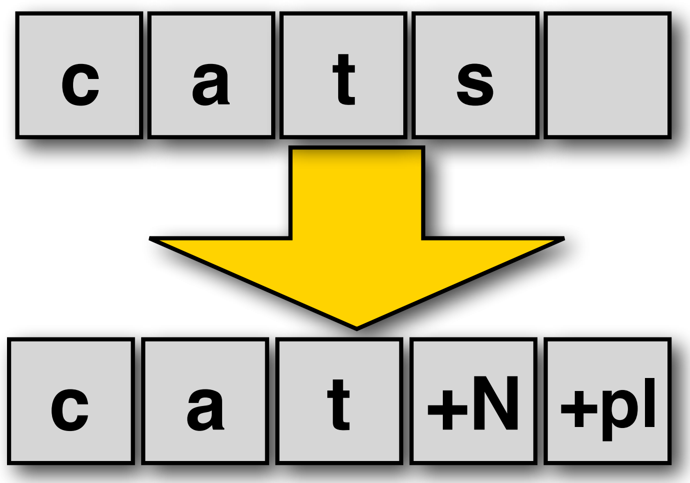
noun$_3$ = *{natur, structur,…}*

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:

Input (Surface form)



Output (Lexical form)

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.
  (NB: Jurafsky&Martin define $\sigma: Q \times \Sigma^* \rightarrow \Delta^*$. Why is this equivalent?)

Finite-state transducers

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

L_{in} = {**cat, cats, fox, foxes, ...**}

L_{out} = {*cat+N+sg, cat+N+pl, fox+N+sg, fox+N+pl ...*}

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

Peculiarities of English spelling (orthography)

The same underlying morpheme (e.g. *plural-s*) can have different orthographic "surface realizations" (-s, -es)

This leads to 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

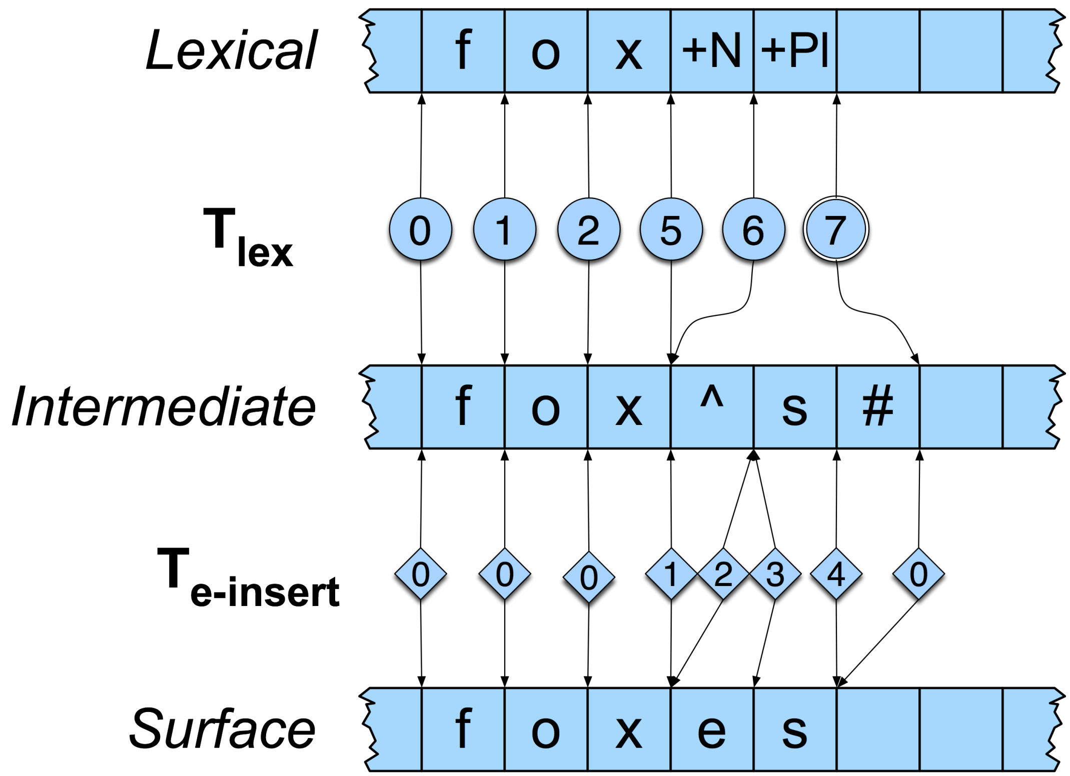
We define an intermediate representation to capture morpheme boundaries (^) and word boundaries (#):

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

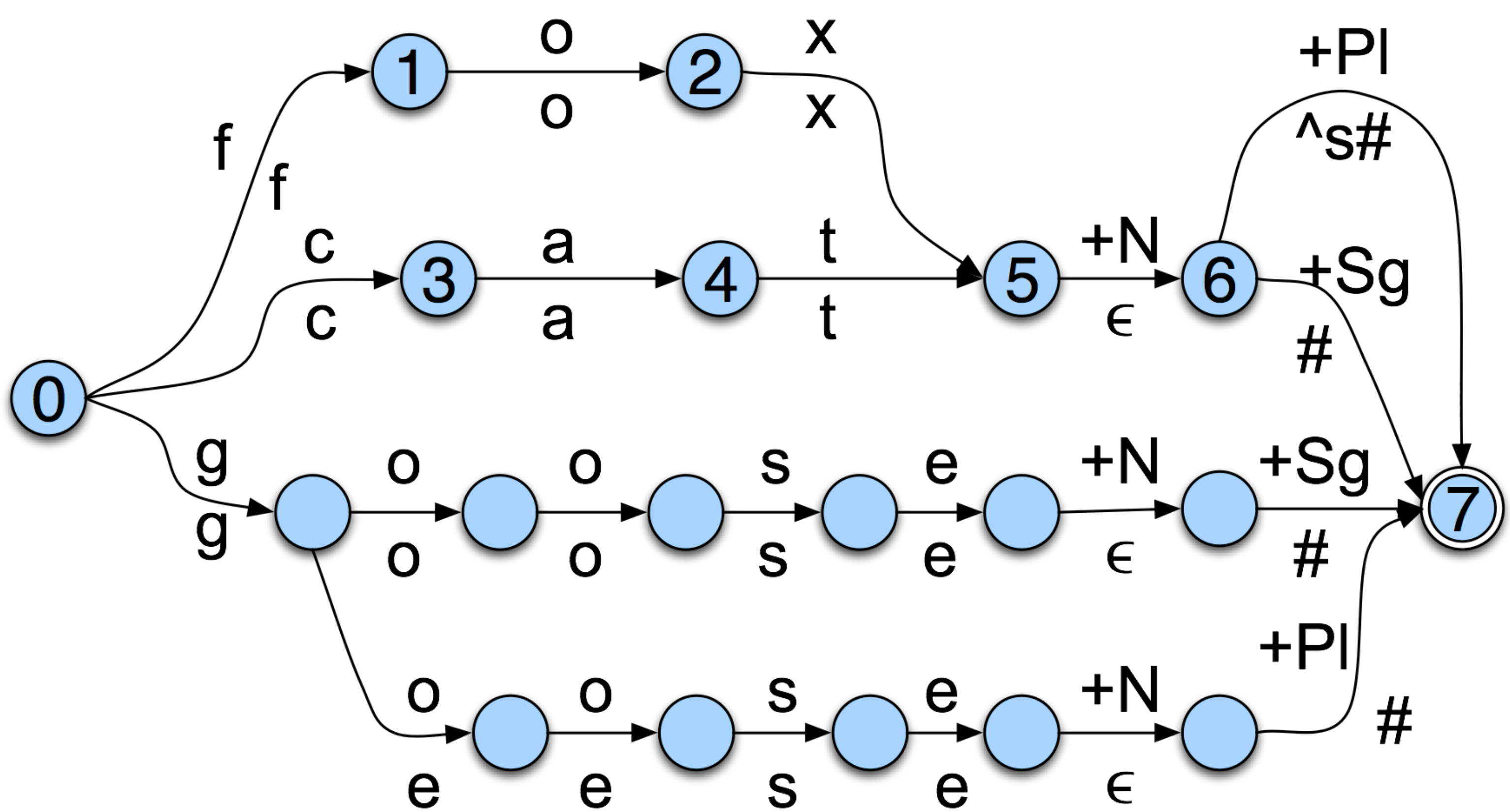
Intermediate-to-Surface Spelling Rule:

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

FST composition/cascade:

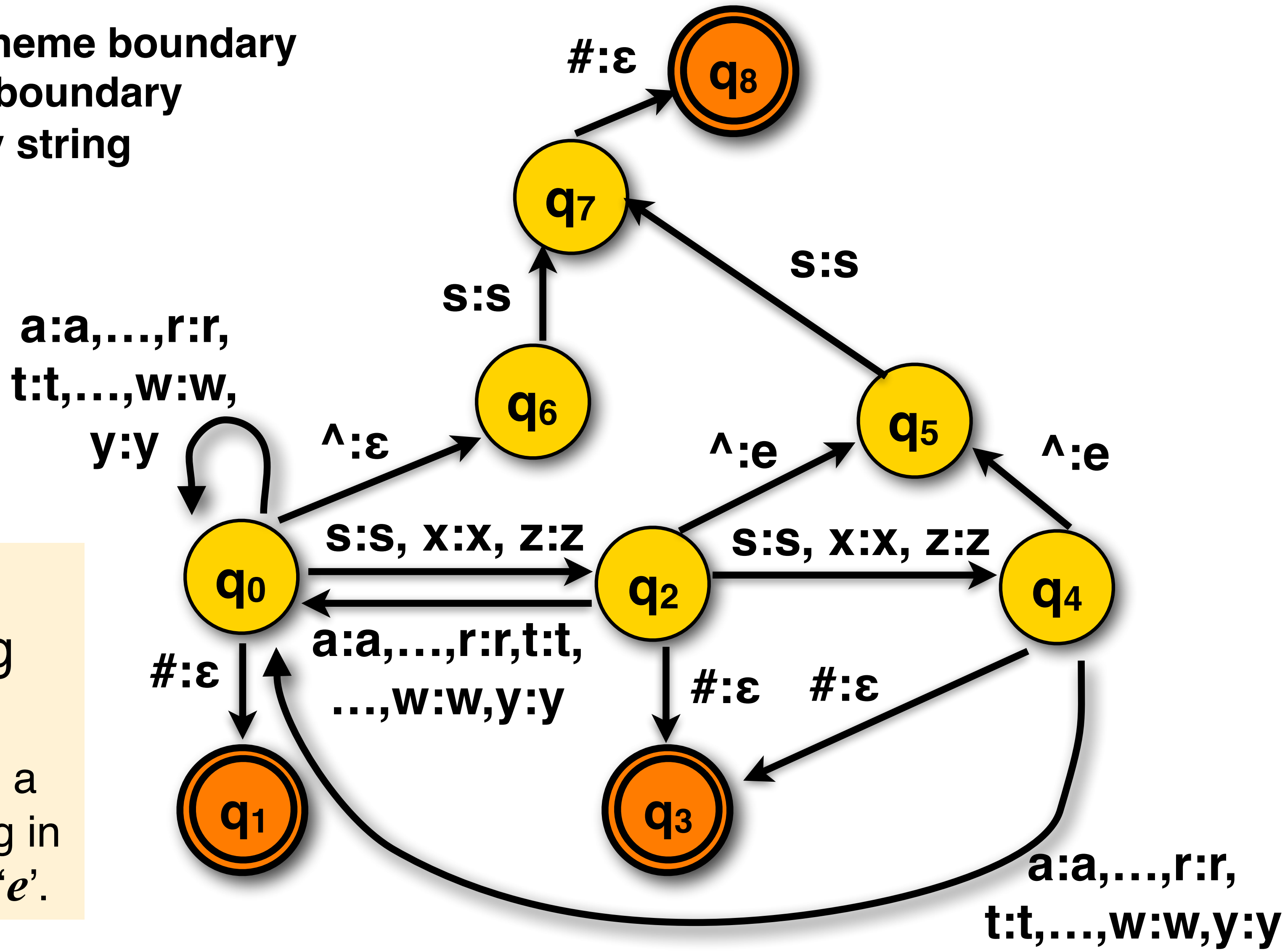


T_{lex}: Lexical to intermediate level



$T_{e\text{-}insert}$: intermediate to surface level

^ = morpheme boundary
# = word boundary
ε = empty string



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.

We need a nondeterministic FST.

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

We also need a scoring function to identify which analysis is more likely.

- We may need to know the context in which the word appears: (I read a **book** vs. I **book** flights)

Other applications of FSTs

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 other NLP tasks?

Could we write an FST for machine translation?

What about compounds?

Semantically, 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.

Today’s key concepts

Morphology (word structure): stems, affixes
Derivational vs. inflectional morphology
Compounding
Stem changes
Morphological analysis and generation

Finite-state automata
Finite-state transducers
Composing finite-state transducers

Today's reading

This lecture follows closely
Chapter 3.1-7 in J&M 2008

Optional readings (see website)
Karttunen and Beesley '05, Mohri (1997), the Porter stemmer, Sproat et al. (1996)