CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 2: Tokenization and Morphology

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Lecture 2:
What will we
discuss today?

Lecture 2: Overview

Today, we'll look at **words**:

— How do we identify words in text?
— Word frequencies and Zipf's Law
— What is a word, really?
— What is the structure of words?
— How can we identify the structure of words?

To do this, we'll need a bit of linguistics,
some data wrangling, and a bit of automata theory.

Later in the semester we'll ask more questions about words:

How can we identify different word classes (parts of speech)?
What is the meaning of words? How can we represent that?

Lecture 2: Reading

Most of the material is taken from Chapter 2 (3rd Edition)

I won't cover regular expressions (2.1.1) or edit distance (2.5), because I assume you have all seen this material before.
I you aren't familiar with regular expressions, read this section because it's very useful when dealing with text files!

The material on finite-state automata, finite-state transducers and morphology is from the 2nd Edition of this textbook, but everything you need should be explained in these slides.

Lecture 2: Key Concepts

You should understand the distinctions between
— Word forms vs. lemmas
— Word tokens vs. word types
— Finite-state automata vs. finite-state transducers
— Inflectional vs. derivational morphology

And you should know the implications of Zipf's Law for NLP (coverage!)

Lecture 2: Tokenization

Tokenization: Identifying word boundaries

Text is just a sequence of characters:

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

How do we split this text into words and sentences?

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

How do we identify the words in a text?

For a language like English, this *seems* like a really easy problem:

A word is any sequence of alphabetical characters between whitespaces that's not a punctuation mark?

That works to a first approximation, but…

… what about abbreviations like *D.C.*?
… what about complex names like *New York*?
… what about contractions like *doesn't* or *couldn't've*?
… what about *New York-based* ?
… what about names like *SARS-Cov-2*, or R2-D2?
… what about languages like Chinese that have no whitespace, or languages like Turkish where one such "word" may express as much information as an entire English sentence?

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: Contractions and Clitics

English: "doesn't" , "I'm" ,

Italian: "dirglielo" = dir + gli(e) + lo
tell + him + it

Tokenization Standards

Any actual NLP system will assume a particular tokenization standard.

Because so much NLP is based on systems that are trained on particular corpora (text datasets) that everybody uses, these corpora often define a de facto standard.

Penn Treebank 3 standard:

Input:

"The San Francisco-based restaurant,"
they said, "doesn't charge $10".

Output:

“_ The _ San _ Francisco-based _ restaurant _ , _” _
they_ said_ ,_ "_ does _ n’t _ charge_ $_ 10 _ " _ . _

Aside: What about sentence boundaries?

How can we identify that this is two sentences?

Mr. Smith went to D.C. Ms. Xu went to Chicago instead.

Challenge: punctuation marks in abbreviations (Mr., D.C, Ms,...)

[It's easy to handle a small number of known exceptions,
but much harder to identify these cases in general]

See also this headline from the NYT (08/26/20):

Anthony Martignetti ('Anthony!'), Who Raced Home for Spaghetti, Dies at 63

How many sentences are in this text?

"The San Francisco-based restaurant," they said, "doesn't charge $10".

Answer: just one, even though "they said" appears in the middle of another sentence.

Similarly, we typically treat this also just as one sentence:

They said: "The San Francisco-based restaurant doesn't charge $10".

Spelling variants, typos, etc.

The same word can be written in different ways:

- with different **capitalizations**:
  lowercase "cat" (in standard running text)
  capitalized "Cat" (as first word in a sentence, or in titles/headlines),
  all-caps "CAT" (e.g. in headlines)
- with different **abbreviation** or **hyphenation** styles:
  US-based, US based, U.S.-based, U.S. based
  US-EU relations, U.S./E.U. relations, …
- with **spelling variants** (e.g. regional variants of English):
  labor vs labour, materialize vs materialise,
- with **typos** (teh)

Good practice: Be aware of (and/or document) any normalization (lowercasing, spell-checking, …) your system uses!

Lecture 2: Word Frequencies and Zipf's Law

Counting words: tokens vs types

When counting words in text, we distinguish between word types and word tokens:

— The vocabulary of a language
is the set of (unique) word types:
V = {a, aardvark, …., zyzzva}

— The tokens in a document include all occurrences of the word types in that document or corpus
(this is what a standard word count tells you)

— The frequency of a word (type) in a document
= the number of occurrences (tokens) of that type

How many different words are there in English?

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

Vocabulary size = the number 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.

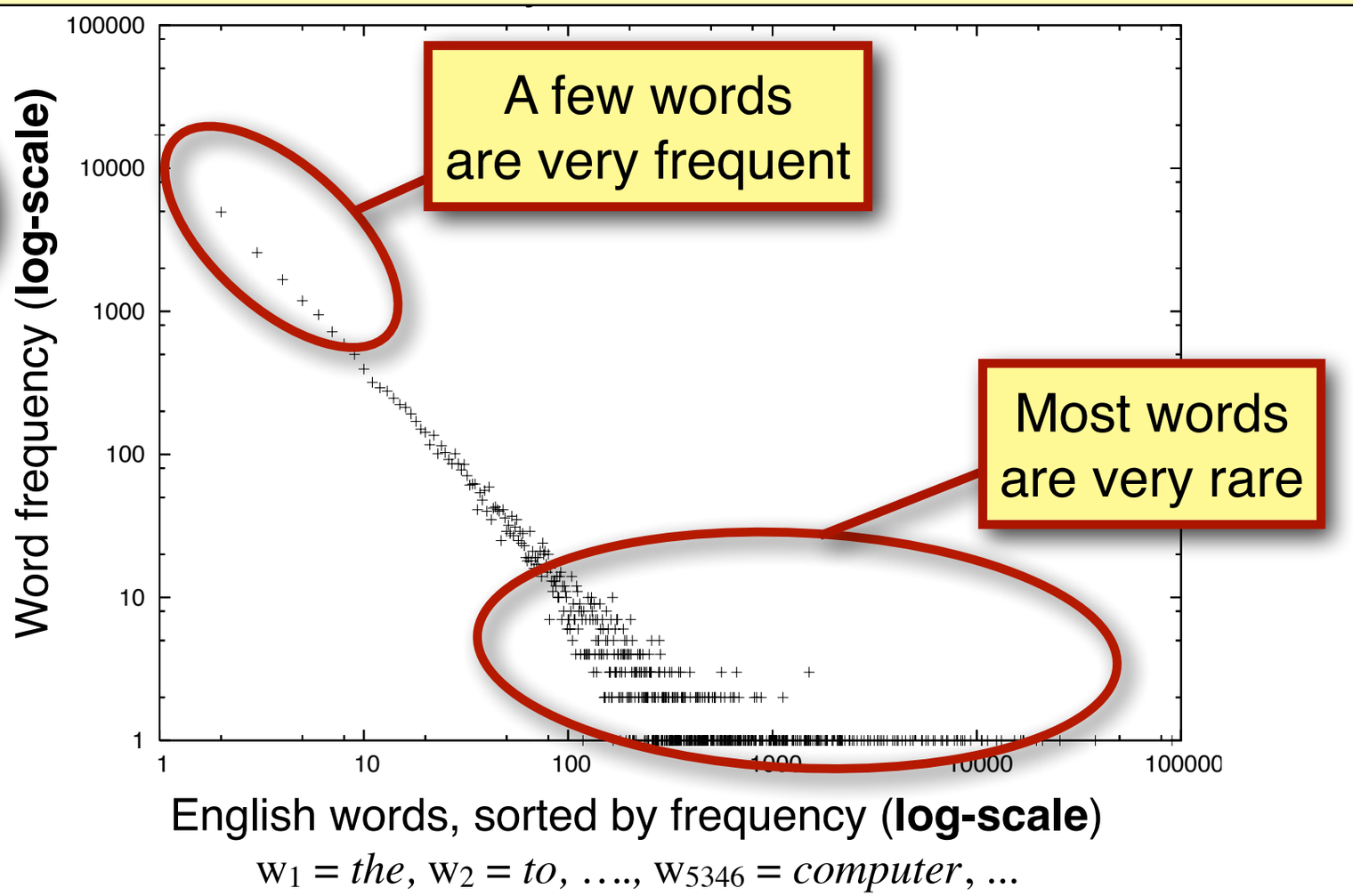
… even if you've read a lot of text,
you will keep finding words you haven't seen before.

Word frequency: the number of occurrences of a word type in a text (or in a collection of texts)

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 $P(w_r) \propto 1/r$



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**

- Each (surface) word form is its own symbol
- Add some generalization by mapping different forms of a word to the same symbol
  - **Normalization:** map all variants of the same word (form) to the same canonical variant (e.g. lowercase everything, normalize spellings, perhaps spell-check)
  - **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)

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 (`"book"`)
plus morphological information (`"N pl"` i.e. plural noun)

This is particularly useful for highly inflected languages, e.g. Czech, Finnish, Turkish, etc. (less so for English or Chinese):

In Czech, you might need to know that *nejnezajímavějším*
is a regular, feminine, plural, dative adjective in the superlative.

How do we represent unknown words?

Many NLP systems assume a fixed vocabulary, but still have to handle **out-of-vocabulary (OOV)** words.

Option 1: the **UNK** token

Replace all rare words (with a frequency at or below a given threshold, e.g. 2, 3, or 5) in your training data with an UNK token (UNK = “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

[often used in neural models]

Represent (rare and unknown) words [“Champaign”] as sequences of characters [‘C’, ‘h’, ‘a’,…,’g’, ’n'] or substrings [“Ch”, “amp”, “ai”, “gn”]

Byte Pair Encoding (BPE): learn which character sequences are common in the vocabulary of your language, and treat those common sequences as atomic units of your vocabulary

Lecture 2: What is a word, really?

How many different words are there in English?

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

Vocabulary size = the number of distinct word types

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

[here, we're treating inflected forms (took, taking) as distinct]

You may have heard statements such as
"adults know about 30,000 words"
"*you need to know at least 5,000 words to be fluent*"

Such statements do not refer to inflected word forms (take/takes/taking/take/takes/took) but to lemmas or dictionary forms (take), and assume if you know a lemma, you know all its inflected forms too.

Which words appear in this text?

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

Actual text doesn't consist of dictionary entries:

wants is a form of want

took is a form of take

courses is a form of course

Linguists distinguish between

— the **(surface) forms** that occur in text:

want, wants, beginners', took,...

— and the **lemmas** that are the uninflected forms of these words:

want, beginner, take, ...

In NLP, we sometimes map words to lemmas (or simpler "stems"), but the raw data always consists of surface forms

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,…

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

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**es**, fly-fl**ies**, child-child**ren**
- Most past tense verbs add ***-ed*** to infinitive: walk-walk**ed**,
  but: like-like**d**, leap-leap**t**

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.

Lecture 2: 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 $\sum$:

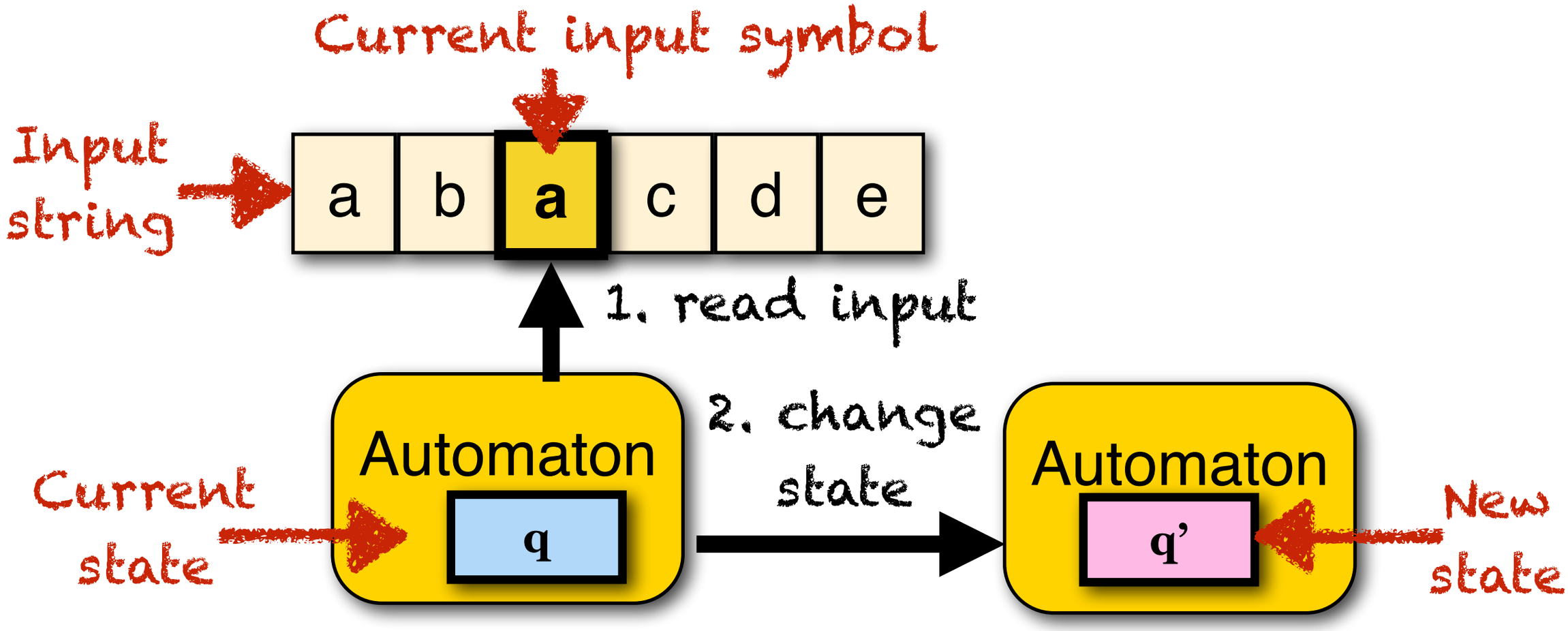
$\sum^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, ...\}$

A language $L \subseteq \sum^*$ 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.

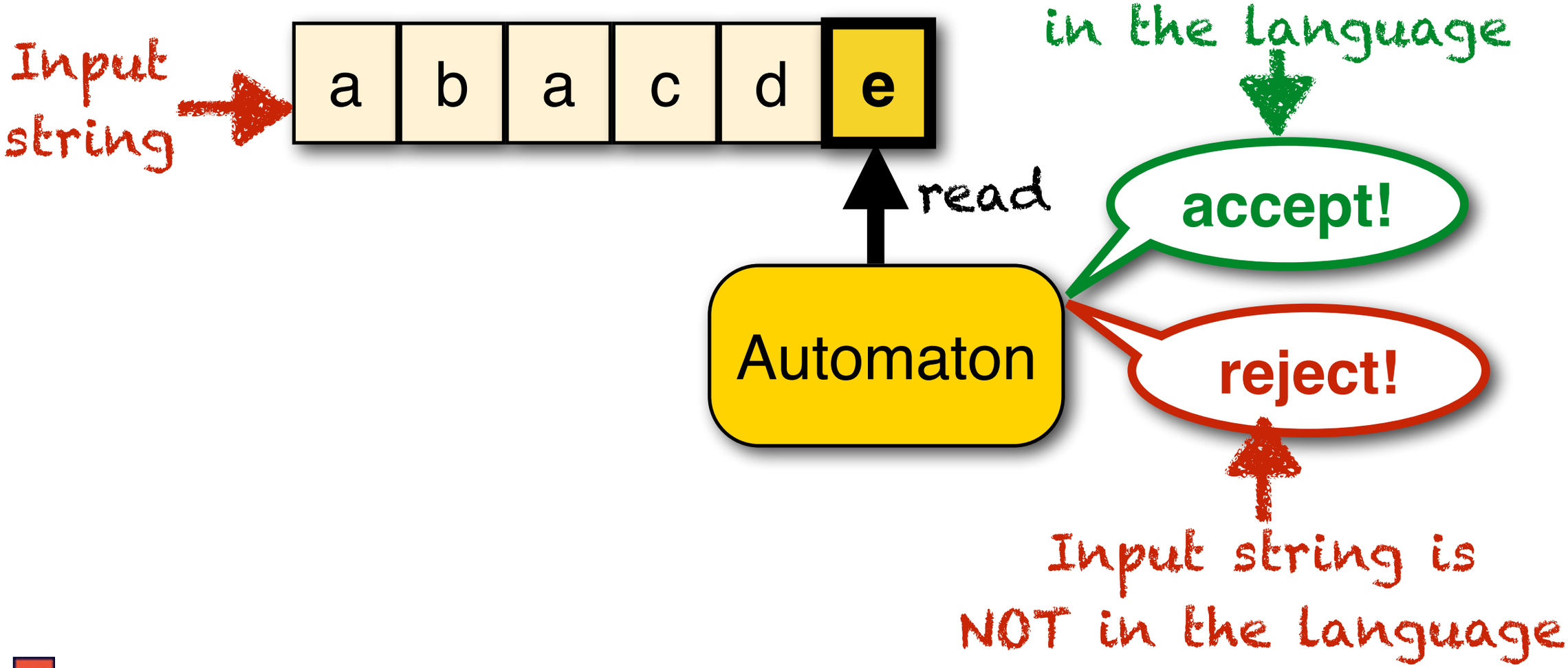


Automata and languages

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

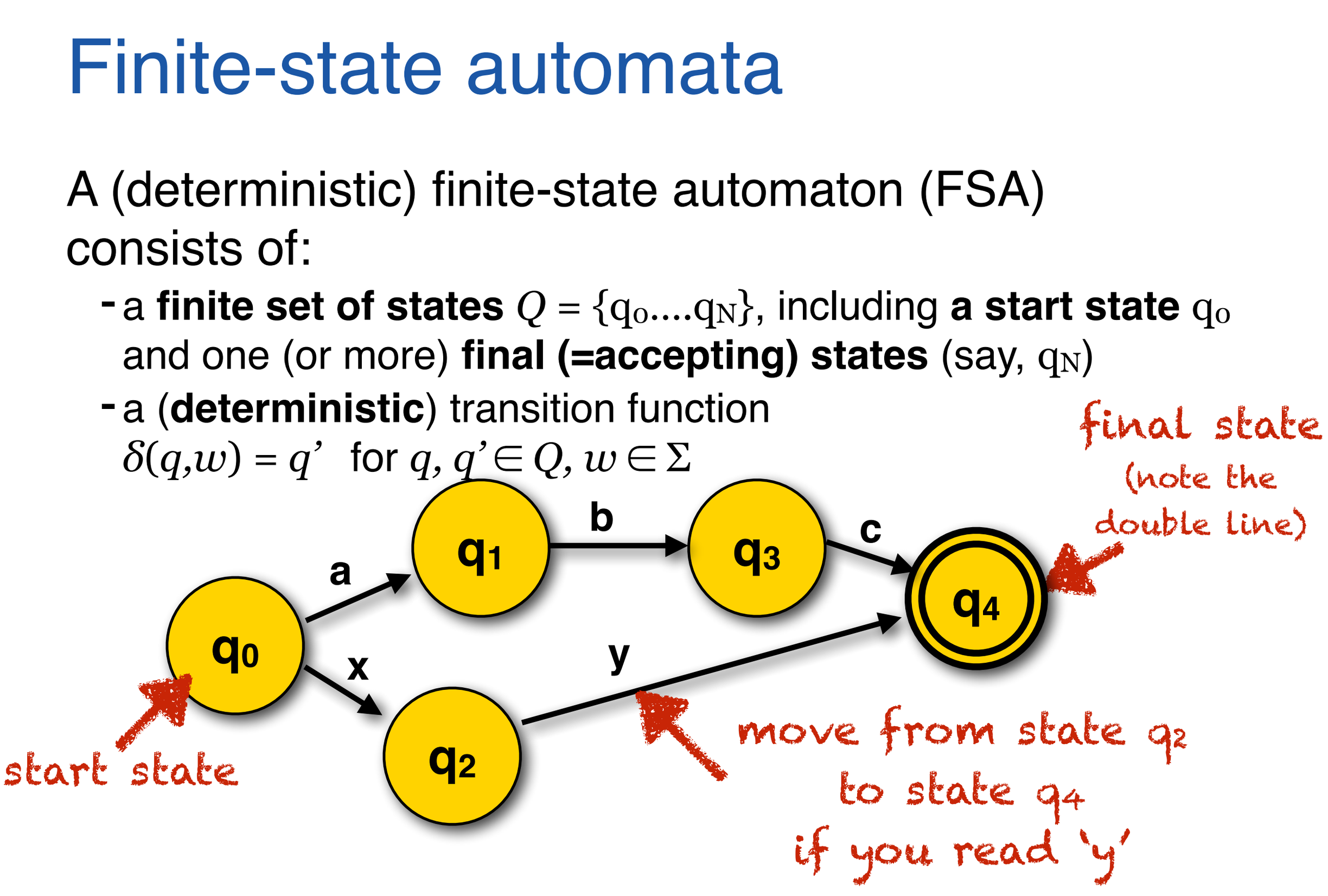
- **Pushdown** automata define **context-free** languages

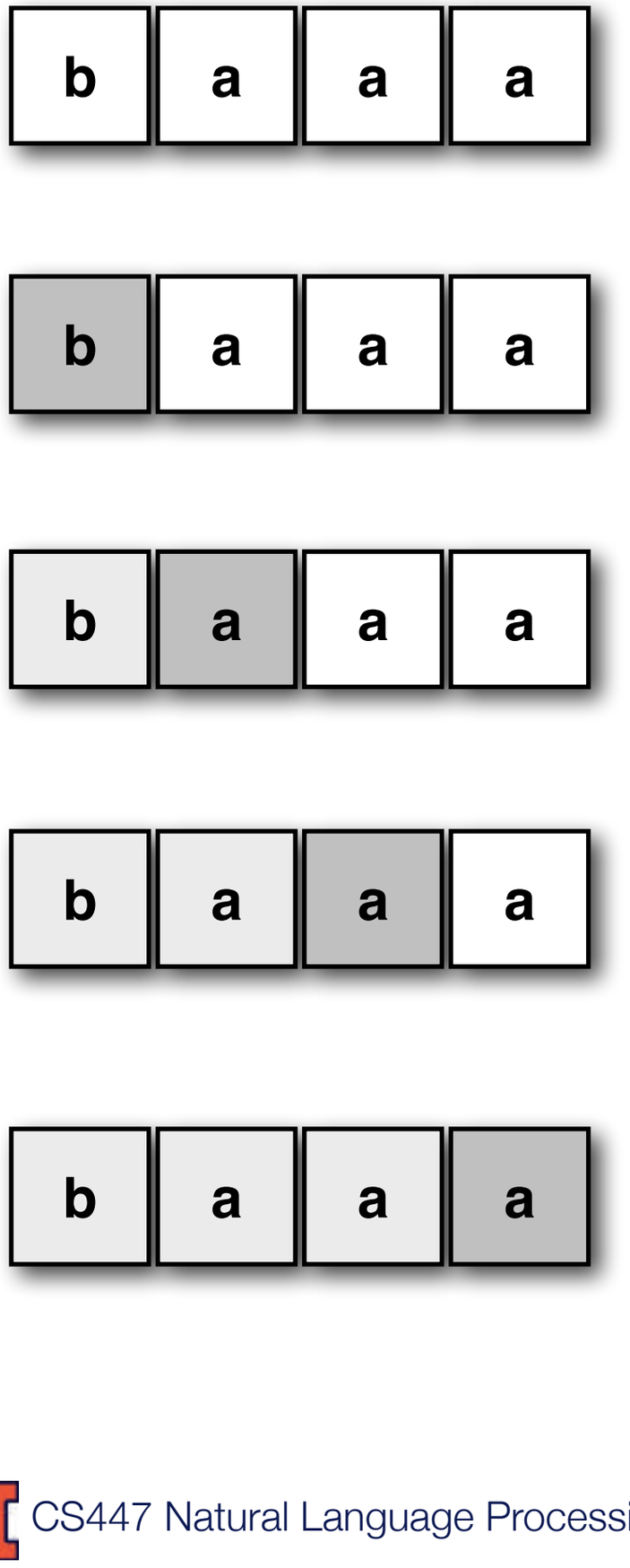
- **Turing machines** define **recursively enumerable** languages

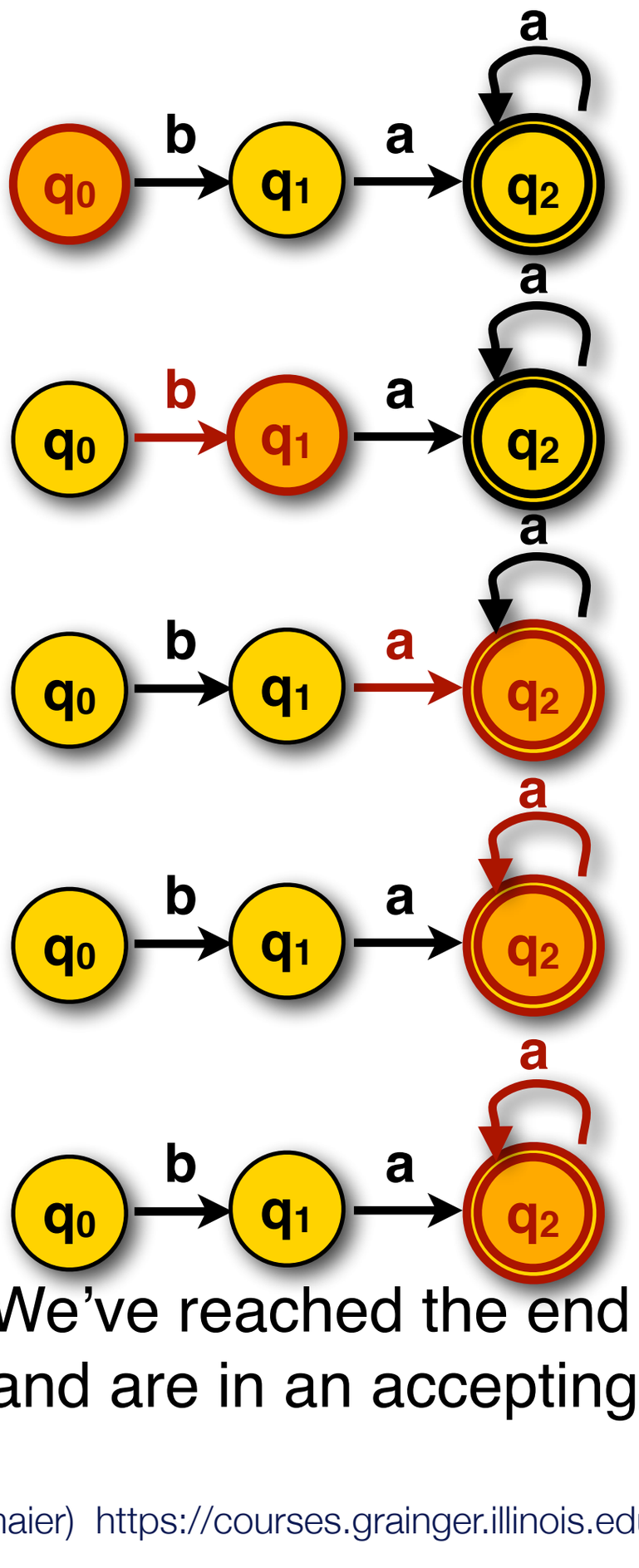
Finite-state automata

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







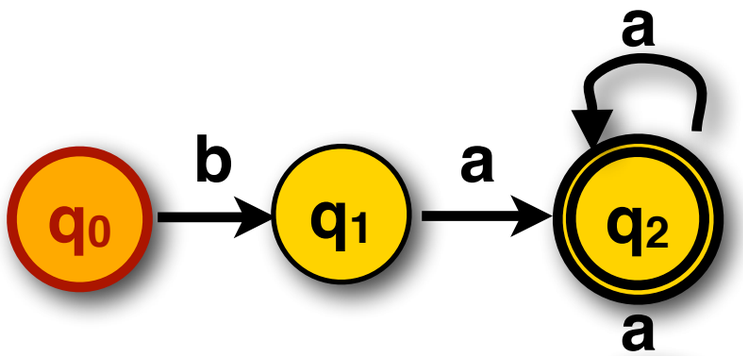
Start in q_0

Accept!

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

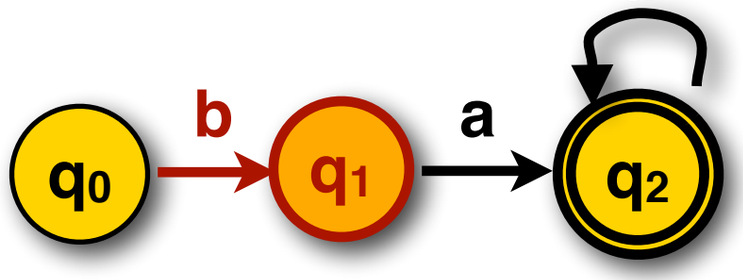
Rejection: Automaton does not end up in accepting state

b



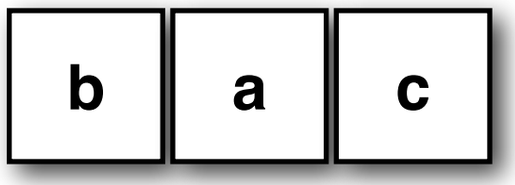
Start in q_0

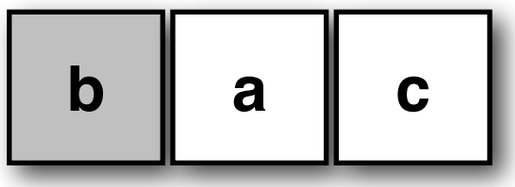
b

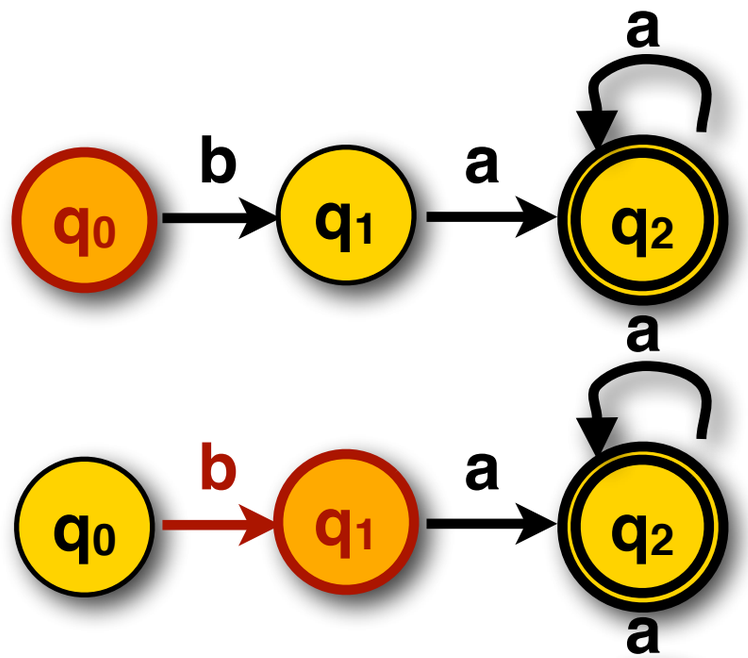


Reject!
(q_1 is not a final state)

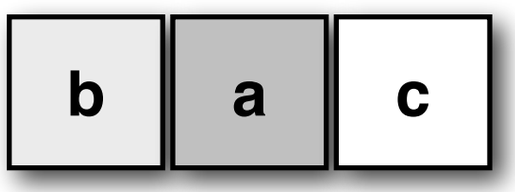
Rejection: Transition not defined

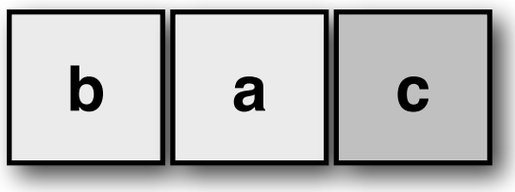


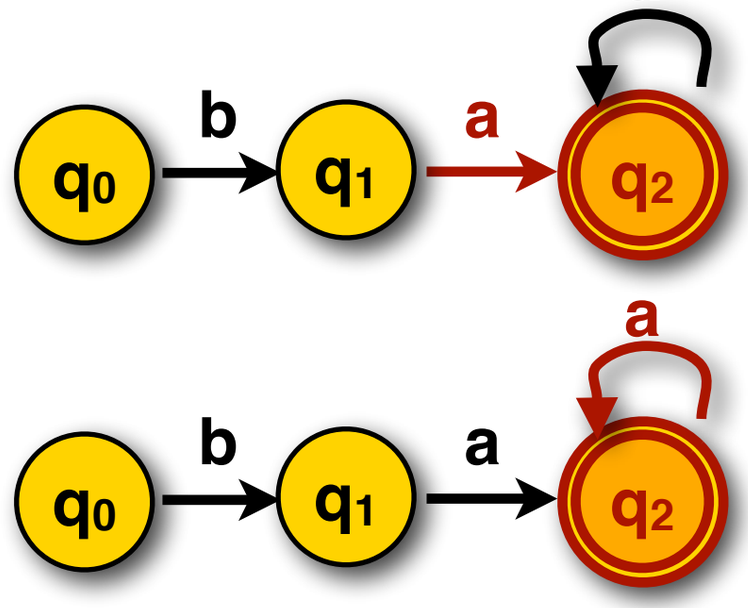




Start in q_0



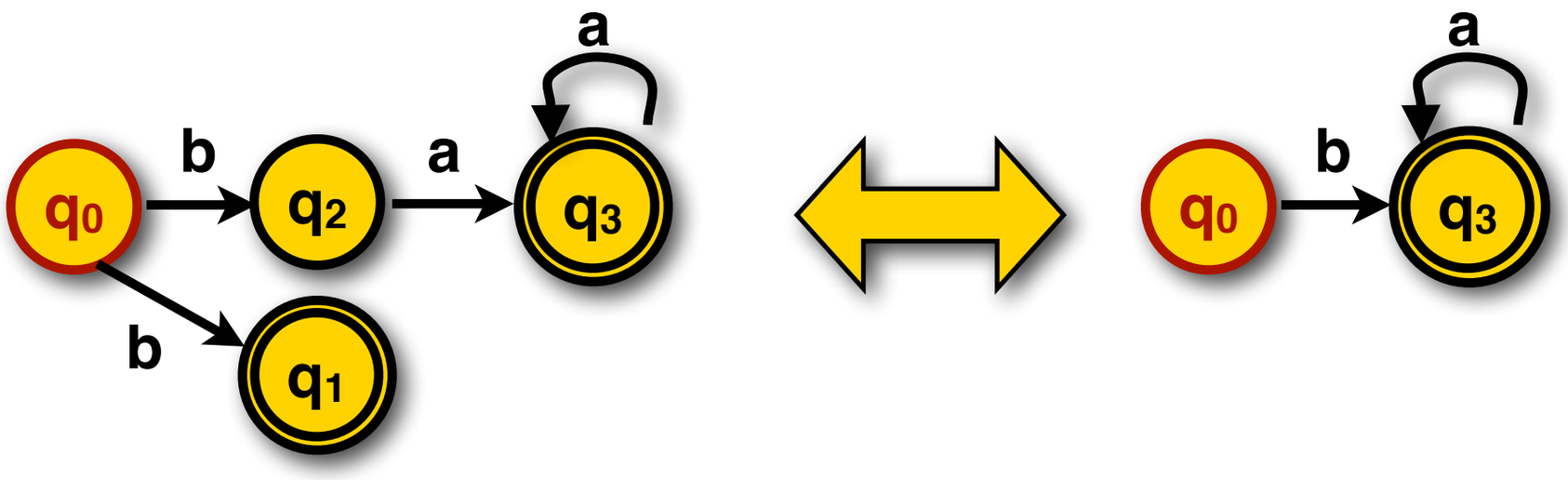




Reject!
(There is no transition labeled ‘c’)

Finite State Automata (FSAs)

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 (regexes) 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 '.'

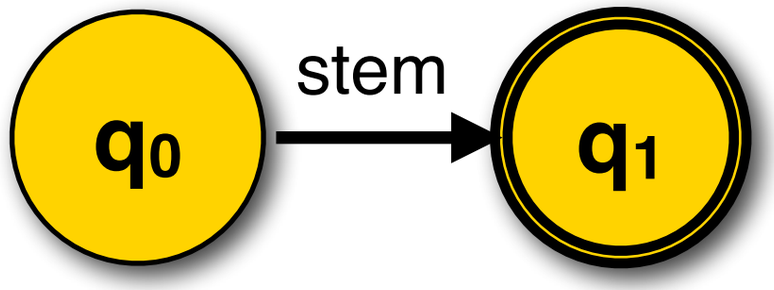
Complex patterns: (e.g. ^[*A-Z*]([*a-z*])+\s)

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

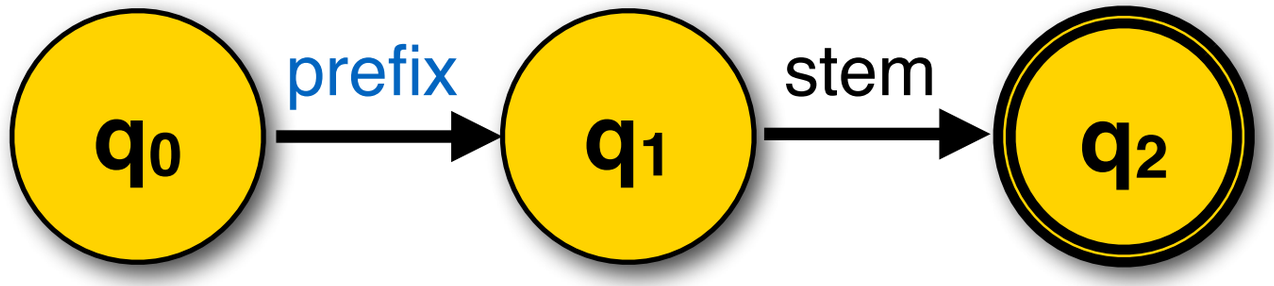
Lecture 2: Finite-state automata for morphology

Finite state automata for morphology

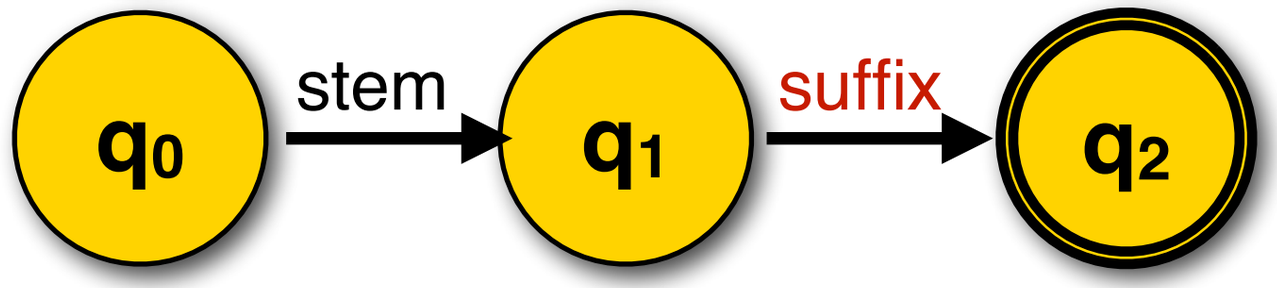
grace:



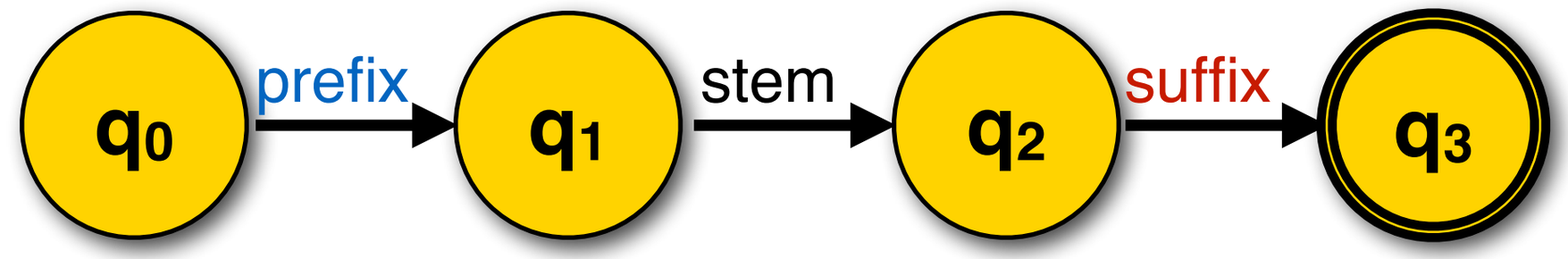
dis-grace:



grace-ful:

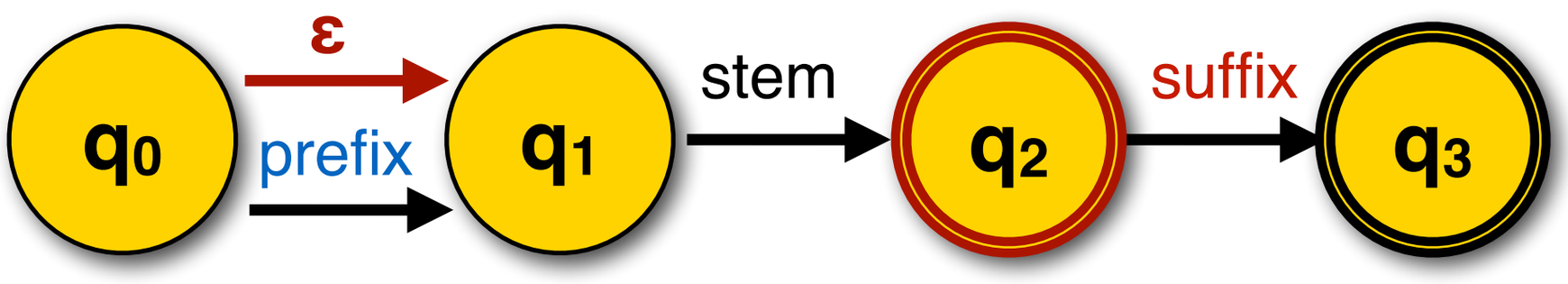


dis-grace-ful:

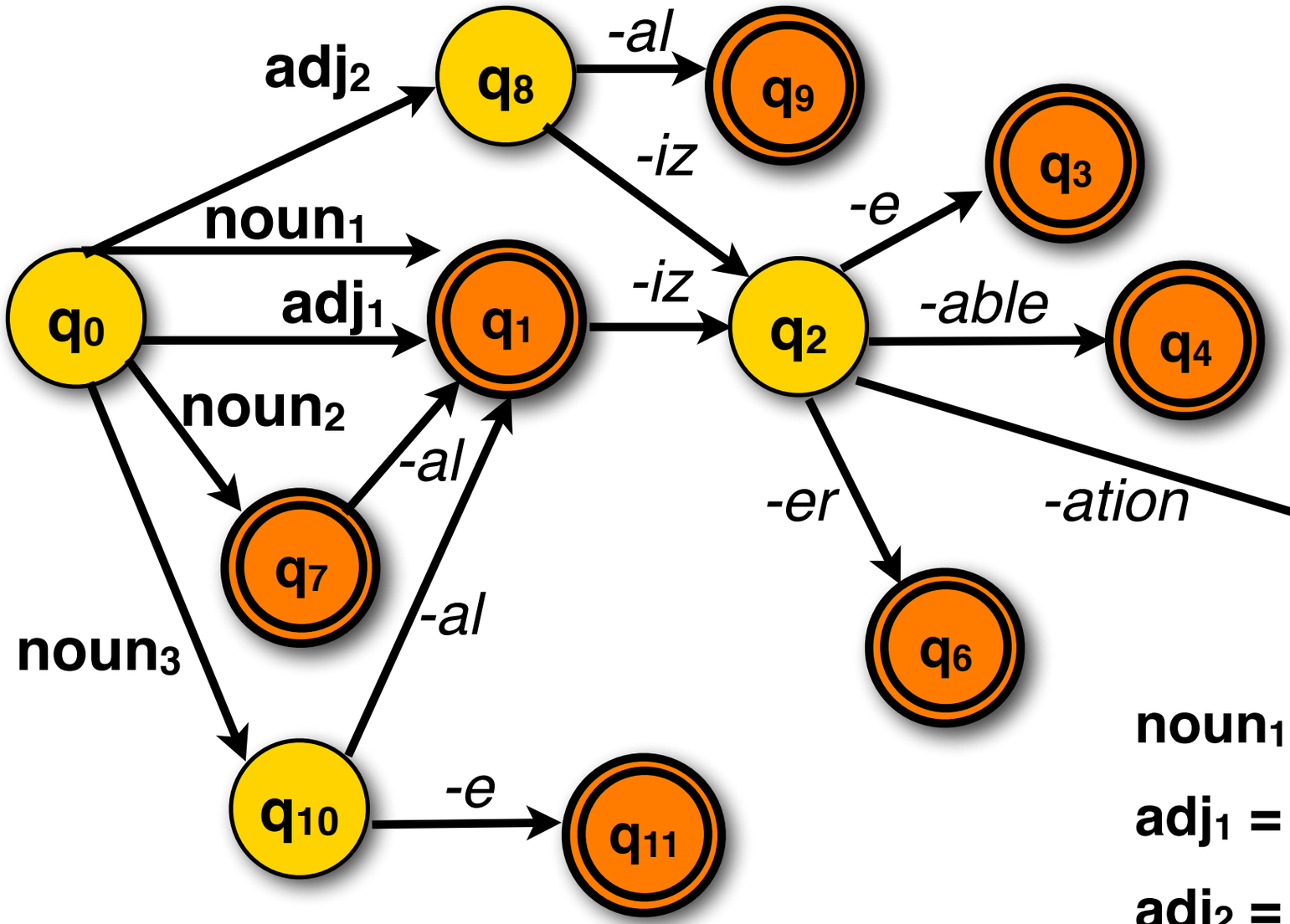


Union: merging automata

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,…}*

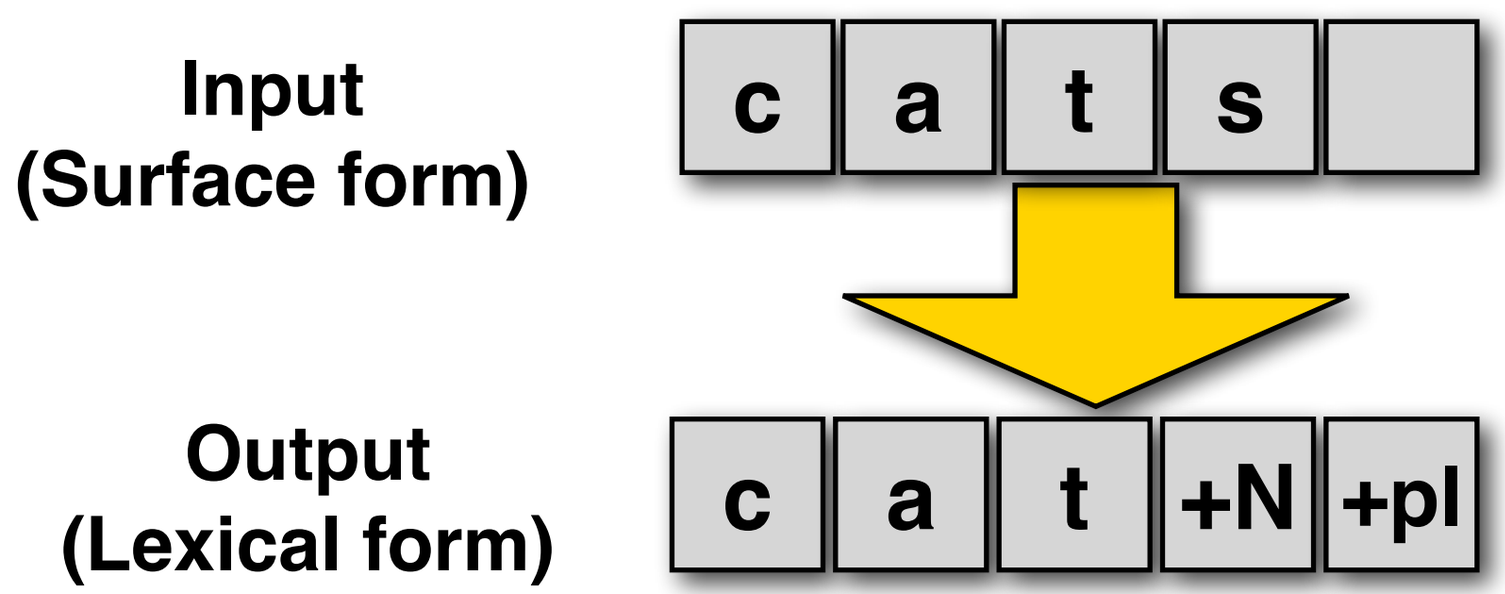
noun$_3$ = *{natur, structur,…}*

Lecture 2: Finite-State Transducers

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:



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"

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. $\Sigma = \{a, b, c, \ldots\}$)
- A designated **start state** $q_0 \in Q$
- A set of **final states** $F \subseteq Q$
- A **transition function** δ:

  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'

  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 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. $\Sigma = \{a, b, c, \ldots\}$)

— A finite alphabet Δ of **output symbols** (e.g. $\Delta = \{+N, +pl, \ldots\}$)

— 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 (2nd ed.) 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}:

$L_{in} = \{\textbf{cat, cats, fox, foxes, ...}\}$

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

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

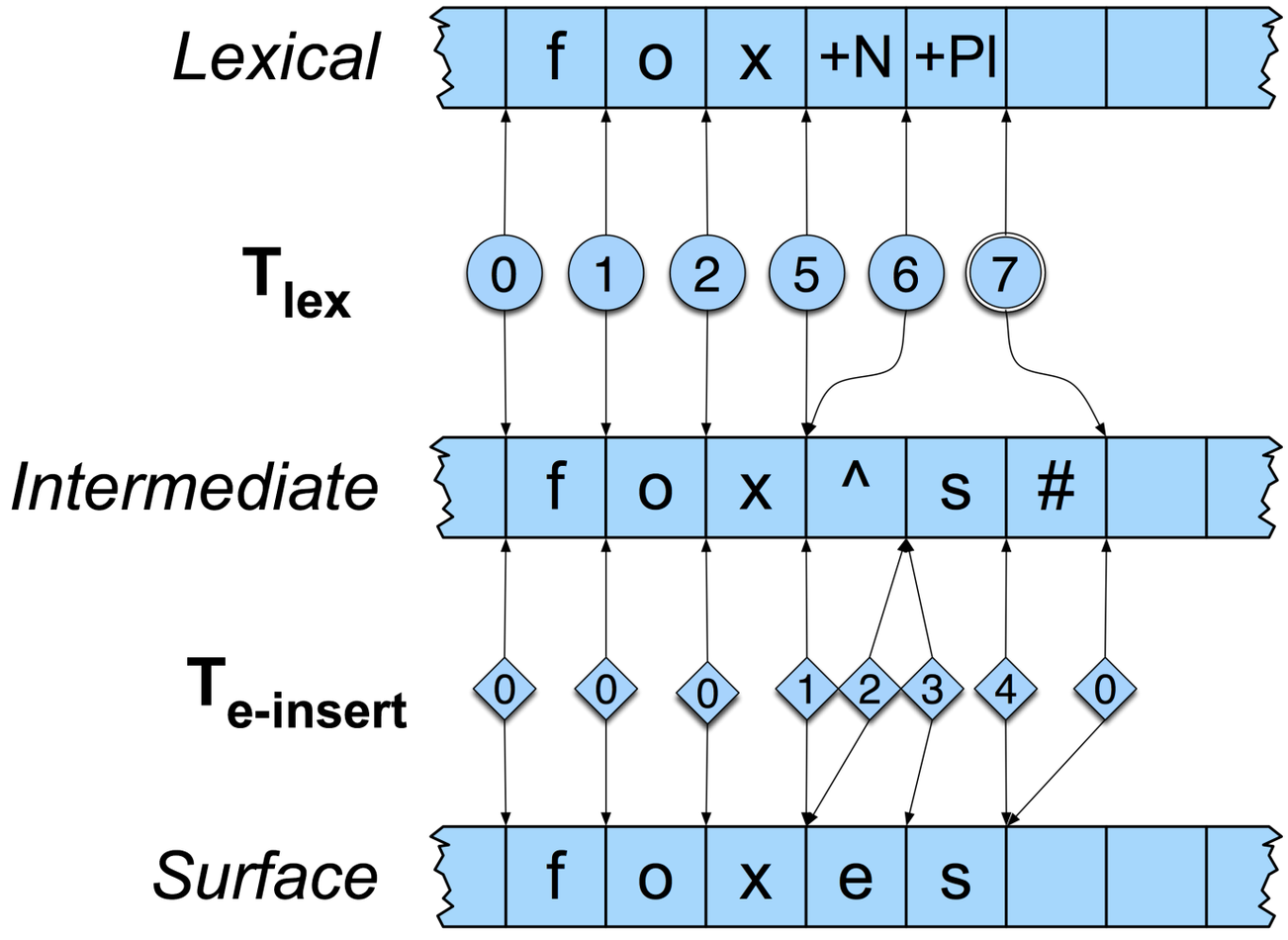
We define an intermediate representation to capture morpheme boundaries (^) and word boundaries (#):

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

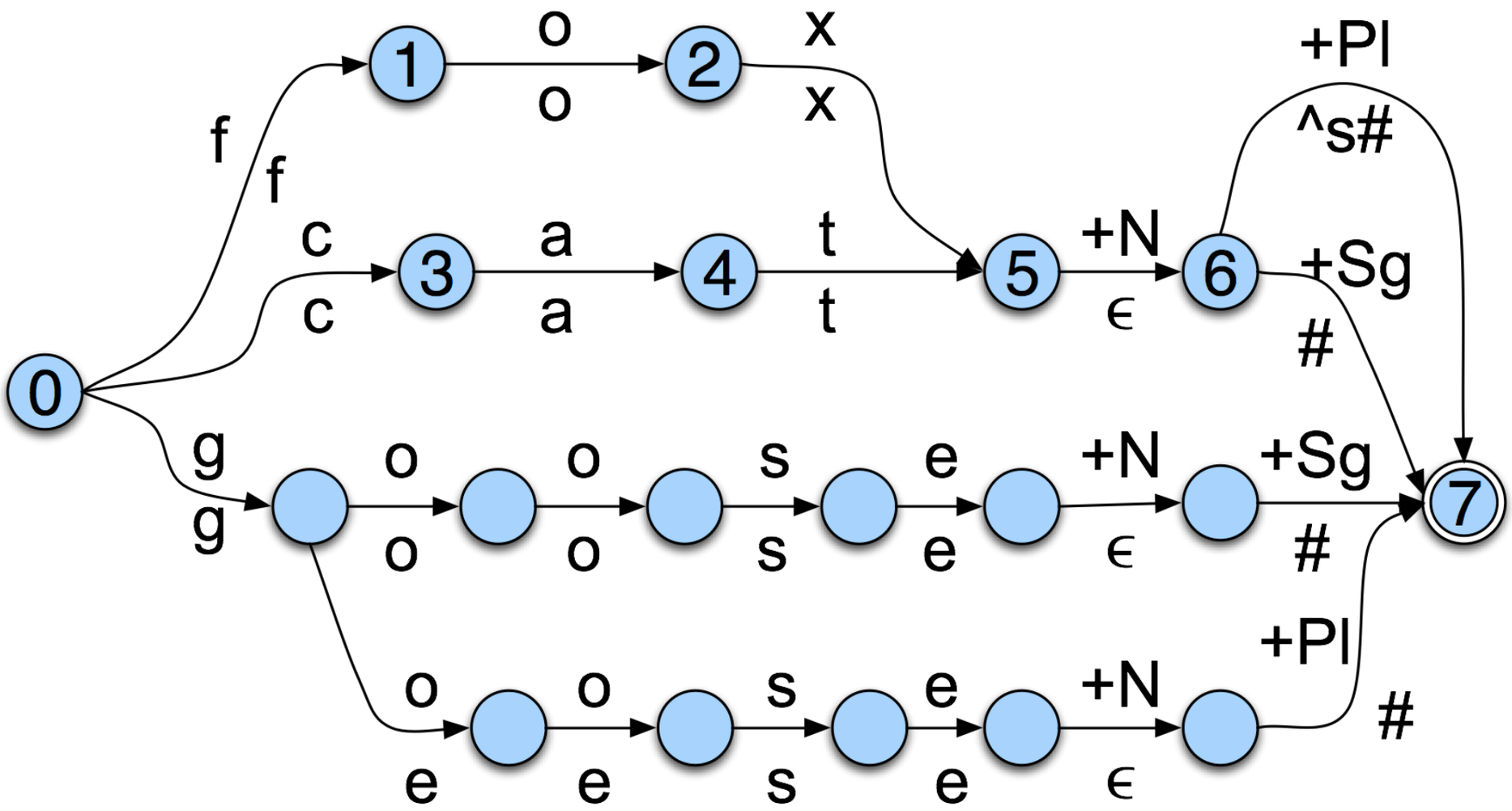
Intermediate-to-Surface Spelling Rule:

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

FST composition/cascade:

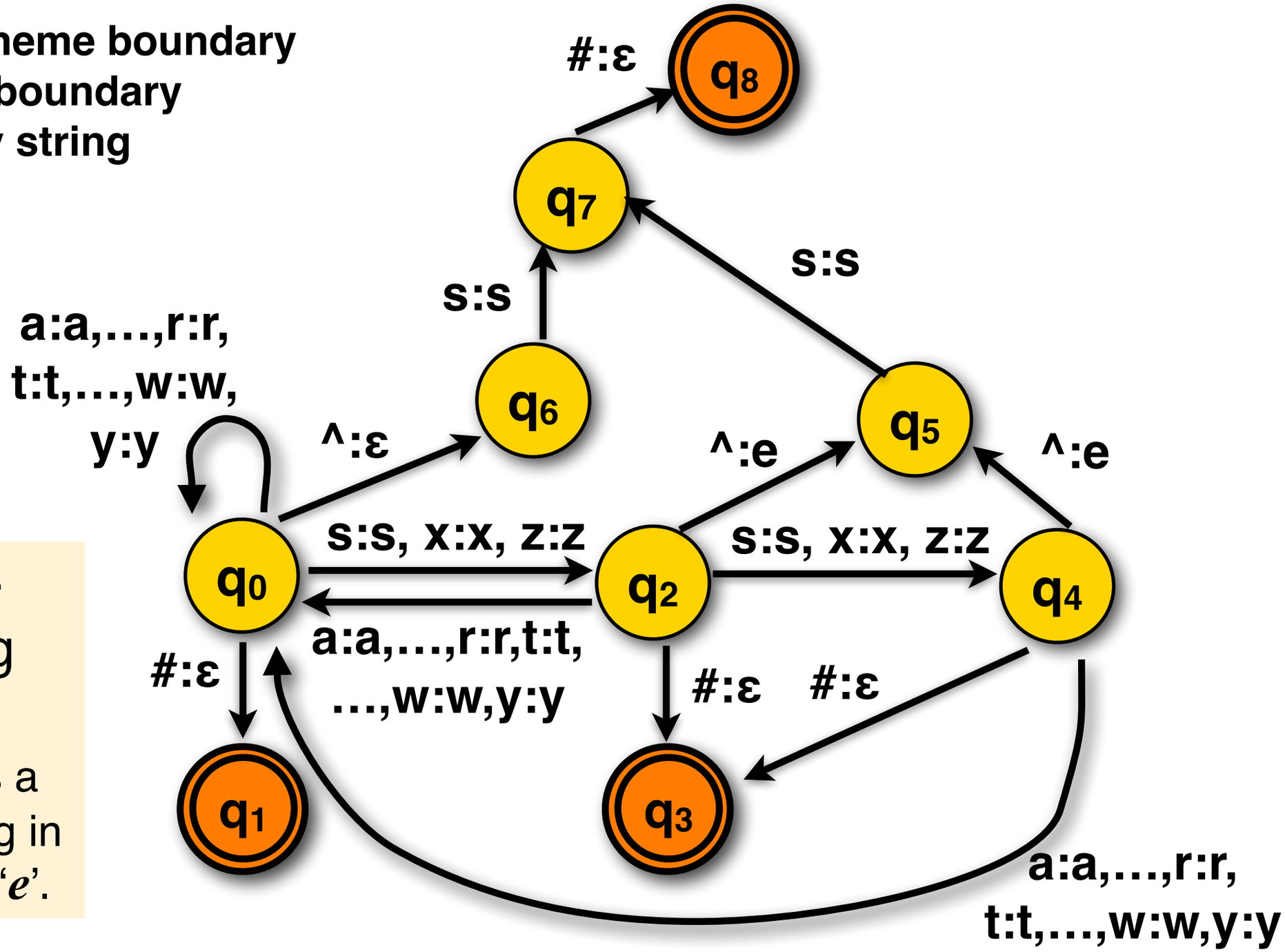


T_{lex}: Lexical to intermediate level



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

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.


The End