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

Lecture 2: Finite-state methods for morphology

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

A bit more admin…

HW 0

HW0 will come out later today
(check the syllabus.html page on the website)

We will assume Python 3.5.2 for our assignments
(you shouldn't have to load any additional modules or libraries besides the ones we provide)

You get 2 points for HW0
(HW1—HW4 have 10 points each)
1 point for uploading something to Compass
1 point for uploading a tar.gz file with the correct name and file structure

---

Compass and enrollment…

We won't be able to grade more than 100 assignments (and HW0 is only worth 2 points)

- Lecture slides and the PDFs for the assignments will always be posted on the class website.
- You don't need to be on Compass to get access.
- Piazza is also available to everybody.

If you are planning to drop this class, please do so ASAP, so that others can take your spot.

If you just got into the class, it is likely to take 24 hours to get access to Compass.

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.

Last lecture

The NLP pipeline:

Tokenization — POS tagging — Syntactic parsing — Semantic analysis — Coreference resolution

Why is NLP difficult?

Ambiguity
Coverage

Today’s lecture

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)

NB: No probabilities or machine learning yet.

We’re thinking about (symbolic) representations today.

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,...

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:
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

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 gesehen) 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**

One 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

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.

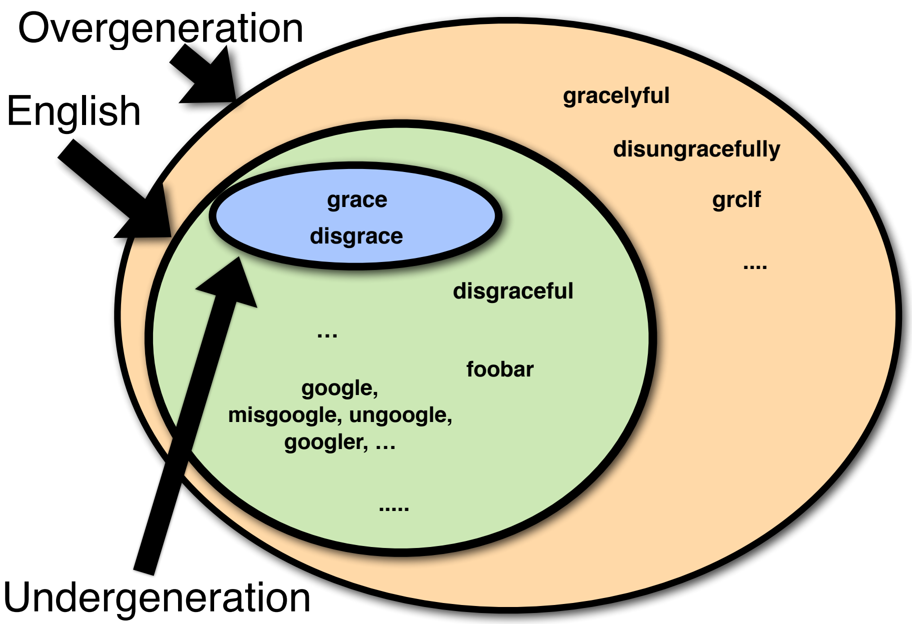
That is, we want a procedure that can generate (or accept) possible English words…

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"



Review:
Finite-State Automata
and
Regular Languages

Formal languages

An alphabet Σ is a set of symbols:

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

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

The empty string ε consists of zero symbols.

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

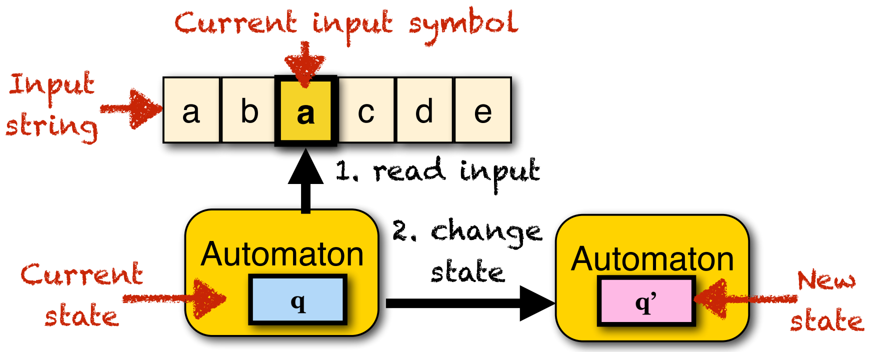
$\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, \ldots\}$

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

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

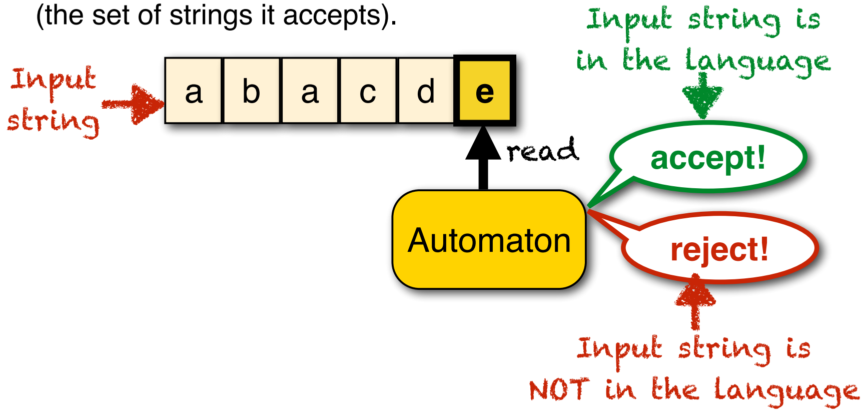
Automata and languages

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).

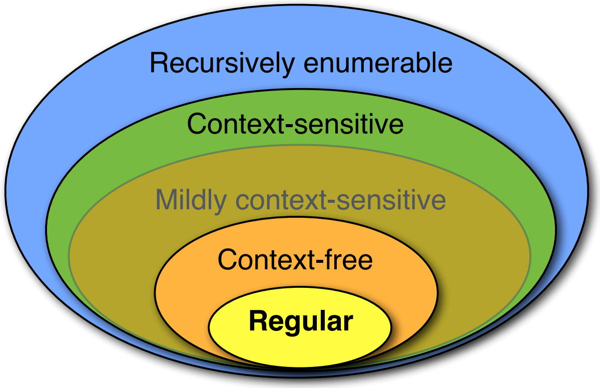


Automata and languages

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

The Chomsky Hierarchy

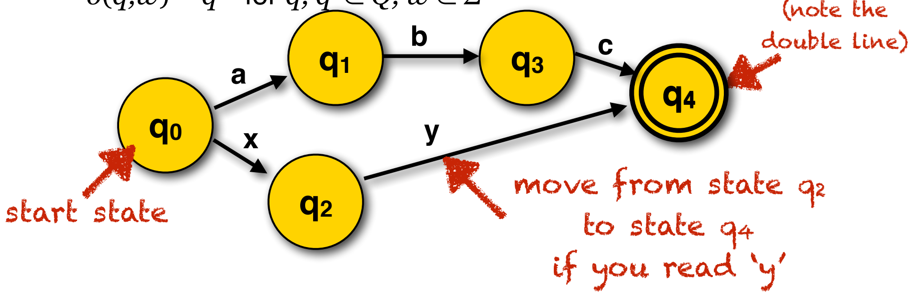


The structure of English words can be described
by a regular (= finite-state) grammar.

Finite-state automata

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

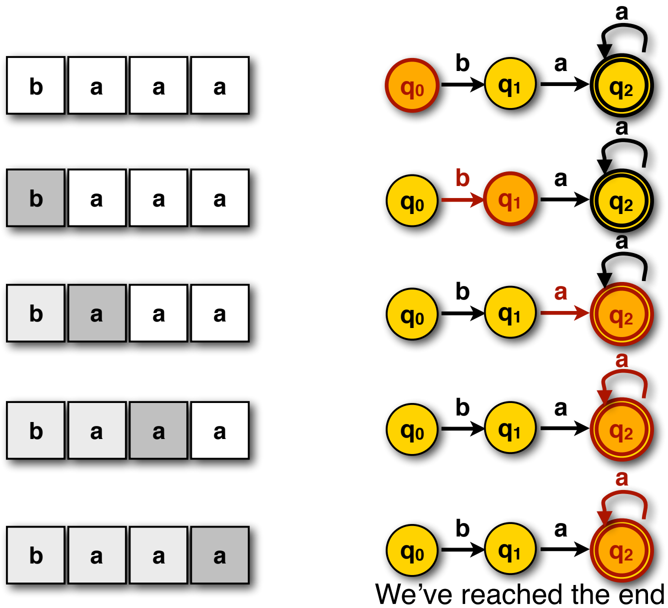
- a **finite set of states** $Q = \{q_0....q_N\}$, including **a start state** q_0 and one (or more) **final (=accepting) states** (say, q_N)
- a (**deterministic**) transition function $\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$



final state (note the double line)

start state

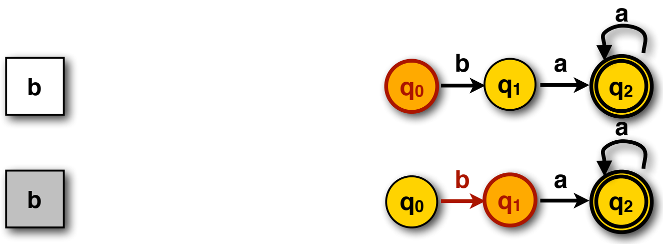
move from state q2 to state q4 if you read 'y'



Start in q_0

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

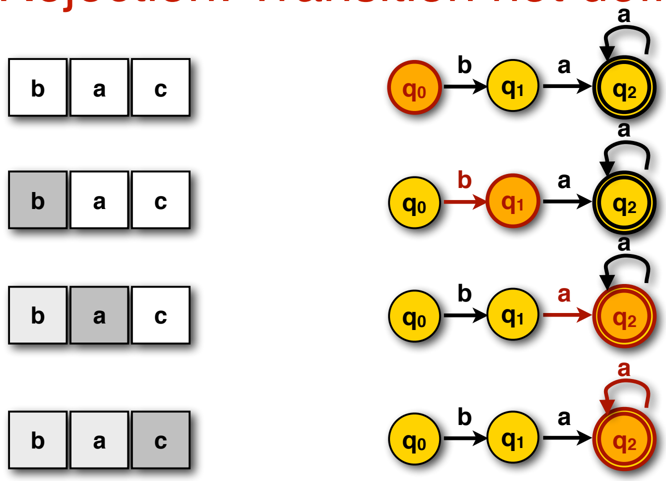
Rejection: Automaton does not end up in accepting state



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

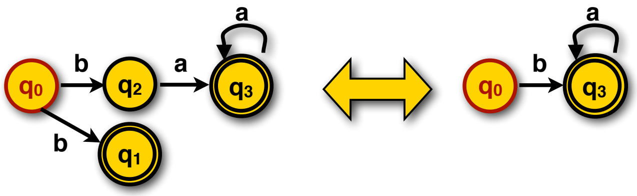
    If the current state is q and the current input is w, go to q'
  - The transition function for a nondeterministic (N)FSA: $Q \times \Sigma \rightarrow 2^Q$

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

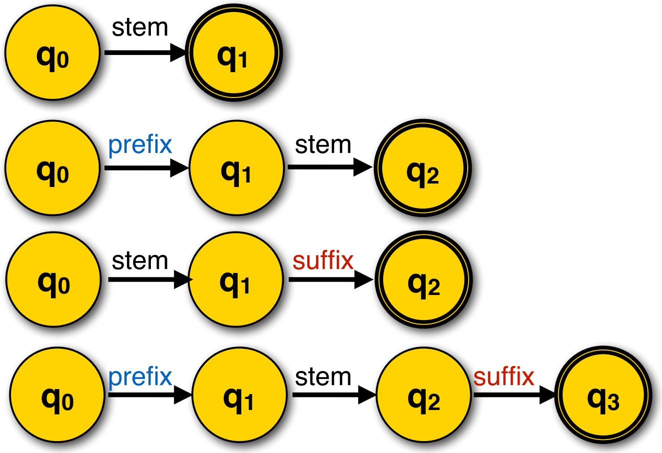
- **Standard characters** match themselves: '*a*', '1'
- **Character classes**: '[*abc*]', '[0-9]', **negation:** '[^*aeiou*]'
  (Predefined: \s (whitespace), \w (alphanumeric), etc.)
- **Any character** (except newline) is matched by '.'

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



grace:

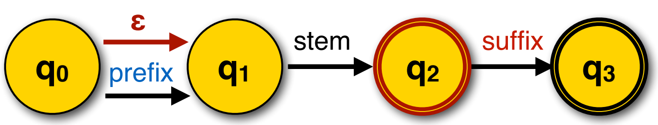
dis-grace:

grace-ful:

dis-grace-ful:

Union: merging automata

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

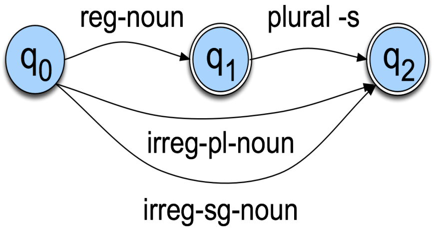


Stem changes

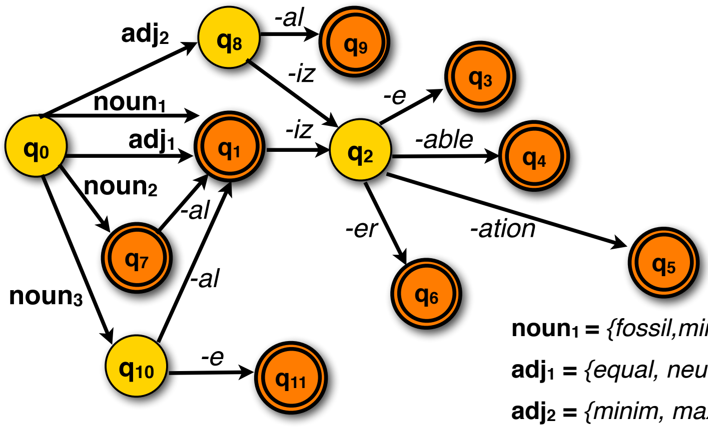
Some irregular words require stem changes:

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

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



FSAs for derivational morphology



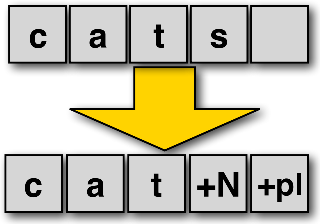
noun$_1$ = *{fossil,mineral,...}*
adj$_1$ = *{equal, neutral}*
adj$_2$ = *{minim, maxim}*
noun$_2$ = *{nation, form,…}*
noun$_3$ = *{natur, structur,…}*

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:

Input
(Surface form)



Output
(Lexical form)

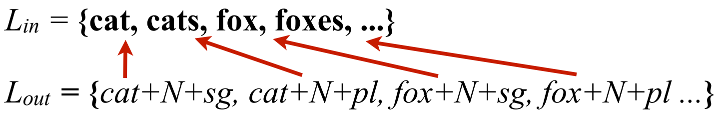
Finite-state transducers

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

Side note: "Surface realization"?

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 with the same meaning.

Intermediate representations

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

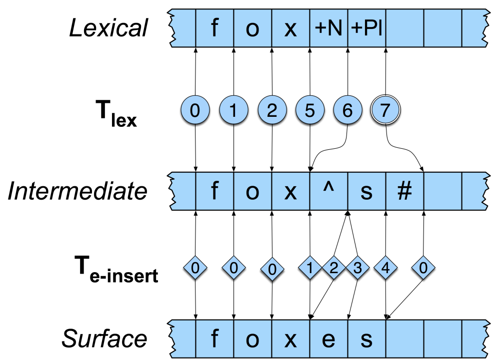
We define an intermediate representation to capture morpheme boundaries (^) and word boundaries (#):

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

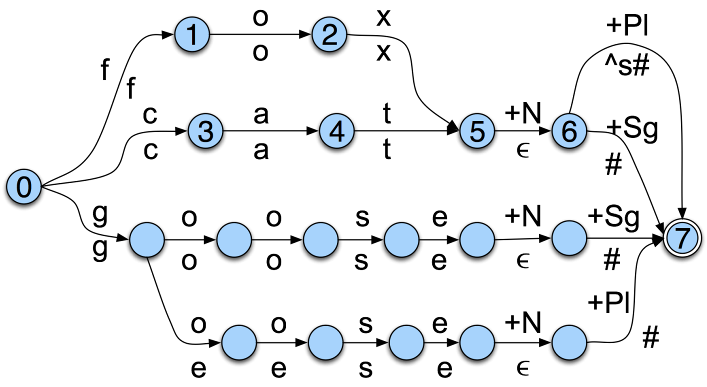
Intermediate-to-Surface Spelling Rule:

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

FST composition/cascade:

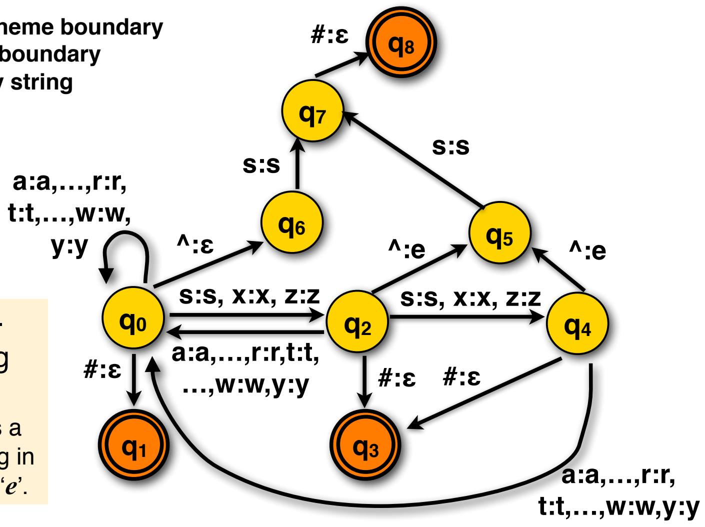


T_{lex}: Lexical to intermediate level



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