Lecture 2: Finite-state methods for morphology

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Last lecture

The NLP pipeline:
- tokenization — POS tagging — syntactic parsing
- semantic analysis — coreference resolution

Why is NLP difficult?
- ambiguity
- coverage

Course admin:
- HW0 is out (due Thursday, Sep 10, not Tuesday, Sep 8)
- Office hours
- Homework policies (no late submissions…)
- Midterm and final exams
- Projects and Literature surveys
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)
Morphology: What is a word?
A Turkish word

uygarlaştıramadıklarımızdanmışsınızcasına
uygar_laş_tır_ama_dık_lar_ımızdanmiş siziniz 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)
_miş: 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:
  \( V + \text{-ation} \): computerization
  \( V+ \text{-er} \): killer
  \( \text{Adj} + \text{-ness} \): fuzziness

Negation:
  \( \text{un-} \): undo, unseen, ...
  \( \text{mis-} \): mistake, ...

Adjectivization:
  \( V+ \text{-able} \): doable
  \( N + \text{-al} \): national
Morphemes: stems, affixes

\textbf{dis-grace-ful-ly}
\textbf{prefix-stem-suffix-suffix}

Many word forms consist of a \textit{stem} plus a number of \textit{affixes} (prefixes or suffixes)

\textit{Infixes} are inserted inside the stem.
\textit{Circumfixes} (German \textit{gesehen}) surround the stem

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

\textit{Stems} (grace) are often \textit{free morphemes}.
Free morphemes can occur by themselves as words.
\textit{Affixes} (dis-, -ful, -ly) are usually \textit{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-books, but: box-boxes, fly-flies, child-children

Past tense verbs add -ed to infinitive: walk-walked, but: like-liked, leap-leapt

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

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, *gracefully, *disungracefully,…

NB: * is linguists’ shorthand for “this is ungrammatical”
Overgeneration

English

Undergeneration

grace

disgrace

disgraceful

foobar

google,
misgoogle, ungoogle,
gogler, ...

gracelyful

disungracefully

grcif
Review:
Finite-State Automata
and
Regular Languages
Formal languages

An alphabet $\Sigma$ is a set of symbols:
e.g. $\Sigma = \{a, b, c\}$

A string $\omega$ is a sequence of symbols, e.g. $\omega = abcb$.
The empty string $\varepsilon$ consists of zero symbols.

The Kleene closure $\Sigma^*$ (‘sigma star’) is the (infinite) set of all strings that can be formed from $\Sigma$:
$\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, \ldots\}$

A language $L \subseteq \Sigma^*$ over $\Sigma$ is also a set of strings.
Typically we only care about proper subsets of $\Sigma^*$ ($L \subset \Sigma$).
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** $Q = \{q_0, \ldots, q_N\}$, 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 $q, q' \in Q$, $w \in \Sigma$

![Diagram of a finite-state automaton]

- **Start state** $q_0$
- **Final state** (note the double line)

Move from state $q_2$ to state $q_4$ if you read 'y'.
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

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 $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet $\Sigma$ 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 $\delta$:
  
  - 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**

$L_1 = \{ a^m b^n \} = \{ab, aab, abb, aaab, abb,… \}$ is a regular language,
$L_2 = \{ a^n b^n \} = \{ab, aabb, aaabbb,… \}$ 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:

\[
\begin{array}{c}
q_0 \quad \text{stem} \quad q_1 \\
\end{array}
\]

dis-grace:

\[
\begin{array}{c}
q_0 \quad \text{prefix} \quad q_1 \quad \text{stem} \quad q_2 \\
\end{array}
\]

grace-ful:

\[
\begin{array}{c}
q_0 \quad \text{stem} \quad q_1 \quad \text{suffix} \quad q_2 \\
\end{array}
\]

dis-grace-ful:

\[
\begin{array}{c}
q_0 \quad \text{prefix} \quad q_1 \quad \text{stem} \quad q_2 \quad \text{suffix} \quad q_3 \\
\end{array}
\]
Union: merging automata

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

noun₁ = \{fossil, mineral, \ldots\}
adj₁ = \{equal, neutral\}
adj₂ = \{minim, maxim\}
noun₂ = \{nation, form, \ldots\}
noun₃ = \{natur, structur, \ldots\}
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** $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet $\Sigma$ of **input symbols** (e.g. $\Sigma = \{a, b, c, \ldots\}$)
- A finite alphabet $\Delta$ of **output symbols** (e.g. $\Delta = \{+N, +p1, \ldots\}$)
- A designated **start state** $q_0 \in Q$
- A set of **final states** $F \subseteq Q$
- A **transition function** $\delta: Q \times \Sigma \rightarrow 2^Q$
  $$\delta(q, w) = Q' \quad \text{for} \ q \in Q, \ Q' \subseteq Q, \ w \in \Sigma$$
- An **output function** $\sigma: Q \times \Sigma \rightarrow \Delta^*$
  $$\sigma(q, w) = \omega \quad \text{for} \ q \in Q, \ w \in \Sigma, \ \omega \in \Delta^*$$

If the current state is $q$ and the current input is $w$, write $\omega$.

(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 $L_{out}$:

$L_{in} = \{\text{cat, cats, fox, foxes, } \ldots\}$

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

$T = \{\langle\text{cat, cat+N+sg}\rangle, \langle\text{cats, cat+N+pl}\rangle, \langle\text{fox, fox+N+sg}\rangle, \langle\text{foxes, fox+N+pl}\rangle\}$
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 ($T \cdot T'$): (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 underlying morphemes (plural-s, etc.) can have different orthographic surface realizations (-s, -es)

Spelling changes at morpheme boundaries:
  - E-insertion:  fox +s = foxes
  - E-deletion:   make +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:  

\[ \text{cat} + \text{N} + \text{PL} \quad \text{fox} + \text{N} + \text{PL} \]

⇒ Intermediate representation:  

\[ \text{cat}^\text{s#} \quad \text{fox}^\text{s#} \]

⇒ Surface string:  

\[ \text{cats} \quad \text{foxes} \]

Intermediate-to-Surface Spelling Rule:

If plural ‘s’ follows a morpheme ending in ‘x’, ‘z’ or ‘s’, insert ‘e’.
FST composition/cascade:

Lexical

Intermediate

Surface

}$T_{\text{lex}}$

}$T_{e\text{-insert}}$
\[ T_{\text{lex}}: \text{Lexical to intermediate level} \]
Intermediate-to-Surface Spelling Rule:
If plural 's' follows a morpheme ending in 'x', 'z' or 's', insert 'e'.

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