Lecture 3: Morphology and Finite-State Methods

Julia Hockenmaier
juliahmr@illinois.edu
What is a word?
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,...
“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
imiz: 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)
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 *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

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

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.
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
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) $q_N$
- 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

Start in $q_0$

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

$\begin{array}{ccc} b & a & c \\ b & a & c \\ b & a & c \\ b & a & c \end{array}$
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

$L_1 = \{ a^n b^m \} = \{ab, aab, abb, aaab, abb,...\}$ 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.
FSAs for derivational morphology

- \( q_0 \): noun
- \( q_1 \): adj
- \( q_2 \): adj
- \( q_3 \): noun
- \( q_4 \): noun
- \( q_5 \): adj
- \( q_6 \): adj
- \( q_7 \): adj
- \( q_8 \): adj
- \( q_9 \): adj
- \( q_{10} \): adj
- \( q_{11} \): adj

- \( \text{noun}_1 = \{\text{fossil, mineral, \ldots}\} \)
- \( \text{adj}_1 = \{\text{equal, neutral}\} \)
- \( \text{adj}_2 = \{\text{minim, maxim}\} \)
- \( \text{noun}_2 = \{\text{nation, form, \ldots}\} \)
- \( \text{noun}_3 = \{\text{natur, structur, \ldots}\} \)
Finite state automata for morphology

**grace:**

- State 0 (q₀)
- State 1 (q₁)

**dis-grace:**

- State 0 (q₀)
- State 1 (q₁)
- State 2 (q₂)

**grace-ful:**

- State 0 (q₀)
- State 1 (q₁)
- State 2 (q₂)

**dis-grace-ful:**

- State 0 (q₀)
- State 1 (q₁)
- State 2 (q₂)
- State 3 (q₃)
Union: merging automata

grace, dis-grace, grace-ful, dis-grace-ful
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** ‘$’
Finite-State Transducers for Morphology
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)  cats
Output (Lexical form)  cat +N +pl
```
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, *gracefully, *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** $Q = \{q_0, q_1, \ldots, q_n\}$
- 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** $\delta$:
  
  For a **deterministic (D)FSA**: $Q \times \Sigma \to Q$
  
  $$\delta(q,w) = q' \text{ for } q, q' \in Q, w \in \Sigma$$

  If the current state is $q$ and the current input is $w$, go to $q'$. 

  For a **nondeterministic (N)FSA**: $Q \times \Sigma \to 2^Q$
  
  $$\delta(q,w) = Q' \text{ for } q \in Q, Q' \subseteq Q, w \in \Sigma$$

  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 \( Q = \{q_0, q_1, \ldots, q_n\} \)
- A finite alphabet \( \Sigma \) of input symbols (e.g. \( \Sigma = \{a, b, c, \ldots\} \))
- A finite alphabet \( \Delta \) of output symbols (e.g. \( \Delta = \{+N, +pl, \ldots\} \))
- A designated start state \( q_0 \in Q \)
- A set of final states \( F \subseteq Q \)
- A transition function \( \delta: Q \times \Sigma \rightarrow 2^Q \)
  \[ \delta(q, w) = Q' \quad \text{for } q \in Q, Q' \subseteq Q, w \in \Sigma \]
- 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 (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 $L_{out}$:

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

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

$T = \{ \langle \text{cat, cat+N+sg} \rangle,$
    $\langle \text{cats, cat+N+pl} \rangle,$
    $\langle \text{fox, fox+N+sg} \rangle,$
    $\langle \text{foxes, fox+N+pl} \rangle \}$
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** ($T \circ T'$): *(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 = foxes
- **E-deletion:** make +ing = making
Intermediate representations

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

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

Lexicon:  
cat+N+PL  
fox+N+PL

$\Rightarrow$ Intermediate representation:  
cat$^s#$  
fox$^s#$

$\Rightarrow$ 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:

Lexical: fo x +N +Pl

Intermediate: fo x ^ s #

T_{lex}: 0 1 2 5 6 7

T_{e-insert}: 0 0 0 1 2 3 4 0

Surface: fo x e s
$T_{\text{lex}}$: Lexical to intermediate level
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:

$$((\text{ice cream}) \ (\text{cone}) \ \text{bakery})$$
not $$\text{ice} \ ((\text{cream} \ (\text{cone}) \ \text{bakery}))$$

$$((\text{computer science}) \ (\text{graduate student}))$$
not $$\text{computer} \ ((\text{science} \ (\text{graduate}) \ \text{student}))$$

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