Finite-State Morphology

CMSC 723 / LING 723 / INST 725

MARINE CARPUAT
marine@cs.umd.edu
Recall: Morphological Analysis

• Morpheme = smallest linguistic unit that has meaning

• Morphemes are combined into words
  – duck + s = [\text{N} \text{ duck}] + [\text{plural s}]
  – duck + s = [\text{V} \text{ duck}] + [\text{3rd person singular s}]
  – happiness = [\text{Adj happy}] + [\text{ness}]
Recall: Complex Morphology

In Turkish, from the root “uyu-” (sleep), the following can be derived...

<table>
<thead>
<tr>
<th>Turkish</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>uyuyorum</td>
<td>I am sleeping</td>
</tr>
<tr>
<td>uyuyorsun</td>
<td>you are sleeping</td>
</tr>
<tr>
<td>uyuyor</td>
<td>he/she/it is sleeping</td>
</tr>
<tr>
<td>uyuyoruz</td>
<td>we are sleeping</td>
</tr>
<tr>
<td>uyuyorsunuz</td>
<td>you are sleeping</td>
</tr>
<tr>
<td>uyuyorlar</td>
<td>they are sleeping</td>
</tr>
<tr>
<td>uyuduk</td>
<td>we slept</td>
</tr>
<tr>
<td>uyudukça</td>
<td>as long as (somebody) sleeps</td>
</tr>
<tr>
<td>uyumalıyız</td>
<td>we must sleep</td>
</tr>
<tr>
<td>uyumadan</td>
<td>without sleeping</td>
</tr>
<tr>
<td>uyuman</td>
<td>your sleeping</td>
</tr>
<tr>
<td>uyurken</td>
<td>while (somebody) is sleeping</td>
</tr>
<tr>
<td>uyuyunca</td>
<td>when (somebody) sleeps</td>
</tr>
<tr>
<td>uyutmak</td>
<td>to cause somebody to sleep</td>
</tr>
<tr>
<td>uyutturmak</td>
<td>to cause (somebody) to cause (another) to sleep</td>
</tr>
<tr>
<td>uyutturtturmak</td>
<td>to cause (somebody) to cause (some other) to cause (yet another) to sleep</td>
</tr>
</tbody>
</table>
Today

• Computational tools
  – Finite-state automata
  – Finite-state transducers

• Morphology
  – Introduction to morphological processes
  – Computational morphology with finite-state methods
Sheeptalk!

Language:

Regular Expression:
/\baa\+!\/
Finite-State Automata

- What are they?
- What do they do?
- How do they work?
FSA: What are they?

- **Q**: a finite set of N states
  - \[ Q = \{q_0, q_1, q_2, q_3, q_4\} \]
  - The start state: \( q_0 \)
  - The set of final states: \( F = \{q_4\} \)

- **\( \Sigma \)**: a finite input alphabet of symbols
  - \[ \Sigma = \{a, b, !\} \]

- **\( \delta(q,i) \)**: transition function
  - Given state \( q \) and input symbol \( i \), return new state \( q' \)
  - \( \delta(q_3,!) \rightarrow q_4 \)

![Diagram of FSA states and transitions](image-url)
**FSA: State Transition Table**

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>!</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>∅</td>
<td>2</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

**Diagram:**

- $q_0$: Start state
- $q_1$: Transition on $b$ to $q_1$
- $q_2$: Transition on $a$ to $q_2$
- $q_3$: Transition on $a$ to $q_3$
- $q_4$: Transition on $!$ to $q_4$

- Arrows represent transitions based on the input.
FSA: What do they do?

• Given a string, a FSA either rejects or accepts it
  – ba! → reject
  – baa! → accept
  – baaaaz! → reject
  – baaaaa! → accept
  – baaaaaaa! → accept
  – baa → reject
  – moooo → reject

• What does this have to do with CL/NLP?
FSA: How do they work?
FSA: How do they work?
function D-RECOGNIZE(tape, machine) returns accept or reject

index ← Beginning of tape

current-state ← Initial state of machine

loop
  if End of input has been reached then
    if current-state is an accept state then
      return accept
    else
      return reject
  elseif transition-table[current-state,tape[index]] is empty then
    return reject
  else
    current-state ← transition-table[current-state,tape[index]]
    index ← index + 1
  end
Accept or Generate?

• **Formal languages** are sets of strings
  – Strings composed of symbols drawn from a finite alphabet

• **Finite-state automata** define formal languages
  – Without having to enumerate all the strings in the language

• Two views of FSAs:
  – **Acceptors** that can tell you if a string is in the language
  – **Generators** to produce all and only the strings in the language
Exercise

Define an FSA representing the language of all non-zero binary strings of even length
Exercise

Define an FSA representing the language of all non-zero binary strings of odd length
Introducing Non-Determinism

- Deterministic vs. Non-deterministic FSAs

- Epsilon (ε) transitions
Using NFSAs to Accept Strings

• What does it mean?
  – Accept: there exist at least one path (need not be all paths)
  – Reject: no paths exist

• General approaches
  – Backup: add markers at choice points, then possibly revisit unexplored arcs at marked choice point
  – Explore paths in parallel
  – Recognition with NFSAs as search through state space
What’s the point?

• NFSAs and DFSA are equivalent
  – For every NFSA, there is an equivalent DFSA (and vice versa)

• Equivalence between regular expressions and FSA

• Why use NFSAs?
Regular Language: Definition

• $\emptyset$ is a regular language
• $\forall a \in \Sigma \cup \varepsilon, \{a\}$ is a regular language
• If $L_1$ and $L_2$ are regular languages, then so are:
  – $L_1 \cdot L_2 = \{x \ y \ | \ x \in L_1 \ , \ y \in L_2 \}$, the concatenation of $L_1$ and $L_2$
  – $L_1 \cup L_2$, the union or disjunction of $L_1$ and $L_2$
  – $L_1^*$, the Kleene closure of $L_1$
Regular Languages: Starting Points

(a) \( r = \epsilon \)

(b) \( r = \emptyset \)

(c) \( r = a \)
Regular Languages: Concatenation
Regular Languages: Disjunction
Regular Languages: Kleene Closure
Finite-State Transducers (FSTs)

- A two-tape automaton that recognizes or generates pairs of strings
- Think of an FST as an FSA with two symbol strings on each arc
Four-fold view of FSTs

- As a recognizer
- As a generator
- As a translator
- As a set relater

Lexical: $\{\text{cat}, \text{+N}, \text{+PL}\}$

Surface: $\{\text{cats}\}$
Today

• Computational tools
  – Finite-state automata
  – Finite-state transducers

• Morphology
  – Introduction to morphological processes
  – Computational morphology with finite-state methods
Computational Morphology

• Definitions and problems
  – What is morphology?
  – Topology of morphologies

• Computational morphology
  – Finite-state methods
Morphology

• Study of how words are constructed from smaller units of meaning

• Smallest unit of meaning = morpheme
  – fox has morpheme fox
  – cats has two morphemes cat and –s
  – Note: it is useful to distinguish morphemes from orthographic rules

• Two classes of morphemes:
  – Stems: supply the “main” meaning
    • Aka root / lemma
  – Affixes: add “additional” meaning
Topology of Morphologies

- Concatenative vs. non-concatenative
- Derivational vs. inflectional
- Regular vs. irregular
Concatenative Morphology

- Morpheme+Morpheme+Morpheme+...
- Stems (also called lemma, base form, root, lexeme):
  - hope+ing → hoping
  - hop+ing → hopping
- Affixes:
  - Prefixes: Antidisestablishmentarianism
  - Suffixes: Antidisestablishmentarianism
- Agglutinative languages (e.g., Turkish)
  - uygarlaştıramadıklarımızdanmışsınızcasına → uygar+laş+tır+ama+dık+lar+ıımız+dan+mış+sınız+casına
  - Meaning: behaving as if you are among those whom we could not cause to become civilized
Non-Concatenative Morphology

- **Infixes** (e.g., Tagalog)
  - hingi (borrow)
  - humingi (borrower)

- **Circumfixes** (e.g., German)
  - sagen (say)
  - gesagt (said)

- **Reduplication** (e.g., Motu, spoken in Papua New Guinea)
  - mahuta (to sleep)
  - mahuta mahuta (to sleep constantly)
  - mahutamahuta (to sleep, plural)
Templatic Morphologies

- Common in Semitic languages
- Roots and patterns

Arabic

مكتوب

maktuub

written

Hebrew

מַתְוַב

ktuuv

written
Derivational Morphology

• Stem + morpheme →
  – New word with different meaning or different part of speech
  – Exact meaning difficult to predict

• Nominalization in English:
  – -ation: computerization, characterization
  – -ee: appointee, advisee
  – -er: killer, helper

• Adjective formation in English:
  – -al: computational, derivational
  – -less: clueless, helpless
  – -able: teachable, computable
Inflectional Morphology

• Stem + morpheme →
  – Word with same part of speech as the stem
• Adds: tense, number, person,...

• Plural morpheme for English noun
  – cat+s
  – dog+s

• Progressive form in English verbs
  – walk+ing
  – rain+ing
Noun Inflections in English

- Regular
  - cat/cats
  - dog/dogs

- Irregular
  - mouse/mice
  - ox/oxen
  - goose/geese
Verb Inflections in English

<table>
<thead>
<tr>
<th>Morphological Class</th>
<th>Regularly Inflected Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>stem</td>
<td>walk</td>
</tr>
<tr>
<td>-s form</td>
<td>walks</td>
</tr>
<tr>
<td>-ing participle</td>
<td>walking</td>
</tr>
<tr>
<td>Past form or -ed participle</td>
<td>walked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Morphological Class</th>
<th>Irregularly Inflected Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>stem</td>
<td>eat</td>
</tr>
<tr>
<td>-s form</td>
<td>eats</td>
</tr>
<tr>
<td>-ing participle</td>
<td>eating</td>
</tr>
<tr>
<td>preterite</td>
<td>ate</td>
</tr>
<tr>
<td>past participle</td>
<td>eaten</td>
</tr>
</tbody>
</table>
Morphological Parsing

• Computationally decompose input forms into component morphemes

• Components needed:
  – A lexicon (stems and affixes)
  – A model of how stems and affixes combine
  – Orthographic rules
## Morphological Parsing: Examples

<table>
<thead>
<tr>
<th>WORD</th>
<th>STEM (+FEATURES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cats</td>
<td>cat +N +PL</td>
</tr>
<tr>
<td>cat</td>
<td>cat +N +SG</td>
</tr>
<tr>
<td>cities</td>
<td>city +N +PL</td>
</tr>
<tr>
<td>geese</td>
<td>goose +N +PL</td>
</tr>
<tr>
<td>ducks</td>
<td>(duck +N +PL) or (duck +V +3SG)</td>
</tr>
<tr>
<td>merging</td>
<td>merge +V +PRES-PART</td>
</tr>
<tr>
<td>caught</td>
<td>(catch +V +PAST-PART) or (catch +V +PAST)</td>
</tr>
</tbody>
</table>
Different Approaches

• Lexicon only
• Rules only
• Lexicon and rules
  – finite-state automata
  – finite-state transducers
Lexicon-only

- Simply enumerate all surface forms and analyses

<table>
<thead>
<tr>
<th>Word</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>acclaim</td>
<td>acclaim $N$</td>
</tr>
<tr>
<td>acclaim</td>
<td>acclaim $V+0$</td>
</tr>
<tr>
<td>acclaimed</td>
<td>acclaim $V+ed$</td>
</tr>
<tr>
<td>acclaimed</td>
<td>acclaim $V+en$</td>
</tr>
<tr>
<td>acclaming</td>
<td>acclaim $V+ing$</td>
</tr>
<tr>
<td>acclaims</td>
<td>acclaim $N+s$</td>
</tr>
<tr>
<td>acclaims</td>
<td>acclaim $V+s$</td>
</tr>
<tr>
<td>acclamation</td>
<td>acclamation $N$</td>
</tr>
<tr>
<td>acclamations</td>
<td>acclamation $N+s$</td>
</tr>
<tr>
<td>acclimate</td>
<td>acclimate $V+0$</td>
</tr>
<tr>
<td>acclimate</td>
<td>acclimate $V+ed$</td>
</tr>
<tr>
<td>acclimate</td>
<td>acclimate $V+en$</td>
</tr>
<tr>
<td>acclimates</td>
<td>acclimate $V+s$</td>
</tr>
<tr>
<td>acclimating</td>
<td>acclimate $V+ing$</td>
</tr>
</tbody>
</table>
Rule-only

• Cascading set of rules
  – $s \rightarrow \varepsilon$
  – ation $\rightarrow$ e
  – ize $\rightarrow$ ε
  – ...

• Example
  – generalizations
    $\rightarrow$ generalization
    $\rightarrow$ generalize
    $\rightarrow$ general
  – organizations
    $\rightarrow$ organization
    $\rightarrow$ organize
    $\rightarrow$ organ
Lexicon + Rules

• FSA: for recognition
  – Recognize all grammatical input and only grammatical input

• FST: for analysis
  – If grammatical, analyze surface form into component morphemes
  – Otherwise, declare input ungrammatical
FSA: English Noun Morphology

<table>
<thead>
<tr>
<th>reg-noun</th>
<th>irreg-pl-noun</th>
<th>irreg-sg-noun</th>
<th>plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>geese</td>
<td>goose</td>
<td>-s</td>
</tr>
<tr>
<td>cat</td>
<td>sheep</td>
<td>sheep</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>mice</td>
<td>mouse</td>
<td></td>
</tr>
</tbody>
</table>

Note problem with orthography!
FSA: English Noun Morphology
FSA: English Adjectival Morphology

• Examples:
  – big, bigger, biggest
  – small, smaller, smallest
  – happy, happier, happiest, happily
  – unhappy, unhappier, unhappiest, unhappily

• Morphemes:
  – Roots: big, small, happy, etc.
  – Affixes: un-, -er, -est, -ly
FSA: English Adjectival Morphology

adj-root₁: \{happy, real, \ldots\}
adj-root₂: \{big, small, \ldots\}
Morphological Parsing with FSTs

- Limitation of FSA:
  - Accepts or rejects an input... but doesn’t actually provide an analysis

- Use FSTs instead!
  - One tape contains the input, the other tape as the analysis

<table>
<thead>
<tr>
<th>Lexical</th>
<th>c</th>
<th>a</th>
<th>t</th>
<th>+N</th>
<th>+Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface</td>
<td>c</td>
<td>a</td>
<td>t</td>
<td>s</td>
<td></td>
</tr>
</tbody>
</table>
Terminology

• Transducer alphabet (pairs of symbols):
  – \( a:b = a \) on the upper tape, \( b \) on the lower tape
  – \( a:\varepsilon = a \) on the upper tape, nothing on the lower tape
  – If \( a:a \), write \( a \) for shorthand

• Special symbols
  – \# = word boundary
  – \(^\wedge = \) morpheme boundary
  – (For now, think of these as mapping to \( \varepsilon \))
FST for English Nouns

- What’s the problem here?
FST for English Nouns
Handling Orthography

**Lexical**
- **cat**
- **+N**
- **+Pl**

**Surface**
- **cats**

**Surface**
- **foxes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description of Rule</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consonant doubling</td>
<td>1-letter consonant doubled before <em>-ing/-ed</em></td>
<td>beg/begging</td>
</tr>
<tr>
<td>E deletion</td>
<td>silent e dropped before <em>-ing</em> and <em>-ed</em></td>
<td>make/making</td>
</tr>
<tr>
<td>E insertion</td>
<td>e added after <em>-s,-z,-x,-ch,-sh</em> before <em>-s</em></td>
<td>watch/watches</td>
</tr>
<tr>
<td>Y replacement</td>
<td>-y changes to <em>-ie</em> before <em>-s, -i</em> before <em>-ed</em></td>
<td>try/tries</td>
</tr>
<tr>
<td>K insertion</td>
<td>verbs ending with vowel + <em>-c</em> add <em>-k</em></td>
<td>panic/panicked</td>
</tr>
</tbody>
</table>
Complete Morphological Parser

LEXICON-FST

$FST_1$  orthographic rules  $FST_n$

fox $+_N$ $+_PL$

fox $^\wedge$ s #

foxes
FSTs and Ambiguity

• unionizable
  – union +ize +able
  – un+ ion +ize +able

• assess
  – assess +V
  – ass +N +essN
Practical NLP Applications

• In practice, it is almost never necessary to write FSTs by hand...

• Typically, one writes rules:
  – Chomsky and Halle Notation: $a \rightarrow b / c\_d$
    = rewrite $a$ as $b$ when occurs between $c$ and $d$
  – E-Insertion rule

$$\varepsilon \rightarrow e / \left\{\begin{array}{c}x \\ s \\ z\end{array}\right\} ^\_\_ s \#$$

• Rule $\rightarrow$ FST compiler handles the rest...
What we covered today...

• Computational tools
  – Finite-state automata (deterministic vs. non-deterministic)
  – Finite-state transducers

• Morphology
  – Overview of morphological processes
  – Computational morphology with finite-state methods
Before next class...

• Sign up for Piazza
  https://piazza.com/umd/fall2015/cmsc723/home

• Email me dates of religious holidays you will observe this semester

• Do the readings

• Submit HW1