Finite-State Morphology

CMSC 723 / LING 723 / INST 725

MARINE CARPUAT
marine@cs.umd.edu
Today

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

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

Language:

\[ \text{baa!} \]
\[ \text{baaa!} \]
\[ \text{baaaaa!} \]
... 

Regular Expression:

\[ /baa+!/ \]

Finite-State Automaton:
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 \( q' \)
  - \( \delta(q_3,!) \rightarrow q_4 \)
# FSA: State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>b</th>
<th>a</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>ø</td>
<td>ø</td>
<td>ø</td>
</tr>
<tr>
<td>1</td>
<td>ø</td>
<td>2</td>
<td>ø</td>
<td>ø</td>
</tr>
<tr>
<td>2</td>
<td>ø</td>
<td>3</td>
<td>ø</td>
<td>ø</td>
</tr>
<tr>
<td>3</td>
<td>ø</td>
<td>3</td>
<td>ø</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>ø</td>
<td>ø</td>
<td>ø</td>
<td>ø</td>
</tr>
</tbody>
</table>
FSA: What do they do?

- Given a string, a FSA either rejects or accepts it
  - ba! → reject
  - baa! → accept
  - baaaz! → reject
  - baaaa! → accept
  - baaaaaa! → accept
  - baaaaaaa! → accept
  - baa → reject
  - moooo → reject

- What does this have to do with CL/NLP?
FSA: How do they work?

$q_0$ $q_1$ $q_2$ $q_3$ $q_3$ $q_4$

$b$ $a$ $a$ $a$ $!$

$q_0$ $q_1$ $q_2$ $q_3$ $q_3$ $q_4$

$\text{ACCEPT}$
FSA: How do they work?

 transition diagram with states $q_0, q_1, q_2$ and input symbols $b, a, !$. The sequence of symbols $b a ! ! !$ leads to a reject state $q_4$. The diagram shows the transitions for different symbols.
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
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
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
  – Parallelism
  – Look ahead
What’s the point?

• NFSAs and DFSAs are equivalent
  – For every NFSA, there is a 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:
  - $L_1 \cdot L_2 = \{x \, y \mid x \in L_1, \, y \in L_2\}$, 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**

```
{ c a t +N +PL }
```

**Surface**

```
{ c a t s }
```
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

• 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)
  – uygarlaştırmaademiklerimizdanimişsinizcasına → uygar+laş+tır+ama+dık+lar+ımız+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)
Templatic Morphologies

- Common in Semitic languages
- Roots and patterns

Arabic

كتب

مكتوب

maktuub

written

Hebrew

כתיב

כתב

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, merge, try, map</td>
</tr>
<tr>
<td>-s form</td>
<td>walks, merges, tries, maps</td>
</tr>
<tr>
<td>-ing participle</td>
<td>walking, merging, trying, mapping</td>
</tr>
<tr>
<td>Past form or -ed participle</td>
<td>walked, merged, tried, mapped</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, catch, cut</td>
</tr>
<tr>
<td>-s form</td>
<td>eats, catches, cuts</td>
</tr>
<tr>
<td>-ing participle</td>
<td>eating, catching, cutting</td>
</tr>
<tr>
<td>preterite</td>
<td>ate, caught, cut</td>
</tr>
<tr>
<td>past participle</td>
<td>eaten, caught, cut</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>Surface Form</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>acclaim</td>
<td>$N$</td>
</tr>
<tr>
<td>acclaim</td>
<td>$V+0$</td>
</tr>
<tr>
<td>acclaimed</td>
<td>$V+ed$</td>
</tr>
<tr>
<td>acclaimed</td>
<td>$V+en$</td>
</tr>
<tr>
<td>acclaiming</td>
<td>$V+ing$</td>
</tr>
<tr>
<td>acclaims</td>
<td>$N+s$</td>
</tr>
<tr>
<td>acclaims</td>
<td>$V+s$</td>
</tr>
<tr>
<td>acclamation</td>
<td>$N$</td>
</tr>
<tr>
<td>acclamations</td>
<td>$N+s$</td>
</tr>
<tr>
<td>acclimate</td>
<td>$V+0$</td>
</tr>
<tr>
<td>acclimated</td>
<td>$V+ed$</td>
</tr>
<tr>
<td>acclimated</td>
<td>$V+en$</td>
</tr>
<tr>
<td>acclimates</td>
<td>$V+s$</td>
</tr>
<tr>
<td>acclimating</td>
<td>$V+ing$</td>
</tr>
</tbody>
</table>
Rule-only

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

• Example
  – generalizations
    $\rightarrow$ generalization
  – 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
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 a t +N +P l</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface</td>
<td>c a t s</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
  – ^ = morpheme boundary
  – (For now, think of these as mapping to $\varepsilon$)
FST for English Nouns
FST for English Nouns
## Handling Orthography

### Lexical vs. Surface Orthography

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lexical</strong></td>
<td>cat +N +Pl</td>
</tr>
<tr>
<td><strong>Surface</strong></td>
<td>cats</td>
</tr>
<tr>
<td></td>
<td>foxes</td>
</tr>
</tbody>
</table>

### Consonant Doubling
- 1-letter consonant doubled before -\textit{ing}/-\textit{ed}
- Example: beg/begging

### E Deletion
- Silent e dropped before -\textit{ing} and -\textit{ed}
- Example: make/making

### E Insertion
- E added after -\textit{s}, -\textit{z}, -\textit{x}, -\textit{ch}, -\textit{sh} before -\textit{s}
- Example: watch/watches

### Y Replacement
- Y changes to -\textit{ie} before -\textit{s}, -\textit{i} before -\textit{ed}
- Example: try/tries

### K Insertion
- Verbs ending with vowel + -c add -k
- Example: panic/panicked
Complete Morphological Parser

f o x +N +PL

LEXICON-FST

f o x ^ s #

FST_1 orthographic rules FST_n

f o x e s
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

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

• Rule $\rightarrow$ FST compiler handles the rest...
FSTs and Ambiguity

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

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

• Morphology
  – Overview of morphological processes
  – Computational morphology with finite-state methods