More on text searching

- Searching for *woodchuck* (singular) and *woodchucks* (plural)
  - Regular expression `/woodchucks/`
- What about:
  - fox -> foxes, peccary -> peccaries, fish -> fish, goose -> geese
- Morphological parsing
  - Recognizing that a word breaks down into component morphemes and building a structured representation of this fact
Morphological parsing

• Where needed?
  ♦ Web search (for morphologically complex languages)
    ▪ Search for inflected forms of the base form the user typed in
  ♦ Part-of-Speech tagging
  ♦ Dictionaries for spell-checking
Morphological parsing

- Why not store all word forms in a dictionary?
  - Many formations are predictable and productive
    - Automatically includes any new word
    - New words constantly enter into a language
- Better to separate the lexicon and morphological rules
Words

- Finite-state methods are particularly useful in dealing with a lexicon
- Many devices, most with limited memory, need access to large lists of words
- And they need to perform fairly sophisticated tasks with those lists
- So we’ll first talk about some facts about words and then come back to computational methods
English Morphology

- Morphology is the study of the ways that words are built up from smaller meaningful units called morphemes.
- We can usefully divide morphemes into two classes:
  - **Stems**: The core meaning-bearing units.
  - **Affixes**: Bits and pieces that adhere to stems to change their meanings and grammatical functions.
Example

- **Word form**: cats
  - Stem: cat
  - Suffix: s

- **Word form**: foxes
  - Stem: fox
  - Suffix: es

- **Word form**: rewrites
  - Stem: write
  - Prefix: re
  - Suffix: s
English Morphology

• We can further divide morphology up into two broad classes
  ♦ Inflectional
  ♦ Derivational
Inflectional Morphology

• Inflectional morphology concerns the combination of stems and affixes where the resulting word:
  - Has the same word class as the original
  - Serves a grammatical/semantic purpose that is
    - Different from the original
    - But is nevertheless transparently related to the original
Nouns and Verbs in English

• Nouns are simple
  - Markers for plural and possessive
• Verbs are only slightly more complex
  - Markers appropriate to the tense of the verb
Regulars and Irregulars

• It is a little complicated by the fact that some words misbehave (refuse to follow the rules)
  - Mouse/mice, goose/geese, ox/oxen
  - Go/went, fly/flew
• The terms regular and irregular are used to refer to words that follow the rules and those that don’t
Regular and Irregular Verbs

- Regulars...
  - walk, walks, walking, walked, walked

- Irregulars
  - eat, eats, eating, ate, eaten
  - catch, catches, catching, caught, caught
  - cut, cuts, cutting, cut, cut
Inflectional Morphology

- So inflectional morphology in English is fairly straightforward
- But is complicated by the fact that are irregularities
Derivational Morphology

- Derivational morphology is the messy stuff that no one ever taught you.
  - Quasi-systematicity
  - Irregular meaning change
  - Changes of word class
### Derivational Examples

- Verbs (V) and Adjectives (A) to Nouns (N)
  - nominalization

<table>
<thead>
<tr>
<th></th>
<th>Verbs</th>
<th>Nouns</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ation</td>
<td>compute (V)</td>
<td>computerization (N)</td>
</tr>
<tr>
<td>-ee</td>
<td>appoint (V)</td>
<td>appointee (N)</td>
</tr>
<tr>
<td>-er</td>
<td>kill (V)</td>
<td>killer (N)</td>
</tr>
<tr>
<td>-ness</td>
<td>fuzzy (A)</td>
<td>fuzziness (N)</td>
</tr>
</tbody>
</table>
## Derivational Examples

- **Nouns (N) and Verbs (V) to Adjectives (A)**

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Noun</th>
<th>Verb</th>
<th>Adjective</th>
</tr>
</thead>
<tbody>
<tr>
<td>-al</td>
<td>computation (N)</td>
<td>computational (A)</td>
<td></td>
</tr>
<tr>
<td>-able</td>
<td>embrace (V)</td>
<td>embraceable (A)</td>
<td></td>
</tr>
<tr>
<td>-less</td>
<td>clue (N)</td>
<td>clueless (A)</td>
<td></td>
</tr>
</tbody>
</table>
Example: *Compute*

- Many paths are possible...
- Start with *compute*
  - Computer -> computerize -> computerization
  - Computer -> computerize -> computerizable
- But not all paths/operations are equally good (allowable?)
  - Clue
    - Clue -> *clueable*
Morphological parsing

• Our goal is to convert input like:
  ◦ cats => cat + N + PL
  ◦ cat => cat + N + SG
  ◦ cities => city + N + PL
  ◦ geese => goose + N + PL
  ◦ merging => merge + V + PresPart

• We need a morphological parser/analyzer
Morphological parser

1. Lexicon
   - the list of stems and affixes

2. Morphotactics
   - Explains which classes of morphemes can follow other class of morphemes inside a word

3. Orthographic rules
   - Spelling rules used to model the changes that occur in a word
Morphology and FSAs

- We’d like to use the machinery provided by FSAs to capture these facts about morphology
  - Accept strings that are in the language
  - Reject strings that are not
  - And do so in a way that doesn’t require us to in effect list all the words in the language
Finite-State Lexicon construction

- Start simple
- Regular singular nouns are ok
- Regular plural nouns have an -s on the end
- Irregulars are ok as is
Simple Rules

\[ q_0 \xrightarrow{\text{reg-noun}} q_1 \xrightarrow{\text{plural -s}} q_2 \]

irreg-pl-noun

irreg-sg-noun
Now Plug in the Words
If everything is an accept state how do things ever get rejected?
We can now run strings through these machines to recognize strings in the language.

But recognition is usually not quite what we need:
- Often if we find some string in the language we might like to assign a structure to it (parsing).
- Or we might have some structure and we want to produce a surface form for it (production/generation).

Example:
- From “cats” to “cat +N +PL”
Finite State Transducers

• The simple story
  - Add another tape
  - Add extra symbols to the transitions

  - On one tape we read “cats”, on the other we write “cat +N +PL”
FSTs

Lexical: cats +N +Pl

Surface: cats
Applications

• The kind of parsing we’re talking about is normally called **morphological analysis**

• It can either be
  • An important stand-alone component of many applications (spelling correction, information retrieval)
  • Or simply a link in a chain of further linguistic analysis
Transitions

- c:c means read a c on one tape and write a c on the other
- +N: ε means read a +N symbol on one tape and write nothing on the other
- +PL:s means read +PL and write an s
Typical Uses

• Typically, we’ll read from one tape using the first symbol on the machine transitions (just as in a simple FSA).
• And we’ll write to the second tape using the other symbols on the transitions.
The Gory Details

- Of course, it's not as easy as
  - “cat +N +PL” <-> “cats”
- As we saw earlier there are geese, mice and oxen
- But there are also a whole host of spelling/pronunciation changes that go along with inflectional changes
  - Fox and Foxes
Multi-Tape Machines

• To deal with these complications, we will add more tapes and use the output of one tape machine as the input to the next
• So to handle irregular spelling changes we’ll add intermediate tapes with intermediate symbols
Multi-Level Tape Machines

- We use one machine to transduce between the lexical and the intermediate level, and another to handle the spelling changes to the surface tape.
Lexical to Intermediate Level
Intermediate to Surface

- The add an “e” rule as in $\text{fox}^s# \longleftrightarrow \text{foxes}#$
Foxes

Lexical

Intermediate

Surface

$T_{\text{lex}}$

$T_{\text{e-insert}}$
Note

- A key feature of this machine is that it doesn’t do anything to inputs to which it doesn’t apply.
- Meaning that they are written out unchanged to the output tape.
Overall Scheme

• We now have one FST that has explicit information about the lexicon (actual words, their spelling, facts about word classes and regularity).
  • Lexical level to intermediate forms
• We have a larger set of machines that capture orthographic/spelling rules.
  • Intermediate forms to surface forms
Overall Scheme

LEXICON-FST

\[ \text{LEXICON-FST} \]

\[ \text{FST}_1 \text{ orthographic rules } \text{FST}_n \]

<table>
<thead>
<tr>
<th>f</th>
<th>o</th>
<th>x</th>
<th>+N</th>
<th>+PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>o</td>
<td>x</td>
<td>^s</td>
<td>#</td>
</tr>
<tr>
<td>f</td>
<td>o</td>
<td>x</td>
<td>e</td>
<td>s</td>
</tr>
</tbody>
</table>
Cascades

• This is an architecture that we’ll see again and again
  • Overall processing is divided up into distinct rewrite steps
  • The output of one layer serves as the input to the next
  • The intermediate tapes may or may not wind up being useful in their own right
HFST

• Helsinki Finite-State Transducer Technology
• http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/
• Includes demos to try