CMPT-825
Natural Language Processing

Anoop Sarkar
http://www.cs.sfu.ca/~anoop
Grammar Development: Inflectional morphology

• Write an NFA for the following data such that each suffix type gets a single transition (e.g. adding an -s is the plural suffix):

  cat    cats
  dog    dogs
  fox    foxes
  mouse  mice

• Note that foxes is not an isolated case (irregular), e.g. suffix, suffixes
Grammar Development: Derivational morphology

- Write an NFA for the following data (ignore the parts of speech, also you can use substrings on the transitions, e.g. a transition can have `demon` on it):

  `demon/N  demon+ize/V`
  `demon+ize+ation/N  demon+ize+able/A`
  `demon+ize+er/N`
  `formal/A  formal+ity/N`
  `formal+ness/N`
Grammar Development: Derivational morphology

- Does your NFA accept the following additional strings (ignore the parts of speech). If not, what do you need to add to your previous NFA?

  formal+ize/V formal+ize+ation/N  
  formal+ize+able/A formal+ize+er/N  
  demon+ize+able+ity/N
Dealing with foxes: Finite-state transducers

- $\alpha:0$ is notation for a map between two alphabets: $\Sigma_1$ and $\Sigma_2$

- FSTs accept pairs of strings. Language accepted by an FST: $L \subseteq (\Sigma_1^*, \Sigma_2^*)$
Dealing with foxes: Finite-state transducers

- FST for \((ab, 00), (aa, 0)\)

![Diagram 1](ab, 00), (aa, 0)

- FST for \((\epsilon, \epsilon), (ab, 01), (abab, 0101), \ldots\)

![Diagram 2](\epsilon, \epsilon), (ab, 01), (abab, 0101), \ldots}
Dealing with *foxes*: Finite-state transducers

- Draw FST for \((\epsilon, \epsilon), (ab, 00), (ab, 01), (ab, 10), (ab, 11), (abab, 0000), \ldots\)
Finite-state Transducers

- The mystery transducer: what does it do?
Morphological Parsing with Transducers

- Simpler to start by thinking of it as generation

- Start with cat +N +PL and then use a FST to produce cats

- Advantage: since we can add/delete material, we can handle fox +N +PL to get the correct form foxes

- To deal with orthographic rules (like the one above), J&M provide an analysis which is made very modular with the use of an intermediate FST
Morphological Parsing with Transducers

- Draw a transducer for the following examples:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat+N+PL</td>
<td>cat's#</td>
<td>cat+N+SG</td>
<td>cat#</td>
</tr>
<tr>
<td>dog+N+PL</td>
<td>dog's#</td>
<td>dog+N+SG</td>
<td>dog#</td>
</tr>
<tr>
<td>fox+N+PL</td>
<td>fox's#</td>
<td>fox+N+SG</td>
<td>fox#</td>
</tr>
<tr>
<td>mouse+N+PL</td>
<td>mice#</td>
<td>mouse+N+SG</td>
<td>mouse#</td>
</tr>
</tbody>
</table>
Morphological Parsing with Transducers

- A transducer for the $e$-insertion rule
  if word ends in $x^s#$ then output $xes$; similarly for $z^#$ and $s^#$
  note the use of the intermediate output from the previous transducer

  define $other = [a-r,t-w,y]$

<table>
<thead>
<tr>
<th>Input</th>
<th>Inter.</th>
<th>Output</th>
<th>Input</th>
<th>Inter.</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat+N+PL</td>
<td>cat$^s#$</td>
<td>cats</td>
<td>cat+N+SG</td>
<td>cat#</td>
<td>cat</td>
</tr>
<tr>
<td>fox+N+PL</td>
<td>fox$^s#$</td>
<td>foxes</td>
<td>fox+N+SG</td>
<td>fox#</td>
<td>fox</td>
</tr>
</tbody>
</table>
Ambiguity when Parsing with FSTs

- Global ambiguity:
  foxes $\rightarrow$ fox+N+PL OR foxes+V+3SG
  I saw two foxes yesterday
  That trickster foxes me every time

- Local ambiguity:
  assess has a prefix string which can be analyzed:
  ass+N+PL $\rightarrow$ asses

- An FST will return the two answers in the first case, but only return one answer in the second case (even though it will consider a false analysis partway through the string)
Deterministic vs. Non-deterministic

- Deterministic transducers are called *subsequential* transducers (no backtracking when translating one string to another)

- Deterministic transducers where all the states are final states are called *sequential* transducers.
Porter Stemmer

- Unlike our previous FSTs, the Porter Stemmer has no stems. This makes the FST much smaller as a result – leading to a simple implementation (available widely on the web in many programming languages).

- *ational → ate*
  *ing → ε* if stem contains vowel (e.g. motoring, motor)

- Performs well enough most of the time, but suffers from problems that the FSTs we saw earlier do not have: *organization → organ*
  Still, it is used often for quick and dirty stemming in many NLP applications due to its simplicity and speed.
FST Software in Research and Industry

- FSTs are used for many applications in NLP: morphology, stemming, segmentation

- FST software:
  
  Van Noord fsa  
  URL:http://odur.let.rug.nl/~Evannoord/Fsa/  
  AT&T fsm toolkit  
  URL:http://www.research.att.com/sw/tools/fsm/  
  Xerox LinguistX  
  URL:http://www.inxight.com/products/oem/linguistx/  
  Teragram  
  URL:http://www.teragram.com/

- FSTs are also widely used in aligning sequences in genomics