Finite state morphology and phonology

Natural Language Processing
LING/CSCI 5832

Mans Hulden
Dept. of Linguistics
mans.hulden@colorado.edu

Jan 22 2014
Composition

im+possibility

im+possible+ity

im+possible+ity

im+possibility

impossibility

impossibility
Composition

NEG+possible+ity+NOUN+PLURAL

in+possible+ity+s

im+possible+ity+s

im+possibility+s

impossibilities

NEG+possible+ity+NOUN+PLURAL

impossibility
Compilers

Several finite-state compilers available to do the hard work

- Xerox xfst (http://www.fsmbook.com)
- SFST (https://code.google.com/p/cistern/wiki/SFST)
- HFST (http://hfst.sf.net)
- OpenFST (http://www.openfst.org)
- Foma (http://foma.googlecode.com)

Demo with foma
Toy grammar of English

Toy lexicon: kiss, hire, spy
Possible suffixes: ed, ing, s
Generate kiss+s/kisses, spy+ed/spied, hire+ing/hiring, hire+ed/hired, etc.

More advanced version of this in tutorial form on:
https://code.google.com/p/foma/wiki/MorphologicalAnalysisTutorial
## Some derivations

<table>
<thead>
<tr>
<th></th>
<th>hire+ing</th>
<th>hire+ed</th>
<th>kiss+s</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edelete</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EInsert</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Delete +</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>hiring</th>
<th>hired</th>
<th>kisses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edelete</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EInsert</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Delete +</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Code

def Stems      s p y | k i s s | h i r e ;
def Suffixes   "+" [ 0 | s | e d | i n g ];

def Lexicon   Stems  Suffixes ;

def YRule1       y -> i e || _ "+" s ;    # spy+s > spie+s
def YRule2       y -> i || _ "+" e d ;    # spy+ed > spi+ed
def Einsert     "+" -> e || s _ s ;         #kiss+s > kisses
def Edelete      e -> 0 || _ "+" [e|i];  #hire+ed > hir+ed, hire+ing > hir+ing
def Cleanup     "+" -> 0 ;                      #hir+ing >hiring, etc.

def Grammar         Lexicon .o. YRule1 .o. YRule2 .o. Einsert .o. Edelete .o. Cleanup;
regex Grammar;
def Stems spy | kiss | hire;
def Suffixes 0:"+" [ "[INF]" : 0 | "[NOUN][SINGULAR]" : 0 | "[PRES]" : s | 
"[NOUN][PLURAL]" : s | "[PASTPART]" : [ed] | "[PRESPART]" : [ing] ];
def Lexicon Stems Suffixes;

def YRule1 y -> i e || _ "+" s;
def YRule2 y -> i || _ "+" ed;
def Einser t "+" -> "+" e || s _ s;
def Edelete e -> O || _ "+" [e|i];
def Cleanup "+" -> O;

def Grammar Lexicon .o. YRule1 .o. YRule2 .o. Einser t .o. Edelete .o. Cleanup;
regex Grammar;
The 2 second spell checker

NEG+possible+ity+NOUN+PLURAL

(1) Extract the possible outputs of the “Grammar” transducer, and convert to automaton (output-side projection)

(2) Test a word against automaton
The 5 second spelling corrector [med]

Assume we have a list of words as a repeating FST as before

hired

W

hired
The 5 second spelling corrector

Assume we have a list of words as a repeating FST as before

Now, create a transducer $C_1$ that makes one change in a word (one deletion, one change, one insertion)

```
abc
```

```
0
1

@<@><@<@><@>
```

```
ab, bc, ac, aba, aac, abca, ...
```
The 5 second spelling corrector

Compose

hired

W

hired

CI

xire, hird, hird, hiredx, ired, hied,...
The 5 second spelling corrector

Compose

W o CI

xire, hird, hird, hiredx, ired, hied,...
Code

analyzer3.foma

def Stems s p y | k i s s | h i r e ;
def Suffixes 0:"+" [ "[INF]":0 | "[NOUN][SINGULAR]":0 | "[PRES]":s | "[NOUN] [PLURAL]":s | "[PASTPART]":[e d] | "[PRESPART]":[i n g] ];

def Lexicon Stems Suffixes ;

def YRule1 y -> i e || _ "+" s ;
def YRule2 y -> i || _ "+" e d ;
def Epenthesis "+" -> "+" e | | s s ;
def Erule e -> 0 | | _ "+" [e|i];
def Cleanup "+" -> 0 ;

def Grammar Lexicon .o. YRule1 .o. YRule2 .o. Epenthesis .o. Erule .o. Cleanup ;

def C1 ?* [?:0|0:?|?:?:-?] ?* ;

regex Grammar.2 .o. C1 ;
Entirely non-orthographic grammar

def Stems  s p Λ I | k ι s | h ι r;
def Suffixes 0:"+" ["[INF]"o | "[PRES]"z | "[PASTPART]"d | "[PRESPART]"[ι η] ];

def Sib [s|z]; # Sibilants
def Unvoiced [h|s]; # Unvoiced phonemes
define ObsAssimilation d -> t | | Unvoiced "+" _ ;
define Epenthesis [..] -> τ | | Sib "+" _ Sib;
define Cleanup "+" -> 0;

def Lexicon    Stems Suffixes ;
def Grammar    Lexicon .o. ObsAssimilation .o. Epenthesis .o. Cleanup;

regex Grammar;
Applications

- Tokenization
- POS tagging
- Shallow parsing (chunking)
- Syntactic parsing
- Information extraction
- Text-to-speech
- Spell checking/correction
- Electronic dictionaries
- Machine translation
- …
Syntactic parsing

- Generally consults a separate morphological analyzer

```
S
  /   \\
NP  VP \\
  /   \\
Det Adj N \\
  /    \\
the mome raths
```

Morphological analyzer:

- `outgrabe +V+Past`
- `outgrabe +N+Sg`
Wrapup

• The above are standard techniques - morphological/phonological grammars have been written for hundreds of languages in this way.

• The calculus is crucial - thinking about states and transitions is counterproductive.

• A well-designed grammar should be very accurate, barring misspellings (easily >99% recall).

• There are also probabilistic extensions to all of the above (to combine with language models, to handle noisy data, etc.).

• These grammars are also used to improve POS-taggers, parsers, chunkers, named entity recognition, etc.