T-(538|725)-MALV, Natural Language Processing

Hrafn Loftsson¹  Hannes Högni Vilhjálmsson¹

¹School of Computer Science, Reykjavik University

October 2009
Outline

1. Two-level rules

2. lexc
Outline

1. Two-level rules

2. lexc
# Two-level rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a:b \Rightarrow lc _ rc)</td>
<td>(a) is transduced as (b) only when it has (lc) to the left and (rc) to the right</td>
</tr>
<tr>
<td>(a:b \Leftarrow lc _ rc)</td>
<td>(a) is always transduced as (b) when it has (lc) to the left and (rc) to the right</td>
</tr>
<tr>
<td>(a:b \Leftrightarrow lc _ rc)</td>
<td>(a) is transduced as (b) always and only when it has (lc) to the left and (rc) to the right</td>
</tr>
<tr>
<td>(a:b /\Leftarrow lc _ rc)</td>
<td>(a) is never transduced as (b) when it has (lc) to the left and (rc) to the right</td>
</tr>
</tbody>
</table>

More detail:

http://www.sil.org/pckimmo/v2/doc/Rules_1.html#subsec:3.1.4
Two-level rules

An example from English

<table>
<thead>
<tr>
<th>Examples</th>
<th>happy+er</th>
<th>party+s</th>
<th>marry+ed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>happi0er</td>
<td>parties</td>
<td>marri0ed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules</th>
<th>Cy+er</th>
<th>Cy+s</th>
<th>Cy+ed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy0er</td>
<td>Cies</td>
<td>Ci0ed</td>
<td></td>
</tr>
</tbody>
</table>

1. \[ y:i \leftarrow C:C \_ +:0 \ e:e \ r:r \]
2. \[ y:i \leftarrow C:C \_ +:e \ s:s \]
3. \[ y:i \leftarrow C:C \_ +:0 \ e:e \ d:d \]

All the rules are applied in **parallel**. Every rule must be successfully applied to the current pair of characters *lexical:surface* before moving to the next pair (see Fig. 5.10 page 139).
A two-level rule can be compiled into an equivalent finite-state transducer.

A program which performs two-level morphological analysis:
- The user writes a set of two-level rules.
- The program compiles the rules into transducers.

Example: **lexc** (Lexicon Compiler), a part of the **XFST** (Xerox Finite-State Tool).

Example: **hfst-lexc** (Helsinki Finite-state tools); http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/

Example: **PC-Kimmo**, http://www.sil.org/pckimmo/
Outline

1. Two-level rules

2. lexc
**Lexicon Compiler**

- A language (and a compiler) for defining automata and transducers.
- Specifically suitable for defining lexicons.
- Based on two-level morphology.

**The format of a lexc file**

- Multichar_Symbols declaration
- Declarations section
- Lexicon Root
- Lexicon X
- Lexicon Y
- ... 
- END
A simple example

! ex1-lex.txt (this line is a comment)
LEXICON Root
dog # ;
cat # ;
bird # ;
END

Running it

xfst ; Starts up xfst, which waits for input
xfst[0]: read lexc < ex1-lex.txt
xfst[1]:
Running it

xfst[1]: up dog ; from surface to lexical
xfst[1]: down bird ; from lexical to surface
xfst[1]: print words
xfst[1]: clear stack
xfst[0]
LEXICON  Root
walk  # ;
walks  # ;
walked  # ;
walking  # ;

talk  # ;
talks  # ;
talked  # ;
talking  # ;

pack  # ;
packs  # ;
packed  # ;
packing  # ;
LEXICON Root
walk V ;
talk V ;
pack V ;

LEXICON V
s # ;
ed # ;
ing # ;
#; ! <- an empty-string entry
Lexical transducers

Upper:Lower records

LEXICON Root
swim:swam # ;
fight:fought # ;

Running it

xfst[1]: up swim ; from surface to lexical
xfst[1]: down fight ; from lexical to surface
xfst[1]: print words