Regular Expressions

Lê Hồng Phương
Hanoi University of Science
Vietnam National University
phuonglh@gmail.com
Regular Expressions

• A formal language for specifying text strings
  – *Finite-state automaton*

• How can we search for any of these?
  – seahorse
  – seahorses
  – Seahorse
  – Seahorses
Disjunctions

- Letter inside square brackets `[]`

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[sS]eahorse</td>
<td>seahorse, Seahorse</td>
</tr>
<tr>
<td>[1234567890]</td>
<td>Any digit</td>
</tr>
</tbody>
</table>

- Ranges `[A-Z]`

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A-Z]</td>
<td>An uppercase letter</td>
</tr>
<tr>
<td>[a-z]</td>
<td>A lower case letter</td>
</tr>
<tr>
<td>[0-9]</td>
<td>A single digit</td>
</tr>
</tbody>
</table>
Negation in Disjunction

- Negation [^Ss]
  - Carat means negation only when first in []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A-Z]</td>
<td>Not an upper case letter</td>
</tr>
<tr>
<td>[^Ss]</td>
<td>Neither 'S' nor 's'</td>
</tr>
<tr>
<td>a^b</td>
<td>The pattern a carat b</td>
</tr>
</tbody>
</table>
More Disjunction

• The pipe | for disjunction

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>groundhog</td>
<td>woodchuck</td>
</tr>
<tr>
<td>yours</td>
<td>mine</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>[gG]roundhog</td>
<td>[Ww]oodchuck</td>
</tr>
</tbody>
</table>
### Multiplicity: ? * + .

#### Multiplicity notations

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>colou?r</td>
<td>Optional previous char</td>
<td>color  colour</td>
</tr>
<tr>
<td>oo*h!</td>
<td>0 or more of previous char</td>
<td>oh!  ooh!  oooh! oooh!</td>
</tr>
<tr>
<td>o+h!</td>
<td>1 or more of previous char</td>
<td>oh!  ooh!  oooh! oooh!</td>
</tr>
<tr>
<td>baa+</td>
<td></td>
<td>baa, baaa, baaaa</td>
</tr>
<tr>
<td>beg.n</td>
<td>. is any char</td>
<td>begin begun beg3n</td>
</tr>
</tbody>
</table>
Boundary Matchers

• Anchors:
  – ^ : the beginning of a line
  – $ : the end of a line
  – \b : a word boundary
  – \B : a non-word boundary
# Boundary Matchers

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^[A-Z]</code></td>
<td>Palo Alto</td>
</tr>
<tr>
<td><code>[^A-Za-z]</code></td>
<td>1 “Hello”</td>
</tr>
<tr>
<td>.$</td>
<td>The end.</td>
</tr>
<tr>
<td>.$</td>
<td>The end?</td>
</tr>
<tr>
<td>.$</td>
<td>The end!</td>
</tr>
</tbody>
</table>
Example

- Find me all instances of the word “the” in a text.
- Regular expressions:
  - `the` → misses capitalized instances
  - `[Tt]he` → incorrectly matches other or theology
  - `[^A-Za-z][Tt]he[^A-Za-z]`
More Examples

• What do the following regular expressions match?
  
  - ([0-9]*[\.,])?[0-9]+%
  
  - ([A-Z][a-z]+)(\s+[A-Z][a-z]+)+
Summary

- Regular expressions play a surprisingly large role.
  - Sophisticated sequences of regular expressions are often the first model for any text processing task.
- For many hard tasks, we use machine learning classifiers.
  - But regular expressions are used as features in these classifiers
  - Can be very useful in capturing generalization.
Exercises

• Vietnamese tokenizer (vnTokenizer)
  – http://mim.hus.vnu.edu.vn/phuonglh/softwares/vnTokenizer

• Design regular expressions to extract simple text patterns before performing word segmentation

• Java Regex Test GUI
  – http://sourceforge.net/projects/javaregextestgui/