Regular Expressions

- using REs to find patterns
- implementing REs using finite state automata
REs and FSAs

• Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata

• Finite-state automata are a way of implementing regular expressions
Regular expressions

• A formal language for specifying text strings
• How can we search for any of these?
   woodchuck
   woodchucks
   Woodchuck
   Woodchucks
Regular Expressions for Textual Searches

Who does it?

Everybody:

- Web search engines, CGI scripts
- Information retrieval
- Word processing (Emacs, vi, MSWord)
- Linux tools (sed, awk, grep)
- Computation of frequencies from corpora
- Perl
Whenever I learn a new skill I concoct elaborate fantasy scenarios where it lets me save the day.

Oh no! The killer must have followed her on vacation!

But to find them we'd have to search through 200 MB of emails looking for something formatted like an address!

It's hopeless!

Everybody stand back.

I know regular expressions.

Boom! Tap.
Regular Expression

- **Regular expression**: formula in algebraic notation for specifying a set of strings

- **String**: any sequence of alphanumeric characters
  - letters, numbers, spaces, tabs, punctuation marks

- **Regular expression search**
  - **pattern**: specifying the set of strings we want to search for
  - **corpus**: the texts we want to search through
Basic Regular Expression Patterns

• Case sensitive: d is not the same as D
• Disjunctions: [dD]  [0123456789]
• Ranges: [0–9]    [A–Z]
• Negations: [^Ss]  (only when ^ occurs immediately after [ )
• Optional characters: ? and *
• Wild :  .
• Anchors: ^ and $, also \b and \B
• Disjunction, grouping, and precedence:  (pipe)
Caret for negation, `^`, or anchor

<table>
<thead>
<tr>
<th>RE</th>
<th>Match (single characters)</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[^A-Z]</code></td>
<td>not an uppercase letter</td>
<td>“Oyfn pripetchik”</td>
</tr>
<tr>
<td><code>[^Ss]</code></td>
<td>neither ‘S’ nor ‘s’</td>
<td>“I have no exquisite reason for’t”</td>
</tr>
<tr>
<td><code>[^\.]</code></td>
<td>not a period</td>
<td>“our resident Djinn”</td>
</tr>
<tr>
<td><code>[e/]</code></td>
<td>either ‘e’ or ‘^’</td>
<td>“look up ^ now”</td>
</tr>
<tr>
<td><code>a^b</code></td>
<td>the pattern ‘a^b’</td>
<td>“look up a^b now”</td>
</tr>
<tr>
<td><code>^T</code></td>
<td>T at the beginning of a line</td>
<td>“The Dow Jones closed up one”</td>
</tr>
</tbody>
</table>
## Optionality and Counters

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>woodchucks?</td>
<td>woodchuck or woodchucks</td>
<td>“The woodchuck hid”</td>
</tr>
<tr>
<td>colou?r</td>
<td>color or colour</td>
<td>“comes in three colours”</td>
</tr>
<tr>
<td>(he){3}</td>
<td>exactly 3 “he”’s</td>
<td>“and he said hehehe.”</td>
</tr>
</tbody>
</table>

?  zero or one occurrences of previous char or expression

*  zero or more occurrences of previous char or expression

+  one or more occurrences of previous char or expression

{n}  exactly n occurrences of previous char or expression

{n, m} between n to m occurrences

{n, } at least n occurrences
Wild card ‘.’

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>beg.n</td>
<td>any char between <em>beg</em> and n</td>
<td>begin, beg’n, begun</td>
</tr>
<tr>
<td>big.*dog</td>
<td>find lines where big and dog occur</td>
<td>the big dog bit the little dog bit the</td>
</tr>
</tbody>
</table>
Operator Precedence Hierarchy

1. Parenthesis                  ( )
2. Counters                    * + ? {} 
3. Sequences and Anchors      the ^my end$
4. Disjunction                 | 

Examples:
    /moo+/  
    /try|ies/  
    /and|or/  

Example

• Find all instances of the word “the” in a text.
Example

• Find all instances of the word “the” in a text.

♦ /the/
Example

- Find all instances of the word “the” in a text.
  - /the/

Misses capitalized examples
Example

• Find all instances of the word “the” in a text.
  ♦ /the/
  Misses capitalized examples
  ♦ /[tT]he/
Example

- Find all instances of the word “the” in a text.
  - /the/
    - Misses capitalized examples
  - /[tT]he/
    - Finds other or theology
Example

• Find all instances of the word “the” in a text.

♦ /the/

   Misses capitalized examples

♦ /[tT]he/

   ▪ Finds other or theology

♦ /\b[tT]he\b/
Example

- Find all instances of the word “the” in a text.
  - `/the/`
    - Misses capitalized examples
  - `/[tT]he/`
    - Finds other or theology
  - `/[tT]he\b/`
  - `/[^a-zA-Z][tT]he[^a-zA-Z]/`
Example

• Find all instances of the word “the” in a text.
  ✦ /the/
    Misses capitalized examples
  ✦ /[tT]he/
    ▪ Finds other or theology
  ✦ /\b[tT]he\b/
  ✦ /[^a-zA-Z][tT]he[^a-zA-Z]/
    ▪ Misses sentence-initial “the”
Example

• Find all instances of the word “the” in a text.
  ♦ /the/
    Misses capitalized examples
  ♦ /[tT]he/
    Finds other or theology
  ♦ /\b[tT]he\b/
  ♦ /[^a-zA-Z][tT]he[^a-zA-Z]/
    Misses sentence-initial “the”
  ♦ /(^[^a-zA-Z]) [tT]he[^a-zA-Z]/
Errors

• The process we just went through was based on **fixing two kinds of errors**
  - Matching strings that we should not have matched (**there**, **then**, **other**)
    - False positives (**Type I**)
  - Not matching things that we should have matched (**The**)
    - False negatives (**Type II**)

Sunday, 4 December 11
A more complex example

Write a RE that will match “any PC with more than 500MHz and 32 Gb of disk space for less than $1000”.

• First a RE for prices

\$/[0-9]+/ # whole dollars
\$/[0-9]+\.[0-9][0-9]/ # dollars and cents
\$/[0-9]+(\.[0-9][0-9][0-9])*#/ #cents optional
\b\$/[0-9]+(\.[0-9][0-9][0-9])*\b/ #word boundaries
A more complex example

Write a RE that will match “any PC with more than 500MHz and 32 Gb of disk space for less than $1000”.

• First a RE for prices

\$/[0-9]+/  # whole dollars
\$/[0-9]+\.[0-9][0-9]/  # dollars and cents
\$/[0-9]+(\.[0-9][0-9])?/  # cents optional
\b\$/[0-9]+(\.[0-9][0-9])?\b/  # word boundaries
• **Specifications for processor speed**

  `/\b[0-9]+ *(MHz|[Mm]egahertz|Ghz|[Gg]igahertz)\b/`

• **Memory size**

  `/\b[0-9]+ *(Mb|[Mm]egabytes?)\b/`

  `/\b[0-9]+(\.[0-9]+) *(Gb|[Gg]igabytes?)\b/`

• **Vendors**

  `/\b(Win(95|98|NT|dows *(NT|95|98|2000)?)\b/`

  `/\b(Mac|Macintosh|Apple)\b/`
Substitutions and Memory

- **Substitutions:** `s/regexp/pattern/)`

  `s/color/colour/``

- **Memory** (\1, \2, etc. refer back to found matches) e.g.,
  Put angle brackets around all integers in text

  `the 39 students ==> the <39> students`

  `s/ ([0-9]+) /<\1>/`
## Using Backslash

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>an asterisk “*”</td>
<td>“K<em>A</em>P<em>L</em>A*N”</td>
</tr>
<tr>
<td>.</td>
<td>a period “.”</td>
<td>“Dr. Livingston, I presume”</td>
</tr>
<tr>
<td>?</td>
<td>a question mark</td>
<td>“Would you light my candle?”</td>
</tr>
<tr>
<td>\n</td>
<td>a newline</td>
<td></td>
</tr>
<tr>
<td>\t</td>
<td>a tab</td>
<td></td>
</tr>
</tbody>
</table>
### Some Useful Aliases

<table>
<thead>
<tr>
<th>RE</th>
<th>Expansion</th>
<th>Match</th>
<th>Example Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>[0–9]</td>
<td>any digit</td>
<td>Party of 5</td>
</tr>
<tr>
<td>\D</td>
<td>[^0–9]</td>
<td>any non-digit</td>
<td>99p</td>
</tr>
<tr>
<td>\w</td>
<td>[a-zA-Z0–9_]</td>
<td>any alphanumeric</td>
<td>99p</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or underscore</td>
<td></td>
</tr>
<tr>
<td>\W</td>
<td>[^\w]</td>
<td>a non-alphanumeric</td>
<td>!!!</td>
</tr>
<tr>
<td>\s</td>
<td>[ \r\t\n\f]</td>
<td>whitespace (sp, tab)</td>
<td></td>
</tr>
<tr>
<td>\S</td>
<td>[^\s]</td>
<td>Non-whitespace</td>
<td>in Concord</td>
</tr>
</tbody>
</table>
Substitutions and Memory

• **Substitutions:** \( s/regexp/pattern/ \)

\[
\begin{align*}
&\textbf{s/color/colour/}
\end{align*}
\]

• **Memory** (\( \backslash 1, \backslash 2, \text{ etc. refer back to found matches} \)) e.g.,

Put angle brackets around all integers in text

\( \text{the 39 students} \Rightarrow \text{the <39> students} \)

\[
\begin{align*}
&\textbf{s/ \([0-9]+\) /<\1>/}
\end{align*}
\]
Example

Swap first two words of line

s/\(\w+\) + (\w+)/\2 \1/

% perl -de 42
DB<1> $s = "DOES HE LIKE BEER";
DB<2> print $s;
DOES HE LIKE BEER
DB<3> $s =~ s/\(\w+\) + (\w+)/\2 \1/;
DB<4> print $s;
HE DOES LIKE BEER
Finite State Automata & Regular Expressions

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs and their probabilistic relatives are at the core of much of what we’ll do this quarter
FSAs as Graphs

• Let’s start with the sheep language

♦/baa+!/
FSAs as Graphs

• Let’s start with the sheep language

/ baa+ ! /
Sheep FSA

• We can say the following things about this machine
  ✷ It has 5 states
  ✷ b, a, and ! are in its alphabet
  ✷ q₀ is the start state
  ✷ q₄ is an accept state
  ✷ It has 5 transitions
More Formally

• You can specify an FSA by enumerating the following things.
  • The set of states: Q
  • A finite alphabet: Σ
  • A start state
  • A set of accept/final states
  • A transition function that maps Q×Σ to Q
Yet Another View

- The guts of FSAs can be represented as tables

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a</th>
<th>!</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Yet Another View

• The guts of FSAs can be represented as tables

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>a</th>
<th>!</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2,3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you’re in state 1 and you’re looking at an a, go to state 2
Recognition

• Recognition is the process of determining if a string should be accepted by a machine
• Or… it’s the process of determining if a string is in the language we’re defining with the machine
• Or… it’s the process of determining if a regular expression matches a string
• Those all amount the same thing in the end
Recognition

- Traditionally, (Turing’s notion) this process is depicted with a tape.
Recognition

• Start in the start state
• Examine the current input
• Consult the table
• Go to a new state and update the tape pointer.
• Until you run out of tape.
Tracing a Rejection

$q_0$

a b a ! b
Tracing a Rejection

Slide from Dorr/Monz
Tracing a Rejection

\[ q_0 \]

\[ a \quad b \quad a \quad ! \quad b \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \]

\[ b \quad a \quad a \quad a \quad ! \]

REJECT

Slide from Dorr/Monz
Tracing an Accept

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_3 \rightarrow q_4$

$b, a, a, a, !$
Tracing an Accept

Slide from Dorr/Monz
Tracing an Accept

Slide from Dorr/Monz
Regular expression search

http://www.learn-javascript-tutorial.com/RegularExpressions.cfm#h1.2

Search for the following expressions

- Alice
- brillig
- m.m
- c..c
- [A-Z][A-Z]+  
- J|j
- (J|j)
- \(.\)
- \(.*\)
- l.*l
- l.*?l
- l.+l

What does . stand for? (any character)
* is for repetition - zero or more times
[aeiou] is for any vowel
More Examples

Finite State Automata and Regular Expressions
Rara is similar
ra(ra)*
Now we try to write finite state machines that will search for regular expressions.
fl(i|o)p
regular expressions

• any character is a regexp
  • matches itself
• if R and S are regexps, so is RS
  • matches a match for R followed by a match for S
• if R and S are regexps, so is R|S
  • matches any match for R or S (or both)
• if R is a regexp, so is R* (R+)
  • matches any sequence of 0 (1) or more matches for R