Regular Expressions and Finite State Automata

Mausam

(Based on slides by Jurafsky & Martin, Julia Hirschberg)
Regular Expressions and Text Searching

• Everybody does it
  ◦ Emacs, vi, perl, grep, etc..
• Regular expressions are a compact textual representation of a set of strings representing a language.
<table>
<thead>
<tr>
<th>RE</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/woodchucks/</td>
<td>“interesting links to woodchucks and lemur”</td>
</tr>
<tr>
<td>/a/</td>
<td>“Mary Ann stopped by Mona’s”</td>
</tr>
<tr>
<td>/Claire_says,/</td>
<td>“Dagmar, my gift please,” Claire says,”</td>
</tr>
<tr>
<td>/DOROTHY/</td>
<td>“SURRENDER DOROTHY”</td>
</tr>
<tr>
<td>/!/</td>
<td>“You’ve left the burglar behind again!” said Nori</td>
</tr>
<tr>
<td>RE</td>
<td>Match</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>/[wW]oodchuck/</td>
<td>Woodchuck or woodchuck</td>
</tr>
<tr>
<td>/[abc]/</td>
<td>‘a’, ‘b’, or ‘c’</td>
</tr>
<tr>
<td>/[1234567890]/</td>
<td>any digit</td>
</tr>
</tbody>
</table>
## Regular Expressions

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ [A–Z] /</td>
<td>an upper case letter</td>
<td>“we should call it ‘Drenched Blossoms’”</td>
</tr>
<tr>
<td>/ [a–z] /</td>
<td>a lower case letter</td>
<td>“my beans were impatient to be hoed!”</td>
</tr>
<tr>
<td>/ [0–9] /</td>
<td>a single digit</td>
<td>“Chapter 1: Down the Rabbit Hole”</td>
</tr>
</tbody>
</table>
## Regular Expressions

<table>
<thead>
<tr>
<th>RE</th>
<th>Match (single characters)</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A-Z]</td>
<td>not an upper case letter</td>
<td>“Oyfn priпetchik”</td>
</tr>
<tr>
<td>[^Ss]</td>
<td>neither ‘S’ nor ‘s’</td>
<td>“I have no exquisite reason for’t”</td>
</tr>
<tr>
<td>[^.]</td>
<td>not a period</td>
<td>“our resident Djinn”</td>
</tr>
<tr>
<td>[e^]</td>
<td>either ‘e’ or ‘^’</td>
<td>“look up ^ now”</td>
</tr>
<tr>
<td>a^b</td>
<td>the pattern ‘a^b’</td>
<td>“look up a^b now”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>colou?r</code></td>
<td>Optional previous char <code>color</code> <code>colour</code></td>
</tr>
<tr>
<td><code>oo*h!</code></td>
<td>0 or more of previous char <code>oh!</code> <code>ooh!</code> <code>oooh!</code> <code>ooooh!</code></td>
</tr>
<tr>
<td><code>o+h!</code></td>
<td>1 or more of previous char <code>oh!</code> <code>ooh!</code> <code>oooh!</code> <code>ooooh!</code></td>
</tr>
<tr>
<td><code>baa+</code></td>
<td><code>baa  baaa  baaaa  baaaaa</code></td>
</tr>
<tr>
<td><code>beg.n</code></td>
<td><code>begin  begun  begun  begun  beg3n</code></td>
</tr>
</tbody>
</table>
### Regular Expressions: Anchors

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>^[A-Z]</td>
<td><em>Palo Alto</em></td>
</tr>
<tr>
<td>^[^A-Za-z]</td>
<td><em>&quot;Hello&quot;</em></td>
</tr>
<tr>
<td>.$</td>
<td><em>The end</em></td>
</tr>
<tr>
<td>$</td>
<td><em>The end? The end!</em></td>
</tr>
</tbody>
</table>
# Regular Expressions

<table>
<thead>
<tr>
<th>RE</th>
<th>Expansion</th>
<th>Match</th>
<th>Examples</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>Blue moon</td>
</tr>
<tr>
<td>\w</td>
<td>[a-zA-Z0-9_]</td>
<td>any alphanumeric/underscore</td>
<td>Daiyu</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 (space, tab)</td>
<td></td>
</tr>
<tr>
<td>\S</td>
<td>[^\s]</td>
<td>Non-whitespace</td>
<td>in Concord</td>
</tr>
</tbody>
</table>
### Regular Expressions

<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>“Why don’t they come and lend a hand?”</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>
Example

• Find all the instances of the word “the” in a text.
  ◦ /the/
  ◦ /[tT]he/
  ◦ /\b[tT]he\b/
  ◦ [^a-zA-Z][tT]he[^a-zA-Z]
  ◦ (^[^a-zA-Z]) [tT]he ($) [^a-zA-Z] )
Errors

• The process we just went through was based on two fixing 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)
Errors

- We’ll be telling the same story for many tasks, all semester. Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).
Finite State Automata

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs capture significant aspects of what linguists say we need for **morphology** and parts of **syntax**.
FSAs as Graphs

• Let’s start with the sheep language from Chapter 2
  ♦ /baa+/
Sheep FSA

- We can say the following things about this machine:
  - It has 5 states
  - $b, a, \text{ and }!$ are in its alphabet
  - $q_0$ is the start state
  - $q_4$ is an accept state
  - It has 5 transitions
But Note

- There are other machines that correspond to this same language.
More Formally

• You can specify an FSA by enumerating the following things.
  - The set of states: $Q$
  - A finite alphabet: $\Sigma$
  - A start state
  - A set of accept/final states
  - A transition function that maps $Q \times \Sigma$ to $Q$
Dollars and Cents
Dollars and Cents
• The guts of FSAs can ultimately be represented as tables

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

• Simply a process of starting in the start state
• Examining the current input
• Consulting the table
• Going to a new state and updating the tape pointer.
• Until you run out of tape.


\textbf{D-Recognize}

\begin{verbatim}
function D-RECOGNIZE(tape, machine) returns accept or reject

index ← Beginning of tape
current-state ← Initial state of machine
loop
  if End of input has been reached then
    if current-state is an accept state then
      return accept
    else
      return reject
  elseif transition-table[current-state, tape[index]] is empty then
    return reject
  else
    current-state ← transition-table[current-state, tape[index]]
    index ← index + 1
  end
end
\end{verbatim}
Key Points

• Deterministic means that at each point in processing there is always one unique thing to do (no choices).
• D-recognize is a simple table-driven interpreter
• The algorithm is universal for all unambiguous regular languages.
  ♦ To change the machine, you simply change the table.
Key Points

- Crudely therefore... matching strings with regular expressions (ala Perl, grep, etc.) is a matter of
  - translating the regular expression into a machine (a table) and
  - passing the table and the string to an interpreter
Generative Formalisms

- **Formal Languages** are sets of strings composed of symbols from a finite set of symbols.
- Finite-state automata define formal languages (without having to enumerate all the strings in the language)
- The term *Generative* is based on the view that you can run the machine as a generator to get strings from the language.
Generative Formalisms

• FSAs can be viewed from two perspectives:
  - Acceptors that can tell you if a string is in the language
  - Generators to produce *all and only* the strings in the language
Non-Determinism

\[ q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3 \xrightarrow{a} ! \]

\[ q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3 \xrightarrow{a} ! \]
Non-Determinism cont.

- Yet another technique
  - Epsilon transitions
  - Key point: these transitions do not examine or advance the tape during recognition
Equivalence

• Non-deterministic machines can be converted to deterministic ones with a fairly simple construction
• That means that they have the same power; non-deterministic machines are not more powerful than deterministic ones in terms of the languages they can accept
ND Recognition

- Two basic approaches (used in all major implementations of regular expressions, see Friedl 2006)
  1. Either take a ND machine and convert it to a D machine and then do recognition with that.
  2. Or explicitly manage the process of recognition as a state-space search (leaving the machine as is).
Non-Deterministic Recognition: Search

- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.
Non-Deterministic Recognition

- So **success** in non-deterministic recognition occurs when a path is found through the machine that ends in an accept.
- **Failure** occurs when all of the possible paths for a given string lead to failure.
Example

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

b \rightarrow a \rightarrow a \rightarrow a \rightarrow ! \rightarrow \backslash
Example

1

b a a !

q₀  q₁  q₂  q₃  q₄

b  a  a  !
Example

1

\[
\begin{array}{cccc}
\underline{b} & a & a & a \\
q_0 & q_1
\end{array}
\]

2

\[
\begin{array}{cccc}
\underline{b} & a & a & a \\
q_0 & q_1 & q_2 & q_3 & q_4
\end{array}
\]

\[
\begin{array}{cccc}
b & a & a & ! \\
q_0 & q_1 & q_2 & q_3 & q_4
\end{array}
\]
Example
Example

1. \( baaa! \)

2. \( baaa! \)

3. \( baaa! \)

4. \( baaa! \)
Example

1. b a a a!

2. b a a a!

3. b a a a!

4. b a a a!

5. b a a a!

Diagram:

$\begin{array}{c}
q_0 \\
q_1 \\
q_2 \\
q_3 \\
q_4 \\
\end{array}$

b → a → a → !
Example

1. b a a a!

2. b a a a!

3. b a a a!

4. b a a a!

5. b a a a!

6. b a a a!

Diagram:

$q_0 \rightarrow b \rightarrow a \rightarrow a \rightarrow a \rightarrow !$

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

1

2

3

4

5

6

7

q₀

q₁

q₂

q₃

q₄

Speech and Language Processing - Jurafsky and Martin

11/3/2020
Example
Key Points

• States in the search space are pairings of tape positions and states in the machine.
• By keeping track of as yet unexplored states, a recognizer can systematically explore all the paths through the machine given an input.
FSTs (Contd)
FST Fragment: Lexical to Intermediate

- ^ is morpheme boundary; # is word boundary
Putting Them Together

Lexical: f o x +N +Pl

Intermediate: f o x ^ s #

Te-insert: 0 0 0 1 2 3 4 0

Surface: f o x e s
Practical Uses

• This kind of parsing is normally called morphological analysis

• Can be
  • An important stand-alone component of an application (spelling correction, information retrieval, part-of-speech tagging,...)
  • Or simply a link in a chain of processing (machine translation, parsing,...)
#!/usr/bin/perl

$letternumber = "[A-Za-z0-9]";
$notletter = "[^A-Za-z0-9]";
$alwayssep = "[/\?(!()";/\"";
$clitic = "('|:|--'|S'|D'|H'|LL'|RE'|VE'|N'T'|'n'|'d'|'m'|'ll'|'re'|'ve'|'n't');

$abbr("Co.") = 1; $abbr("Dr.") = 1; $abbr("Jan.") = 1; $abbr("Feb.") = 1;

while ($line = <>){
    # read the next line from standard input
    $line =~ s/$letternumber/ $& /g;
    $line =~ s/\s+/\s+ /g;
    $line =~ s/\s+(\d+\d)/ $& , /g;
    $line =~ s/\s+([\d+\d])/ , $&/g;
    $line =~ s/\s+([^\s]+)/ $& , $1/g;

    # distinguish single quotes from apostrophes by
    # segmenting off single quotes not preceded by letter
    $line =~ s/\"\'/\$/g;
    $line =~ s/$notletter/\" \'/g;

    # segment off unambiguous word-final clitics and punctuation
    $line =~ s/$clitics\$/ $&/g;
    $line =~ s/$clitic($notletter)/ $1 $2/g;

    # now deal with periods. For each possible word
    @possiblewords = split(/\s+/, $line);
    foreach $word (@possiblewords) {
        # if it ends in a period,
        if ($word =~ /\$letternumber\$/)
            # and isn't on the abbreviation list
            # and isn't a sequence of letters and periods (U.S.)
            # and doesn't resemble an abbreviation (no vowels; Inc.)
            && ($word =~ "([A-Za-z]\.([A-Za-z]\.)+[A-Z]\[bcdfgh]\-nptvxz]+\.\s/)
        # then segment off the period
            $word =~ s/\.$/ ./;
    }
    # expand clitics
    $word =~ s/\vehave;/
    $word =~ s/\m/am/
        print $word, " ";
    }

    print "\n";
}
Porter Stemmer (1980)

• Common algorithm for stemming English

• Conventions + 5 phases of reductions
  ♦ phases applied sequentially
  ♦ each phase consists of a set of commands
  ♦ sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*
Porter Stemmer (1980)

- Standard, very popular and usable stemmer (IR, IE) – identify a word’s stem
- Sequence of cascaded rewrite rules, e.g.
  - IZE → ε (e.g. unionize → union)
  - CY → T (e.g. frequency → frequent)
  - ING → ε, if stem contains vowel (motoring → motor)
- Can be implemented as a lexicon-free FST (many implementations available on the web)
- http://text-processing.com/demo/stem/
User₁: Men are all alike.
ELIZA₁: IN WHAT WAY
User₂: They’re always bugging us about something or other.
ELIZA₂: CAN YOU THINK OF A SPECIFIC EXAMPLE
User₃: Well, my boyfriend made me come here.
ELIZA₃: YOUR BOYFRIEND MADE YOU COME HERE
User₄: He says I’m depressed much of the time.
ELIZA₄: I AM SORRY TO HEAR YOU ARE DEPRESSED
Eliza FST

```
s/.\* I’M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.\* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.\* all .*/IN WHAT WAY/
s/.\* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```
### RelNoun: Nominal Open IE

<table>
<thead>
<tr>
<th>Constructions</th>
<th>Phrase</th>
<th>Extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb1</td>
<td>Francis Collins is the director of NIH</td>
<td>(Francis Collins; is the director of; NIH)</td>
</tr>
<tr>
<td>Verb2</td>
<td>the director of NIH is Francis Collins</td>
<td>(Francis Collins; is the director of; NIH)</td>
</tr>
<tr>
<td>Appositive1</td>
<td>Francis Collins, the director of NIH</td>
<td>(Francis Collins; is the director of; NIH)</td>
</tr>
<tr>
<td>Appositive2</td>
<td>the director of NIH, Francis Collins</td>
<td>(Francis Collins; is the director of; NIH)</td>
</tr>
<tr>
<td>Appositive3</td>
<td>Francis Collins, the NIH director</td>
<td>(Francis Collins; is the director [of]; NIH)</td>
</tr>
<tr>
<td>AppositiveTitle</td>
<td>Francis Collins, the director,</td>
<td>(Francis Collins; is the director)</td>
</tr>
<tr>
<td><strong>CompoundNoun</strong></td>
<td>NIH director Francis Collins</td>
<td><strong>(Francis Collins; is director [of]; NIH)</strong></td>
</tr>
<tr>
<td>Possessive</td>
<td>NIH’s director Francis Collins</td>
<td>(Francis Collins; is director [of]; NIH)</td>
</tr>
<tr>
<td>PossessiveAppositive</td>
<td>NIH’s director, Francis Collins</td>
<td>(Francis Collins; is director [of]; NIH)</td>
</tr>
<tr>
<td>AppositivePossessive</td>
<td>Francis Collins, NIH’s director</td>
<td>(Francis Collins; is director [of]; NIH)</td>
</tr>
<tr>
<td>PossessiveVerb</td>
<td>NIH’s director is Francis Collins</td>
<td>(Francis Collins; is director [of]; NIH)</td>
</tr>
<tr>
<td>VerbPossessive</td>
<td>Francis Collins is NIH’s director</td>
<td>(Francis Collins; is director [of]; NIH)</td>
</tr>
</tbody>
</table>
Compound Noun Extraction Baseline

- NIH Director Francis Collins

(Francis Collins, is the Director of, NIH)

- Challenges
  - New York Banker Association
  - German Chancellor Angela Merkel
  - Prime Minister Modi
  - GM Vice Chairman Bob Lutz
Rule-Based System

- Classifies and filters orgs

- List of demonyms
  - appropriate location conversion

- Bootstrap a list of relational noun prefixes
  - vice, ex, health, ...
Summing Up

- Regular expressions and FSAs can represent subsets of natural language as well as regular languages
  - Both representations may be difficult for humans to use for any real subset of a language
  - But quick, powerful and easy to use for small problems

- Finite state transducers and rules are common ways to incorporate linguistic ideas in NLP for small applications

- Particularly useful for no data setting