Regular Languages
Lecture #2

Computational Linguistics
CMPSCI 591N, Spring 2006

University of Massachusetts Amherst

Andrew McCallum
A Little About Yourselves

• Have you programmed before?
  – Almost none at all.
  – Not much.
  – I work for a software company.
  – Fortran, C, C++, C#, Lisp, Perl, Python, Java,...
  – Only Basic on my Tandy 286!
A Little About Yourselves

• Hobbies?
  – Fencing!
  – Hiking, Singing, Cooking, Poker, ...
  – Working on machines, like cars, motorcycles, airplanes.
  – Drinking, Smoking.
  – Fencing.
  – Watching movies, especially awesomely bad ones.
A Little About Yourselves

• Favorite authors:
  – Kurt Vonnegut, George Orwell, Noam Chomsky
  – Asimov, Tolkein, Pinger,
  – I avoid reading, sorry.
  – Tolkein (x6), CS Lewis, etc.
  – Stroustrup
  – Arthur C. Clark
  – Hemmingway, x2
  – Salman Rushdie
  – Obscure foreign names like Savyon Librecht.
  – Karel Capek, Milan Kundera, Bulgahov.
A Little About Yourselves

• Why are you in the class?
  – Practical skills to help in my linguistic research: accessing data, building grammars...
  – Interested in how probabilistic methods can be integrated with algebraic grammars.
  – Possibilities of a computer that can make sense of language are very exciting!
  – I want to expand my knowledge of AI.
  – I want to focus my career in CL, especially translation.
  – Want to simulate “the mind’s big bang”.
  – I think this will help me get a job!
Today’s Main Points

• Examples of computation helping in Linguistic goals

• What are regular languages, finite state automata and regular expressions?

• Writing regular expressions (in Python)
• Examples on several large natural language corpora

• Finite-state transducers, and morphology

• Homework assignment #1
Some brief history: 1950s

- Early CL on machines less powerful than pocket calculators.
- Foundational work on automata, formal languages, probabilities and information theory.
- First speech systems (Davis et al, Bell Labs).
- MT heavily funded by military, but basically just word substitution programs.
- Little understanding of natural language syntax, semantics, pragmatics.
Some brief history: 1960s

• Alvey report (1966) ends funding for MT in America - the lack of real results realized
• ELIZA (MIT): Fraudulent NLP in a simple pattern matcher psychotherapist
  – It’s true, I am unhappy.
  – *Do you think coming here will make you not to be unhappy?*
  – I need some help; that much is certain.
  – *What would it mean to you if you got some help?*
  – Perhaps I could earn to get along with my mother.
  – *Tell me more about your family.*
• Early corpora: Brown Corpus (Kudera and Francis)
Some brief history: 1970s

- Could interpret questions, statements commands.
  - Which cube is sitting on the table?
  - *The large green one which supports the red pyramid.*
  - Is there a large block behind the pyramid?
  - *Yes, three of them. A large red one, a large green cube, and the blue one.*
  - Put a small one onto the green cube with supports a pyramid.
  - *OK.*
Some brief history: 1980s

- Procedural --> Declarative (including logic programming)
- Separation of processing (parser) from description of linguistic knowledge.
- Representations of meaning: procedural semantics (SHRDLU), semantic nets (Schank), logic (perceived as answer; finally applicable to real languages (Montague))
- Perceived need for KR (Lenat and Cyc)
- Working MT in limited domains (METEO)
Some brief history: 1990s

- Resurgence of finite-state methods for NLP: in practice they are incredibly effective.
- Speech recognition becomes widely usable.
- Large amounts of digital text become widely available and reorient the field. The Web.
- Resurgence of probabilistic / statistical methods, led by a few centers, especially IBM (speech, parsing, Candide MT system), often replacing logic for reasoning.
- Recognition of *ambiguity* as key problem.
- Emphasis on machine learning methods.
Some brief history: 2000s

• A bit early to tell! But maybe:
  – Continued surge in probability, Bayesian methods of evidence combination, and joint inference.
  – Emphasis on meaning and knowledge representation.
  – Emphasis on discourse and dialog.
  – Strong integration of techniques, and levels: bringing together statistical NLP and sophisticated linguistic representations.
  – Increased emphasis on unsupervised learning.
Examples of Computation Helping Linguistics

• Kevin Knight
  “A Computational Approach to Deciphering Unknown Scripts”
  Mayan Writing
  Pronunciation model, by Expectation Maximization
  (which we will study in about 5 weeks)

Figure 1: The Phaistos Disk (c. 1700BC). The disk is six inches wide, double-sided, and is the earliest known document printed with a form of movable type.
Examples of Computation Helping Linguistics

Other examples coming later:

• Learning Lexical Semantics
  – Augmenting WordNet by mining the Web.

• Automatically discovering English versus Japanese word order by grammar induction.

• Neural Network learners go through the same periods mistakes on irregular verbs as children do.

• ...and others.
Noun phrase parsing...?
Ed Hovy’s thing?
Noam Chomsky
1928 -

Chomsky Hierarchy
Generative Grammar
Liberatarian-Socialist

The most cited person alive.
A Language

Some sentences in the language
• The man took the book.  From [Chomsky, 1956], his first context-free parse tree.
• The purple giraffe hopped through the clouds.
• This sentence is false.

Some sentences not in the language
• *The girl, the sidewalk, the chalk, drew.
• *Backwards is sentence this.
• *IoDvaD tlhIngan Hol ghojmoH be.
Compact description of a language

• Start with some “non-terminal” symbol, $S$.

• Expand that symbol, using some substitution rules.

• ...keep applying rules until all non-terminals are expanded to terminals.

• The string of terminals is in the sentence.
C%u00F3nsky Hierarchy

• Type 0 languages (Turing-equivalent)
  Rewrite rules $a \rightarrow b$
  where $a$, $b$ are any string of terminals and non-terminals

• Context-sensitive languages
  Rewrite rules $aXb \rightarrow acb$
  where $X$ is non-terminal and $a$, $b$ as above

• Context-free languages
  Rewrite rules $X \rightarrow a$
  where $X$, $a$, $b$ as above

• Regular languages
  Rewrite rules $X \rightarrow aY$
  where $X$, $Y$ are non-terminals and $a$ is a string of terminals

More detail on all this again later.

Linguistic example:

- ATNs
- TAGs
- PSGs
- FSAs

Andrew McCallum, UMass Amherst,
including material from Chris Manning and Jason Eisner
Regular language example

- Non-terminals: S, X, Y, Z
- Terminals: m, o
- Rules:
  - S → mX
  - X → oY
  - Y → o
  - Y →
- Start symbol: S

An expansion:
- S
- mX
- moY
- mooY
- mooo
Example: Sheep Language

Strings in and out of the example Regular Language:

- In the language:
  “ba!”, “baa!”, “baaaaa!”
- Not in the language:
- “ba”, “b!”, “ab!”, “bbaaa!”, “alibaba!”

Finite-state Automata

Regular Expression

\[ baa^* \]
Recognizer

- A recognizer for a language is a program that takes as input a string $W$ and answers “yes” if $W$ is a sentence in the language, and answers “no” otherwise.

- We can think of this as a machine that emits only two possible responses it input.
Regular Languages: related concepts

Regular Languages
the accepted strings

Finite-state Automata
machinery for accepting

Regular Expressions
a way to type the automata
Finite State Automata, more formally

- A finite state automata is a 5-tuple: \((Q, \Sigma, q_0, F, \delta(q,i))\)
  - \(Q\): finite set of \(N\) states, \(q_0, q_1, q_2, \ldots, q_N\) (non-terminals)
  - \(\Sigma\): finite set of (terminals)
  - \(\delta(q,i)\): transition function, given state and input, returns next state (production rules)
  - \(q_0\): the start state
  - \(F\): the set of final states

The FSA

State marker

Input tape

We will later return to a probabilistic version of this with Hidden Markov Models!
## Transition Table, $\delta$

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$b$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
## Regular Expressions

**The “foundational” operations**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>abc</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>(a</td>
<td>bb)d</td>
</tr>
<tr>
<td>a*</td>
<td>ε a aa aaa ...</td>
</tr>
<tr>
<td>c (a</td>
<td>bb)*</td>
</tr>
</tbody>
</table>

Regular expressions / Finite-state automata are “closed under these operations”
Stephen Kleene, 1909 - 1994

Attended Amherst College!

Best known for founding the branch of mathematical logic known as recursion theory, together with Alonzo Church, Kurt Gödel, Alan Turing and others; and for inventing regular expressions.

“Kleeneliness is next to Gödeliness.”
Practical Applications of RegEx’s

- Web search
- Word processing, find, substitute
- Validate fields in a database (dates, email addr, URLs)
- Searching corpus for linguistic patterns
  - and gathering stats...

- Finite state machines extensively used for
  - acoustic modeling in speech recognition
  - information extraction (e.g. people & company names)
  - morphology
  - ...

Andrew McCallum, UMass Amherst,
including material from Chris Manning and Jason Eisner
Two types of characters in REs

• **Literal**
  – Every normal text character is an RE, and denotes itself.

• **Meta-characters**
  – Special characters that allow you to combine REs in various ways

  – Example:
    - \( a \) denotes \( a \)
    - \( a^* \) denotes \( \varepsilon \) or \( a \) or \( aa \) or \( aaa \) or ...
## Basic Regular Expressions

<table>
<thead>
<tr>
<th>Character Concat</th>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>went</td>
<td>went</td>
<td>went</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alternatives</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(go</td>
<td>went)</td>
<td>go</td>
</tr>
<tr>
<td>[aeiou]</td>
<td>a</td>
<td>o</td>
</tr>
<tr>
<td>[^aeiou]</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>.</td>
<td>a</td>
<td>z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loops &amp; skips</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a*</td>
<td>ε</td>
<td>a</td>
</tr>
<tr>
<td>a+</td>
<td>a</td>
<td>aa</td>
</tr>
<tr>
<td>colou?r</td>
<td>color</td>
<td>colour</td>
</tr>
</tbody>
</table>
More Fancy Regular Expressions

- **Special characters**
  - `\t` tab
  - `\v` vertical tab
  - `\n` newline
  - `\r` carriage return

- **Aliases (shorthand)**
  - `\d` digits
    - `\D` non-digits
  - `\w` alphabetic
    - `\W` non-alphabetic
  - `\s` whitespace
    - `\S` non-whitespace
  - `\w` alphabetic
    - `\W` non-alphabetic

- **Examples**
  - `\d+ dollars` 3 dollars, 50 dollars, 982 dollars
  - `\w*oo\w*` food, boo, oodles

- **Escape character**
  - `\` is the general escape character; e.g. `\.` is not a wildcard, but matches a period .
  - if you want to use `\` in a string it has to be escaped `\\`
Yet More Fancy Regular Expressions

- **Anchors.** AKA, “zero width characters”.
- They match positions in the text.
  - `^` beginning of line
  - `$` end of line
  - `\b` word boundary, i.e. location with `\w` on one side but not on the other.
  - `\B` ???
- **Examples:**
  - `\bthe\b` the together

- **Counters** `{1}`, `{1,2}`, `{3,}`
Even More Fancy Regular Expressions

• Grouping
  – a (good|bad) movie
  – He said it (again and )*again.

• Parens also indicate Registers (saved contents)
  – b(\w+)b\1
    matches boohoo and baha, but not boohaa
    The digit after the \ indicates which of multiple paren groups, as ordered by when then were opened.

• Grouping without the cost of register saving
  – He went (?:this|that) way.
Extra Fancy Regular Expressions

• Non-consuming tests
  – (?=...) - Positive lookAHEAD
  – (?!...) - Negative lookAHEAD
  – (?<=...) - Positive lookBEHIND
  – (?<!...) - Negative lookBEHIND

• Example
  – (?<=[Hh]e) \w+ed (?=\w+ly)
Yowza! Regular Expressions

- Multi-line, with comments, etc.

```
FROM = re.compile("""
    ^From: # Anchor to start of line
    \s*  # maybe some spaces
    (?P<user>\w+) # ‘user’: group of word characters
    @
    (?P<domain>\S+) # the ‘domain’:
    \s+  # some non-space characters
    \s   # finally, a space character
""",re.VERBOSE)
```
Oral Quiz:
Describe the strings these will accept

• \texttt{ab+a}

• \texttt{(ab)*}

• \texttt{([^[aeiou][aeiou])\1}

• \texttt{\bdis\w+\b...}
Hands-on!
With little strings

- re_show function from NLTK

```python
import re
def re_show(regexp, string):
    print re.compile(regexp, re.M).sub("\g<0>", string.rstrip())
```
Hands-on!
Investigating corpora

- `regexs.py`
- `regexcount.py`

- Corpora
  - 200k+ words of Wall Street Journal articles
  - 3 books by Jane Austin
  - Several plays by William Shakespeare
  - Several books by Mark Twain
  - and various other text from Walt Whitman, & others...

I will make these available on the course Web site
**ELIZA (Weizenbaum 1966)**

<table>
<thead>
<tr>
<th>User:</th>
<th>Men are all alike.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELIZA:</td>
<td>IN WHAT WAY?</td>
</tr>
<tr>
<td>User:</td>
<td>They’re always bugging us about something or other.</td>
</tr>
<tr>
<td>ELIZA:</td>
<td>CAN YOU THINK OF A SPECIFIC EXAMPLE?</td>
</tr>
<tr>
<td>User:</td>
<td>Well, my boyfriend made me come here.</td>
</tr>
<tr>
<td>ELIZA:</td>
<td>YOUR BOYFRIEND MADE YOU COME HERE.</td>
</tr>
<tr>
<td>User:</td>
<td>He say’s I’m depressed much of the time.</td>
</tr>
<tr>
<td>ELIZA:</td>
<td>I’M SORRY TO HEAR THAT YOU ARE DEPRESSED.</td>
</tr>
</tbody>
</table>

---

**Implemented with regular expression substitution!**

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR THAT YOU ARE \1/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/
```
Non-deterministic FSAs

More than one outgoing transition on ‘a’

State | Input |  
-------|-------|
| 0 | b 1 | a 0 | ! 0 |
| 1 | ∅ | 1,2 1 | ∅ |
| 2 | ∅ | 0 0 | 3 0 |
| 3 | ∅ | 0 0 | 0 0 |

Transition \textit{relation}, rather than transition function.
Non-deterministic finite-state automata as “Recognizers”

• The problem:

When processing a string, we might follow the wrong transition, and reject the string when we should have accepted it!

One solution: turn the NFA into a DFA... (See CMPSCI 250)

• Ubiquitous problem in this course: How to efficiently search through various possible “paths” (parses) to find one that works / the most likely one, etc.

How do humans do this?!
Solutions

• **Look-ahead**
  – Peek ahead to help decide which path to take.

• **Parallelism**
  – At each choice, take every path in parallel.

• **Backup**
  – At each choice point, mark the input / state
  – If we fail, go back and try another path
    Need a *stack* (or *queue*) of markers
  – Marker = “Machine state”
  – Collection of current state & markers = “Search state”
  – Depth-first search (or Breadth-first search).

  “Smart” heuristic search, “A*”. See CMPSCI 383
  (Artificial Intelligence)
RE / FSA equivalence proof

• How would you do it?
Morphology
The study of the sub-word units of meaning.

Making a word plural:
- If word is regular, add s
- If word ends in y, change y to i, and add s
- If word ends in x, add -es

Examples:
- dog becomes dogs
- baby becomes babies
- fox becomes foxes

Recognizing that foxes breaks down into morphemes fox and -es called Morphological Parsing

Parsing = taking an input and producing some sort of structure for it.

Andrew McCallum, UMass Amherst,
including material from Chris Manning and Jason Eisner
Morphology, briefly

• **morpheme**: minimal meaning-bearing unit
  – **stem**: “main” morpheme of a word, e.g. *fox*
  – **affixes**: add “additional” meanings, e.g. *+es*
    includes *prefixes, suffixes, infixes, circumfixes*,
    e.g. *un-, -ly, ... ...
  – concatenative morphology, non-concatenative

• **inflection**: stem+morpheme in the same class as stem.
  – e.g. nouns plural *+s*, possessive *+'s*

• **derivation**: stem+morpheme in different class...
  – e.g. *+ly* makes and adverb from an adjective
Morphological Parsing with Finite State Transducers

• We want a system that given *foxes* will output a parse: *fox+es* or *fox +PL*

• FSAs will take input, but not produce output (other than “accept”/”reject”)

• Solution: **Finite State Transducers** (FST):
  – A FST is a two-tape automaton that recognizes or generates **pairs** of strings.
FSTs can be used to transform a word surface form into morphemes (or vice-versa!) 

An entire lexicon can be encoded as a FST.
# FST transition table

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>h:h</th>
<th>a:a</th>
<th>p:p</th>
<th>y:y</th>
<th>i:y</th>
<th>ε:+</th>
<th>e:e</th>
<th>r:r</th>
<th>s:s</th>
<th>t:t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>1</td>
<td>Ø</td>
<td>2</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>2</td>
<td>Ø</td>
<td>Ø</td>
<td>3</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>3</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>4</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>4</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>5</td>
<td>6</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>5:</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>6</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>7</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>7</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>8</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>8</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>10</td>
<td>9</td>
<td>Ø</td>
</tr>
<tr>
<td>9:</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
</tbody>
</table>
Fragment of a lexicon in a FST
Further Closure Properties of FSAs

Regular languages are also closed under the following operations

• **Reversal**: If $L_1$ is regular, so is the language consisting of the set of all reversals of strings in $L_1$.

• **Intersection**: if $L_1$ and $L_2$ are regular languages, so is the language consisting of all strings that are in both $L_1$ and $L_2$.

• **Difference**: If $L_1$ and $L_2$ are regular languages, so is the language consisting of all strings in $L_1$ that are not in $L_2$.

• **Complementation**: If $L_1$ is a regular language, so is the set of all possible strings that are not in $L_1$. 
Announcement: Undergraduate CMPSSCI Meeting

• “First Friday”
  – Curriculum Information
  – Spring Events
  – Jobs/Co-ops/Research positions in and out of the Department
  – Library Carrels
  – And More!

• Friday, February 3, 2005
  3:30 - 5:00 PM
  CMPS 150/151 (Computer Science Building)
  Refreshments will be served.
Next class (Tuesday Feb 7)

• Learning Python
  – Variables, operators, conditionals, iteration, etc.
  – functions, classes, modules
  – Gather statistics from Python-ized Penn Treebank.
  – Calculate statistics from 200k words of WSJ
  – Implement a phrase structure grammar, and generate sentences from it.

• Install Python, and bring your laptop with you!
First Homework, assigned today!

- Essentially:
  - Write some regular expressions
  - Run them on some corpora
  - Write ~1 page about your experience and findings
  - Extra credit for creativity and interesting application!

- Feel free to come do it in office hours!

- Due next Thursday, one week from today.
  (Don’t wait until Wednesday to install Python!

- Recommended schedule:
  - Idea by Saturday
  - Coded/tested by Monday
  - Write-up by Wednesday
Office Hours, CS Building, Rm 264

- Friday, 2-4pm
- Monday, 10:30am-1pm
- Tuesday, 10:30am-1pm
- Wednesday, 10:30am-1pm
- Thursday, 10:30am-12:30pm

If you can’t make these times, let me know.
Aside: Grammar Induction

• Also called “Grammatical Inference”
• “Learning” finite-state automata from many examples of strings in (and out of) the language.
• Learning FSA and CFG structure from data!
Thank you!