CMSC 723 / LING 723: Computational Linguistics I

September 19, 2007: Dorr
Reg. Expressions, FS Automata (J&M 2)
Morphology (J&M 3)
Prof. Bonnie J. Dorr
Co-Instructor: Nitin Madnani
TA: Hamid Shahri

Regular Expressions and Finite State Automata

- REs: Language for specifying text strings
- Search for document containing a string
  - Searching for “woodchuck”
    - How much wood would a woodchuck chuck if a woodchuck would chuck wood?
  - Searching for “woodchucks” with an optional final “s”
    - Finite-state automata (FSA)
      - (singular: automaton)
### Regular Expressions

**Basic regular expression patterns**

**Disjunctions** \([wW]oodchuck\)

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>[/wW]oodchuck/</td>
<td>Woodchuck or woodchuck</td>
<td>“Woodchuck”</td>
</tr>
<tr>
<td>[/abc]/</td>
<td>‘a’, ‘b’, or ‘c’</td>
<td>“In uomini, in soldati”</td>
</tr>
<tr>
<td>[/1234567890]/</td>
<td>any digit</td>
<td>“plenty of 7 to 5”</td>
</tr>
</tbody>
</table>

- **Perl**: Enclose regular expressions in slashes, e.g., `/[abc]/`
- **Python**: Use ‘r’ (for “raw”) with single quotes, e.g., `r’[abc]’`

### Regular Expressions

**Ranges** \([A–Z]\)

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[A–Z]/</td>
<td>an uppercase letter</td>
<td>“we should call it ‘Drenched Blossoms’”</td>
</tr>
<tr>
<td>/[a–z]/</td>
<td>a lowercase 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>

**Negations** \([^Ss]\)

<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 uppercase letter</td>
<td>“Oyfn pripetchik”</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^n b now”</td>
</tr>
</tbody>
</table>
Regular Expressions

- Optional characters ?, *, and +
  - ? (0 or 1)
    - colou?r → color or colour
  - * (0 or more)
    - oo*th! → oh! or ooh! or ooooh!
  - + (1 or more)
    - o+h! → oh! or ooh! or ooooh!

- Wild cards .
  - beg.n → begin or began or begun

Regular Expressions

- Anchors ^ and $
  - ^[A-Z] → “Ramallah, Palestine”
  - ^[^A-Z] → “¿verdad?” “really?”
  - \.$ → “It is over.”
  - .$ → ?

- Boundaries \b and \B
  - \bon\b → “on my way” “Monday”
  - \Bon\b → “automaton”

- Disjunction |
  - yours|mine → “it is either yours or mine”
**Disjunction, Grouping, Precedence**

✱ Column 1  Column 2  Column 3 …
How do we express this?

**Column [0-9]+ * **
(Column [0-9]+ *)*

✱ Precedence
– Parenthesis ()
– Counters *  +  ?  {}
– Sequences and anchors the ^my end$
– Disjunction |

✱ REs are greedy!

**Python Commands (See Chapter 7 of Dive Into Python)**

```python
import re
lines = ["The cat sat",
        "What is the number?",
        "Can you tell me the time?"]
for line in lines:
    if re.match(r’the’, line):
        print “MATCH:”, line
```

Important: Difference between match() and search()!
For more info:
[http://www.diveintopython.org/regular_expressions/index.html](http://www.diveintopython.org/regular_expressions/index.html)
Writing correct expressions

Exercise: Write a Python regular expression to match the English article “the”:

```
\'the\'
\'[tT]he\'
\'\b[tT]he\b\'
\'[^a-zA-Z][tT]he[^a-zA-Z]\'
\'(^[^a-zA-Z])[tT]he[^a-zA-Z]\'
```

Aliases for Common Sets of Characters

<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, 0123, Blue, moon</td>
</tr>
<tr>
<td>\D</td>
<td>[^0-9]</td>
<td>any non-digit</td>
<td></td>
</tr>
<tr>
<td>\w</td>
<td>[a-zA-Z0-9_]</td>
<td>any alphanumeric or underscore</td>
<td>Daiyu, party</td>
</tr>
<tr>
<td>\W</td>
<td>[^\w]</td>
<td>a non-alphanumeric</td>
<td></td>
</tr>
<tr>
<td>\s</td>
<td>[\s\t\n\f]</td>
<td>whitespace (space, tab)</td>
<td></td>
</tr>
<tr>
<td>\S</td>
<td>[^\s]</td>
<td>Non-whitespace</td>
<td></td>
</tr>
</tbody>
</table>
More Advanced Operators

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>zero or more occurrences of the previous char or expression</td>
<td>“K<em>A</em>P<em>L</em>A*N”</td>
</tr>
<tr>
<td>+</td>
<td>one or more occurrences of the previous char or expression</td>
<td>“Dr. Livingston, I presume”</td>
</tr>
<tr>
<td>?</td>
<td>exactly zero or one occurrence of the previous char or expression</td>
<td>“Would you light my candle?”</td>
</tr>
<tr>
<td>{n}</td>
<td>n occurrences of the previous char or expression</td>
<td></td>
</tr>
<tr>
<td>{n, m}</td>
<td>from n to m occurrences of the previous char or expression</td>
<td></td>
</tr>
<tr>
<td>{n,}</td>
<td>at least n occurrences of the previous char or expression</td>
<td></td>
</tr>
</tbody>
</table>

Substitutions and Memory

- **Substitutions**
  
  ```python
  line = 'this colour is the right colour'
  re.sub(r'colour', r'color', line, 1)
  re.sub(r'colour', r'color', line)
  re.sub(r'\([0-9]+\)', r'<\1>', '1 and 2')
  re.sub(r'(odd|weird)', r'\1er', 'they were getting odd and weird')
  ```

  Substitute as many times as possible!

- **Memory (\1 used to refer back to matches)**
  
  ```python
  re.sub(r'(\([0-9]+\))', r'\1er', '1 and 2')
  ```

  ```python
  re.sub(r'(odd|weird)', r'\1er', 'they were getting odd and weird')
  ```
Eliza [Weizenbaum, 1966]

User: Men are all alike
ELIZA: IN WHAT WAY
User: They’re always bugging us about something or other
ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE?
User: Well, my boyfriend made me come here
ELIZA: YOUR BOYFRIEND MADE YOU COME HERE
User: He says I’m depressed much of the time
ELIZA: I AM SORRY TO HEAR THAT YOU ARE DEPRESSED

Eliza-style regular expressions

Step 1: replace first person references with second person references
re.sub(r"\bI('m| am)\b",r'YOU ARE', response)
re.sub(r'\bmy\b',r'YOUR', response)
re.sub(r'\bmine\b',r'YOURS', response)

Step 2: use additional regular expressions to generate replies
.* I am (depressed|sad) ➔ I AM SORRY TO HEAR YOU ARE
.* I am (depressed|sad) ➔ WHY DO YOU THINK YOU ARE
.* all ➔ IN WHAT WAY
.* always ➔ CAN YOU THINK OF A SPECIFIC EXAMPLE

Step 3: use scores to rank possible transformations
Finite-state Automata

- Finite-state automata (FSA)
- Regular languages
- Regular expressions

Finite-state Automata (Machines)

- State transition
- Final state
- Regular expressions
- Finite automata
- Regular languages
Input Tape

```
q_0

0 b 1 a 2 a 3 a 4 !
```

REJECT

Input Tape

```
q_0 q_1 q_2 q_3 q_3 q_4

0 b 1 a 2 a 3 a 4 !
```

ACCEPT
**Finite-state Automata**

- $Q$: a finite set of $N$ states
  - $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma$: a finite input alphabet of symbols
  - $\Sigma = \{a, b, !\}$
- $q_0$: the start state
- $F$: the set of final states
  - $F = \{q_4\}$
- $\delta(q,i)$: transition function
  - Given state $q$ and input symbol $i$, return new state $q'$
  - $\delta(q_3,!) \rightarrow q_4$

**State-transition Tables**

<table>
<thead>
<tr>
<th>State</th>
<th>b</th>
<th>a</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4:</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**D-RECOGNIZE**

function D-RECOGNIZE (tape, machine) returns accept or reject

  index $\leftarrow$ Beginning of tape
  current-state $\leftarrow$ 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
    end
  elsif transition-table [current-state, tape[index]] is empty then
    return reject
  else
    current-state $\leftarrow$ transition-table [current-state, tape[index]]
    index $\leftarrow$ index + 1
  end

end

---

**Adding a failing state**

![Diagram of a failing state transition graph]

- States: $q_0, q_1, q_2, q_3, q_4, q_f$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
  - $q_3 \xrightarrow{b} q_f$
  - $q_f \xrightarrow{a} q_4$
  - $q_4 \xrightarrow{b} q_f$
  - $q_f$ is a failing state
Adding an “all else” arc

Languages and Automata

- Can use FSA as a generator as well as a recognizer
- Formal language $L$: defined by machine $M$ that both generates and recognizes all and only the strings of that language.
  - $L(M) = \{\text{baa!}, \text{baaa!}, \text{baaaa!}, \ldots\}$
- Regular languages vs. non-regular languages
Languages and Automata

- **Deterministic vs. Non-deterministic FSAs**

- **Epsilon (ε) transitions**

Using NFSAs to accept strings

- **Backup**: add markers at choice points, then possibly revisit unexplored arcs at marked choice point.
- **Look-ahead**: look ahead in input
- **Parallelism**: look at alternatives in parallel
Using NFSAs

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

More about FSAs

- Transducers
- Equivalence of DFSAs and NFSAs
- Recognition as search: depth-first, breadth-search
Recognition using NFSAs

Function \textsc{ND-RECOGNIZE}(tape, machine) returns accept or reject

\begin{verbatim}
agenda \leftarrow \{(initial state of machine, beginning of tape)\}
current-search-state \leftarrow \textsc{NEXT}(agenda)

loop:
  if \textsc{ACCEPT-STATE}(current-search-state)
    return accept
  else
    if agenda is empty
      return reject
    else
      current-search-state \leftarrow \textsc{NEXT}(agenda)

end
\end{verbatim}

Function \textsc{GENERATE-NEW-STATE}(current-state) returns a set of search states:

\begin{verbatim}
current-node \leftarrow the node the current search-state is in
index \leftarrow the point on the tape the current search-state is looking at
return a list of search states from transition table as follows:
(transitions-table[current-node], index)
end
\end{verbatim}

Function \textsc{ACCEPT-STATE}(search-state) returns true or false:

\begin{verbatim}
if index is at the end of the tape and current-node is an accept state of machine
  return true
else
  return false
\end{verbatim}

NFSA Recognition of “baaa!”

Diagram of NFSA recognizing the string “baaa!”
Breadth-first Recognition of “baaa!”

Regular languages

- Regular languages are characterized by FSAs
- For every NFSA, there is an equivalent DFSA.
- Regular languages are closed under concatenation, Kleene closure, union.
Concatenation

Kleene Closure
Digression: Composition (for FSTs)

In Homework 2: You will work with FST composition

- Similar to concatenation of FSA’s, but two internal conversions compiled into more efficient single transition a:c
- FST’s have input/output pair on arcs instead of a single read-only input
- Compose: Performs transitive closure to get single arc with input symbol of first automaton and output symbol of last automaton
**Composition Example**

```python
from soundexutils import compose
output = compose(S, f1, f2, f3)
```

The above function call computes \( f_1 \circ f_2 \circ f_3 \). Obviously, it will raise an error if one or more of the input transducers produce no output.

![Diagram](diagram.png)

**Morphology**

- **Definitions and Problems**
  - What is Morphology?
  - Topology of Morphologies

- **Approaches to Computational Morphology**
  - Lexicons and Rules
  - Computational Morphology Approaches
**Morphology**

- The study of the way words are built up from smaller meaning units called *Morphemes*.
- Abstract versus Realized
  - HOP + PAST $\rightarrow$ hop + ed $\rightarrow$ hopped $\rightarrow$ /hapt/

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Lexeme/Inflected Lexeme</th>
<th>Grammars</th>
<th>sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morphology</td>
<td>Morpheme/Allomorph</td>
<td>Morphotactics</td>
<td>words</td>
</tr>
<tr>
<td>Phonology</td>
<td>Phoneme/Allophone</td>
<td>Phonotactics</td>
<td>letters</td>
</tr>
</tbody>
</table>

**Phonology and Morphology**

- Phonology vs. Orthography
- Historical spelling
  - night, nite
  - attention, mission, fish
- Script Limitations
  - Spoken English has 14 vowels
    - heed hid hayed head had hoed hood who’d hide how’d taught
    - Tut toy enough
  - English Alphabet has 5
    - Use vowel combinations: far fair fare
    - Consonantal doubling (hopping vs. hoping)
Syntax and Morphology

- Phrase-level agreement
  - Subject-Verb
    - John studies hard (STUDY+3SG)
  - Noun-Adjective
    - Las vacas hermosas

- Sub-word phrasal structures
  - חיובם
  - ש+ב+מסuder+ו+ל
  - That+in+book+PL+Poss:1PL
  - Which are in our books

Topology of Morphologies

- Concatenative vs. Templatic
- Derivational vs. Inflectional
- Regular vs. Irregular
**Concatenative Morphology**

- **Morpheme** + **Morpheme** + **Morpheme** + ...
- Stems: also called lemma, base form, root, lexeme
  - hope + ing → hoping   hop → hopping
- Affixes
  - Prefixes: Antidisestablishmentarianism
  - Suffixes: Antidisestablishmentarianism
  - Infixes: hingi (borrow) – humingi (borrower) in Tagalog
  - Circumfixes: sagen (say) – gesagt (said) in German

**Agglutinative Languages**

- u y g a r l a š tııramadı
- u y g a r + l a š tııramadı
- Behaving as if you are among those whom we could not cause to become civilized

**Templatic Morphology**

- **Roots and Patterns**

  ك ت ب
  مكتوب  written
  ك ت ب
  ktuu  written
**Templatic Morphology: Root Meaning**

- **KTB:** writing “stuff”

**Derivational vs. Inflectional**

- **Word Classes**
  - Parts of speech: noun, verb, adjectives, etc.
  - Word class dictates how a word combines with morphemes to form new words
Derivational morphology

- Nominalization: computerization, appointee, killer, fuzziness
- Formation of adjectives: computational, clueless, embraceable
- CatVar: Categorial Variation Database http://clipdemos.umiacs.umd.edu/catvar/

Inflectional morphology

- Adds: Tense, number, person, mood, aspect
- Word class doesn’t change
- Word serves new grammatical role
- Five verb forms in English
- Other languages have (lots more)
**Nouns and Verbs (in English)**

- Nouns have simple inflectional morphology
  - cat
  - cat+s, cat+s
- Verbs have more complex morphology

**Regulars and Irregulars**

- Nouns
  - Cat/Cats
  - Mouse/Mice, Ox, Oxen, Goose, Geese
- Verbs
  - Walk/Walked
  - Go/Went, Fly/Flew
### Regular (English) Verbs

<table>
<thead>
<tr>
<th>Morphological Form Classes</th>
<th>Regularly Inflected Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stem</td>
<td>walk, merge, try, map</td>
</tr>
<tr>
<td>-s form</td>
<td>walks, merges, tries, maps</td>
</tr>
<tr>
<td>-ing form</td>
<td>walking, merging, trying, mapping</td>
</tr>
<tr>
<td>Past form or -ed participle</td>
<td>walked, merged, tried, mapped</td>
</tr>
</tbody>
</table>

### Irregular (English) Verbs

<table>
<thead>
<tr>
<th>Morphological Form Classes</th>
<th>Irregularly Inflected Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stem</td>
<td>eat, catch, cut</td>
</tr>
<tr>
<td>-s form</td>
<td>eats, catches, cuts</td>
</tr>
<tr>
<td>-ing form</td>
<td>eating, catching, cutting</td>
</tr>
<tr>
<td>Past form</td>
<td>ate, caught, cut</td>
</tr>
<tr>
<td>-ed participle</td>
<td>eaten, caught, cut</td>
</tr>
</tbody>
</table>
“To love” in Spanish

<table>
<thead>
<tr>
<th>Present Indicative</th>
<th>Imperfect Indicative</th>
<th>Future</th>
<th>Preterite</th>
<th>Present Subjunctive</th>
<th>Conditional</th>
<th>Imperfect Subjunctive</th>
<th>Future Subjunctive</th>
</tr>
</thead>
<tbody>
<tr>
<td>amo</td>
<td>amas</td>
<td>amaba</td>
<td>amaré</td>
<td>amo</td>
<td>amaría</td>
<td>amaras</td>
<td>amares</td>
</tr>
<tr>
<td></td>
<td>ames</td>
<td>amábamos</td>
<td>amaremos</td>
<td>ame</td>
<td>amarias</td>
<td>amaríamos</td>
<td>amaremos</td>
</tr>
<tr>
<td>ama</td>
<td>amamos</td>
<td>amaba</td>
<td>amaré</td>
<td>ame</td>
<td>amaría</td>
<td>amaríamos</td>
<td>amaríamos</td>
</tr>
<tr>
<td>amás</td>
<td>amad</td>
<td>amabas</td>
<td>amáremos</td>
<td>ames</td>
<td>amarias</td>
<td>amamáríamos</td>
<td>amamárame</td>
</tr>
<tr>
<td>amar</td>
<td>amas</td>
<td>amaras</td>
<td>amarías</td>
<td>ames</td>
<td>amarías</td>
<td>amaríamos</td>
<td>amaríos</td>
</tr>
</tbody>
</table>

For next time ...

- Finish Chapter 3
- Start Chapter 5