**CS 181:**
**Natural Language Processing**

*Lecture 2: FSA’s & Regular Expressions*

**Kim Bruce**

**Pomona College**

**Spring 2008**

Disclaimer: Slide contents borrowed from many sources on web!

---

**History**

- McCulloch & Pitts (1943) neural networks
- Kleene (1956) equiv. of fsa & reg exps
- Mealy, Moore (1955-56) generalized
- Scott-Rabin (1959) ndfa & decisions
- S. Ginsburg (1962) finite transducers

---

**Language**

- Set of "sentences"
- Characterize with recognizer or generator
- \( L(M) = \{ w \mid M \text{ recognizes } w \} \)
- \( L(M) = \{ w \mid M' \text{ generates } w \} \)
- Chomsky hierarchy
  - Type 0 (recursively enumerable)
  - Type 1 (context sensitive)
  - Type 2 (context free)
  - Type 3 (regular)

**Regular Language**

- \( S \rightarrow bX \)
- \( X \rightarrow aY \)
- \( Y \rightarrow aY \)
- \( Y \rightarrow ! \)

Productions must be of form: \( U \rightarrow a \) or \( U \rightarrow aV \) or \( U \rightarrow \epsilon \)

- \( S \rightarrow bX \Rightarrow baY \Rightarrow baaY \Rightarrow baaaY \Rightarrow baaa! \)

---

**Finite-State Automaton & Regular expressions**

- **Finite State Automaton**
  - \( M = (Q, \Sigma, s, F, \delta) \) where
  - \( Q \) is a finite set of states
  - \( \Sigma \) is finite input alphabet
  - \( s \in Q \) is start state
  - \( F \subseteq Q \) is set of final states
  - \( \delta : Q \times \Sigma \rightarrow Q \) is state transition function
  - \( \text{Sheep} = \{ (s0, s1, s2, s3), (b, a, !), s0, \{ s3 \}, \delta_{\text{sheep}} \} \)

---

All 3 models equivalent in power!
FSA Accepts Language

- Define $\delta^*(q_0, a_1 \ldots a_n) = q_{n+1}$ iff $\exists q_2 \ldots q_n$ s.t. for $i = 1..n$, $\delta(q_0, a_i) = q_i+1$.
- String $w \in L(M)$ iff $\delta^*(s, w) \in F$.

Generalized FSA

- Add epsilon-moves -- don’t eat up input
- Can have non-deterministic fsa
- May have 0 or more choices
- Can always find equivalent deterministic fsa w/no epsilon-moves.
- Subset construction!

Regular Expressions Generate Language

- Regular expressions over $\Sigma$:
  - $\emptyset$ and each $a \in \Sigma$
  - If $\alpha$ and $\beta$ are regular, then so is $\alpha | \beta$
  - If $\alpha$ is regular, then so is $\alpha^*$
- Examples: baa*, (b|d)ad

RegExp in Linguistics

<table>
<thead>
<tr>
<th>Description</th>
<th>Pattern</th>
<th>Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char concat</td>
<td>stay</td>
<td>stay</td>
</tr>
<tr>
<td>Alternatives</td>
<td>(go</td>
<td>stay)</td>
</tr>
<tr>
<td>Disjunction</td>
<td>[aeiou][b-d]</td>
<td>a,e,i,o,u &amp; b,c,d</td>
</tr>
<tr>
<td>Disjunction negation</td>
<td>^[aeiou]</td>
<td>b,c,d,f,...,z</td>
</tr>
<tr>
<td>Wildcard char</td>
<td>.</td>
<td>a,b,c,....</td>
</tr>
<tr>
<td>Zero or more</td>
<td>$a^*$</td>
<td>e, a, aa, aaaa, ...</td>
</tr>
<tr>
<td>One or more</td>
<td>$a^+$</td>
<td>a, aa, aaa, ...</td>
</tr>
<tr>
<td>Optional (0 or 1)</td>
<td>color?r</td>
<td>color, colour</td>
</tr>
</tbody>
</table>

More Shorthand

- Special chars and abbreviations:
  - \n, \t, ...
  - \d - digit, \D - non-digit,
  - \w - alphanumeric or "_", \W
  - \s - whitespace, \S
  - " is escape character - "\" diff from ".
- Anchors (location not character)
  - ^ - beginning of line, $ - end of line
  - \b - word boundary, \B - not word boundary

Examples

- "\w*the\w*"
- "\w*\bthe\b\w*" matches only "the"
- "[^a-z]+d$"
- "^[a-z]+d$"
**More Regular Expressions**

- Groupings: if use parentheses, only reports that portion in output:
  - `(\[a-z\]+)([^a-z]+)` - matches expression fat73, but only returns fat.

- Registers can remember substrings
  - `(\[a-z\]+)` matches "haha" (and returns "ha"), but "hah" does not match.

  *Takes you beyond fsa’s!*

**Closure**

- Regular expressions are closed under:
  - intersection, difference, complementation, reversal, concatenation, Kleene * closure, union

- ndfa’s = dfa’s
  - = regular languages
  - = regular expressions

  *We’ll be using regular expressions, knowing that can automatically construct dfa’s to recognize them!*

**Python**

*Wikipedia:*

- Python is a multi-paradigm programming language (primarily functional, object oriented and imperative) which has a fully dynamic type system and uses automatic memory management; it is thus similar to Perl, Ruby, Scheme, and Tcl.

* Borrowed from other languages:
  - list comprehensions and white space as delimiters borrowed from Miranda, predecessor of Haskell

**More Python**

- Python generally interpreted, though some compilers available

- Python type-safe, but not statically (like Scheme)

  *Because dynamically typed, must work harder to make sure no errors.*

- Main advantage is huge libraries.

- Can link with C or C++. 
Data Types

- Lists: `a = ["hello", 17, "there", 4.53]`
  `a[1]`, `a[-1]`, `a[:3]`, `a[:3]`
  `for elt in a:`
- Strings similar: "hello" or 'hello'
- String & list ops: `b = ["hello","17","there"]`
  `str = ' '.join(a)`
  `str.split('e')` returns `['h', 'llo 17 th', 'r', '']`
- List comprehensions:
  - `[x for x in b if x[0] == 'h']`
  - `+`: concatenation

Tuples immutable: ("hello",17)

Dictionaries - address by key: `d = {}`
- `d["bob"] = "123 Norse Way"`
- `d["ann"] = "54 Andover"`
- `print d["bob"]`
- `d.has_key("ann")` returns True
- `d.keys()` returns `["bob", "ann"]`
- `d.items()` returns
  - `["bob", "123 Norse Way"], ("ann", "54 Andover")`

Running Python

- At command line type: python
  - use xserv.cs.pomona.edu not linus
- IDLE: editing and execution environment
- Emacs (aquamacs)
  - create new window in Python mode
  - select start interpreter
  - can compute interactively
  - to execute file, in edit window type C-c C-c
  - put text files in same folder or use complete path

Example

```python
import nltk
count = {}
for word in nltk.corpus.gutenberg.words('shakespeare-macbeth'):
    word = word.lower()
    if word not in count:
        count[word] = 0
    count[word] += 1

print 'Scotland occurs', count['scotland'], 'times.'
```

NLTK & Regular Expressions

- NLTK library provides access to corpora (Brown, Penn Treebank, etc.) and tools
- Python has re module, while nltk provides functions like re_show.
- Start file (module) with
  - `import nltk or from nltk import ...`

Using re and nltk

```python
>>> import nltk, re
>>> sent = "The quick brown fox jumped over the lazy dog"
>>> nltk.re_show('he',sent)
'The quick brown fox jumped over the lazy dog'
>>> re.sub('er','ah',sent)
'The quick brown foxjumped over the lazy dog'
>>> nltk.re_show('(fox|dog|rat)',sent)
The quick brown fox jumped over the lazy dog
```

Using Treebank (Brown) and Tagged Tokens

```python
>>> nltk.re_tree('dog',sent)
The quick brown fox jumped over the lazy dog
```
More RE’s

```python
>>> nltk.re_show('[^aeiou][aeiou]', sent)
T[he] {qu}ick b{ro}wn {fo}x {ju}m{pe}d{ o}{ve}r t{he} {la}zy {do}g
>>> re.findall('[^aeiou][aeiou]', sent)
['he', 'qu', 'ro', 'fo', 'ju', 'pe', 'o', 've', 'he', 'la', 'do']
Parentheses select a subpart to be returned:
>>> re.findall('([^aeiou])([aeiou])', sent)
[('h', 'e'), ('q', 'u'), ('r', 'o'), ('f', 'o'), ('j', 'u'), ('p', 'e'), (' ', 'o'), ('v', 'e'), ('h', 'e'), ('l', 'a'), ('d', 'o')]
```

NLTK & WEB PAGES

```python
>>> page = urlopen('http://www.nytimes.com').read()
>>> text = nltk.clean_html(page)
>>> print text[16:50]
The New York Times - Breaking News
```

Using Brown Corpus

```python
>>> nltk.corpus.gutenberg.items
>>> count = 0
>>> for word in nltk.corpus.gutenberg.words('whitman-leaves'):
...     count += 1
...
>>> print count
154898
```

More Programming

```python
from nltk import defaultdict
sentence = "she sells sea shells by the sea shore"
dict = defaultdict(int)
words = sentence.split()
for word in words:
    word = word.lower()
    dict[word] += 1
for key in sorted(dict.keys):
    print "%s: %d" % (key,dict[key])
```

Using Brown Corpus

```python
>>> nltk.corpus.brown.items
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'n', 'p', 'r')
>>> print nltk.corpus.brown.words('a')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> print nltk.corpus.brown.tagged_sents('a')
[['The', 'AT'], ['Fulton', 'NN-TL'], ['County', 'NN-TL'], ['Grand', 'NN-TL'], ['Jury', 'NN-TL'], ['said', 'VBD'], ...]
```

Any Questions?