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 | M recognizes w }
  - L(M) = { w | M' generates w }
- Chomsky hierarchy *recognizer*
  - Type 0 (recursively enumerable) Tm
  - Type 1 (context sensitive) lba
  - Type 2 (context free) pda
  - Type 3 (regular) fsa

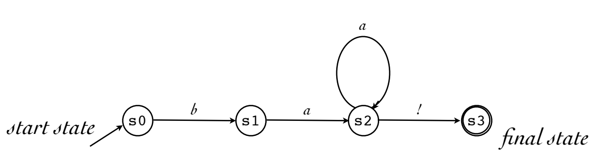
Regular Language

- $S \rightarrow bX$
- $X \rightarrow aY$
- $Y \rightarrow aY$
- $Y \rightarrow !$

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



baa*!

All 3 models equivalent in power!

Finite State Automaton

- M = (Q, Σ, s, F, δ) where
  - Q is a finite set of states
  - Σ is finite input alphabet
  - s ∈ Q is start state
  - F ⊆ Q is set of final states
  - δ: Q × Σ → Q is state transition function
- Sheep = (`{s0,s1,s2,s3}`,`{b,a,!}`,`s0`, `{s3}`, δ_{sheep})

FSA Accepts Language

- Define $\delta^*(q_1,a_1\ldots a_n)=q_{n+1}$ iff $\exists q_2 \ldots q_n$ s.t for i = 1..n, $\delta(q_i,a_i) = q_{i+1}$.
- 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 Σ:
  - $\emptyset$ and each $a \in \Sigma$
  - If α and β are regular, then so is $\alpha \mid \beta$
  - If α is regular, then so is α^*
- Examples: baa*!, (b|d)ad

RegExp in Linguistics

Description	*Pattern*	*Matches*
Char concat	stay	stay
Alternatives	(go | stay)	go, stay
Disjunction	[aeiou],[b-d]	a,e,i,o,u & b,c,d
Disjunction negation	[^aeiou]	b,c,d,f,...,z
wildcard char	.	a,b,1,!,...
zero or more	a*	ε, a, aa, aaa, ...
one or more	a+	a, aa, aaa, ...
optional (0 or 1)	colou?r	color, colour

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$

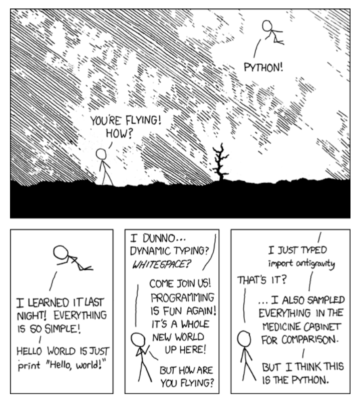
More Regular Expressions

- 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]+)\1 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



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 delimitors 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[1:3], a[:3], a[-2:]
  - for elt in a:
- Strings similar: “hello” or ‘hello’
- String & list ops: b = [“hello”,”17”,”there”]
  - str = ‘ ‘.join(a) ⇒ “hello 17 there”
  - str.split(‘e’) ⇒ ['h', 'llo 17 th', 'r', '']
  - list comprehensions:
    - [(x,len(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

```
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

```
>>> import nltk, re
>>> sent = “The quick brown fox jumped over the lazy dog”
>>> nltk.re_show(‘he’,sent)
T{he} quick brown fox jumped over t{he} lazy dog
>>> re.sub('er','ah',sent)
'The quick brown fox jumped ovah the lazy dog'
>>> nltk.re_show(‘(fox|dog)’,sent)
The quick brown {fox} jumped over the lazy {dog}
>>> re.findall(‘(fox|dog|rat)’,sent)
['fox', 'dog']
```

More RE's

>>> 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', 'q', 'r', 'f', 'j', 'p', ' ', 'v', 'h', 'l', 'd']

>>> 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

>>> 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

>>> nltk.corpus.gutenberg.items

('austen-emma', 'austen-persuasion', 'austen-sense', 'bible-kjv', 'blake-poems', 'blake-songs', 'chesterton-ball', 'chesterton-brown', 'chesterton-thursday', 'milton-paradise', 'shakespeare-caesar', 'shakespeare-hamlet', 'shakespeare-macbeth', 'whitman-leaves')

>>> count = 0

>>> for word in nltk.corpus.gutenberg.words('whitman-leaves'):

...   count += 1

...

>>> print count

154898

More Programming

```
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

>>> 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', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-TL'), ('Jury', 'NN-TL'), ('said', 'VBD'), ...], ...]

Any Questions?