Chapter 2
Regular Expressions & Finite State Automata

April 1, 2009

Announcement

- Assignment 1 posted
  Due April 8

Overview

- Formal laguages (review)
- Why FSAs?
- Formal definition of regular languages (reprise)
- Extension to regular expressions
- Finite state automata
- Definition, DFSA vs. NFSA, algorithms for using FSAs, pseudocode, search
- Regular languages vs. applications

Formal languages (review)

- From the point of view of formal languages theory, a *language* is a set of strings defined over some alphabet.
- The *Chomsky hierarchy* is a description of classes of languages.
- Languages from a single level in the hierarchy can be described in terms of the same formal devices.
- *Regular languages* can be described by *regular expressions* and by *finite-state* automata.
- Regular languages < context-free languages < context-sensitive languages < all languages

Three views on the same subject (review)

- Regular language: a set of strings
- Regular expression: an expression from a certain formal language which describes a regular language
- Finite-state automaton: a simple computing machine which accepts or generates a regular language

Why FSAs?

- Relationship between Chomsky Hierarchy, Generative Grammar
- Useful in morphological analysis, POS tagging, and more
- Point of entry for linguists into computational linguistics: Moving from abstract characterization (what) to algorithm (how).

Formal definition of regular languages: Symbols (reprise)

- ϵ is the *empty string*
- ø is the *empty set*
- Σ is an alphabet (set of symbols)

Formal definition of regular languages (reprise)

- The class of regular languages over Σ is formally defined as:
  - $\emptyset$ is a regular language
  - $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language.
  - If L_1 and L_2 are regular languages, then so are
    - $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$ (concatenation)
    - $L_1 \cup L_2$ (union or disjunction)
    - $L_1{}^*$ (Kleene closure)

(Jurafsky & Martin 2009:39)

Examples

- `abc`
- `a|bc`
- `(a|b)c`
- `a*b`
- `[^a]*th[aeiou]+[a-z]*`

Regular Expressions: (Re)view

- What are the three fundamental operators?
- What other operators are defined in Perl/Python (syntactic sugar)?
- What kind of applications might you use regular expressions in?

Regular Expressions: An Extension (1/2)

- Parentheses – ( ) in Python or Perl, \(\) in grep – allow you to ‘save’ part of a string and access it again…
  - … to specify regexps with repetition: `/([a-z]+) \1/`
  - … when you’re using regular expressions to rewrite strings:

```
regexp = re.compile(‘dog(s?)’)
regexp.sub(dawg\g<1>, input)
regexp = re.compile(r’\.(\s+[a-z])’)
regexp.sub(KEEPER\g<1>, input)
```

Regular Expressions: An Extension (2/2)

- NB: This extension to Python/grep/MS regular expression syntax actually takes them beyond the realm of regular expressions. The languages generated by regular expressions augmented with this kind of memory device are NOT regular languages – i.e., cannot be recognized by FSAs.

So what's an FSA anyway? (1/2

- An abstract computing machine
- Consists of a set of states (or nodes in a directed graph) and a set of transitions (labeled arcs in the graph)
- Three kinds of states: plain, start, final

So what's an FSA anyway? (2/2)

- FSAs can also be represented as tables:

	Input		
State	a	b	c
0	1	3	-
1:	1	2	3
2:	-	3	-
3:	-	-	-

Recognizing a regular language

- FSAs can be used to *recognize* a regular language.
- Take the FSA and a “tape” with the string to be recognized.
- Start with the start of the tape and the FSA’s start state.
- For each symbol on the tape, attempt to take the corresponding transition in the machine.
- If no transition is possible: reject.
- When the string is finished, check whether the current state is a final state.
- Yes: accept. No: reject.

Notes on Pseudocode

- Basic components of algorithms:
  - Loops
  - Conditionals
  - Variable assignment
  - Evaluating expressions (e.g. $i + 1$)
  - Input values
  - Return values

D-RECOGNIZE *in pseudocode*

```
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
    elsif transition-table[current-state, tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state, tape[index]]
      index ← index + 1
  end
```

Formal definition of regular languages (for reference)

- The class of regular languages over Σ is formally defined as:
  - $\emptyset$ is a regular language
  - $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language.
  - If L_1 and L_2 are regular languages, then so are
    - $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$ (concatenation)
    - $L_1 \cup L_2$ (union or disjunction)
    - $L_1{}^*$ (Kleene closure)

(Jurafsky & Martin 2009:39)

Proof of equivalence between FSAs and regular languages

- Three basic operations:
  - Union
  - Concatenation
  - Kleene closure
- Why isn't Kleene closure a special case of concatenation?

Regular languages are also closed under:

- Complementation: Interchange final states and non-final states
- Intersection: DeMorgan's theorem
- Reversal: Use final states as start states, the start state as the final state, and reverse all arcs.
- Difference: L – M = the intersection of L and the complement of M

NFSAs

- The FSAs considered so far are deterministic (DFSAs): there's only one choice at each node.
- NFSAs include more than one choice at at least one node.
- Those choices might include ϵ-transitions, or unlabeled arcs that allow one to jump from one node to another without reading any input.
- Recognizing strings with an NFSA is thus our first example of "search".

Two parameters

- Handling choices: backup, look-ahead, or parallelism
- Systematic exploration: depth-first, breadth-first, dynamic programming, A*, …

ND-RECOGNIZE *(1/3)*

```
Function ND-RECOGNIZE(tape, machine) returns accept or reject
  agenda ← {(Initial state of machine, beginning of tape)}
  current-search-state ← NEXT(agenda)
  loop
    if ACCEPT-STATE?(current-search-state) returns true then
      return accept
    else
      agenda ← agenda
                 ∪
               GENERATE-NEW-STATES(current-search-state)
    if agenda is empty then
      return reject
    else
      current-search-state ← NEXT(agenda)
  end
```

ND-RECOGNIZE *(2/3)*

function Generate-New-States(*current-state*)
    returns a set of search-states

    current-node ← the node the current search state is in
    index ← the point on the tape the current search-state is looking at
    return a list of search-states from transition table as follows:
        (*transition-table*[*current-node*, ϵ], *index*)
        $\cup$
        (*transition-table*[*current-node*, *tape*[*index*]], *index* + 1)

ND-RECOGNIZE *(3/3)*

Function ACCEPT-STATE?(*search-state*) **returns** true or false

current-node ← the node the search-state is in

index ← the point on the tape the search-state is looking at

if *index* is at the end of the tape

and *current-node* is an accept state **then**

return true

else

return false

A bit more on NFSAs

- ND-RECOGNIZE leaves the search strategy (depth-first or breadth-first) underspecified. Why?
- Any NFSA can be converted to a DFSA. How?

Regular languages vs. applications of regexps

- Regular expressions can define sets of strings.
- Applications of regular expressions include:
  - Search (and replace)
  - Spell checking
- In this case, the strings matched by the regexp are substrings of larger strings.
- Regexp matching is greedy.
- May or may not match multiple instances.
- Anchors become useful in search.

Look ahead: Morphology

- Find a partner
- Come up with a morphological subsystem that can be modeled by an FSA
- Write the FSA

Overview

- Formal languages (review)
- Why FSAs?
- Formal definition of regular languages (reprise)
- Extension to regular expressions
- Finite state automata
  - Definition, DFSA vs. NFSA
  - Algorithms for using FSAs
  - Search
- Regular languages vs. applications