October 2, 2003

Chapter 2

Regular Expressions & Finite State Automata

Leftover Administrivia

- Syllabus choose your own adventure (disjunction):
  - Ch 13: Language and Complexity
  - Ch 5: Probabilisitc Models of Pronunciation and Spelling
  - Ch 12: Lexicalized and Probabilistic Parsing
- Self-intro emails

  name, major, year, linguistics background, CS/programming background, languages spoken, what you want to get out of this class

Regular Expressions: Review

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

Regular Expressions: An Extension

- Parentheses — ( ) in 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:

    s/dog(s?)/dawg\1/g;

    s/\.(\s+[a-z])/KEEPER\1/g;

Regular Expressions: An Extension

- NB: This extension to Perl/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?

- An abstract computing machine
- Consists of a set of states (or nodes in a DAG) and a set of transitions (labelled arcs in a DAG).
- Three kinds of states: plain, start, and final

So what's an FSA anyway?

- FSAs can also be represented as tables:

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

Notes on Pseudocode

- Basic components of algorithms:
  - loops
  - conditionals
  - variable assignment
  - evaluating expressions (e.g., i + 1)
  - function arguments
  - return values

D-RECOGNIZE *in pseudocode*

```
function D-RECGONIZE(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
```

Short-comings of D-RECOGNIZE

- To apply D-RECOGNIZE to the problem of finding a string in a text, we'd need to solve the following:
  - D-RECOGNIZE assumes that it is already pointing at the string to be checked.
  - D-RECOGNIZE assumes that the end of a string to be checked must coincide with the end of any legitimate string it recognizes.
- How might we solve these problems?

Intuition of proof of equivalence between FSAs and regular languages

- Concatenation
- Closure
- Union

Regular languages are also closed under:

- Intersection: DeMorgan's theorem
- Complementation: Interchange final states and non-final states
- 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

```
function ND-RECOGNIZE(tape,machine) returns accept or reject
    agenda ← {(Initial state of machine, beginnging 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

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

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

current-node ← the node search-state is in
index ← the point on the tape 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?