Speech and Language Processing

Lecture 2
Chapter 2 of SLP
Today

- Finite-state methods
Regular Expressions and Text Searching

• Everybody does it
  ◆ Emacs, vi, perl, grep, etc..

• Regular expressions are a compact textual representation of a set of strings representing a language.
Example

- Find all the instances of the word “the” in a text.
  - `/the/`
  - `/[tT]he/`
  - `/[tT]he\b/`
Errors

- The process we just went through was based on two fixing kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)
Errors

• We’ll be telling the same story for many tasks, all semester. Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).
Finite State Automata

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs and their probabilistic relatives are at the core of much of what we’ll be doing all semester.
- They also capture significant aspects of what linguists say we need for morphology and parts of syntax.
Let’s start with the sheep language from Chapter 2

/\baa+/!
Sheep FSA

- We can say the following things about this machine:
  - It has 5 states
  - \( b, a, \) and \( ! \) are in its alphabet
  - \( q_0 \) is the start state
  - \( q_4 \) is an accept state
  - It has 5 transitions
But Note

• There are other machines that correspond to this same language

• More on this one later
More Formally

• You can specify an FSA by enumerating the following things.
  • The set of states: Q
  • A finite alphabet: Σ
  • A start state
  • A set of accept/final states
  • A transition function that maps QxΣ to Q
About Alphabets

• Don’t take term *alphabet* word too narrowly; it just means we need a finite set of symbols in the input.
• These symbols can and will stand for bigger objects that can have internal structure.
Dollars and Cents
Yet Another View

- The guts of FSAs can ultimately be represented as tables.

If you’re in state 1 and you’re looking at an a, go to state 2.

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

Graph:

- States: q₀, q₁, q₂, q₃, q₄
- Transitions: b → q₁, a → q₂, a → q₃, ! → q₄
Recognition

- Recognition is the process of determining if a string should be accepted by a machine
- Or... it’s the process of determining if a string is in the language we’re defining with the machine
- Or... it’s the process of determining if a regular expression matches a string
- Those all amount the same thing in the end
Recognition

- Traditionally, (Turing’s notion) this process is depicted with a tape.
Recognition

- Simply a process of starting in the start state
- Examining the current input
- Consulting the table
- Going to a new state and updating the tape pointer.
- Until you run out of tape.
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
        elseif transition-table[current-state, tape[index]] is empty then
            return reject
        else
            current-state ← transition-table[current-state, tape[index]]
            index ← index + 1
        end
    end
Key Points

- Deterministic means that at each point in processing there is always one unique thing to do (no choices).
- D-recognize is a simple table-driven interpreter
- The algorithm is universal for all unambiguous regular languages.
  - To change the machine, you simply change the table.
Key Points

- Crudely therefore... matching strings with regular expressions (ala Perl, grep, etc.) is a matter of
  - translating the regular expression into a machine (a table) and
  - passing the table and the string to an interpreter
You can view this algorithm as a trivial kind of *state-space search*. States are pairings of tape positions and state numbers. Operators are compiled into the table. Goal state is a pairing with the end of tape position and a final accept state. It is trivial because?
Generative Formalisms

- **Formal Languages** are sets of strings composed of symbols from a finite set of symbols.
- Finite-state automata define formal languages (without having to enumerate all the strings in the language).
- The term *Generative* is based on the view that you can run the machine as a generator to get strings from the language.
Generative Formalisms

• FSAs can be viewed from two perspectives:
  ✷ Acceptors that can tell you if a string is in the language
  ✷ Generators to produce *all and only* the strings in the language
Non-Determinism cont.

- Yet another technique
  - Epsilon transitions
  - Key point: these transitions do not examine or advance the tape during recognition
Equivalence

• Non-deterministic machines can be converted to deterministic ones with a fairly simple construction
• That means that they have the same power; non-deterministic machines are not more powerful than deterministic ones in terms of the languages they can accept
ND Recognition

- Two basic approaches (used in all major implementations of regular expressions, see Friedl 2006)
  1. Either take a ND machine and convert it to a D machine and then do recognition with that.
  2. Or explicitly manage the process of recognition as a state-space search (leaving the machine as is).
Non-Deterministic Recognition: Search

- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.
Non-Deterministic Recognition

- So **success** in non-deterministic recognition occurs when a path is found through the machine that ends in an accept.
- **Failure** occurs when all of the possible paths for a given string lead to failure.
Example

The diagram shows a finite state machine (FSM) with states $q_0$ to $q_4$. The transitions are labeled with symbols $b$, $a$, and exclamation point (!) and arrows indicating the direction of the transition. The table below the diagram lists the states corresponding to each symbol:

- $b$ transitions from $q_0$ to $q_1$
- $a$ transitions from $q_1$ to $q_2$
- $a$ transitions from $q_2$ to $q_3$
- The exclamation point (!) transitions from $q_3$ to $q_4$
Example

1

\[ b\ a\ a\ a\ ! \]

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \]

\[ b \rightarrow a \rightarrow a \rightarrow ! \]
Example

1  
\[ b a a a ! \]

2  
\[ b a a a ! \]

\[ q_0, q_1 \]

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \]

\[ b \rightarrow a \rightarrow a \rightarrow ! \]
Example

1  b a a a !

2  b a a a !

3  b a a a !
Example

1  b  a  a  a  !

2  b  a  a  a  !

3  b  a  a  a  !

4  b  a  a  a  !
Example
Example

1
\[ b \ a \ a \ a ! \]

2
\[ b \ a \ a \ a ! \]

3
\[ b \ a \ a \ a ! \]

4
\[ b \ a \ a \ a ! \]

5
\[ b \ a \ a \ a ! \]

6
\[ b \ a \ a \ a ! \]

7
\[ b \ a \ a \ a ! \]
Example

1. b a a a !
2. b a a a !
3. b a a a !
4. b a a a !
5. b a a a !
6. b a a a !
7. b a a a !
8. b a a a !

Chart:
- q0
- q1
- q2
- a
- q3
- q4

Sequence:
- q0
- q1
- q2
- a
- q3
- q4

Transition:
- b
- a
- a
- !
Key Points

• States in the search space are **pairings of tape positions and states** in the machine.
• By keeping track of **as yet unexplored states**, a recognizer can systematically explore all the paths through the machine given an input.
Why Bother?

• Non-determinism doesn’t get us more formal power and it causes headaches so why bother?
  ◦ More natural (understandable) solutions
Compositional Machines

- Formal languages are just sets of strings
- Therefore, we can talk about various set operations (intersection, union, concatenation)
- This turns out to be a useful exercise
Union

FSA_1

q_0 \rightarrow q_0 \rightarrow q_f

\epsilon

FSA_2

q_0 \rightarrow q_f

\epsilon

q_0 \rightarrow q_f

\epsilon

7/29/08 Speech and Language Processing - Jurafsky and Martin 42
Concatenation

\[ \epsilon \]

\[ FSA_1 \]

\[ FSA_2 \]

\[ q_0 \rightarrow q_f \]

\[ q_0 \rightarrow q_f \]
Negation

- Construct a machine M2 to accept all strings not accepted by machine M1 and reject all the strings accepted by M1
  - Invert all the accept and not accept states in M1
- Does that work for non-deterministic machines?
Intersection

• Accept a string that is in both of two specified languages

• An indirect construction...
  ◆ A\(^\wedge\)B = \sim(\sim A \text{ or } \sim B)