Speech and Language Processing

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/
  ◦ /\b[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.

FSAs as Graphs

- 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
  - \(b, a, \text{ and }!\) 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: \( \Sigma \)
  - A start state
  - A set of accept/final states
  - A transition function that maps \( Q \times \Sigma \) 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

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

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

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

Recognition as Search

• 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
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
  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

Example
Example

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)