Outline

- Other areas of AI (machine learning, reflex agents)
- English morphology
- Finite state transducers
- Finite state automata (this chapter)

Words

- Fun with FSA
- ND-recognition
- Recognition as Search
- Non-Determinism
- Context-free Grammars and Formal Languages
- Deterministic Recognition
- Regular Expressions and Finite State Automata

Administration

- Helpp://techcoreruleus/history/2003/jang.htm
- Weizenbaum, Invention of ELIZA, born (1952)

Today in Technology

History Archives January 8 - Computer scientist Joseph

Regular Expressions
Build simple interactive applications like Eiza

With other tools, we can also build powerful tools such as grep and find allow large corpus analytics and "shallow" processing

Uses of Regular Expressions in NLP

Basic Regular Expression Patterns (Perl notation): 2-3.27

Regular Expressions: Text Searches

As we will see, these are equivalent

As a way to specify the design of a particular kind of machine

As a practical way to specify textual search strings
ELIZA: I’m sorry to hear you are depressed.
I’m (depressed) and I’m sorry to hear you are.

ELIZA: I hear my mother says I’m depressed.

ELIZA: Can you think of a specific example.

ELIZA: They’re always bringing me about something or other.

ELIZA: In what way.

ELIZA: All of these.

ELIZA: Here are all alike.

ELIZA: Substitutions Using Memory

This is pretty much going to be the story of the rest of the course.

The process we just went through was based on fixing errors in the regular expression.

The second RE catches “the” when it belongs in a sentence, while the third RE does not match “the” when embedded in another word such as “other.”

Finding instances of the determiner “the” (using a pattern over a corpus)
More on this later...

A graphical view of the sheep language

Alternatives for our Sheep Language

FSAs as Directed Graphs (continued)

FSAs and their close cousins are a theoretical foundation of much of the field of NLP.
A Note on Alphabets

- doesn't accept hundred dollars
  accepts one dollar

- doesn't accept three dollars
  accepts three
  accepts any

- doesn't accept five dollars
  accepts five

A More Formal View

- the transition function that maps $s \times O$ to $O$
- the set of final states: $F \subseteq O$
- the start state: $s_0$
- a finite set of symbols $\Sigma$
- $\Gamma^*$ can be formally specified as a 5-tuple
Historically, recognition is viewed as processing an input written on a tape.

In terms of FAs, it's the process of determining whether or not a given input should be accepted by a given machine.

Recognition (or acceptance) is the process of determining whether or not a given input is accepted by a given machine.

### Recognition

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

### State Transition Table

- $i$: $q_1$
- $q$: $q_2$
- $q'$: $q_3$
- $q''$: $q_4$

- $i$: $q_1$
- $q$: $q_2$
- $q'$: $q_3$
- $q''$: $q_4$

- $i$: $q_1$
- $q$: $q_2$
- $q'$: $q_3$
- $q''$: $q_4$

- $i$: $q_1$
- $q$: $q_2$
- $q'$: $q_3$
- $q''$: $q_4$

- $i$: $q_1$
- $q$: $q_2$
- $q'$: $q_3$
- $q''$: $q_4$

### Another Representation

- $i$: $q_1$
- $q$: $q_2$
- $q'$: $q_3$
- $q''$: $q_4$
Key points about D-Recognize:

- D-Recognize is essentially a table-driven interpreter.
- The recognition code is universal for all PFA's. To change to a new formal language, you merely change the alphabet and the table. The recognition in the process.
- The recognition code always knows what to do at each point in the process.
- This technique for a string using a RE involves compiling the RE into a table and passing the table to an interpreter.

**D-Recognize (continued)**
Language.

FSAs can be useful tools for recognizing and generating languages. Subsets of regular languages.

Recall that deterministic machines are easy tape-driven.

Acceptance/representation by a given FSA.

Recognition is the process of determining if an input is in the language.

Recall that representations of formal languages (sets of strings) are called regular languages.

Here are compact textual representations of FSAs, their visualizations, and graph representations.

Example

<table>
<thead>
<tr>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
</tr>
</tbody>
</table>

Example representation by \( L \)

Given a machine \( M \) (such as a particular FSA \( L(M) \)) means the formal language

\( L \) explicitly enumerates the set.

Example representation of \( L(M) \) and regular expressions denote formal languages (without having to name symbols from the alphabet).

A formal language is a set of strings composed of symbols from a finite set.

Another Example Trace
We will talk later about representing regular languages with rules like these. There are three formally equivalent ways of (not including tables) looking at what we do:

1. regular expressions
2. finite automata
3. regular languages

Three Views

Review Examples
ND-machine to a D-machine and then do recognition with D-recognition. It also means that one way of doing recognition would be to convert an NFA to a DFA with respect to the language. Non-determinism does not add formal

test to FSA's. This means that formally they have the same power. They can recognize

Any ND machine can be converted to a D machine by a fairly simple con-

---

**Equivalence of D and ND Machines**

**NFSA:** Transitions do not examine / advance the tape.

**DFA:** Another way is to use transitions (arcs with no symbols)

---

**Non-Deterministic (cont'd)**

---

**Table:**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Diagram:**

---

**States can be lists of nodes**

---

**Non-Deterministic FSA (NFSA)**
The Problem of Choice

- Choice in non-deterministic models comes up again and again in NLP.

---

Non-Deterministic Recognition (continued)

---

Several standard solutions

1. Look at all choices in parallel

2. Look ahead in the input to help make choices

- Mark input/state at choice points

- Back up (search this chapter)

---

The Back-up Approach

---

Thus, at each choice point, the search process needs to remember the sequence of choices so far. If it gets to a point where it has to choose another unexplored choice, it has to back up (search this chapter) and try another choice.

---

Non-deterministic Recognition

---

In a non-deterministic machine, a path that is explored through the machine by a non-deterministic recognition succeeds whenever it is driven by an input string outside the language. Leads to an accept condition.

There may exist paths driven by strings in the language, that do not lead to an accept condition.

Back to Recognition...
Why bother?

Given all of the above, non-determinism seems to be more trouble than it's worth. Why introduce it at all?

Exercise

A search-state is a pairing of a single machine-state with a position on the input tape. By keeping track of not yet explored search-states, a recognizer can systematically explore all possible paths through a machine given some input.
Dynamic Programming / CS157

1. First in, first out (FIFO)
2. Second in, first out (LIFO)
3. States considered in creation order

Breadth-First Search

Previous example trace
1. Last in, first out (LIFO)
2. Second in, first out (FIFO)

More recently created states considered first

Depth-First Search

Note that NEXT is undefined in ND-Recognition

ND-Recognition as State-Space Search

Example Revised

Transition table for (q0, a) examined twice, but tape pointer different

- Each # is the current search state

Space-Space Search: Algorithms such as ND-Recognition, which systematically search for solutions
The primitive concatenation operation of a regular expression can be simulated by an automaton.

**Concatenation**

\[ L_1 \circ L_2 = \{ w_1 w_2 : w_1 \in L_1, w_2 \in L_2 \} \]

**Regular Languages**

- The class of languages that are characteristic by FAS (finite automata).
- The class of languages that are definable by regular expressions.

**Fun With Automata**

- Regular languages are closed under regular expressions.
- Closure with respect to memory.
- Instead of picking one choice and following it, examine all possible choices.

**Breadth-First Trace**
What might a better solution be?

One solution (a bad one) is to build one big complex regular expression:

Monday, February 29, 1900 •
Wednesday, January 27, 2000 •
January 27, 2000 •

Consider the problem of recognizing dates like the following.

| Composite Automata |

Add a new initial state with epsilon transitions to all the former initial states.

| Composite Automata (continued) |

Regular languages are also closed under other operations (e.g., intersection).

How would you implement Kleene + ?

Directly link initial and final state (εo)

Connect all final states back to the initial state (repetition).