Finite Automata.

CS154

Chris Pollett

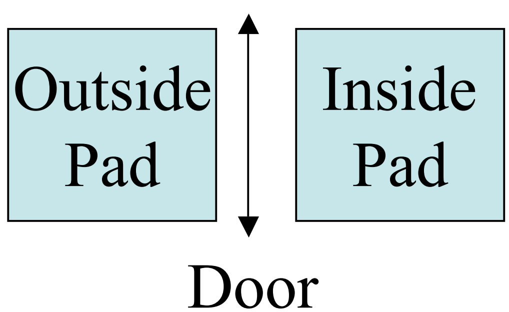
Feb. 6, 2006.

Outline

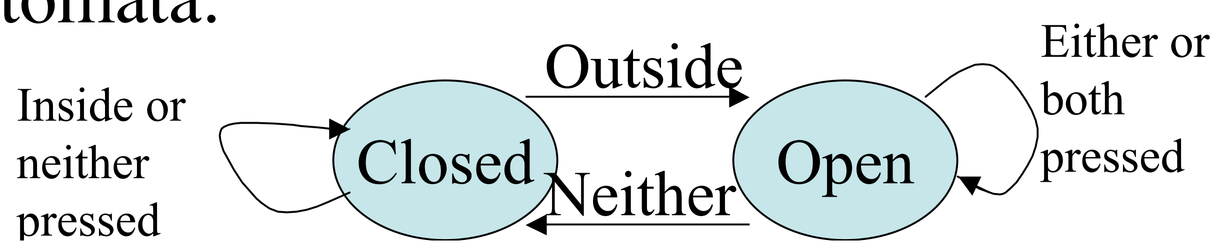
- Introductory Examples
- Formal Definition
- Example of the Definition
- Formal Definition of Accepts
- Designing Finite Automata
- Regular Operations

Introductory Examples

- Finite automata are computer models which are useful when one has very limited memory availability.
- Consider an automatic door say at a grocery store.



- We can model the door state this using a finite automata:



More on Door Example

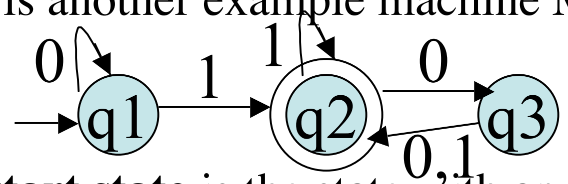
- The controller might start in a CLOSED state and receive the signals: OUTSIDE, INSIDE, NEITHER, INSIDE, BOTH, OUTSIDE, INSIDE NEITHER.
- It would then transition between the states CLOSED (start), OPEN, OPEN, CLOSED, CLOSED, CLOSED, OPEN, OPEN, CLOSED.
- Notice only need 1-bit of memory to keep track of state.
- It is also straightforward to represent transitions in a table:

	Neither	Outside	Inside	Both
Closed	Closed	Open	Closed	Closed
Open	Closed	Open	Open	Open

- Finite automata and the their probabilistic counterparts called **Markov chains** are also useful for pattern recognition. For example, recognizing keywords in programming languages. Or figuring out which word English is likely based on the previous ones seen.

Names for things

- The picture we drew of our automata a couple slides back is called a **state diagram**.
- We will usually use the variables M, N,… for machines.
- Here is another example machine M_1:



- The **start state** is the state with an arrow going from nowhere into it.
- If we are recognizing strings then when we stop process when we get to the end of a string of inputs.
- If we are in a double circled state at that point we accept the string otherwise we reject it. So double circled states called **accept states**.
- Arrows going from one state to another are called **transitions**.
- You might want to see if you can figure out if the above automata accepts each of the following strings: 000, 0110, 1101.

Formal Definition

- A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  1. Q is a finite set called the **states**.
  2. Σ is a finite set called the **alphabet**.
  3. $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**.
  4. $q_0 \in Q$ is the **start state**, and
  5. $F \subseteq Q$ is the **set of accept states**.
- The transition function tells us if we are in a given state reading a given symbol what is the next state to go to.

Example of the Definition

- The machine M_1 of a couple slides back can be described as:
  1. Q = {q1, q2, q3}
  2. ∑ = {0, 1}
  3. δ can be described as:

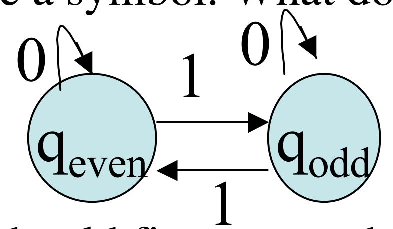
     (q1, 0) --> q1 (q1, 1) --> q2
     (q2, 0) --> q3 (q2, 1) --> q2
     (q3, 0) --> q2 (q3, 1) --> q2
  4. q1 is the start state, and
  5. F = {q2}
- We write L(M) for the language that M accepts. That is, those strings that M accepts.
- Given a set of strings S, we say **M recognizes S** if L(M)=S.
- So M_1 recognizes { w | w contains at least one 1 and an even number of 0s follow the last 1}

Formal Definition of Accepting a String

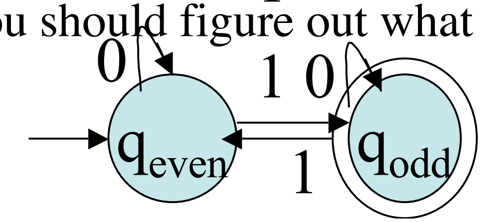
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string. Then **M accepts w** if a sequence of states $r_0 r_1 \ldots r_n$ in Q exist satisfying:
  1. $r_0 = q_0$
  2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i=0,1,\ldots, n-1$, and
  3. $r_n \in F$.
- We say **M recognizes language A** if $A = \{w \mid M$ accepts $w\}$.
- A language is called a regular language if some finite **automaton recognizes** it.

Designing Finite Automata

- Suppose we want to recognize the language that consists of an odd number of 1s.
- One approach is to “pretend to be the automaton”.
- You get symbols from {0,1} one by one.
- Ask yourself how much of the string so far do I read to remember in order to decide whether to accept or not. In this case,
  1. even so far
  2. odd so far
- Make each of these possibilities states. Next pretend you are in one of the states and see a symbol. What do you do?



- Finally you should figure out what your accept and final states are:



Regular Operations

- Just as the natural number are closed under operations like addition and multiplication, the regular languages enjoy some closure properties:
  - Union $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
  - Concatenation $A o B = \{xy \mid x \in A \text{ and } y \in B\}$
  - Star $A^* = \{x_1 x_2 \ldots x_k \mid k >= 0 \text{ and each } x_i \in A\}$