More Finite Automata

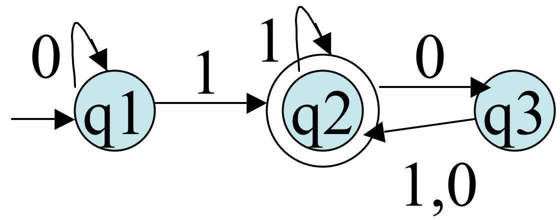
CS154
Chris Pollett
Feb 7, 2007.

Formal Definition

- A deterministic finite automaton (DFA) 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. $\delta: Q \times \Sigma \dashrightarrow Q$ 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.
- Note: the book calls these deterministic finite acceptors

Example of the Definition

- Consider the machine:



1. Q = {q1, q2, q3}
2. Σ = {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}

The Extended Transition Function

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let w be a string.
- We define the extended transition $\delta^*: Q \times \Sigma^{*} \text{-->} Q$ function inductively:

1. $\delta^*(q, \lambda) \text{ --> } q$
2. $\delta^*(q, wa) \text{ --> } \delta(\delta^*(q,w),a)$

- Intuitively, the equation $\delta^*(q, w) = q'$ tells us that if we are in state q when we begin to process the string w, then after processing w we will be in state q´.
- We say **M accepts w** if for some $q \in F$ we have $\delta^*(q_0, w)=q$. i.e., we start in the start state q_0, process w, and end up in an accept state.
- We say **M recognizes language A** if A= {w | M accepts w}.
- A language is called a **regular language** if some finite automaton recognizes it.

Trap States

- For DFA, we require δ to be a **total function**.
- This means for every state q and every alphabet symbol a, δ(q,a) must return some state q´.
- That δ is total will force δ* to be total as well.
- Consider the problem of designing an automaton for the language: {w | w = a^nb for some n≥0}.
- On a string like aabab, after the third a we know the string is not in the language; nevertheless, since the transition function is total we still need to continue processing the string until it is done.
- This can be done with a **trap state**, which has a loop back to itself for every alphabet symbol.

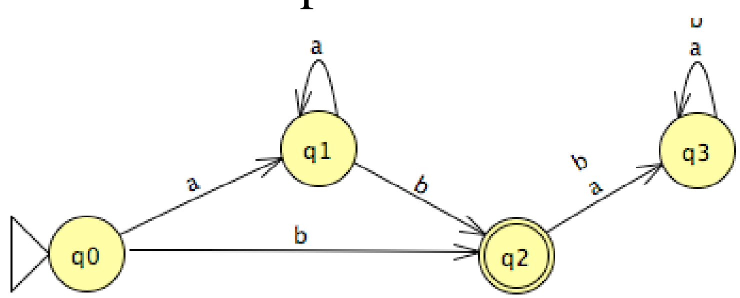
Theorem

Let M= (Q, Σ, δ, q_0, F) be a DFA, and G_M be its associated transition graph. Then for every q_i, q_j in Q and w in Σ^* , $\delta^*(q_i,w) = q_j$ iff there is in G_M a walk with label w (that is, the edge labels written down as a string are w) from q_i to q_j.

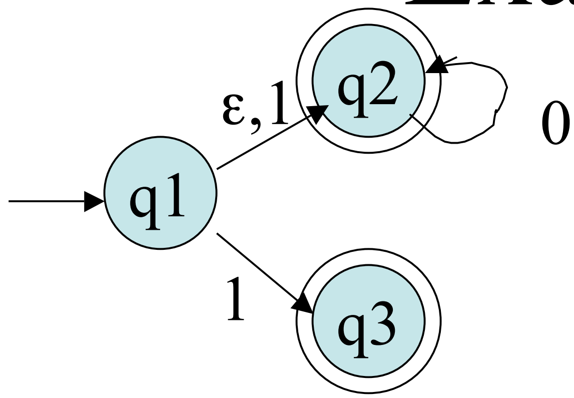
Proof. We give a proof by induction on the length n≥0 of w. Notice $\delta^*(q, \lambda)=q$ corresponds exactly with a walk of length 0 in G_M . So the $|w|=0$ case hold. Assume the statement is true up to some n. Let w=va where $|v|=n$. (The induction step case.) Suppose $\delta^*(q_i,v) = q_k$. By the induction hypothesis there is a walk W of length n in G_M from q_i,to q_k. If $\delta^*(q_i,w) = q_j$ we must have $\delta^*(q_i,w) =\delta(\delta^*(q_i,v),a) = q_j$ by the definition of δ^*. So we have $\delta(q_k,a) = q_j$. Thus, if we add to W the edge (q_k, q_j), the label on the resulting walk will be va=w as desired. This new walk has length n+1. It is also not hard to turn this argument around to show if one stated with a walk of length n+1 with label w, that one could show $\delta^*(q_i,w) = q_j$. You should do this at home.

Nondeterministic Finite Automata

- Having trap states, and transitions in general for every alphabet symbol can make ones diagrams looks messy and be a pain to maintain.
- To get around this one could imagine using a rule which says a string is automatically rejected if it ever happens that one cannot transition out of a state by reading the next symbol of the string.
- It is also sometimes convenient if we want to build bigger automata out of smaller automata, to allow two transitions with the same alphabet symbol out of a state. Or even to allow transitions where we don’t read a symbol at all!
- These ideas motivate the concept of nondeterministic machine.



Example NFA



- Notice we have more than one transition out of a state, we can have ε-transitions, and we don't need to have a transition from every alphabet symbol from a state.
- We say the NFA accepts w roughly if there is some sequence of transitions beginning with the start state, that processes each character of w and ends in an accept state.
- For instance, the machine above accept ε, 0, 00, 000, 1; but rejects 01, 11, 0001. It rejects 01 because although it can get to state q2 after seeing ε0 = 0, it has nowhere to go when it sees a 1 so it can't process the 1 so it rejects. No other path in the machine processes 01 even this far.

Formal Definition of an NFA

- Recall the power set of a set Q, P(Q), is the set of all subsets of Q.
- A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  1. Q is a finite set of states,
  2. Σ is an alphabet,
  3. $\delta: Q \times \Sigma \cup \{\lambda\} \rightarrow P(Q)$ is the transition function,
  4. $q_0 \in Q$ is the start state, and
  5. $F \subseteq Q$ is the set of accept states.

Example

- The machine a couple slides back is defined as $(Q, \Sigma, \delta, q1, F)$ where

1. Q={q1, q2, q3}
2. $\Sigma = \{0, 1\}$
3. δ is given by:

   δ(q1, ε)--> {q2}   δ(q2, ε)--> {}   δ(q3, ε)--> {}

   δ(q1, 0)--> {}   δ(q2, 0)--> {q2}   δ(q3, 0)--> {}

   δ(q1, 1) --> {q2,q3}   δ(q2, 1)--> {}   δ(q3, 1)--> {}
4. q1 is the start state
5. F = {q2, q3}

Formal Definition of Accepts

- To define what it means for an NFA to accept we can modify the definition of δ^* so that it works on a set of states. i.e., $\delta^*(q_i, w) = Q_j$, where Q_j is the set of possible state one could be in after processing w starting in state q_i.
- We say M **accepts** w then if $\delta^*(q_0, w) \cap F$ is nonempty.
- We write L(M) for the set of strings M accepts.