CMPE598.01

Lecture 1 - Notes

Kemal Berk Kocabağlı, 2013400045, kberkkocabagli@gmail.com, authored the first 3 sections
Baris Can Esmer, 2014400033, bariscan.esmer@gmail.com, authored the last 2 sections

February 11, 2018

1 CMPE 350 overview

Starting point of the class: CMPE350 (Finite Automata)

$$\text{Input} \xrightarrow[\textit{Algorithm}]{} \text{Output}$$

In the finite automata setting, we have a **finite (fixed) memory**. No matter how large the input is, the machine is forced to use that fixed memory to produce the output. In addition, the machine should read the input from left to right just **once**. So, n steps for n symbols.

(Note: In other traditional algorithms where memory is not a constraint, the space and time complexity is observed to compare the performance.)

For example, if we have k bits in our machine, there are
2^k states $\rightarrow$ finite!

A finite automaton receives an input string such as “aaabbab” and outputs either “Yes” or “No”, dividing the input space into 2 disjoint sets. The set of strings for which the machine says “Yes” constitutes a language.

2 Real-time Deterministic Finite Automata (Real-time DFA)

2.1 Classical DFA

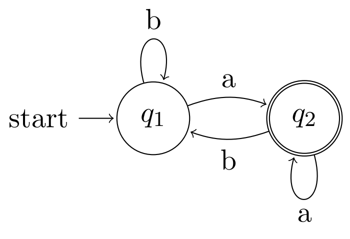
A DFA can be defined as the following:

$$M = (Q, \sum, \delta, q_s, F)$$

where

$$\begin{aligned} Q &= \text{the state set} \\ \sum &= \text{the input alphabet} \\ \delta &= \text{the transition function (the actual algorithm)} \\ q_s &= \text{the initial (starting) state} \\ F &= \text{the set of accept states} \end{aligned}$$

For example, one finite automaton could be:



DFA Diagram

$$
\begin{aligned}
Q &= \{q_1, q_2\} \\
\sum &= \{a, b\} \\
\delta(q_1, a) &= q_2 \\
\delta(q_1, b) &= q_1 \\
\delta(q_2, a) &= q_2 \\
\delta(q_2, b) &= q_1 \\
q_s &= q_1 \\
F &= \{q_2\}
\end{aligned}
$$

This machine accepts any string that ends with an "a". All strings that go by this rule is the language recognized by this machine.

2.2 Matrix Representation of DFA

A finite automaton can also be represented by matrices, which provides us with computational benefits. Converting the previous DFA into matrix notation, we get:

$$M_a = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \ M_b = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \ q_s = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

where M_a represents the transitions resulting from receiving an "a" as input and M_b "b" as input. The rows are the next state while the columns are the current state. Therefore, $M_a[1, 0] = 1$ because when the current state is q_1 and the input is "a", the next state according to δ is q_2.

The initial (starting) state q_s is a vector and since q_1 is the initial state in our example, the first entry of the vector which corresponds to q_1 is 1.

Now, if we would like to find if the machine will accept the string "aabaab", we need to compute: $M_b M_a M_a M_b M_a M_a q_s = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Since the machine ends up in the state: $q_1 \notin F$, it does not accept "aabaab". This is what we expected as we designed it so that it only accepts strings that end with an "a".

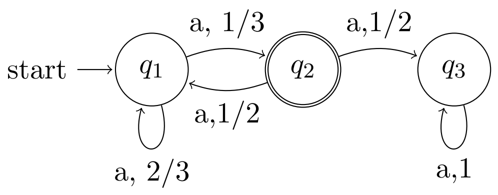
Notice that the matrices we formed have two important special properties:

1. They consist of only zeros and ones.

2. The sum of the numbers along each column is one.

3 Real-time Probabilistic Finite Automata (Real-time PFA)

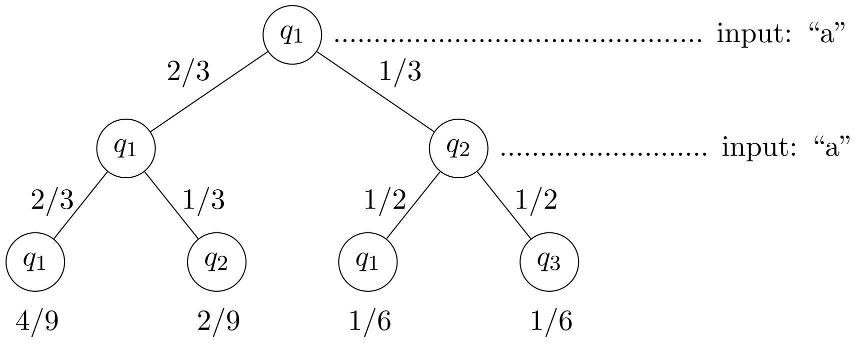
Deterministic Automata is in fact a special case of Probabilistic Automata, where all the probabilities are either 0 or 1. In Probabilistic Automata, each transition is associated with a probability $\in [0, 1]$.
Let's look at an example:



PFA Diagram

$$
\begin{aligned}
Q &= \{q_1, q_2, q_3\} \\
\sum &= \{a\} \\
\delta(q_1, a, q_1) &= 2/3 \\
\delta(q_1, a, q_2) &= 1/3 \\
\delta(q_1, a, q_3) &= 0 \\
\delta(q_2, a, q_1) &= 1/2 \\
\delta(q_2, a, q_2) &= 0 \\
\delta(q_2, a, q_3) &= 1/2 \\
\delta(q_3, a, q_1) &= 0 \\
\delta(q_3, a, q_2) &= 0 \\
\delta(q_3, a, q_3) &= 1 \\
q_s &= q_1 \\
F &= \{q_2\}
\end{aligned}
$$

If we would like to calculate the probability of this machine saying "Yes" to the input string "aa", we could form a binary tree that covers all possibilities.



Summing the probabilities of the paths that end at $\{q|q \in Q, F\}$, we get the answer 2/9. As seen above, the process is not that practical. Matrices come to rescue when handling probabilistic finite automata.

In the matrix notation, the same problem can be solved in the following way:

$$M_a = \begin{bmatrix} 2/3 & 1/2 & 0 \\ 1/3 & 0 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}, \; q_s = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Observe $M_a M_a q_s = \begin{bmatrix} 11/18 \\ \mathbf{4/18} \\ 3/18 \end{bmatrix}$

(corresponding states of the bold numbers are in F)
Therefore, the probability of the machine accepting "aa" is $4/18 = 2/9$.

Question: If we are dealing with probabilities of accepting strings, then what is the language associated with a PFA?

4 Error Tolerance in PFAs

In probabilistic machines, there are many bureaucratic procedures that determine a machine recognizing a language. DFA's either accept or reject a particular string, no matter how many times it operates on the input. However, unlike the DFA's, a PFA can reject and accept a string at different runs. So, how do we define the set of accepted strings (language) of a PFA ?

We can define which strings are accepted by a PFA in terms of an error notion. Suppose A is a PFA, we can define L to be the language recognized by A in many ways. Some of them are,

Zero Error : A accepts every string in set L with probability 1 and rejects every other string with probability 1

One-Sided Error : A accepts every string in L with non-zero probability and rejects every other string with probability 1.

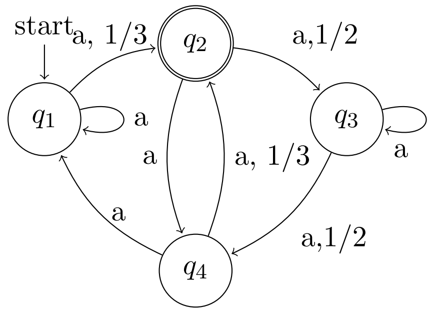
Bounded Error : A gives the correct response ("yes" for members of L, "no" for others) with probability greater than a, where $a > 1/2$

Theorem 4.1. *Zero error PFA and DFA recognize the same set of languages.*

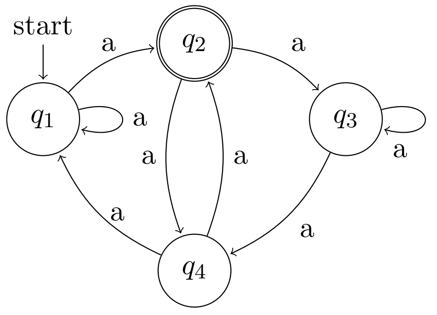
Proof. In a zero error PFA, at a given state any transition is "same". By same, it is meant that there is absolutely no difference which transition is taken, in terms of the response of automaton. Because by the definition of zero error, either all paths beginning from start state end up either at one of the accept states (if the particular string is an element of L) or at one of the reject states (if the string is not an element of L.) Therefore for each state, we can delete all transitions but one and make its probability 1. As stated before, this does not change anything in terms of the response of automaton, the languages recognized by this new PFA is the same. But now, this just constructed automaton is basically a DFA, all transitions between states are deterministic. Therefore the set of languages recognized by a zero error PFA is the same as the ones recognized by a DFA. $\square$

Theorem 4.2. *PFA with one-sided error and DFA recognize the same set of languages.*

Proof. Suppose there exists a PFA P, recognizing language L with one-sided error as defined above. A string s is recognized by P if and only if there exists a path beginning from the start state to one of the accept states, by following the transitions for each letter of s. We already know that NFA's have that exact property. Therefore from P we can construct an NFA (non-deterministic finite automata) P', using the same state set and transitions (of course, without the probabilities). One can easily see that a string is accepted by P if and only if it is accepted by P'. Since NFA's recognize the same set of languages as DFA's,[1] we conclude that PFA with one-sided error also recognizes the same set of languages. $\square$



PFA - with one sided error



Constructed NFA

[1]A proof can be found in this link

In the light of the two theorems above, we can intuitively say that “making mistakes” is better in terms of computational power. Restricting the automaton from making mistakes makes the probabilistic approach useless. Furthermore, making mistakes is not so bad as it seems at first glance, because one can always repeat the process multiple times and probability of majority of the outcomes being wrong is very low.

5 Introduction to Quantum Algorithms

According to the Bohr atom model, electrons are always moving in specific circular paths (orbits). However quantum mechanical model of the atom states that it is possible for an electron to be in many places at the same time, which is called quantum superposition of different states. With observation, it collapses into one state.

If we make use of this matter and build Quantum Automata, do these new machines recognize languages that PFA's were not able to ? Is there a computational advantage ?

For these new type of automata, we should update our matrix/vector models. For example,

$$\begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ -1/\sqrt{3} \end{bmatrix}$$

is a valid vector in this system. There is a negative number so we can not think of these numbers as probabilities. What we do instead is to interpret them as square roots of probabilities. After squaring the numbers, we get probabilities and by summing them we get 1. The transition matrices should also be modified such that the end vector is valid.

If quantum physics is a generalization of classical physics, then so must be our new machines compared to PFA's. This means every PFA could be converted to a Quantum Automaton.