LECTURE 23

10.1 Formal Definition of a DFA

Definition 10.1 *A* Deterministic Finite Automaton *(DFA) M, is a five-tuple:*

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- *Q is a* **finite** *set of states. It is important that the set of states be finite, otherwise it wouldn't be a* **finite** *Automaton.*
- *Σ is the alphabet over which the transitions are defined.*
- *δ is the transition function. The transition function describes each transition that takes as input a state and a symbol from the alphabet, and produces another state. In the example (vending machine) we would say that given state 0 and input symbol d, the transition function would produce state 10, or that $\delta(0, d) = 10$, therefore, $\delta : Q \times \Sigma \rightarrow Q$.*
- *q_0 is the start state.*
- *F is the set of final (accepting) states. The DFA will compute by processing one input symbol at a time, until all the input has been used. If the state that the machine is in when all the input has been used is one of those in F, the machine is said to "Accept", otherwise it is said to "Reject".*

EXAMPLE. We are asked to design a DFA that accepts only those strings of a's and b's that have two consecutive b's.

When designing a DFA it is important to assign meaning to each of the states. For example, in the case of the vending machine, each state represents the amount of money the vending machine has at that point.

We will keep track of how many consecutive b's have been read at a given time, by having 3 states. If the machine is in the first one (state 0) it means that it has not found two consecutive b's and the last symbol read is not a b. In the second state (state 1), the machine has not found two consecutive b's yet, and the last symbol read is a b, in the third state (state 2), the machine has found two consecutive b's.

We can describe the transitions in the following way:

- From state 0:

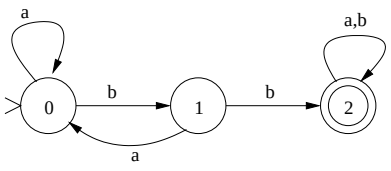
  - If an a is found, continue in state 0, no new b has been found.
  - If a b is found, move to state 1. The last symbol read was a b.

- From state 1:
  - If an a is found, move to state 0. The last symbol read is not a b.
  - If a b is found, move to state 2. The last two symbols read were b's.

- From state 2:
  - Once in state 2, we know that the string has two consecutive b's regardless of other symbols read. So the machine will remain in state 2 when an a or a b are found.

The start state is state 0, and the final state is state 2.

The previous description can be represented by the following DFA.



A machine can be described in two ways

1. A *transition diagram.* As shown above, a transition diagram contains all the information needed to describe the DFA. It includes, the states, the alphabet, the transition function, the start state and the set of final states (with double circle).

2. A *formal description* is given by explicitly showing all the five elements of a DFA.

The following is a *formal description* of the machine shown above in diagram form.

$$M_1 = (Q_1, \Sigma_1, \delta, q_0, F) \quad \text{where:}$$

- $Q = \{0, 1, 2\}$
- $\Sigma = \{a, b\}$
- $\delta : Q \times \Sigma \rightarrow Q$. δ can be described as a table, called a *transition table.*

δ	a	b
0	0	1
1	0	2
2	2	2

- $q_0 = 0$
- $F = \{2\}$

10.1.1 Computation Tracing

To trace the computation of a DFA when reading an input string w we use instantaneous configurations. We already used instantaneous configurations when tracing the computation of a vending machine. Each instantaneous configuration shows the current state of the DFA as well as the unused input, and is represented by the notation [current state, unused string]. From each instantaneous configuration we can go to the next configuration by following the transition table. In this case we say that the first configuration *yields* the next configuration. Formally,

$$[q_i, ax] \vdash [\delta(q_i, a), x]$$

Where $a \in \Sigma$ and $x \in \Sigma^*$.

EXAMPLE. Let us trace the computation of machine M_1, described above, on input *ababb*.

$$\begin{aligned}
[0, ababb] &\vdash [0, babb] \text{ since } \delta(0, a) = 0 \\
&\vdash [1, abb] \text{ since } \delta(0, b) = 1 \\
&\vdash [0, bb] \text{ since } \delta(1, a) = 0 \\
&\vdash [1, b] \text{ since } \delta(0, b) = 1 \\
&\vdash [2, \epsilon] \text{ since } \delta(1, b) = 2
\end{aligned}$$

The computation ends when there is no more input to read, in which case we write an ϵ (empty string) as the unused string. Since the computation ended in state 2, which is a final state $2 \in F$, then the M_1 accepts the string *ababb*. Sometimes we want to say that a computation yields some configuration after several steps, in which case we use the $\vdash^*$ symbol. In the example, we could use the notation $[0, ababb] \vdash^* [2, \epsilon]$ to represent a sequence of steps from the initial configuration $[0, ababb]$ to the final configuration $[2, \epsilon]$

We say that a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string w, if the computation of M on input w ends in a final state. Formally,

Definition 10.2 *M accepts $w \in \Sigma$ if $[q_0, w] \vdash^* [q_f, \epsilon]$, and $q_f \in F$.*

The language of a machine M is the set of all those strings that are accepted by the machine. Formally,

Definition 10.3 *Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, the language of the machine M is $L(M) = \{w \in \Sigma * | M \ \ accepts \ \ w\}$*

EXAMPLE. We are asked to design a DFA for the language

$$L = \{w \in \Sigma^* | w \text{ has an even number of } a\text{'s and an odd number of } b\text{'s}\}$$

As we said before, it is useful to design a DFA by assigning some meaning to each one of the states. In this case there are several option for the string read at a given point by our machine. The string could have either

- an even number of a's, and an even number of b's ($q_{e,e}$)
- an even number of a's, and an odd number of b's ($q_{e,o}$)
- an odd number of a's, and an even number of b's ($q_{o,e}$)
- an odd number of a's, and an odd number of b's ($q_{o,o}$)

The labels appearing in parentheses represent the names of the states. So we will have four states. We can easily build the transition function based on the description of these states. For instance, if the machine is in state $q_{e,o}$ reading an a, we know that the machine already read an even number of a's, after reading the a, the machine will have read an odd number of a's, but the number of b's won't change. Therefore the new state should be $q_{o,o}$. Figure 10.1.1 shows the transition diagram of machine M.

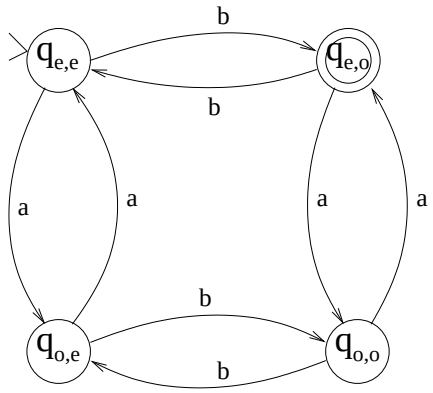


Figure 1: Machine that accepts strings with even number of a's and odd number of 'bs