Lecture 4

Designing Finite Automata

Lecture-4	**PO 1 & PSO 1**

4.1 Examples of Deterministic Finite Automata

Example 4.1:

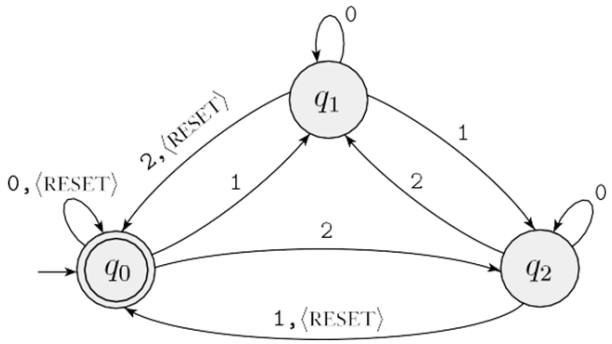

Figure 4.1: Diagram for finite automaton M_5.

In the formal description, M_5 is $(\{q_0, q_1, q_2\}, \{\langle RESET\rangle, 0, 1, 2\}, \delta, q_0, \{q_0\})$. The transition function δ is

States	a	b
$\rightarrow s$	q_1	r_1
q_1*	q_1	q_2
q_2	q_1	q_2
r_1*	r_2	r_1
r_2	r_2	r_1

Machine M_5 keeps a running count of the sum of the numerical input symbols it reads, modulo 3. Every time it receives the $\langle RESET\rangle$ symbol, it resets the count to 0. It accepts if the sum is 0 modulo 3, or in other words, if the sum is a multiple of 3.

Consider a generalization of Example 3.5, using the same four-symbol of alphabet Σ. For each $i \geq 1$ let A_i be the language of all strings where the sum of the numbers is a multiple of i, or the sum is a multiple of i. The sum is reset to 0 whenever the symbol $\langle$RESET$\rangle$ appears. For each A_i we give a finite automaton M_i, recognizing A_i. We describe the machine Bi formally as follows: $M_i = (Q, \Sigma, \delta, q_0, \{q_0\})$, where q_i is the set of i states $\{q_0, q_1, q_2, ..., q_{i-1}\}$, and we design the transition function δ_i so that for each j, if M_i is in q_j, the running sum is j, modulo i.
For each q_j the transitions will be

$\delta_i(q_j, 0) = q_j$,
$\delta_i(q_j, 1) = q_k$, where $k = j + 1$ modulo i,
$\delta_i(q_j, 2) = q_k$, where $k = j + 2$ modulo i, and
$\delta_i(q_j, \langle RESET \rangle) = q_0$.

4.2 Formal Definition of Computation

Here we will discuss about computations in finite automata. We already have an informal idea of the way it computes, and we now formalize it mathematically.
Let $M = (Q, \Sigma, \delta, q_0, F)$, be a finite automaton and let $w = w_1 w_2 ... w_n$ be a string where each w_i is a member of the alphabet Σ. Then M accepts w if a sequence of states $r_0, r_1, ..., r_n$ in Q exists with the following three conditions:

1. $r_0 = q_0$,
2. $(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, ..., n - 1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M recognizes language A if $A =$\{w | M accepts w\}.
A language is called a regular language if some finite automaton recognizes it.
Example 4.2: Consider the machine presented M_5 in Example 3.6. Let w be the string $10\langle RESET \rangle 22\langle RESET \rangle 012$.
Then M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is
$q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$,
which satisfies the three conditions. The language of M_5 is
$L(M_5) = \{w \mid$ the sum of the symbols in w is 0 modulo 3, except that $\langle RESET \rangle$ resets the count to 0$\}$.
As M_5recognizes this language, it is a regular language.

4.3 Designing Finite Automata

Suppose that we are given some language and want to design a finite automaton that recognizes it. Pretending to be the automaton, we receive an input string and must determine whether it is a member of the language the automaton is supposed to recognize. We get to see the symbols in the string one by one. After each symbol, we must decide whether the string seen so far is in the language. The reason is that we, like the machine, don't know when the end of the string is coming, so we must always be ready with the answer. Based on the observations the finite automaton is to be designed. A finite automaton has only a finite number of states, which means a finite memory.
For example: Suppose that the alphabet is {0, 1} and that the language consists of all strings with an odd number of 1's. We want to construct a finite automaton D_1 to recognize this language. Pretending to be the automaton, we start getting an input string of 0's and 1's symbol by symbol. Simply remember whether the number of 1s seen so far is even or odd and keep track of this information as we read new symbols. If we read a 1, flip the answer; but if we read a 0, leave the answer as is it is.
Represent this information as a finite list of possibilities. At a particular instance, the possibilities would be:

1. even so far and
2. odd so far.

Assign a state to each of the possibilities. These are the states of D_1, as shown here.

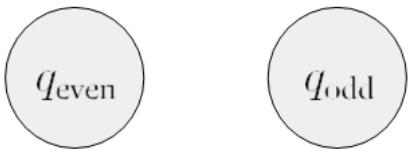

Figure 4.2: The two states q_{even} and q_{odd}.

Next, we assign the transitions by seeing how to go from one possibility to another upon reading a symbol. So, if state q_{even} represents the even possibility and state q_{odd} represents the odd possibility, we would set the transitions to flip state on a 1 and stay put on a 0, as shown here in figure 4.3. Next, we set the start state to be the state corresponding to

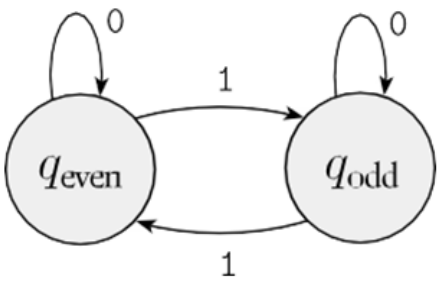

Figure 4.3: Transitions telling how the possibilities rearrange.

the possibility associated with having seen 0 symbols so far (the empty string). In this

case, the start state corresponds to state q_{even} because 0 is an even number. Last, set the accept states to be those corresponding to possibilities where we want to accept the input string. Set q_{odd} to be an accept state because we want to accept when we have seen an odd number of 1s. These additions are shown in the following figure 4.4.

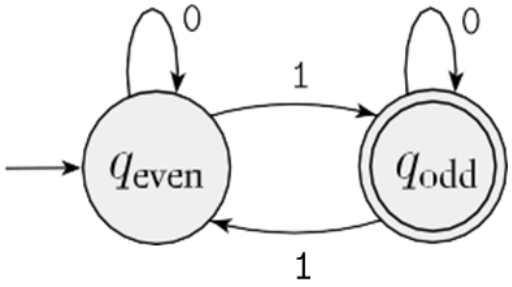

Figure 4.4: Adding the start and accept states.

Example 4.3: We have to design a finite automaton D_2 to recognize the regular language of all strings that contain the string 001 as a substring. For example, the strings 0010, 1001, 001, and 11111110011111 are all in the language, but 11 and 0000 are not. As symbols come in, we would initially skip over all 1s. If we come to a 0, then we note that we may have just seen the first of the three symbols in the pattern 001. If at this point we see a 1, there were too few 0s, so we go back to skipping over 1s. But if we see a 0 at that point, we should remember that we have just seen two symbols of the pattern. Now we simply need to continue scanning until we see a 1. If we find it, remember that we succeeded in finding the pattern and continue reading the input string until we get to the end. So there are four possibilities: We

1. haven't just seen any symbols of the pattern, or
2. have just seen a 0, or
3. have just seen 00, or
4. have seen the entire pattern 001.

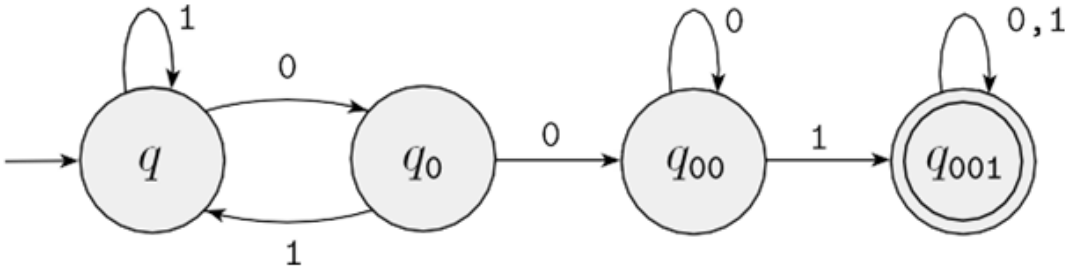

Figure 4.5: Automaton that accepts strings containing 001.

Assign the states q, q_0, q_{00}, and q_{001} to these possibilities. We can assign the transitions by observing that from q reading a 1 we stay in q, but reading a 0 we move to q_0. In q_0 reading a 1 we return to q, but reading a 0 we move to q_{00}. In q_{00} reading a 1 we move to q_{001}, but reading a 0 leaves we in q_{00}. Finally, in q_{001} reading a 0 or a 1 leaves we in

q_{001}. The start state is q, and the only accept state is q_{001}, as shown in Figure 4.5.