Discrete Mathematical Structures

Dr. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

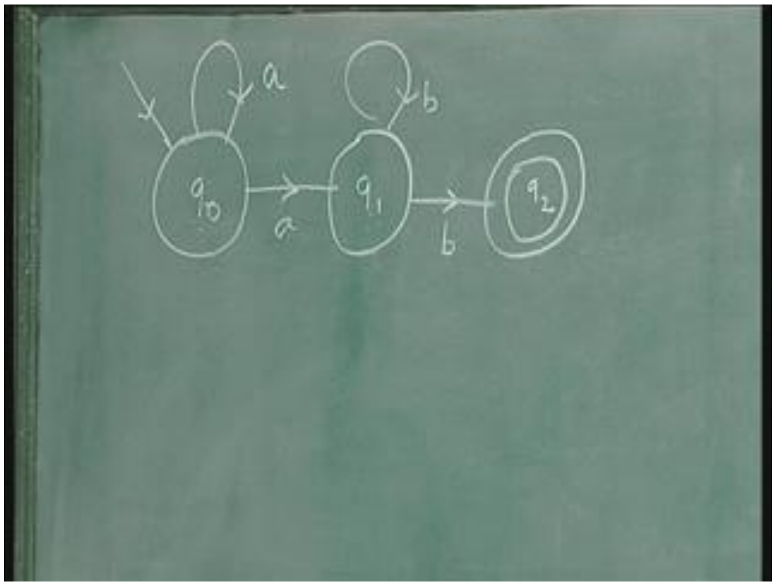
Lecture # 39

Finite State Automaton (Contd…..)

In the last lecture we saw about finite state automaton, how a finite state automaton can be used to represent FORTRAN identifiers or Pascal identifiers, integers, decimals and things like that. And we also saw that finite state automaton is an abstract model of a synchronous sequential machine. In the abstract model we can also think about a non-deterministic finite state automaton. In all the examples we considered in the last lecture you may remember that, given a state and an input symbol the next state was uniquely determined. That need not be the case, suppose given a state and the input symbol there is some choice of moving into the next state then what happens, that is called as a non-deterministic finite state automaton.

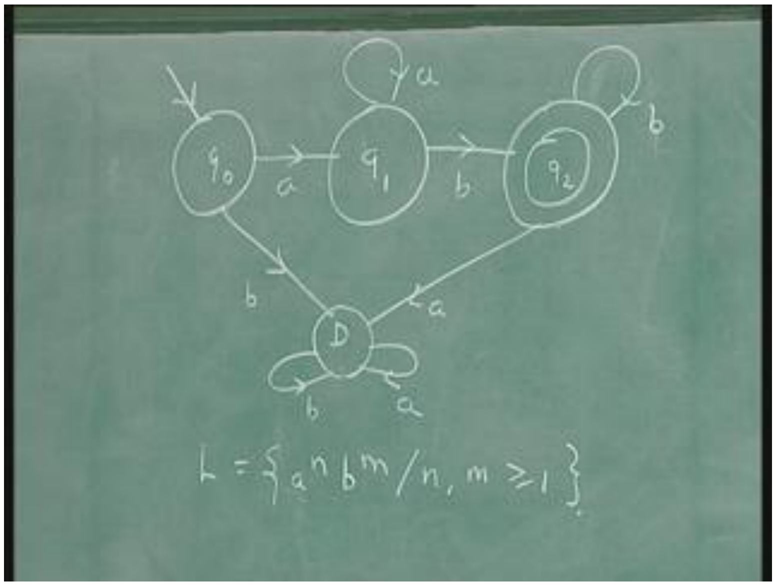
Let us take an example, consider this example:
The transitions are marked like this: q_0 is the initial state and q_2 is the final state, what is the set of strings accepted by this machine? First of all see that if you are in state q_0 and you get the input a then you have a choice of either you can go to q_0 or you can go to q_1, there are two choices. Similarly, when you are in q_1 and you get the input b you have two choices either you can go to q_1 itself or you can go to q_2. Now, from the diagram you can see a string of the form a number of a's followed by a number of b's will be accepted by this machine. We also had a deterministic automaton for that, if you remember

[Refer Slide Time: 03:20]



This is the deterministic automaton we had for this q_0 q_1 q_2 with a dead state a a b and then b. This was a deterministic automaton which accepted the same language L is equal to a power n b power m a string of a's followed by string of b's where n m are greater than or equal to 1, there should be at least one a and one b.

[Refer Slide Time: 04:18]



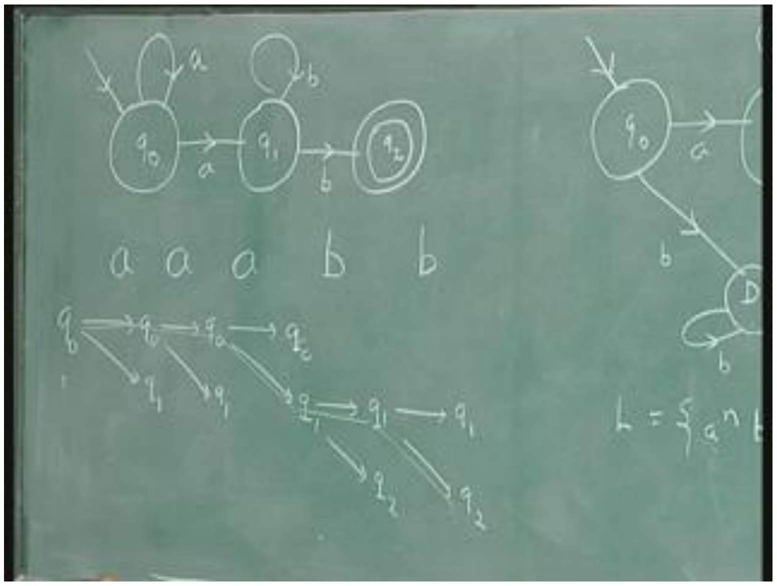
Now look at this diagram, let us see how a string of a's and b's will be accepted. Take this string a a a b b, start from q_0 that is the initial state. After reading a you have two choices either you can go to q_0 or you can go to q_1. You can mark it like this. Now, if you go to q_0 again after reading a there are two possible choices either you can go to q_0 or to q_1. But if you are in q_1 you cannot read a, because there is no transition mapping there for that so you cannot proceed here. Now again if you are in q_0 and you get a the next instance you can go to q_0 again or q_1 but from this direction you cannot proceed further.

Continuing like this in q_0 if you get b you cannot proceed further because there is no transition marked there. But in q_1 if you get b you have two choices either you can go to $q_1$or you can go to q_2. Again in q_1 when you get b you have two choices you can go to q1 or you go to q_2 but in q_2 you cannot read any symbol because no transition is there from q_2.

So, looking at this way from q_0 after reading a, it is possible that you can be in q_0 or q_1. After reading a a you can be in q_0 or q_1, after reading three a's again you can be in q_0 or q_1, after reading a a a b you can be in q_1 or q_2, after reading a a a b b you can be in q_1 or q_2. But one of the state q is a final state, q_2 is a final state so this string will be accepted by the machine. So, after reading the whole string there are a number of possibilities for the machine to be in and if one of them is a final state the string will be accepted. So the sequence of states which leads you to acceptance is here. This is the sequence of states which leads you to the acceptance: q_0 q_0 q_0 q_1 q_1 q_1.

The other sequences may stop in the middle or it may go to a non final state and that does not matter. The string is accepted if there is one sequence of states which leads you to acceptance that is the definition of a non-deterministic automaton. So when you connect it with sequential circuits you generally think about only deterministic automaton. But when we consider it as an abstract model we can think both of deterministic machine and non-deterministic machine.

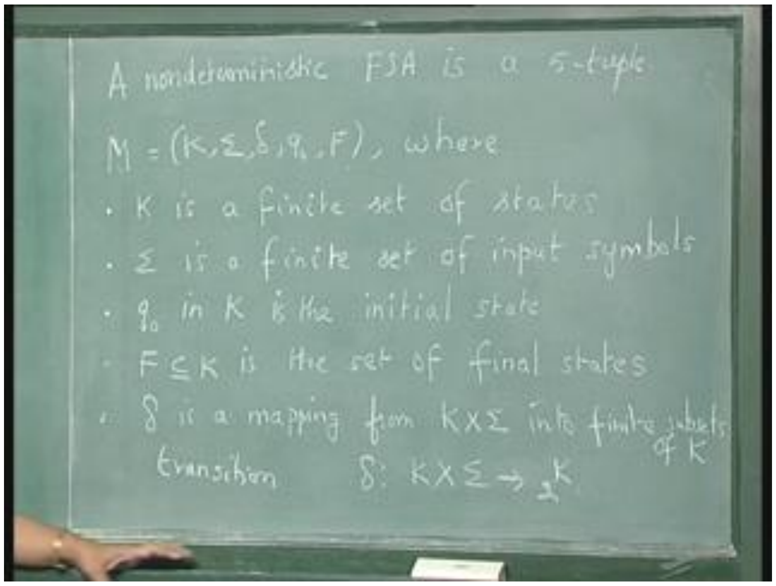
[Refer Slide Time: 07:30]



Now, in this particular example you see that the same language a power n b power m n m greater than or equal to 1 can be accepted by a deterministic machine also. Is it true that in general this will happen?
We find that in general also any language accepted by a non-deterministic automaton can be accepted by a deterministic automaton and we shall prove that. But before going in to that let us go to the formal definition of a non-deterministic finite automaton.

A non-deterministic finite state automaton is a five tuple m is equal to k sigma delta q_0f where again k is the finite set of states, sigma is a finite set of input symbols, q_0 in k is the initial state, f contained in k is the set of final states, delta is a mapping from k into sigma into finite subsets of k and this is where the difference comes. In the deterministic model it is from k into sigma into k but now it is from k into sigma into finite subsets of k and you denote it like this k into sigma into 2 power k this denotes all the subsets of k so this is called the transition function.

[Refer Slide Time: 08:56]



A nondeterministic FSA is a 5-tuple

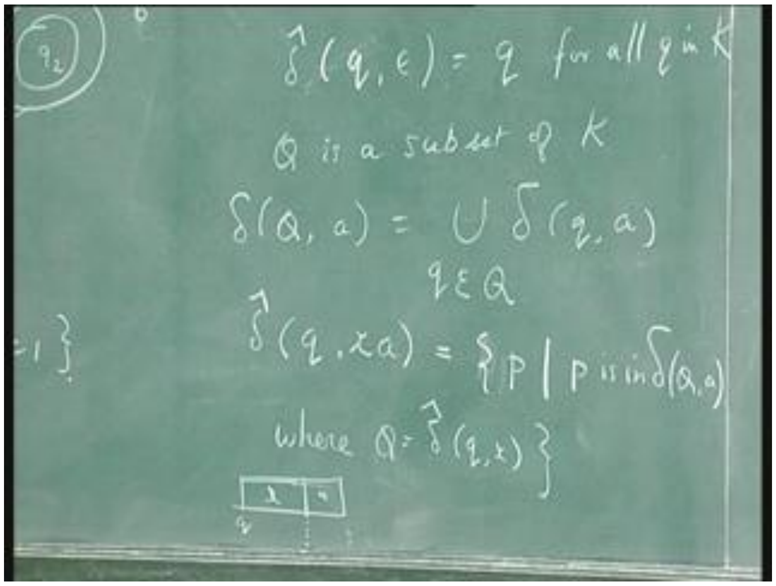
$M = (K, \Sigma, \delta, q_0, F)$, where

- K is a finite set of states
- Σ is a finite set of input symbols
- q_0 in K is the initial state
- $F \subseteq K$ is the set of final states
- δ is a mapping from $K \times \Sigma$ into finite subsets of K

transition $\delta : K \times \Sigma \rightarrow 2^K$

Now, proceeding further to determine the language accepted extend delta to k into sigma star call it as delta cap so delta cap epsilon delta cap q of epsilon is q for all q in K. And we also extend those two subsets of K. So, if q is a subset of K delta of (Q, a) is union of delta of (Q, a) where q belongs to Q. So you are extending the mapping to subsets of k, we need not use a different symbol because delta of (Q, a) is the same now how do you define delta cap of q, xa a string that is equal to a set of states p of the form p is in delta of (Q, a) where what is Q? Q is equal to delta cap of q(x). In essence it means like this; starting from a state q after reading x you can be in a finite number of states after reading the symbol a again from each one of them you can be in a finite subset that is denoted by this.

[Refer Slide Time: 11: 27]



$\hat{\delta}(q, \epsilon) = q$ for all q in K

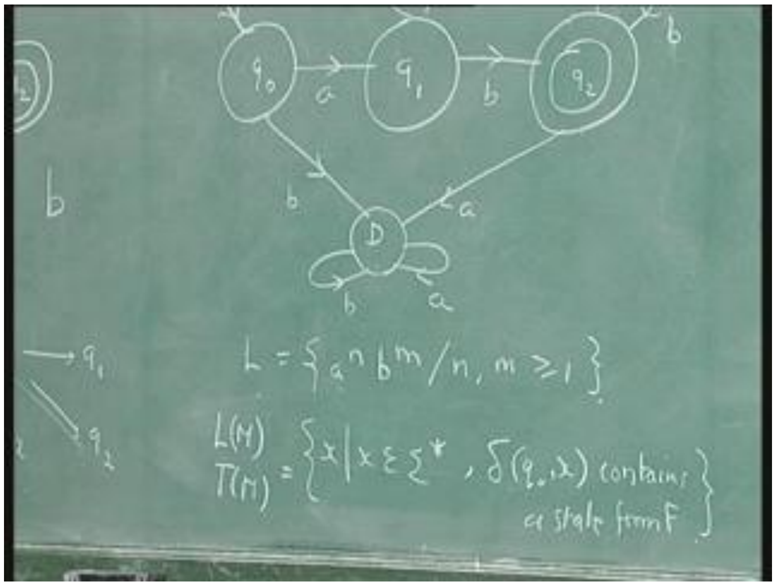
Q is a subset of K

$$\delta(Q, a) = \bigcup_{q \in Q} \delta(q, a)$$

$$\hat{\delta}(q, xa) = \{ p \mid p \text{ is in } \delta(Q, a) \text{ where } Q = \hat{\delta}(q, x) \}$$

So actually delta cap (Q, a) is the same as delta of (Q, a). Since this holds you need not have to distinguish between delta and delta cap and the language accepted is denoted by L(M) or T(M) and that is equal to the set of strings x where x belongs to sigma star delta of (q_0, x) contains a state from F. Actually you know that in the case of deterministic automaton this is a single state but in the case of non-deterministic automaton it is a collection of states delta of (q_0, x) will be a collection of states. If it contains one state from F then you say that the string is accepted. So this is a formal definition of acceptance.

[Refer Slide Time: 12: 45]



$$L = \{ a^n b^m / n, m \geq 1 \}$$

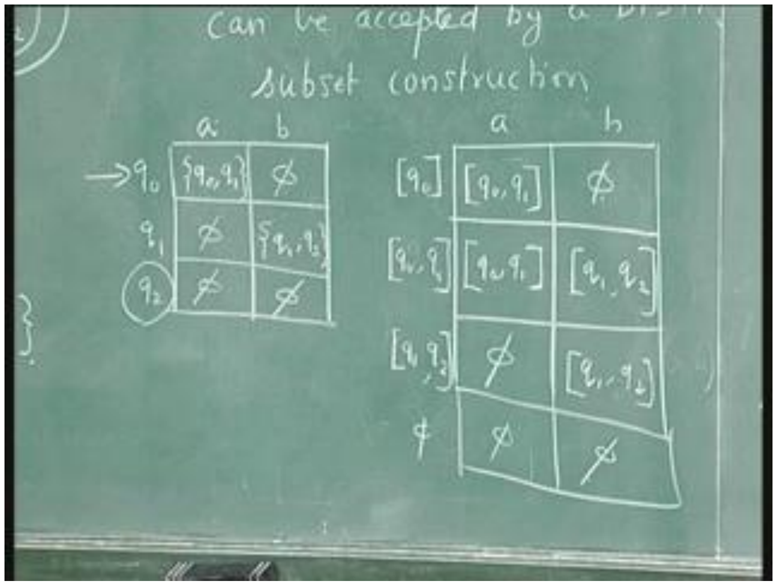
$$\begin{array}{l} L(M) \\ T(M) \end{array} = \{ x \mid x \in \Sigma^*, \delta(q_0, x) \text{ contains a state from } F \}$$

Now, let us see how we can go from non-deterministic automaton to deterministic automaton. So, any set accepted by a NFSA is Non-deterministic Finite State Automaton can be accepted by a deterministic finite state automaton. Now, before going into the regular proof we can consider this example, how we can get this from this? Take this example then we shall go to the formal definition. Let us consider the method which is used here called as subset construction.

Now the non-deterministic automaton state can be represented by a table like this: a b q_0 after reading a you have two possibilities that you denote as q_0 q_1 you cannot read b from q_0, then from q_1 if you read b you go to q_1 or q_2 you cannot read a, from q_2 you cannot read anything. So the state table for a non-deterministic automaton will be like this. For this example it is like this. You see that from q_0 if you read a, you can go to q_0 and q_1 that is written as a set like this because the mapping is from K into sigma into finite subsets of K.

Now let us try to draw another state table. What is the initial state here?
This is the initial state and this is the final state. So start with the initial state as a set like this. We are using square brackets to denote that it is a single state. Now from q_0 if you read a, you have the possibilities of q_0 and q_1. Actually you try to construct a deterministic automaton where corresponding to each subset you will have a state here. So each subset of the non-deterministic machine will be a single state here. So from q_0 if you get a, you can go to q_0 q_1 that is written like this and if you go to phi then from q_0 q_1 consider this. From q_0 if you read a, you can go to q_0 or q_1, from q_1 you cannot read a. So by making use of that union operation from q_0 or q_1 if you get a, you can go to the union of these two that is only q_0 q_1. Similarly, from q_0 q_1 if you get b from q_0 if you get b you cannot do anything but from q_1 if you get b you can go to q_1 or q_2. So from q_0 or q if you get b you can go to the union of these two that is q_1 and q_2.

Now we have considered this, we have not considered phi phi in the end we will consider that, this again has already been considered, now consider q_1 q_2 then you will have phi, phi and phi. Now from q_1 and q_2 if you get a, you have to take the union of this which is just phi, from q_1 and q_2 if you get b you have to take the union of these two that is q_1, q_2. This you have already considered so the whole table is over now. Here remember that each one is a single state.


can be accepted by a DFSA
subset construction

	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	ϕ
q_1	ϕ	$\{q_1, q_2\}$
q_2	ϕ	ϕ

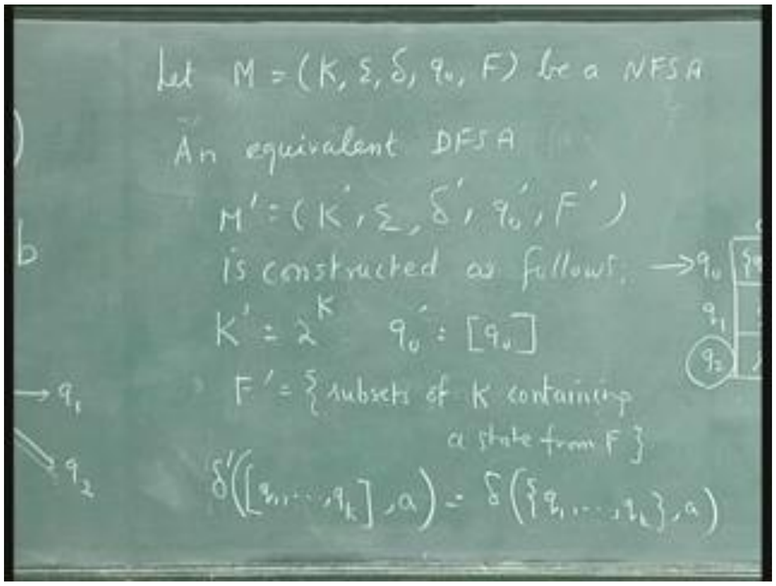
	a	b
$[q_0]$	$[q_0, q_1]$	ϕ
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	ϕ	$[q_1, q_2]$
ϕ	ϕ	ϕ

Now look at this diagram, I will re-label that states like this: q_0 q_0 q_1 q_1 q_2, this denotes the same table, this one is phi, I can even put phi within square brackets it is only a sort of a symbol it does not matter. Look at this transition $[q_0]$ a is $[q_0\ q_1]$; $[q_0]$ b is phi, $[q_0\ q_1]$ a is $[q_0\ q_1]$ $[q_0\ q_1]$ b is $[q_1\ q_2]$ from this table, $[q_1$ q2] a is phi, $[q_1\ q_2]$ b is this. Now how do you designate the final state and the initial state?
The set containing q_0 alone is taken as the initial state and in this table q_2 is a final state so any subset containing q_2 will be a final state. So this subset contains q_2 so that is a final state so this will become a final state. So you see that you have the same diagram but I have only relabeled the states and you are getting the deterministic automaton for this automaton. This is to illustrate with an example.

Now the proof also is like this:
Let M is equal to (K sigma delta q_0, F) be a NFSA an equivalent DFSA M dash is equal to (K dash sigma delta dash q_0 dash F dash) sigma is the same because you are considering over the same alphabet and is constructed as follows. K dash is 2 power k that is all subsets of K will be K dash, q_0 dash will be the set containing $[q_0]$ alone that is each subset of K will be a state in K dash and the subset containing q0 alone will be the initial state and F dash is equal to subsets of K containing a state from F. All subsets which contain a state from F form F dash.

Now you have to define delta dash, delta dash is defined like this:
Delta dash some q_1 is a subset qka this will be delta of q_1, qka. That is, you take delta of $[q_1$, a] $[q_2$, a] $[q_3$, a] etc find the union that is given like this. This is the way we have constructed like this.

[Refer Slide Time: 22:21]



Let $M = (K, \Sigma, \delta, q_0, F)$ be a NFSA

An equivalent DFSA

$M' = (K', \Sigma, \delta', q_0', F')$

is constructed as follows:

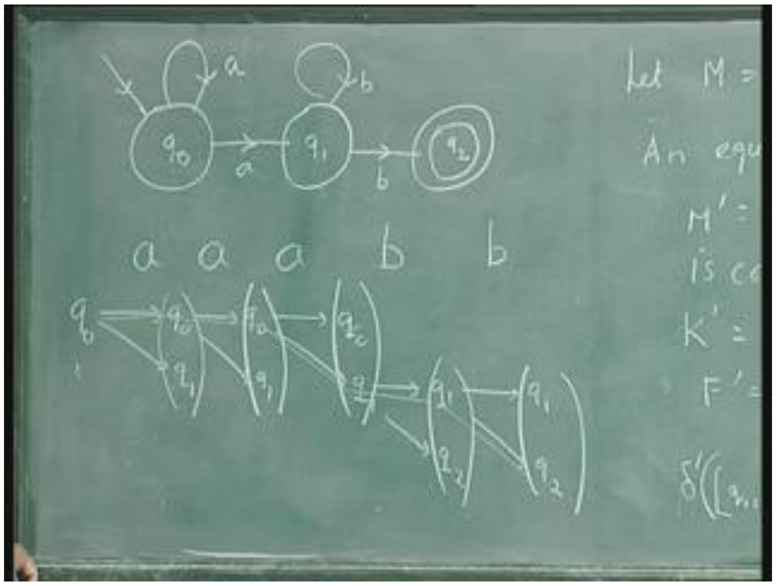
$K' = 2^K \quad q_0' = [q_0]$

$F' = \{$ subsets of K containing a state from $F \}$

$\delta'([q_1, \ldots, q_k], a) = \delta(\{q_1, \ldots, q_k\}, a)$

Now, looking at this table also you see that from q_0 after reading a you go to this state in the deterministic one, you go to this state in the deterministic one, you go to this one, this one and this one. So if this subset contains a state from F the string will be accepted.

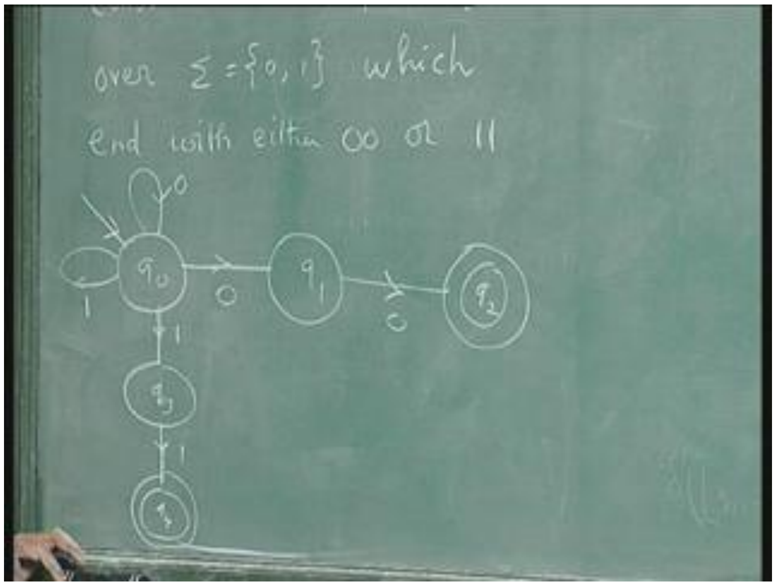
[Refer Slide Time: 22:43]



This is the construction for non-deterministic automaton and the proof is like this. I shall not go into the details of the proof because it may take quite some time. The main idea is how you go about the subset construction. You can see that use induction and the equivalence can be proved, use induction on the length of the string. Instead of going to this induction portion of the proof formally we shall consider one more example.

Consider the set of strings over sigma is equal to 0, 1 which end with either 0, 0 or 1, 1. Now the non-deterministic diagram for that will be like this, it will be q_0 then q_1 q_2 q_3 this is q_2 q_3 q_4. You can very easily draw the non-deterministic diagram like this. First you can read any string of a's 0s and 1s. And once you get two 0s the string will be accepted. It should end with 0, 0 or with 1, 1. You can see that this is a non-deterministic state diagram it is a non-deterministic finite state automaton because from q_0 if you get 0 you have the possibility of going to q_0 or q_1, from q_0 if you get 1 you can have the possibility of going to q_0 or q_3, this is the non-deterministic diagram.

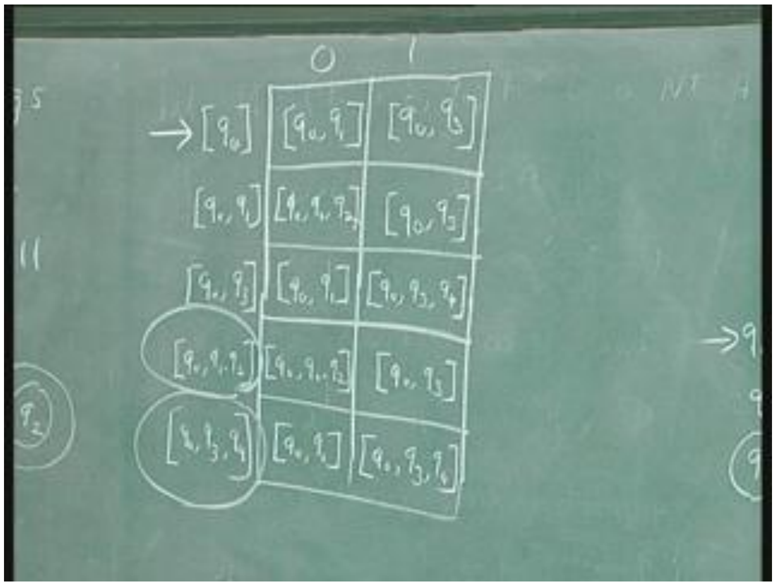
[Refer Slide Time: 26:02]



Let us construct the deterministic diagram draw the state table {0, 1}. So you have to start with q_0 q_0 0 is q_0 q_1 and q_0 1 will be q_0 q_3. Next you have to consider with this q_0 q_1 q_0 q_3. From q_0 or q_1 you get 0 so you go to q_0 q_1 or q_2, from q_0 or q_1 if you get 1 you go to q_0 or q_3 and here again you get this, from q_0 or q_3 if you get 0 you again get q_0 or q_1. If you get 1 you can go to q_0 q_3 or q_4. Now for this you have already considered, you have to consider for this so next you have to consider for the state q_0 q_1q_2 and also for the state q_0 q_3 q_4. Now, from q_0 q_1 or q_2 if you get 0 you again go to q_0 q_1 or q_2 this from the diagram you get f.

From q_0 q_1 or q_2 if you get 1 you get to q_0 or q_3 only so you get q_0 q_3, from q_0 q_3 or q_4 if you get a 0 you get to q_0 or $q_1$only so q_0 q_1, from q_0 q_3 or q_4 if you get 1 you get q_0 q_3 q_4. Now this is the initial state the singleton containing q_0 is the initial state. What are the final states here? Here q_2 and q_4 are final states so any subset containing them will be a final state so this is a final state and this is a final state.

[Refer Slide Time: 29:12]



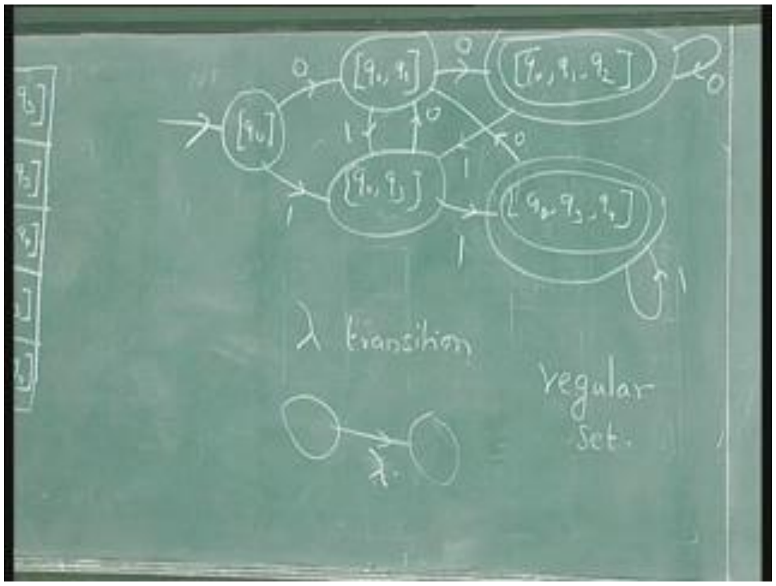
	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_0, q_3]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3, q_4]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$

So, if you draw the state diagram for this it will be like this; q_0 then q_0, q_1 q_0, q_3 q_0, q_1, q_2 then q_0, q_3, q_4 each one of them is a state this is the initial state and these are final states, mark the transitions [q_0, 0] is this and q_0, q_1, 0 is this. [q_0, 1] is this, from here q_0 q_1 if you get 1 you go here, from here if you get 0 you go here if you get 1 you go here, from q_0 q_1 q_2 if you get 0 you remain in the same state, if you get 1 you go to q_0 q_3, from q_0 q_3 q_4 if you get 1 you remain in the same state if you get 0 you go here.

Now here again you can very easily see that if you get two 0s you reach the final state, if you get two 1s you reach the final state. If you get some sequence and at the end if you get two 1s from here if you get two 1s 1 1 you will get, from here if you get two 1s 1 1 you will reach the final state. Similarly, if you get any sequence of 0s and 1s and if you are in this state when you get 0 0 you go here and accept if you are here you will get a 0 0 and you accept. If you are here and then afterwards you get a 1 1 1 you will accept 1 0 you will go here then either you must get two 0s or two 1s. So you can very easily see that from this diagram any string which ends with two 0s or two 1s will be accepted. And also you can very easily see that this is a deterministic diagram so from the non-deterministic automaton like this by use of subset construction we have got a deterministic automaton. So we have seen how to construct a deterministic automaton from a non-deterministic automaton.

We can also have lambda transitions where from one state by reading an epsilon or lambda you go can go to another state. And even with this facility the power of the finite state automaton will not be increased it will just accept only regular sets. The set accepted by a non-deterministic automaton or deterministic automaton is called a regular set.

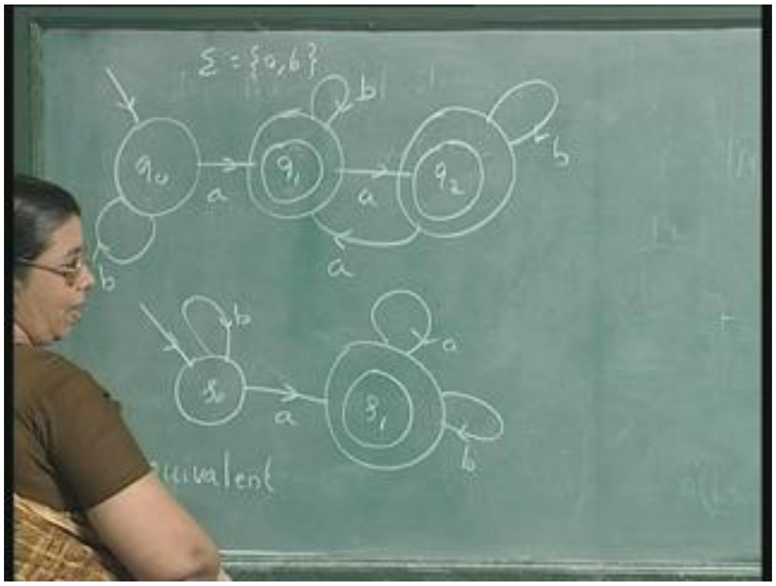
[Refer Slide Time: 33:14]



In these cases we have seen this as a acceptor, next we shall talk about a minimization procedure. Look at this diagram, the alphabet is a, b you have three states these two are final states q_0 q_1 q_2 the state diagram is given like this b a b b a a what is the language accepted by this machine. If you look at the diagram carefully, you see that if you get only a string of b's it will not be accepted. But once you get one a, you go here afterwards you will be either in q_1 or q_2 only so it can be followed by any string. So any string of a's and b's where there at least one a it will be accepted by this machine. If you do not have any a, the string of b's it will not be accepted. So the language consists of set of strings in which at least one a is present.

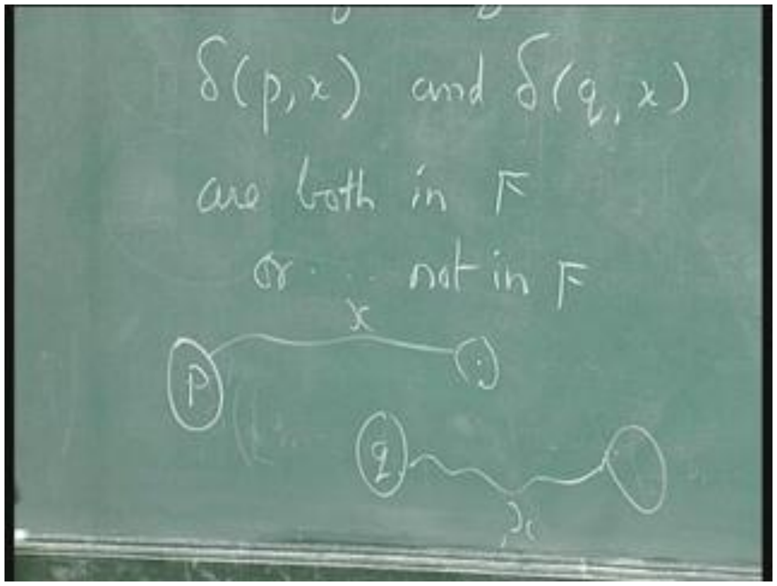
Now look at this diagram, I will call the states just s_0 s_1 instead of q_0 and all, b a a b what is the language accepted by this machine? It is the same, any string of b's will not be accepted but if you get at least one a you go here and you can read any other string afterwards. So these two automata are equivalent they accept the same set of strings, they are equivalent. This is a minimal state automaton but this is not the minimum state automaton because you can reduce the number of states.

[Refer Slide Time:35:34]



How do we go about reducing the number of states?
But before that we need some definitions of equivalent states and so on. Two states p and q you are considering a DFSA. Consider a DFSA two states p and q are equivalent if and only if for any string x in sigma star that is you are considering a DFSA for which the input alphabet is sigma. Two states p and q are equivalent if and only if for any string x in sigma star delta of p(x) and delta of q(x) are both in F or both not in F. If you take the state p and the state q after reading the string x you reach a state, after reading x from here you reach a state both of them should be final states or non final states whatever may be x. Hence, for a particular x both of them may be final for another x both of them may be non final. So in that case you say that the two states p q are equivalent.

[Refer Slide Time: 37:38]

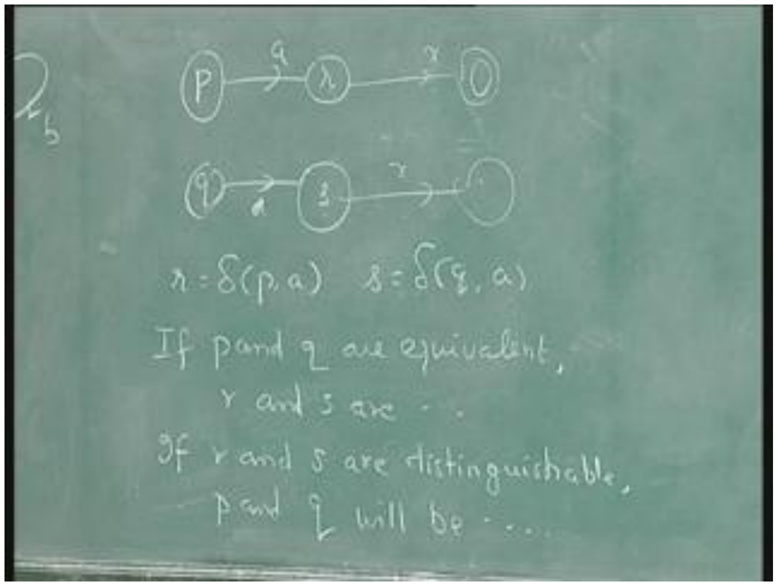


p and q are distinguishable if there exists a string x such that delta of p(x) belongs to F and delta of q(x) does not belong to F or the other way round. So two states p and q are distinguishable if there exists a string x such that delta of p(x) belongs to F and delta of q(x) does not belong to F, the other way round means delta of q(x) belongs to F and delta of p(x) does not belong to F. that is from p and q this x takes you to a state and from q it takes to you to a first state, one of them will be a final state and one of them is not a final state then p and q are distinguishable.

Then you see that in that case x is called the distinguishing string or sequence. This idea of equivalence states the distinguishing sequences; this plays a very important role in computer networks. Though I cannot describe everything without proper background you must remember that this sort of an idea of a finite state automaton where you talk about equivalent strings and distinguishing states and about the distinguishing sequences plays a very very important role in computer networks.

Therefore, if p and q are two states and their successors are r and s respectively. That is r is equal to delta of p of a and s is equal to delta of q of a. If p and q are equivalent then what can you say about r and s? Any string if it takes p to a final state it should also take q to a final state, need not be the same final state it can be different. And if it takes to a non final state this also should be taken to a non final state, so that is true for any string x. If you consider a string of the form x is equal to ay then the string y will take r to a final state if and only if it will take s to a final state. And if it takes r to a non final state s also will go to a non final state after reading y. So, if p and q are equivalent r and s are equivalent. And if r and s are distinguishable what does that mean? There is a string x which takes one to a final state and another to a non final state. So the string ax takes p to a final state and takes q to a non final state. So if x is a distinguishing string for r and s ax will be a distinguishing string for p and q. So if r and s are distinguishable p and q will be distinguishable.

[Refer Slide Time:42:42]



$r = \delta(p, a)$ $s = \delta(q, a)$

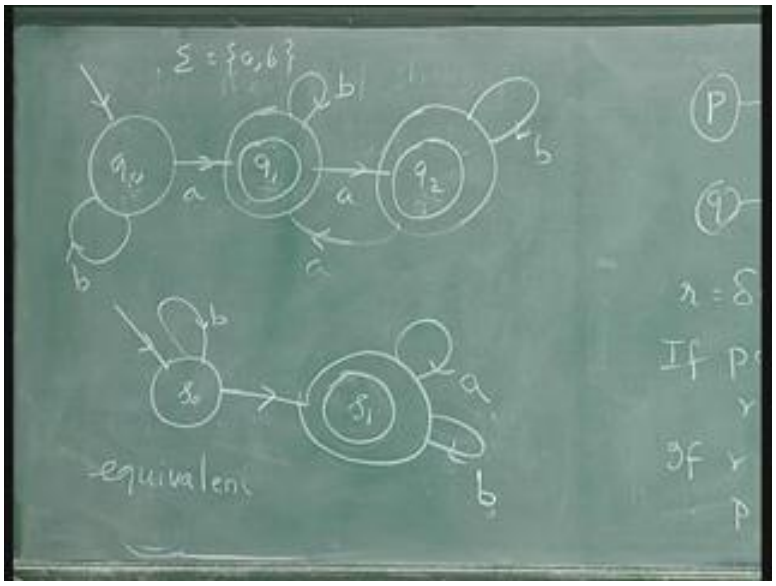
If p and q are equivalent, r and s are . .

If r and s are distinguishable, p and q will be . . .

So making use of this you can try to find the minimum state automaton for a final state automaton. For example for this one the minimum state automaton is this. How do we get this?
You can see that the states q_1 and q_2 are equivalent because from here if you read any string at the end you will only go to q_1 or q_2 which is a final state. From q_2 if you start reading a string you will again go to q_1 or q_2 only you cannot go back to q_0 so that will be a final state. So any string if you take starting from q_1 it will be accepted starting from q_2 also it will be accepted in this particular example. Whereas take q_0 and q_1 if you take b starting from q_0 you go to q_0 starting from q_1 you go to q_1 is a final state and another is a non final state. So these two are distinguishable and the distinguishing string is b, b alone will take you to a non final state here, it will take you to a final state here.

Similarly, if you consider q_0 and q_2 they are distinguishable because if you take b it takes you to a non final state, here b will take you to a final state. So q_0 and q_1 are distinguishable, q_0 and q_2 are distinguishable, but q_1 and q_2 are equivalent. So we can merge the equivalent states and when you merge these two q_0 is here as s_0 but q_1 and q_2 are merged as s_1. So starting from q_0 when you get a you go here when you get b you go here. This way you can find the minimal state automaton you have to merge the equivalent states.

[Refer Slide Time: 44:44]



Let us consider one more example and consider the minimal state automaton. Take this example, you are having six states q_0 q_1 q_3 q_4 q_5 q_6 and what are the sets of strings accepted?
If a string contains just 1 1 any 0s followed by 1 it may be followed by more 0s also it will be accepted. But if a string does not contain 1 it will not be accepted. If it contains two or more 1s also you go to q_6 and it will not be accepted. So the set of strings accepted will be a sequence of 0s with 1 1 followed by a sequence of 0s. Just 1 alone can also be accepted. Now this is not the minimum state automaton how will you find the minimum state automaton?
To find the minimum state automaton you have to take all states try to partition them further and further until we are able to get no more partitions.

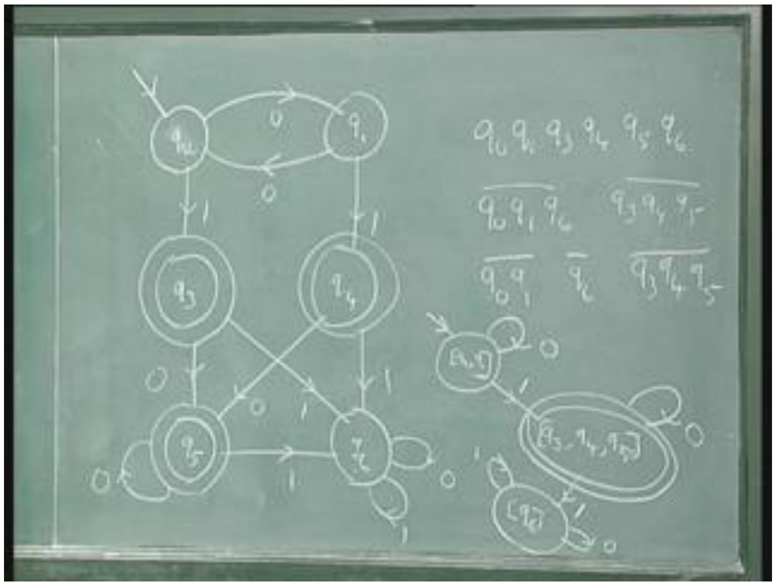
For example; take all of them in one partition q_0 q_1 q_3 q4 q_5 q_6 take all of them. Now, if you take epsilon as the distinguishing sequence starting from here if you take epsilon you will go to a non final state, starting from here if you take epsilon you go to a final state. So first partition into final and non final states q_0 q_1 q_6 this is one partition and q_3 q_4 q_5 and to partition this you have made use of the empty string as a distinguishing sequence. So non final states form one block and final states form one block. The possibility is these may be equivalent and these three may be equivalent, you have to test further whether they are equivalent.

Now consider q_0 q_1 q_6 what are the 0 successors for q_0 q_1 and q_6? The 0 successors of q_0 q_1 and q_6 are q_0 q_1 and q_6 that does not make any difference. What are the 1 successors q_0 q_1 q_6? The 1 successor of q_0 and q_1 is q_3 and q_4 but the 1 successor of q_6 is q_6. So, 1 takes these two to the final states whereas one takes this to a non final state. So you can split it like this q_0 q_1 and q_6 and 1 is the distinguishing sequence.

Now look at q_3 q_4 q_5 what are the 0 successor? The 0 successors of q_3 q_4 q_5 are q_5 only. The 1 successors of q_3 q_4 q_5 are q_6. So further split here is not possible, the 0 successors of all the three is a single state, the 1 successors are also a single state so you cannot split it further. Now we have to test whether you can split this one q_0 q_1. Look at q_0 q_1 what are the 0 successors? The 0 successors are q_0 q_1 they are in the same block.

What are the 1 successors of q_0 q_1?
They are q_3 and q_4 and again they are in the same block. So, further split is not possible here. So the minimum state automaton will have this. You can merge q_0 q_1 and you can merge q_3 q_4 q_5 then q_5 alone will be separate. So this is the final state q_3 q_4 q_5 this is the final state this is the initial state. If you read 1 1 you go here if you read two 1s you go here when you read 0 you are here, when you read a 0 you are here when you read a 0 or a 1 you remain here. So you can see that any string having a number of 0s with 1 1 followed by 0s will be accepted. If you have no 1s or if you have two 1s or more 1s it will go to q_6 which is a non final state and such a string will not be accepted. So this is the minimum state automaton for this deterministic automaton.

[Refer Slide Time: 50:41]



So we have considered a non-deterministic FSA. We have also seen that non-deterministic FSA are equivalent to DFSA and they accept only what are known as regular sets. And we have also seen how to construct the minimum state automaton this also we have seen. And we have also seen what equivalent states are, what is a distinguishing sequence and things like that.

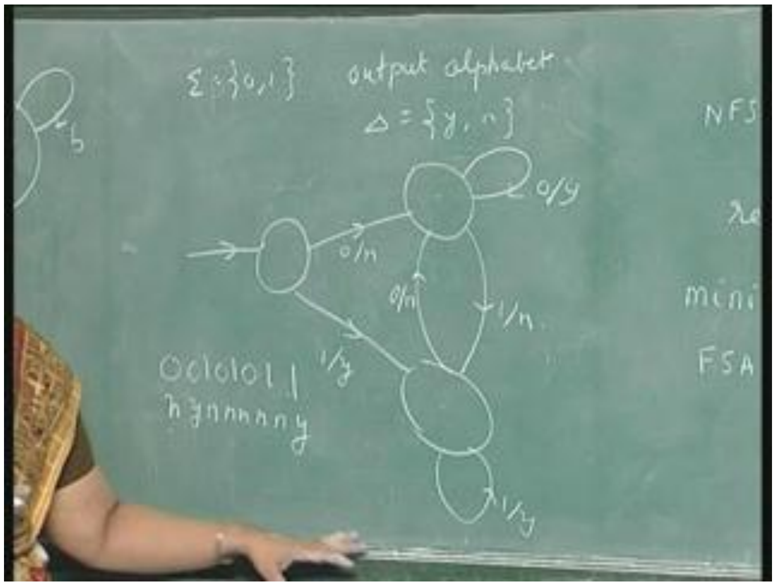
In the last lecture we have considered FSA with output. For example, you can consider this one with an input alphabet sigma is equal to 0 1 and output alphabet delta is equal to yes or no. The string will be accepted means you say yes and if it is not accepted you say no and any string ending with two 0s will be accepted and any string ending with two 1s will be accepted otherwise it will not be accepted. You can have an automaton like this, look at this automaton and consider the sings strings 0 0 1 0 1 0 1 1 and the output will be 0 no 0 0 yes then again 1 no no no no no yes the output sequence will be like this. Whenever you have two 0s it will output a yes when you have two 1s it outputs a yes.

Now you are able to achieve this with just three states having an output mentioned any string ending with two 0s or two 1s you can have an output sequence which outputs a yes when the string ends with a two 0 or two ones otherwise it outputs a no.

We have also considered the same thing as an accepting device and we had a diagram earlier which had five states. Actually we constructed a non-deterministic automaton and from the non-deterministic automaton we constructed the deterministic automaton which also had five states. If we use the minimization procedure by finding equivalent states and all that you will realize that you cannot reduce the number of states you will require five states for that if you want to look at it as an acceptance device whereas making use of outputs we are able to reduce the number of states to three.

Here again there are two types of machines called Moore machines and Mailey machines this is called a Mailey machine.

[Refer Slide Time: 55:05]



We have seen about the final state automaton which has got lot of applications in the lexical part of the compiler and also in text editing. There are other things like weighted finite state automata, distributed finite state automata and integer weighted finite state automata.

These concepts are very useful in image compression and image representation. There are so many results about finite state automaton. What we have seen is only a brief, glimpse of what is a finite state automaton, how non-deterministic automaton and deterministic automaton are equivalent. The language accepted is a regular set and it can also be represented by what is known as a regular machine. So this is just a brief introduction about finite state automaton.