Theory of Computation

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

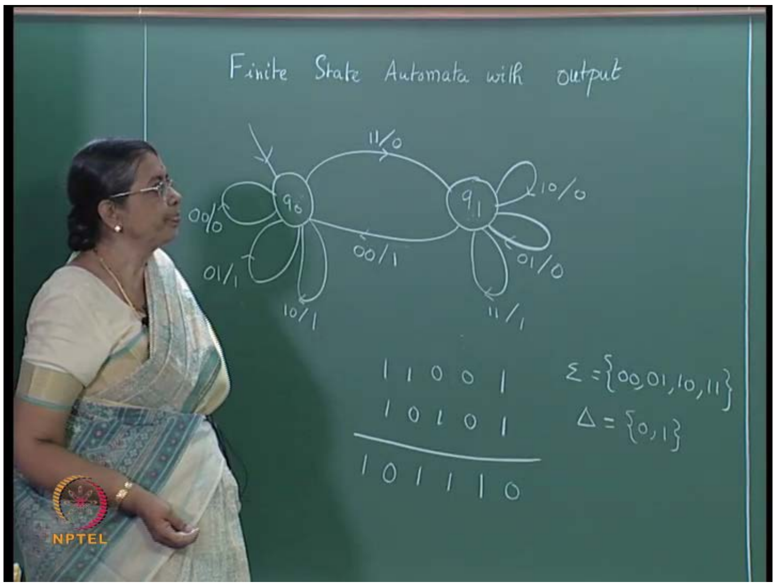
Lecture No. # 19

FSA with Output Moore and Mealy Machines

So, today we shall consider finite state automata with output. Actually, we have considered this in the first class. In the first class, we considered the example of a serial adder. This was the state diagram.

(No audio 00:25 to 01:21)

(Refer Slide Time: 00:25)



This is the state diagram of a serial adder and the inputs are two binary digits 0 0 0 1 1 0 or 1 1. So, suppose, you want to add two numbers say 1 1 0 0 1, 1 0 1 0 1. What will be the output? 1 1 gives you 0 with the carry and the carry this two 0s you get a 1, 0 1 will give you a 1 without a carry, 1 0 will give you a 1 without a carry, 1 1 will give you a 0; there will be a carry which will be output in the next instance.

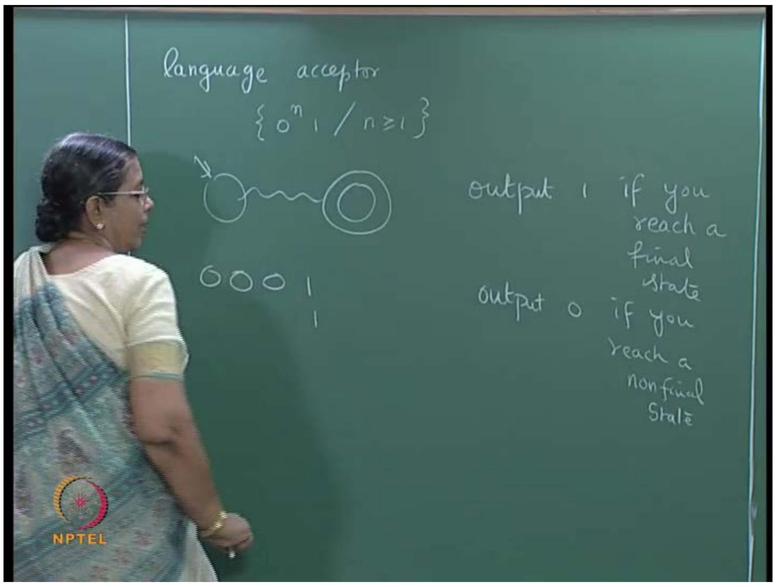
So, you can see that this is the initial state with which you start. We do not mark any final state when we consider final state automata with output. The first input is a 1 1.

This corresponds to a no carry state and this corresponds to a carry state. So, when you get a 1 1 the output is a 0, but you go to the state q 1, which is the carry state - that means there is a 1 here. And the next instance when you get a 0 0, you go to the no carry state but, you output a 1. In this state, a no carry state, when you get a 0 1, you output a 1 there is no carry and we get a 1 0, you output a 1 and there is no carry.

Now, in the no carry state when you again get a 1 1 you output a 0 and go to the carry state and in this state you are not getting any input. That is you have to consider it as 0 0. So, next instance, the last instance 1 will be output. Here, the input symbols are parts of digits. So, you can look at them as 0 0 0 1 1 0 1 1. The output symbols which we denote by delta or either a 0 or a 1. Inputs are parts of digits 0 or 1 0 0 0 1 1 0 1 1, output alphabet is 0 and 1.

So, this automaton works in such a way that when it receives some input, it outputs something. You can very easily see that this output depends on the state and the input symbol. So, in this example, the output depends on both the state and the input symbol. Sometimes, the output may depend just on the state alone. So, that we call as a Moore machine and this type of a machine we call as a mealy machine and let us define it formally. We also consider in the first class, a 1 1 moment delay machine and parity checker those machines also had output. So, we can look at the final state automaton as an input output device. And generally when you look at it as a language recognition device, the input is accepted or rejected.

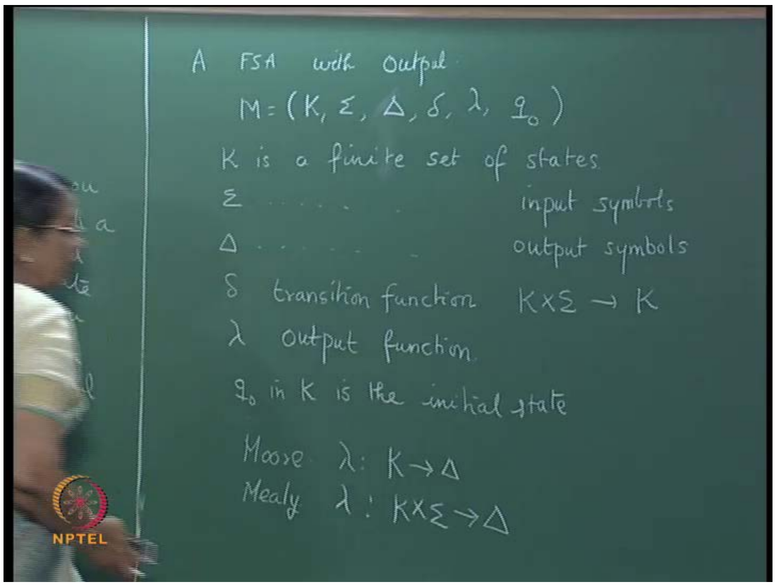
(Refer Slide Time: 05:09)


language acceptor
$\{0^n 1 / n \geq 1\}$
0 0 0 1
1
output 1 if you reach a final state
output 0 if you reach a non final state
NPTEL

Language acceptor when you look at it as a language acceptor for example, it can accept something like 0 power n 1; n greater than or equal to 1, a language like this. In this case, starting from the initial state, after reading some input, if you reach a final state, the input is set to be accepted. You can have a machine, where the output will be 1, if you reach a final state; output is 0 if you reach a non final state. If you look at it this way, even the acceptor can be looked at as an input output device.

It outputs a 1 when you when the string is accepted for example, here if you have a string like this. It will be outputting a 1 here so that the string is accepted. If it is some other string like 0 0 1 0 something like that, it should not be accepted. So, at the end it will put a 0. In fact, at every state, it will output something, which will tell you that portion of that string is accepted or not.

(Refer Slide Time: 07:01)



A FSA with output

$M = (K, \Sigma, \Delta, \delta, \lambda, q_0)$

K is a finite set of states

Σ . . . . . input symbols

Δ . . . . . output symbols

δ transition function $K \times \Sigma \to K$

λ output function

q_0 in K is the initial state

Moore $\lambda : K \to \Delta$

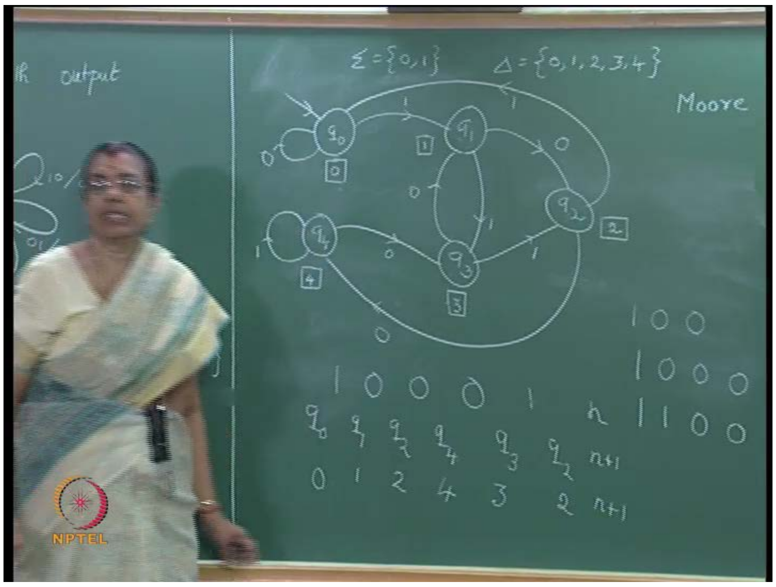
Mealy $\lambda : K \times \Sigma \to \Delta$

Now, let us go to the formal definition of final state automata with output. So, A FSA with output defined as M is equal to K sigma delta delta lambda q naught (( )) like this where K is the finite set of states, sigma is the finite set of input symbols, delta is a finite set of output symbols, delta is the state transition function. It is called the transition function and it maps K into sigma into K. We are considering deterministic machine here, lambda is the output function. Let us see what it is in a moment. We do not specify any final state because we are bothered about the output emitted at each instance of time. So, we do not consider a final state.

q naught is the initial state; q naught in K is the initial state. Now, what about the output function? The output function can depend just on the state alone or on the state and the input symbol. So, in one case when the machine is a Moore machine, output function is K to delta and in the case of a mealy machine, the output function is, it depends both on the state and the input symbol. It is from K into K cross sigma into delta.

Let us consider one example of a Moore machine and one example of a mealy machine. It is not really very different in the sense that from a Moore machine we can go to a mealy machine and from a mealy machine we can go to Moore machine. We shall see that conversion also in due course.

(Refer Slide Time: 10:38)



Let us consider one finite state automata with input alphabet 0 1, output alphabet 0 1 2 3 4, I will draw the state diagram. This is a Moore machine; q naught, q 1, q 2, q 3, q 4. First, let me draw the transition functions 0 1. Since, the input alphabet is 0 and we are considering a deterministic machine from every node, there should be an arc with label 0 leaving and there should be an arc with label 1 leaving. So, from q 1 when you get a 0, you go here. From q 1 when you get a 1, you go here.

From q 2 when you get a 0, you go here, when you get a 1, you go here. From q 3, when you get a 0, you go here, when you get a 1, you go here. From q 4, how do you reach? 1 1 0 0 if you get a 0, you go here, if you get a 1, you remain in 1. Now, I have drawn only the transition function, outputs I have not drawn. The output depends on the state. So, this is a Moore machine. We are considering a Moore machine, so, when the machine is in this state, it outputs a 0. I am drawing the output within the square.

When it is here, it outputs a 1, so, the output for this state is 1, the output for this state is 2, the output for this state is 3, and the output for this state is 4. You see the output alphabet is 0 1 2 3 4. Now, what is the output signify? You just take some string and check, I will take 1 0 0. What is this number in binary or what does this represents? What number does this binary representation represent? That is 4 right; so, 1 0 0 that is you reach state q 4 and output of 4. What does that mean?

When divided by 5, the remainder is 4. Suppose, I take the number 1 0 0 0, when divided by 5, the reminder is 3. Let us see, whether you output a 3. 1 0 0 0 you output a 3, take some other number 12. What is the representation for 12? 1 1 0 0; see what happens. 1 1 0 0, you reach state q 2 and output a 2. 12 divided by 5 gives you a reminder 2. So, the here output really represents, the reminder when divided by 5. You take the input as a binary string, it represents a number.

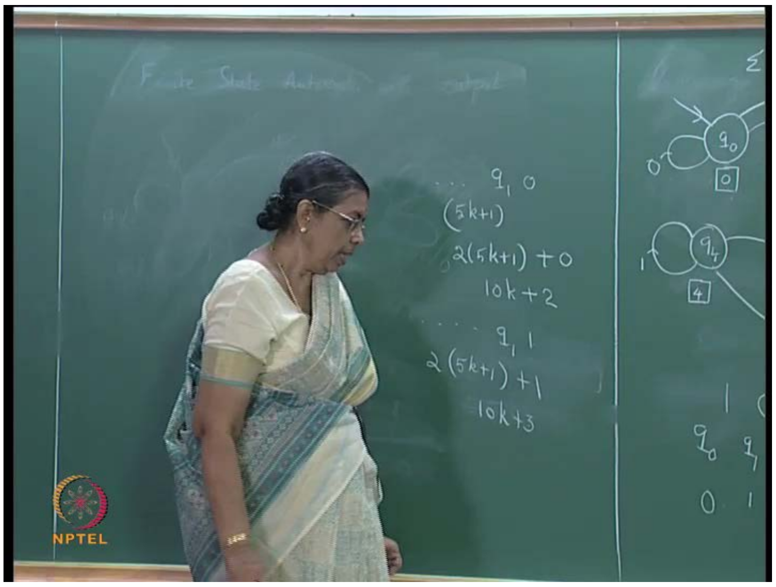
When that number is divided by 5, what is the reminder that is given as the output? Leading 0s are allowed, so I am also allowing numbers like 0 0 1 0 0. This sort of a string is also allowed so, I should forget about the leading 0s. You must remember that at each instance there is an output for example, if I take 8, initially, it starts in straight q naught. After reading a 1, it goes to q 1. After reading another 0, it goes to q 2. Reading one more 0, it goes to q 4. From q 4, after reading a 1, it goes to q 3. So, this is the state sequence.

What will be the output? The output initially will be 0, q naught 0, q 1 output say a 1, q 2 output say 2, q 4 output say 4 and q 3 outputs a 3. So, the output sequence is this 0 1 2 4 3. We can see that, when nothing is there, the output is 0. After you read 1; number 1, when divided by 5 gives you the reminder 1. Then, after reading this, the number is 2. 2 when divided by 5, gives you the reminder 2. After getting one more 0, it is number 4, the remainder is 4. After getting one more 0, it is 8 and the remainder is 3.

Suppose I was to get one more 1 here, what would be the number? 8 into 2 plus 1; 17, q 3 1 is q 2, output will be 2. So, 17 divided by 5, will give you the reminder 2. One thing you must note here is the input. If it is of length 5, the state sequence will be of length 6, if the input is of length n, the state sequence will be of length n plus 1 and output also will be of length n plus 1. This is the case of a Moore machine.

So, this is the way you define a Moore machine. I will keep this diagram and let us consider an example of a mealy machine. This is an example of a mealy machine of course; serial adder is a mealy machine. And before going into that, let me justify this diagram. How are you justified in drawing a diagram? For example, from q 1, I am drawing an arc with label 0 to q 2 and from q 1; I am drawing an arc with label 1 to q 3. How am I justified in that?

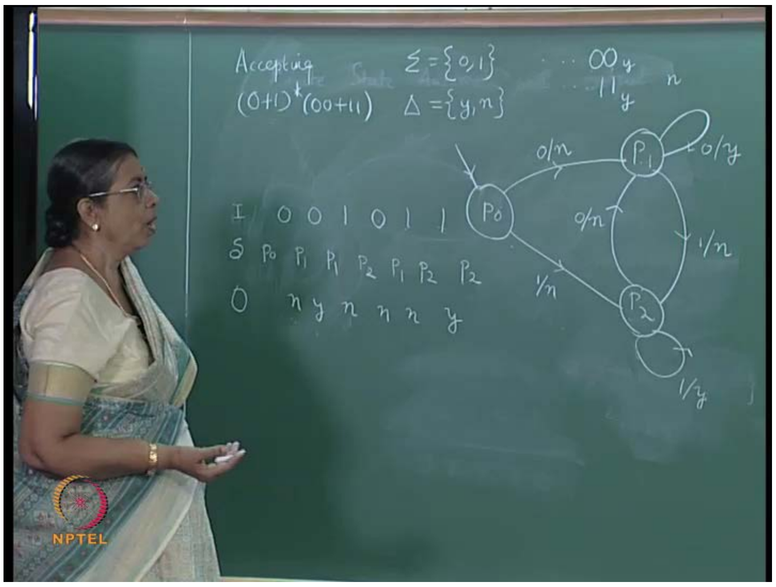
(Refer Slide Time: 18:55)



See when you reach q 1, you read a binary string which leads a reminder 1 when divided by 5. So, the number will be of the form say 5 k plus 1. Now, if we get one more 0, the number will be two times 5 k plus 1 plus 0, which will be 10 k plus 2. So, the number when divided by 5 will give you a remainder 2. That is why, from q 1 there is an arc with label 0 going into q 2. On the other hand, suppose after reading something, I am in state q 1 and I get the input 1. I go to q 3.

How do I get this? This represents a number of the form 5 k plus 1. When I get one more 1, the number represented will be two times 5 k plus 1 plus 1 which is 10 k plus 3 and when divided by 5, it will leave a remainder 3 and that is why, this arc is drawn in this manner. Like that for each one of them you can justify why that arc with label 0 goes here, arc with label 1 goes here. For each state, by writing an expression in this form we can see that you are justified in drawing this state diagram.

(Refer Slide Time: 20:41)



Now, let me go to an example of a mealy machine. You consider strings over the alphabet 0 1 and if the string read so far, at the end you have two consecutive 0s or two consecutive 1s, you output a yes; otherwise you output a no. The last two symbols read if they are 0 0 or 1 1, you output a yes; y symbol. If they are not, you output a n symbol.

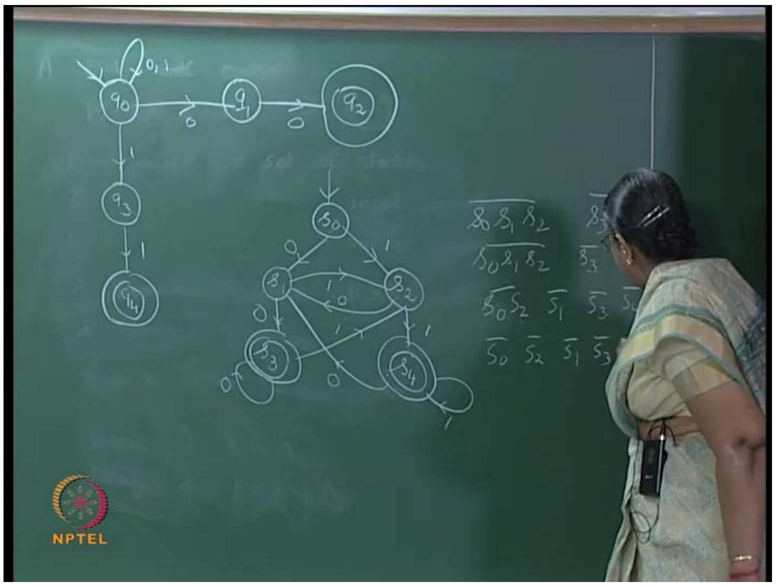
So, what is the output alphabet? The output alphabet is yes or no. So, the state diagram for that will be of this form; p naught p 1 p 2. When you get a 0 output, a no, when I get a 1, again I output a no. One more 0, the last two symbols will be 0 0, so, I output a yes. From here, if I get a 1, the last two symbols are 1 and 1, so, yes. Here, if you read any number of 1s, the last two symbols will be 1 1 1 1 etcetera. So, you will have a yes. But, in this place, if I get a 0, again I have to start outputting a no and if I get a 1 in this place, the output will be a no.

So, this is an example of a mealy machine. Note that the no or yes depends both on the state and the input symbol. In state p naught, if you get a 0, you output a no. In state p naught if you get a 1, you output a no. In state p 1, if you get a 0, you output a y. In state p 1, if we get a 1, you output a no. What will be the some? I will take some string. Suppose, the input is 0 0 1 0 1 1, what will be the state sequence? It will be p naught p naught 0 is p 1, p 1 0 is p 1, p 1 1 will be p 2, p 2 0 will be p 1, p 1 1 will be p 2, p 2 1 will be p 2. This is the state sequence. What would be the output sequence? When you get a p naught 0, you output a no, then p 1 0 you output a yes.

So, when the last two symbols are 0 and 0, you output a yes. Then p 1 1 will be a no, p 2 0 will be a no. The last two symbols are not the same, then p 1 1 will be no, p 2 1 will be yes. Last two symbols are 1 and 1 so, you will get a yes. So, this is the way the diagram is drawn. You can also look at it in a slightly different way. Consider a final state automaton, accepting strings corresponding to regular expression, 0 0 plus 1 1. That is this final state automaton will accept strings if they end with two 0s or with two 1s. Initially, you can have any string so, this is the regular expression for a set of strings which end with either two 0s or two 1.

Suppose, I take this and construct a final state automaton as an acceptor what diagram will I get? Let me draw the diagram. (No audio from 25:39 to 25:52).

(Refer Slide Time: 25:54)



A non deterministic diagram will be like this. Some q naught, q 1, q 2, 0 1 any string of 0s and 1 can be read and it has to end with the 0 0; this is a final state. Or it has to end with a 1 1. So, this is a non deterministic diagram which will accepts strings which end with 0 0 or 1 1. I can use the subset construction and construct the deterministic automaton. A deterministic automaton for that will be like this. I will use s naught s 1 etcetera as state labels, s naught s 1 s 2 s 3 s 4; s 3 and s 4 are final states, s naught is the initial state.

If I get two 0s, it should be accepted. If I get two 1s it should be accepted but, after getting a 0, if I get a 1, I should go here 0 1 1 will accept but, 0 1 cannot be accepted. s 2 and s 1 are non final states. In s 2, if I get a 0, I go here. After getting 0 0 one more 0 I get still, it is a final, it will be accepted. If I get one more 1 here still, it will be accepted, but after getting two 0s or many 0s, if I get a 1, I have go to here.

If I get a 0, I have to go here. Please note that this is a deterministic diagram and how many states we have? We have 5 states. The non deterministic one also we had 5 states, the corresponding deterministic automaton I have not used the subset construction. You can use the subset construction and check; you will end up with 5 states.

Now, is this the minimum state automaton? We know, given a deterministic final state automaton how to construct the minimum state automata. So, is this the minimum state automaton or do we have another automaton which has less number of states? Let us use the minimization procedure and check. Now, first you divide the set of states into final and non final. So, two blocks you have. s naught s 1 s 2 in one block, s 3 and s 4 final states in another block.

Now, in s 3, if I get a 0, I go to this but, from s 4, if I get a 0, I go to another block. So, from s 3, if I get a 0, I go to this block. From s 4, if I get a 0, I go to another block. So, this has to be split in the next step. So, you will get s naught s 1 s 2, s 3 separate, s 4 separate. Now, look at this. What are the 0 successors of s naught s 1 s 2? They are s 1 s naught s 1 s 2 0 successes, they are s 1 s 3 and s 1. s 3 is different, s 1 is in different block and s 1 has a 0 successor which goes to s 3. So, in the next step s naught s 2 will be separated from s 1 and s 3, s 4 are separate.

Now, look at s naught and s 2 - what are the one successes? The 1 successor of s naught is s 2 1, successor of s 2 is s 4. They are in different blocks. So, this again will be split and you will get like this. So, this is the minimum state automaton, you cannot have an automaton with less number of states. A deterministic automaton with less number of states, but, look at the same problem as a mealy machine where the output is yes or no. In fact, in apart from yes or no, you could even say 0 or 1 does not matter yes; 1 for yes, 0 for no.

To make a string accepted or come out with an output yes or if it is not of the form ending with two 0s or two 1s with an output no, you require a machine which has only three states. But, we saw that as an acceptor, when you want to accept strings, ending with two 0s or ending with two 1s, you require 5 states. This is the advantage of having an automaton with output.

So, we have considered both the types Moore machine and mealy machine. Consider one example next; we shall see that they are equivalent. In fact, for a Moore machine, we can construct an equivalent mealy machine and for a mealy machine you can construct an equivalent Moore machine. Can you say a Moore machine is equivalent to a mealy machine? First of all, you must remember that if the input is of size n, in the case of a Moore machine, the output will be of length n plus 1. In the case of a mealy machine, the output will be of length n.

So, there is no point in talking about the equivalence because the length itself is different, but, you must remember that at the initial state, there is an output in the case of a Moore machine. If we ignore that input, if we ignore that output, suppose, I have this as the input and the case of a Moore machine, it will be b 1 b 2 b b n and in the case of a mealy machine, it will be some say c 1 c 2 c n.

If I ignore the first output, then can I say this can be equivalent to this if you constructed in a proper manner and this can be equivalent to this if you construct in a proper manner? That is what we want to see. So, given a Moore machine how to construct an equivalent mealy machine and given a mealy machine how to construct an equivalent Moore machine and ignoring the first output in the case of a Moore machine.

So, you are given a Moore machine k sigma delta delta lambda q naught how will you construct an equivalent mealy machine? The corresponding mealy machine you have the same set of states, same set of input symbols, same set of transition, output symbols, same transition but, the output function will be different. Here, it depends on the state alone; here it depends on both the state and the input symbol. The same initial state you have but, lambda dash how do you define? Lambda dash is a function of q and the input symbols the state and the input symbol.

So, how do you define lambda dash q a? It is defined as lambda of delta q a; delta q a is a state in state q. If you read a symbol, a you go to a particular state, say q dash, the output corresponding to that state is the output for the pair q a. If we define like that, we can get the same machine. So, in this example, this is a Moore machine. Let us change into a mealy machine. How do you change? The outputs will now be defined like this; input, output. So, these outputs we will not mark so, when you go to q naught you are outputting a 0. So, from here, when you go to q naught, you output a 0.

Similarly, from here you go to q naught, so you output a 0. When you go to q 1, you output a 1. So, when you go here, you output a 1. When you go to this state, output a 1. When you go to q 2, you output a 2. Here, you output a 2. When you go to q 3, you output a 3; here 3. When you go to q 4, you output this is i will write it as 1.

So, suppose, the earlier Moore machine, if I had say (( )) the state sequence would have been q naught q 1 q 2 q 4 q 4 and output will be 0 1 2 4 4. In the case of a Moore machine, the output is of length 5. In this case, we have constructed the mealy machine corresponding to that using this definition for the output function. Let us see what is the output? 1. First output we do not have in the case of a mealy machine. 1 1 0 this is 2, then 2 0 output is 4, then q 4 1 output is again 1. You get the same output ignoring the first output in the case of a Moore machine.

So, you can see that given a Moore machine, you can very easily convert that into a mealy machine. Similarly, we can also convert a mealy machine into a Moore machine but, it is slightly more involved than this. Let me consider this example. So, let us start with the mealy machine now. The mealy machine is given by k sigma delta delta lambda q naught then the corresponding Moore machine now, the states are not just k. It is a cross product of k cross delta and you have to the input alphabet is the same, the output alphabet is also the same and you have to define delta dash lambda dash.

The initial state will be again a pair. The first component of the pair will be a state from k. Second component will be a symbol from delta. This b can be any arbitrary symbol. How do we define delta dash? Delta dash you have to have some pair q b a, is not it? The state is now a pair and a symbol read is a symbol. This will be a pair again, the first pair will be delta of q a and the second pair will be lambda of q a. The lambda dash now, depends on the state alone. If you have a pair q b, the output will depend only on this pair and it is given by the second component of the state b.

With this definition let us consider the Moore sorry Moore machine corresponding to this. There are three states p naught p 1 p 2 and two symbols yes and no. So, there will be six states here; p naught no, p naught yes, p 1 no, p 1 yes, p 2 no, p 2 yes. Either of these states, I can take as a input. Initial state I am sorry either of them I can take as the initial state. The transitions are drawn in this manner p naught. Look at the definition delta dash q comma b, if this is the input that will be delta of q a comma lambda of q a.

So, in this case from p naught what state you go to p 1 and what is the output no. So p 1 n you must put. So from this or this when you get a 0, you go here. From p naught, when you get a 1, you go to p 2 and the output is no. So, when you get a 1, you must go here and here also you must go here when you get a 1. From p 1, when you get a 0, you go to p 1 and yes. So, when you get a 0, it will be like this, it will be like this. When you get a 1, you go to p 2 and no so, when you get a 1, will like this. From p 2, when you get a 0, you go to p 1 and no. From p 2, when you get a 0, p 1 and no. From p 2, when you get a 1, you go to p 2 and yes.

So, 1 1 so, this will be the corresponding state diagram of the mealy machine. Now, let me check with one example for an input say, 0 1 1 say, what is the state sequence in the case of a mealy machine? This one it will be p naught. p naught 0 is p 1, p 1 1 is p 2, p 2 1 is p 2. This is the state sequence and what will be the output? Output will be no no yes. This will be the output sequence.

In this case, if you get a 0 1 1 starting from here, 0 1 1 and I have not marked output here. The output depends on the state and it is the second component so, the output will be n here, the output will be yes here. It will be n, I am drawing within a square and it will be yes, it will be yes here. It is a second component of the pair no.

So, what will be the state sequence in this case? I shall write the steady state sequence here. Initially, you start with p naught no. Please note that I could have taken this also as an initial state. Does not matter, either of them I could have taken as the initial state. p naught n 0 is p 1 n, p 1 n and what is p 1 n 1 is p 2 n; p 2 n and p 2 n 1 is p 2 y; p 2 y and output sequence is a second component so, it will be n n n y but, in the case of a Moore machine we have to ignore the first one.

So, the output is n n y, here also the output is n n y. So, you get the same output. So, given a Moore machine, you can construct an equivalent mealy machine and given a mealy machine, you can construct a equivalent Moore machine. Only thing is, in the case of a Moore machine, you have to ignore the first output symbol. So, they are really equivalent models and depending upon the purpose you have in mind, you either construct a Moore machine or you construct a mealy machine whatever you want.

So, this is the main thing about final state automata with output. Related the concept of final state automaton with output, we also have another concept GSM mapping. We study about closure properties of languages, regular languages or closed under intersection complementation, union etcetera similarly, context free languages. We also study something called sets or family of regular sets or family of context free languages. They are closed under GSM mapping.

This sort of a statement will come across in some places mainly, they are useful in finding the closure (( )) homomorphism inverse homomorphism etcetera. What is the G S M? It is called generalized sequential machine; generalized sequential machine. Here, you have states. There will be an initial state, there will also be final states and the transitions will be like i o. So, from this state q, you go to the state by a reading a symbol say a, the output will be a string.

You have an input alphabet sigma and an output alphabet delta a, will be a symbol from sigma, w will be a string from the delta. It can be epsilon also, it can be empty string also; w belongs to delta star. Not only that, what you are interested is when you start from the initial state and reach a final state, you reach some input. What is the corresponding output? For example, I can have something like this say, I have one state here, another state here.

When I read a, I output 2 a s then when I read a b, I output 1 d and when I read more b s, I output epsilon and this I take as the final state say, q naught q 1. So, I have this diagram. This is a sequence generalized sequential machine. Suppose, I have a string a a b b b. how can I read this? a a b b b so what will be the output for the first a a a, second a it will be a a, this b it will be b, for this b it will be epsilon, this b it will be epsilon. So, the output for this will be a power 4 b.

In general, when I have an input a power n, b power m, for each one of the a s, I will be outputting 2 a s and the first b, I will output a b, afterwards I will output an epsilon and so this is the output for this language. This is say n m a greater than or equal to 1. You can say so n greater than or equal to 1. If the input is this language, the output it transforms this language into this language.

A generalized sequential machine transforms one language into another and mainly we see that, if given a regular set, it will transform it into a regular set. Given a context free grammar, it will transform that into a context free language and so on. This again important in the case of closure properties and also weighted final state automata, which are useful for presenting digital pictures.