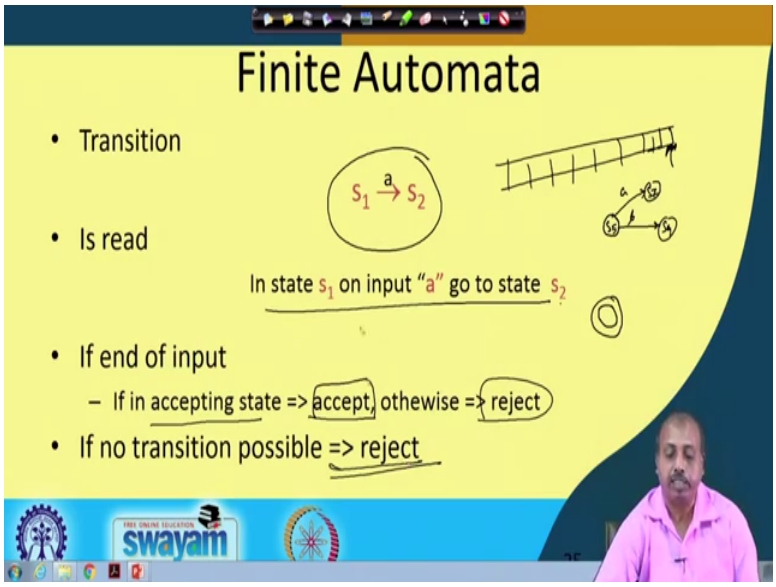
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 10
Lexical Analysis (Contd)

So, any transition in finite automata which is written as this S 1 on input symbol a going to S 2.

(Refer Slide Time: 00:21)



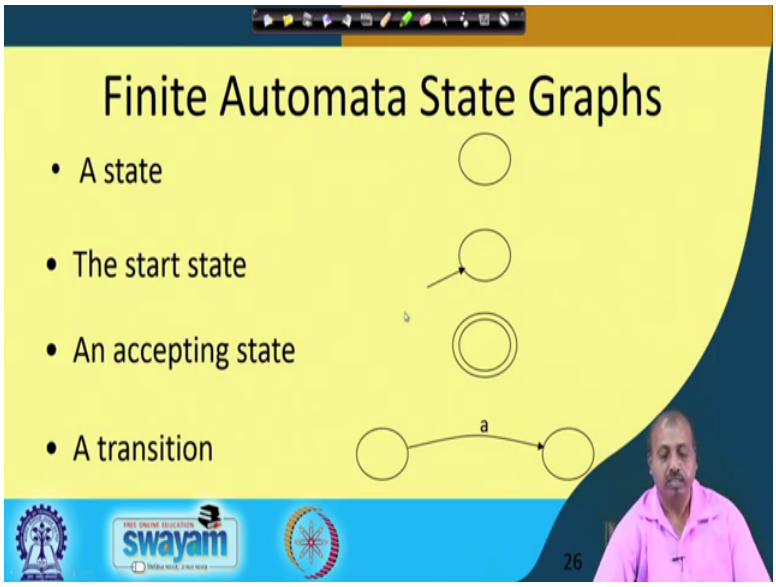
So, this is a; this is a Transition. So, this Transition is read as in state S 1 on input a go to state S 2. So, this is the meaning of this particular symbol S 1 arrow with a level a to S 2. So, this is read as this if you go if you find if you are at the end of the input. So, this was the whole input stream as I was telling. Now, if it is so, if you have reach the end of the input. So, this point no more input is there. Then, we see whether the state at which the automata is at that point of time is an accepting state or not.

So, if it is an accepting state, then the decision is accept; otherwise the decision is reject and the accepting states are the final states and they are identified by this particular notation; the states that will have 2 are concentric circles in it. So, they are the accepting states or final state. So, that is the that may be the; so, if you are at this type at any of this type of state in that case so, it is accepted.

However, if there is no transition possible at some for some state like may be at present, I am at state say S 5 and S 5 the transitions are defined only on the symbols a and b, to say this is to say S 7, this is to say S 9 like that. Now, so if the next input symbol coming does not match with either a or b, then from S 5 the system is unable to go anywhere the automata transition is possible and that and in that case, so it will be a reject set.

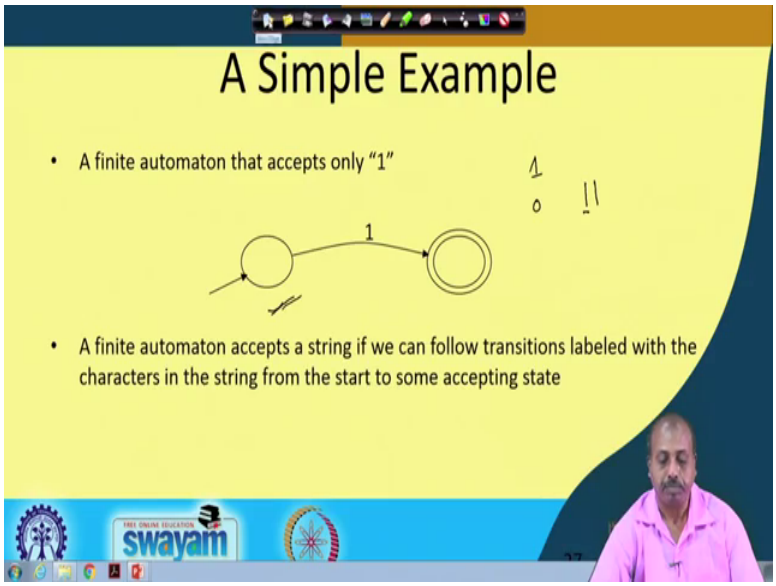
So, it will be it will go to a reject. So, the input stream is not accepted as per that regular definition. So, it may so happen that some other regular definition will match. So, conceptually you can think that as if all the regular definitions are being matched parallelly by the lexical analysis tool and whichever rule gives the maximum match so, that is returned as the token. At the final state of it the token the match token for that is returned to the pastern.

(Refer Slide Time: 02:41)



So, this is the way, this operates the finite automata things. So, these are some of the notations like a state will be represented by a circle or ellipse whatever you do it both are same. The start state will have an arrow pointing to it. So, that will identify a start state and accepting state will have two concentric circles or concentric ellipses and a transition will be level from one state to another with a corresponding level a ok. So, this is a so, these are the notations that are used for defining a state transition graph.

(Refer Slide Time: 03:15)

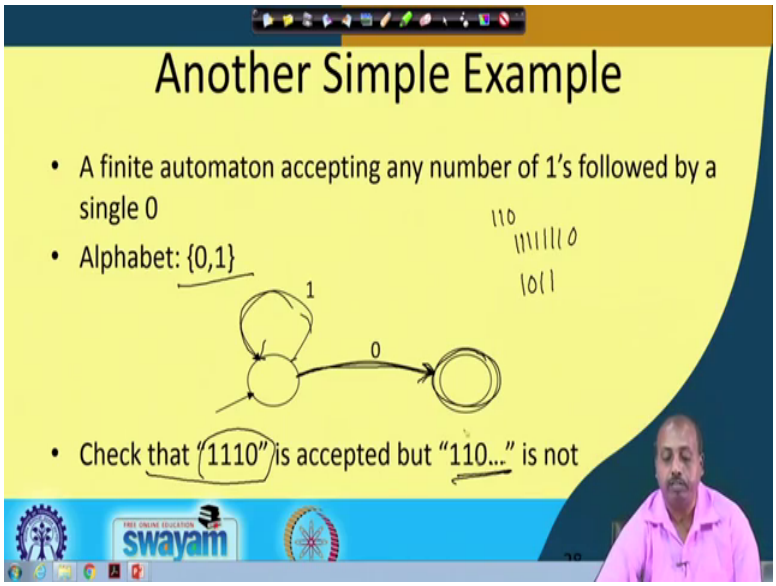


So, this is one simple example. A finite automata that accepts only 1. So, this is the start state so, we have got this is the start state and on getting the input one it comes to a final state. So, if you are giving as an input if you give a single one, then it will come here. But, if you give anything else for example, if you give a 0, then from the start state there is no definition, no transition defined with the level 0. So, it cannot do a transition so, it is in it remains in this state only and that is a error.

Similarly, if you are final state, then suppose I have given two 1's. So, on first one, it comes to this state. But, it does not know what to do on the second one as a result that is also an error. So, when the input is exhausted; so and you are if you are at an accepting state, then it is fine. If you are at some intermediary point; so, if you do not know where to go, transitions are not available for a for the current state so, in that case it is a reject.

So, a finite automat an accepts a string if we can follow transitions level with the characters in the string from the start to some accepting states. So, if you if it can lead you to some accepting state, then only it is the string is accepted.

(Refer Slide Time: 04:43)

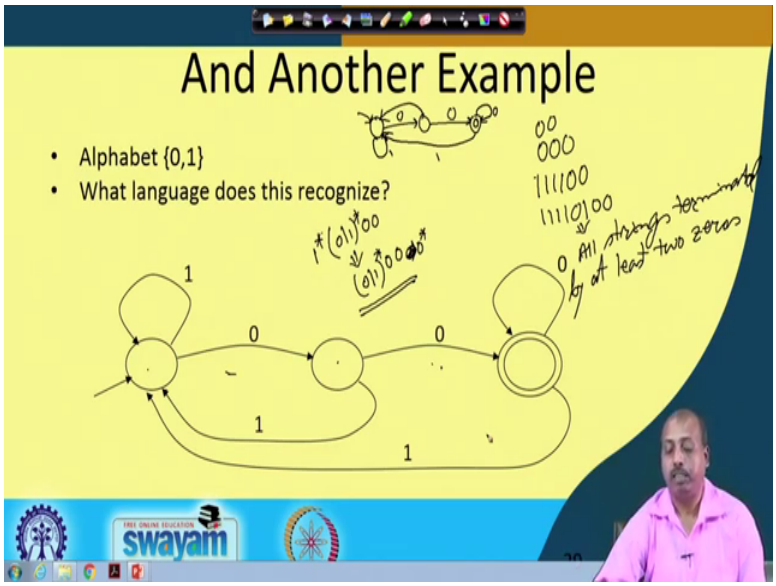


Some other example like a finite automata accepting any number of 1's followed by a 0, single 0. So, all these strings are acceptable; like I can have say 1 1 0; I can have say 1 1 0. Similarly, 1 1 1 1 any number of 1's followed by a 0.

But this is not accepted 1 0 1 1 is not accepted because it is not ending with a 0. So, in as far as this language is concerned, the alphabet set consists of the symbols 0 and 1. Now, the transition the finite automata is defined like this, that this is the start state. From the start state as long as it gets 1, it remains in the start state; if it gets a 0, it comes to a final state.

So, that way it is doing the translations. So, in this case; so, this 1 1 1 0 is a valid string, but 1 1 0 if there is any more character, so that is not a valid string for this language. Because, once it once on a 0, it comes here from these state no transition is defined or neither on 0 nor on 1. So, as a result the second the 2nd one is not a valid string for the language. So, this way we can use this lexical this finite automata for representing the regular expressions.

(Refer Slide Time: 06:07)



What about this particular example? So, you see that if the string as long as it as long as so you start at this point, as long as you get 1, you stay in the start state only. Now, if you get a 0, it comes to the state and then, if you get a 1, it immediately goes back to the state, the start state. So, it can reach final state only when there are two 0's ok. One 0, two 0 and then, it remains there as long as there are 0's. Whenever it gets a 1, it comes back to the first state.

So, some typical language that are accepted by this are say 0 0 0 or simply two 0's; 1st 0 2nd first 0 and 2nd 0, it comes to the state or 0 0 and on 0 it remains here. So, 0 0 0 or it may be any number of 1's, then two 0's. So, that is also fine. So, wherever it gets a 1 1 1 1, then 0 then again 1. So, whenever it gets a 1, it comes back to the first state and to come to the end, the last state it must have at least two 0's at the end. So, to summarize I can say it is it is all strings terminated by; terminated by at least two 0's.

So, this particular finite automata, it corresponds to all the strings that data terminated by at least two 0's 2 then, because whenever it is getting a 1, it is coming back to the initial state. So, from that point it is again starting. So, then it can you can see that it will not be able to it will not be able to go to the accepting state until it gets two successive 0's at the end and there is no more character. So, that way it will do this thing.

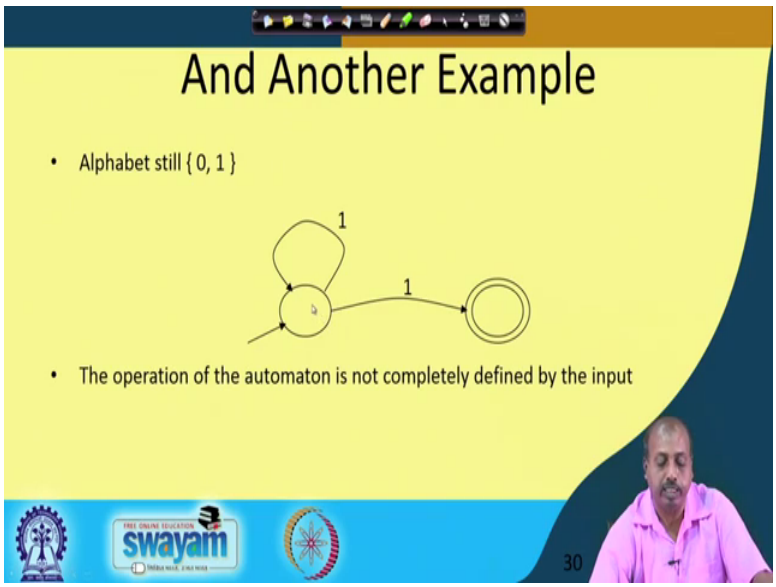
So, given a given such finite automata, so you can trace it and find out the corresponding language and you can try to write down the corresponding regular expression also. Like

here, so I have to write the regular expressions. So, it can be in the state 1 star, then it is any other string over 0 or 1 star any number of 0's and 1's, but then there has to be two 0's that the end ok. In fact, this 1 star is not necessary.

So, you can forget about this 1 star also. So, this can be simplified as 0 or 1 star followed by two 0's. So, if it gets two 0's it will come there, otherwise it will not. So, you can also try to from this regular expressions so, you can try to draw the automata. So, this it has to if it starts at a particular state, this is the initial state; then getting one 0, it comes to this state; getting another 0, it comes to this state.

I am trying to derive this automata from this regular expression. So, for two 0's, it come here. Now, there can be any number of 0's after this so, I should say so this is also not correct. So, there should be a 0 star at the end. So, as long as they are 0's, so it will remain there; but if it is similarly, as long as there are 1's, it will remain here. So, if it gets a 1, it will come back to this point. If it gets a 1, it will come back to this point. So, this way you can draw the corresponding. So, this is the derivations. So, ultimately you see that we have come to the same diagram as we have it here. So, from the regular definition, you can come to the finite automata or from the finite automata you can come back to the regular expression. So, both of them can be done.

(Refer Slide Time: 10:31)



What about this one? So, you see that alphabet is a still 0 and 1. So, it says that as long as you get 1, you remain this state and then, on getting a single 1, you come to this state.

So, the valid strings if we try to enumerate 1is a valid string, but say 1 1 is it a valid string? Like for the 1st one, it remains here; the 2nd one, it goes there.

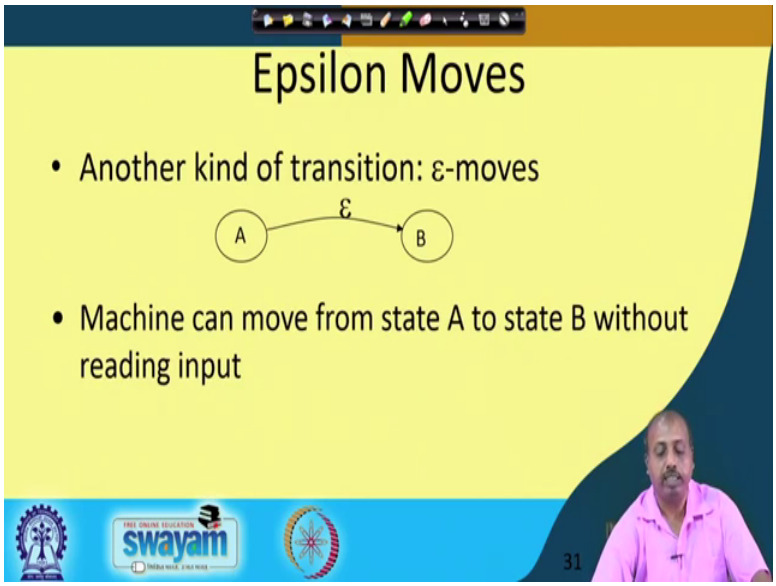
So, that is possible then so, then there were three 1's so, that is also possible because two 1's, it will revolved here and third 1, it will come here. Of course, there is a problem in this particular case as you can see that on 1; so I can go to this state or I can remain in this state. So, there are 2 transitions 2 possible transitions that are their corresponding to the symbol 1 at the start state. So, this is there is some sort of non determinism.

So, apparently it seems that this is not correct, but as such. So, we will see that this defines a new type of finite automata which is known as non deterministic finite automata, where there may be multiple transitions that are possible from one state based on the same input symbol and if any of those transitions can lead me to a final state final accepting state, then we are happy. So, we can do it do this thing.

So, we will come to that as you proceed. Now, the operation of this automata is not completely defined by the input. So, because we have to we have because we have to take some non deterministic decision like at this point on getting on whether we remain here or we go from here to here.

So, there is a decision and that decision is not depicted by the finite automata completely. Whereas, in the previous examples so, here there is no non determinism like at every point. So, if there are two transitions going out from a state. So, they are labeled differently. So, on 0 you go here or 1 you come here. So, this is deterministic, but here we have got non determinism. So, the operation is not completely defined by the input.

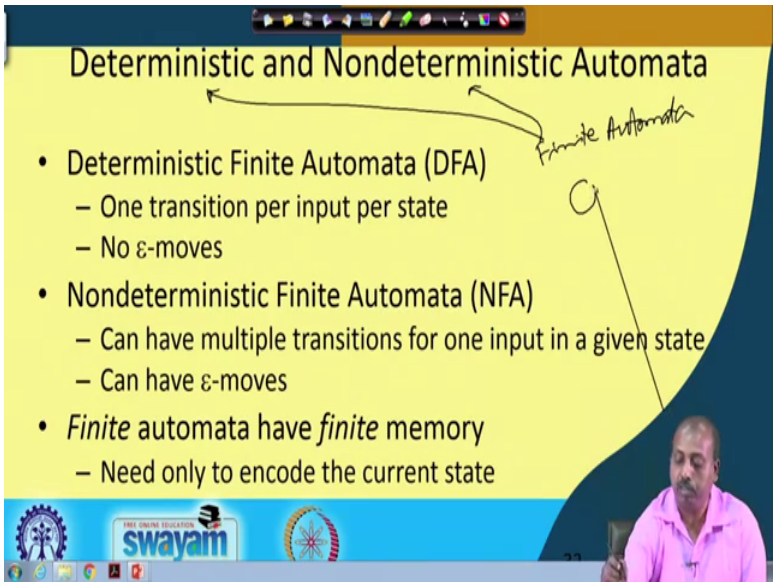
(Refer Slide Time: 12:51)



Another thing that we have is something called epsilon moves. Sometime from state A to state B, we may have a transition whose level is epsilon. So, epsilon means it will no input symbol will be consumed for doing a transition from A to B. The machine simply transits from A to B, but does not consume any input.

So, machine can move from state A to state B without reading the input. So, if we have these two things in our hand that is epsilon moves and this non determinism; so non deterministic decision. So, they will be constituting a different type of automata which is known as Non Deterministic Automata or NFA.

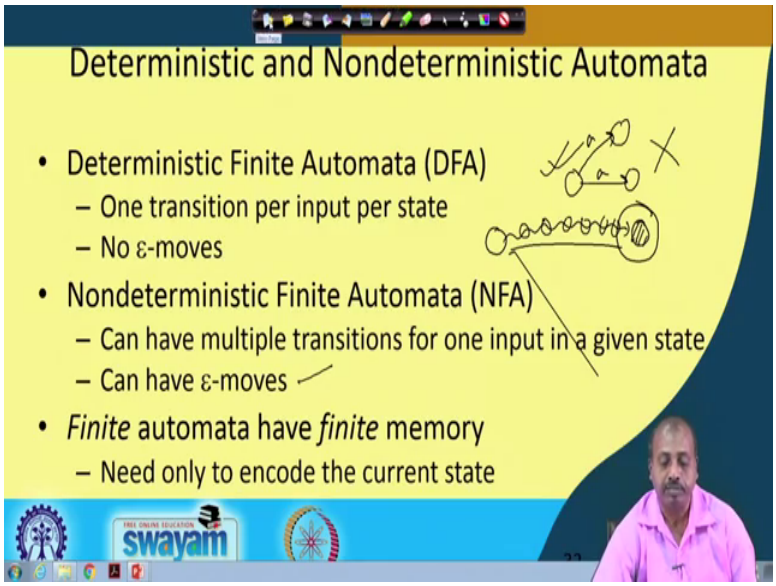
(Refer Slide Time: 13:31)



So, right now we can see that this finite automata, it can be divided into 2 classes. So, this finite automata; so, this is called finite because number of states that we have in it is finite in nature. Now, this finite automata there are two different types; one is deterministic, another is nondeterministic.

So, there are different types of finite automata. So, in a deterministic finite automata so that we have seen at the beginning; so, there is one transition per input per state. So, from every state on some input symbol, there is a single transition that is defined. So, it will not happen that from a particular state; from a particular state, I will have two different transitions and going to two different states and labeled with the same input symbols.

(Refer Slide Time: 14:21)

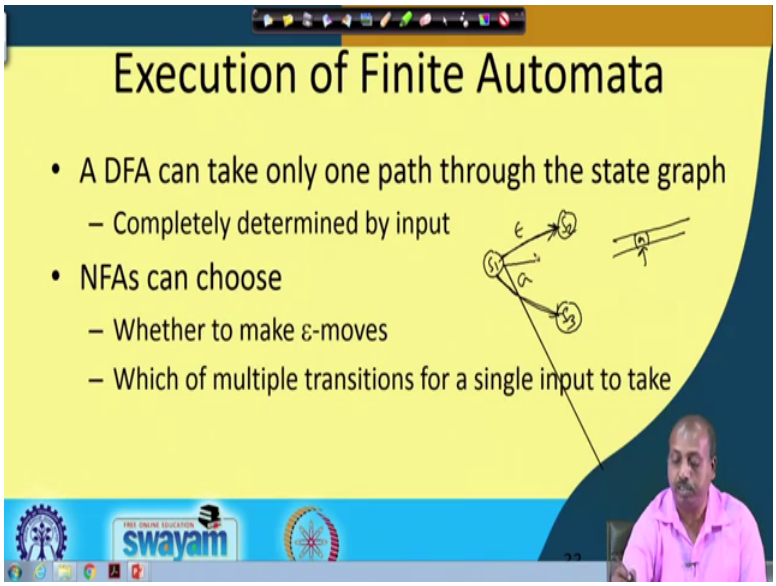


So, this will never happen for a finite deterministic finite automata and also there is no epsilon moves. So, there is no transition whose level is epsilon. On the other hand, this nondeterministic finite automata, so I can have multiple transition so, this is possible. So, from one particular state on getting the same input symbol, so it can go to two different states. So, while doing the operation, it will not deterministically select any of those two states and make a transition there.

So, you can have multiple transition for one input in a given state and also can have epsilon movements, epsilon moves can also be there. So, this defines the Nondeterministic Finite Automata or NFA. Finite automata have finite memory because it needs only to encode the current state. So, only the current state has to be remembered. So, it can forget what that how did it reach that particular state. Like if this is the start state and at present you are in this particular state.

So, you have come to this state where a number of state transitions. So, in between there are many states by which where input consumption we have come to this state. So, for the operation of the finite automata I do not need to remember this path; I do not need to remember this path, only remembering the current state is sufficient. So, that is the advantage that we have with finite automata. So, it can work with finite amount of memory and it needs to encode only the current state.

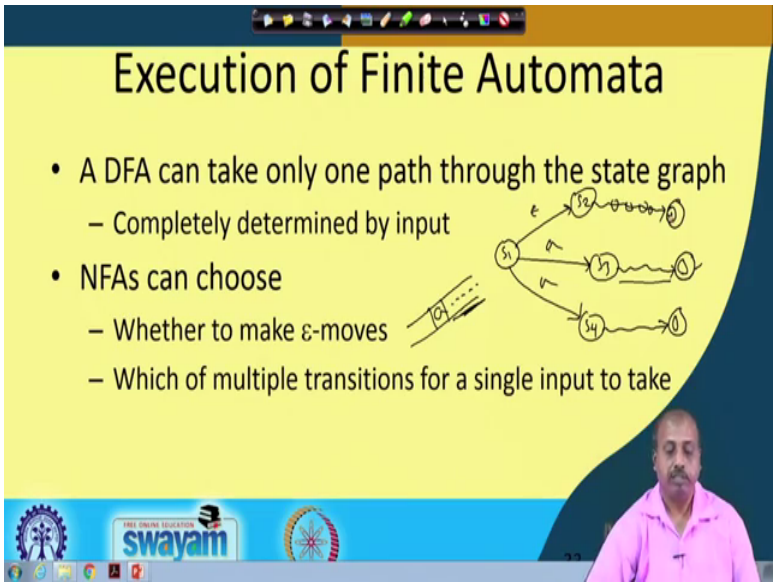
(Refer Slide Time: 16:01)



So, a deterministic finite automata, it executes like this it can take only one path through the state graph. So, it is completely determined by input. So, from 1 state, so there is no possibility that it can go to 2 different states on the same input symbols. So, it is operation is determined by the input. whereas, nondeterministic finite automata there is a choice like if there is a so, it may so happen that some epsilon movements are defined like from state say S 1, there is a transition to state S 2 which is marked as epsilon and there is another transition marked on input a to the state S 3.

Now, if the current input symbol is say a; your input pointer is here and the current input symbol is a. Now, whether you make this transition or you do not consume the input and go to the you follow this epsilon path? So, that decision is nondeterministic in nature. So, whether to take epsilon move or whether to consume the input and go to the next state? So, that is the decision question. So, so that is one problem whether to take epsilon move. Another point is if there are multiple transitions defined on a single input like it may so happen that from S state S 1. So, there is another transition defined.

(Refer Slide Time: 17:25)



So, from S 1 there is one epsilon move to state S 2. There is a transition on a to state S 3 and there is another transition on a to state S 4. Now, the situation is more complex because now on the input symbol a whether it should take epsilon transition or it should take the transition to state S 3 or S 4. So, that becomes non deterministic. So, this nondeterministic automata it can take some non deterministic decision and can and can take any of this transition.

Now, whether the anyhow whether this is valid or not. So, that depends like if there are some final states like this and then, suppose these are the final states and by taking this path may be it can eventually reach this one and by taking this path it can eventually reach here and by taking this path it can eventually reach there. Now, when you have taken this path whether you will reach this final state or not that depends on the transitions that you have on this path and the inputs that you have on your input stream after this.

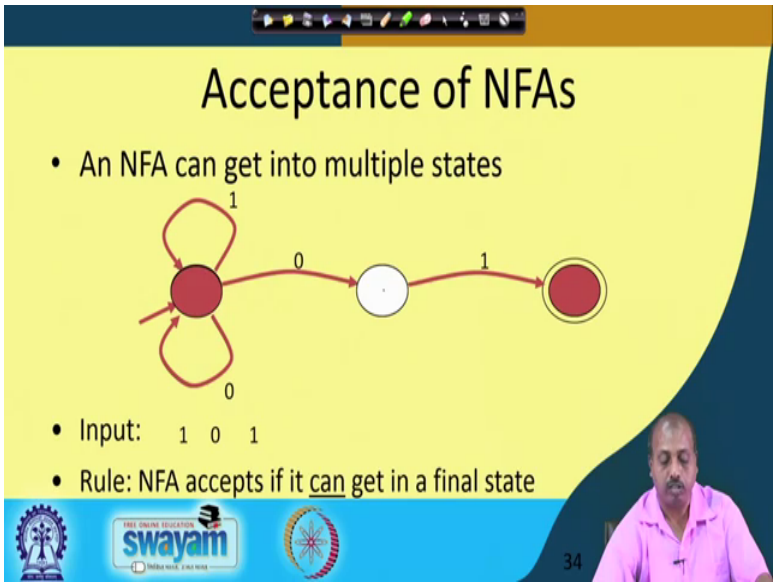
After a whatever input symbols are available. So, if the by that you can reach this one, then it is ok. Similarly, on getting this path, so if it is on a you consume it, then the remaining part if it can reached via this inputs by doing it in proper transitions, then that is also ok.

Now, it is designed in such a fashion that both of them are not possible. So, it cannot be that by taking epsilon path and this remaining input you can reach here as well as taking

a and using remaining inputs you can reach here. So, if that is not there if it is if the automata is design in such a fashion that only one of these final states can be reached by the next few input symbols, then you see that if you take a non deterministic decision which is wrong so, that will not lead you to a final state

And in that case you can come back, you can you can follow the other path to see whether that can lead you to a final state or not. So, that way we can take a decision about this which path to follow and by means of say backtracking and all and this way we can have a automata operations. So, apparently it sees that this is a bit clumsy. But we will see that the this is better in many cases because the total number of states in the system may be less compared to a deterministic finite automata. So, we will see that after sometime.

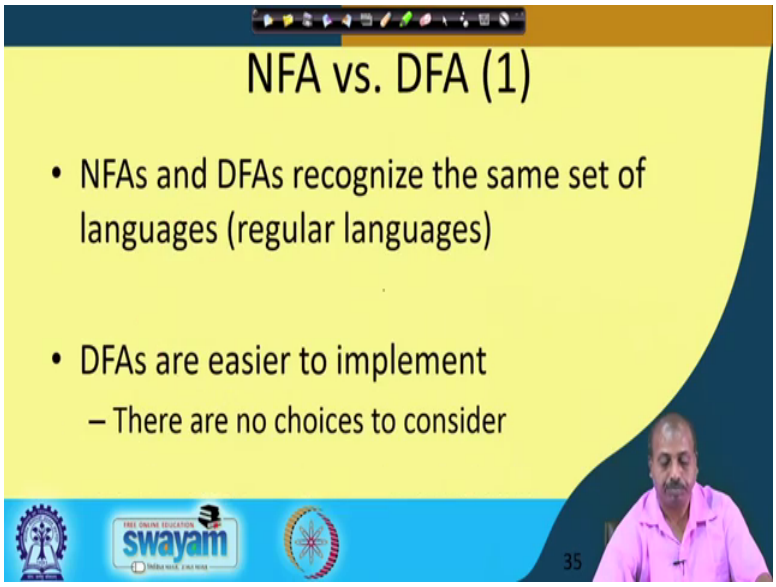
(Refer Slide Time: 20:03)



So, this is an example, like an NFA can get into multiple states like say for this is the start state as I was telling; once I getting 0, it can remain in this state or it can go here. However, on getting one it definitely remains on this state. But on getting 0, it can go to the next state or it can remain in the next state. So, it is in this state so, if the input is 1, then definitely it will make the transition like this. If the input is 0, we have a questions. So, it can either go here or it can go there. So, if it get a 1 here, now you see that from this choice it can go to the final state. But from this choice, it cannot go to the final state or getting 1. So, NFA will accept the string if it can get in a final state.

So, this 1 0 1 is acceptable in this particular language because it can find a path by which it can reach a final state. Of course, in between it write some other path, but that did not reach to the final state so, they are not considered.

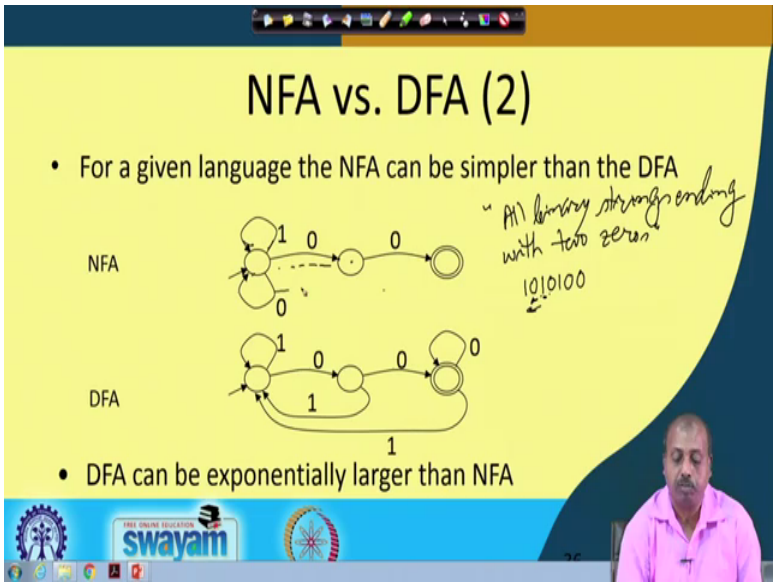
(Refer Slide Time: 21:15)



So, NFA versus DFA; so, apparently as I said that it appears the DFAs are better, but NFAs and DFAs can recognize the same set of languages. So, whatever whichever language is acceptable by an NFA it is also acceptable by a DFA and whichever language we can for whichever language you can construct a DFA, we can also construct an NFA. So, power wise both of them are equally powerful. So, both of them accepts can accepts same set of languages. However, as per as implementation is concerned DFA's are easier to implement because we do not have any choice to considered.

So, we have got only for every state we have got the transition which are well defined. So, there is no choice to be considered. So, DFAs are easier to be implemented so, that is there.

(Refer Slide Time: 22:11)



But, NFAs we have got choice so, we have got difficulty in the implementation because it may need back trekking. However, it may so happened that NFAs may be simpler compared to DFA. Like here I have got an; here I have got an NFA, where the language that we consider here is all strings that ends with two 0's. So, all binary strings ending with two 0's so, this is the for this particular case, we are trying to formulate the NFA and DFA.

Now you see this NFA as long as it there is 1 so, it remains in this state. If it gets a 0, there is a choice it can come to this state or it can remain here, but if it gets another so if it get two 0's, it will go there. So, if we if have a string like say 1 0 1 0 1 0 0, then getting the first one, so it will remain in this state because there is no choice. At the second 0, I have a choice. So, I can go like this or I can go like this. If I go like this and come to this state, then at the next time I get a 1, I find that this is the string becomes unacceptable.

So, that was not a good that was not a valid choice so, we come back ok. So, input pointer is retracted and we come to this state and so, you take this transition now and now after this if I get a 1. So, it will remain here, then again 0. So, there is a choice between this path and this path. So, this way, it can that the NFA can operate ok.

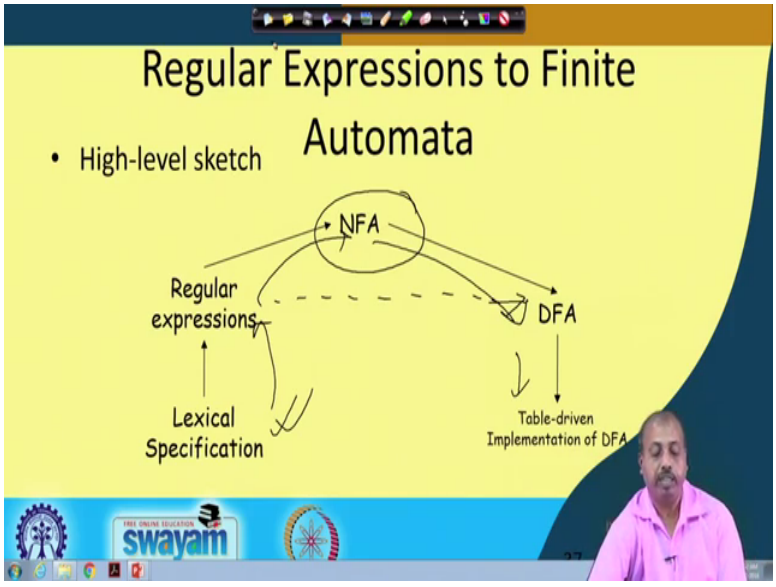
So, it can wherever whenever it is finding that I cannot proceed further, it can retract the input pointer and it can come back to the previous state from where it can it can just try out a new option; whereas, for DFA you do not have that thing. So, when the DFA's

structure is quiet complex. So, as you can see from the diagram itself. So, this also access the same language that is all strings ending with at least two 0's.

So, it is also accepting that, but the point is that this is in this case we have got that that the diagram is much more complex. So, DFA can be exponentially larger than NFA so, this is the thing. So, here of course, the number of states in NFA and DFA are same. So, it does not have much difficulty, but if we will see some example later where we will see that the number of states in a DFA can be much larger than the much much larger than the number of states in the NFA. So, that is why for the say for the representation of a regular expression.

So, it is better that we do it in terms of NFA and while writing the corresponding recognizer. So, if that the program can be made records if so, it can try out the alternatives, it can backtrack and try out the other part like that. So that way, the recognizer can be made complex, but space wise so it will be saving the space because I do not need to have. So, many states the number of states maybe much less compared to the DFA. So, that is the general observation.

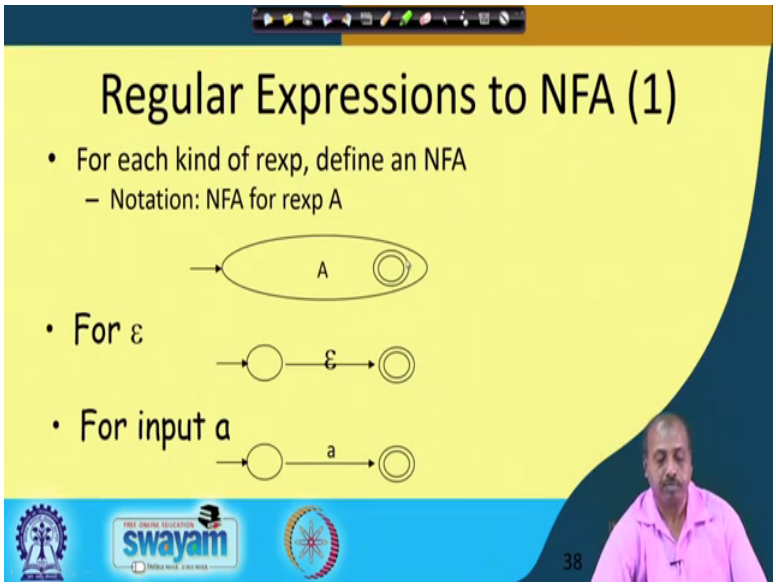
(Refer Slide Time: 25:33)



So, regular expression to finite automata so, we have got we start with this lexical specification and then, from here we come to the regular expression. From the regular expression we converted into NFA and from NFA we convert into DFA and for DFA we have got table driven implementation of DFA that is how this whole thing works ok.

So, sometimes so of course it is possible that you do not make this NFA and go from regular expression directly to DFA that is also possible. But, normally it is not done. So, we follow the other revenue because there are there can be very well defined rule for converting regular expression to NFA. So, we will see some examples later.

(Refer Slide Time: 26:21)

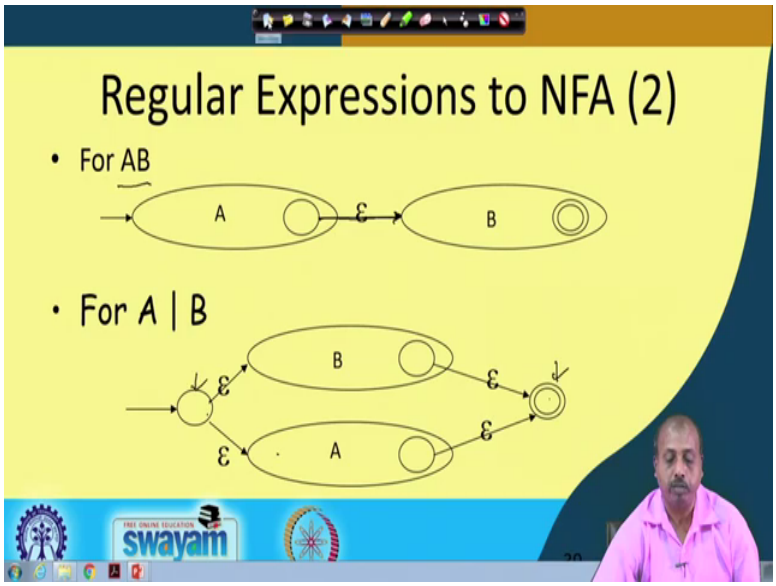


So, like, so this is the so, for say the NFA for this regular expression A. So, for different type of regular expressions, we can have it is like this. So, it has got the input for some regular expression A. So, it has got an input pointer and this is the final states so, there will be a start say.

So, for different types of A, I can have different type of regular expression. For example, if the regular expression A is only epsilon, then it is represented by the NFA while we have got this initial state, then this epsilon telling it to the taking into the final step. If the regular expression A is equal to this small a, it has got only the single character small a in it.

So, it can go from this start state on this small a, it goes to the finals state. So, this way I can define regular expressions for individual symbols that we have for the in the regular expression what you are single symbols appears so I can make it like that.

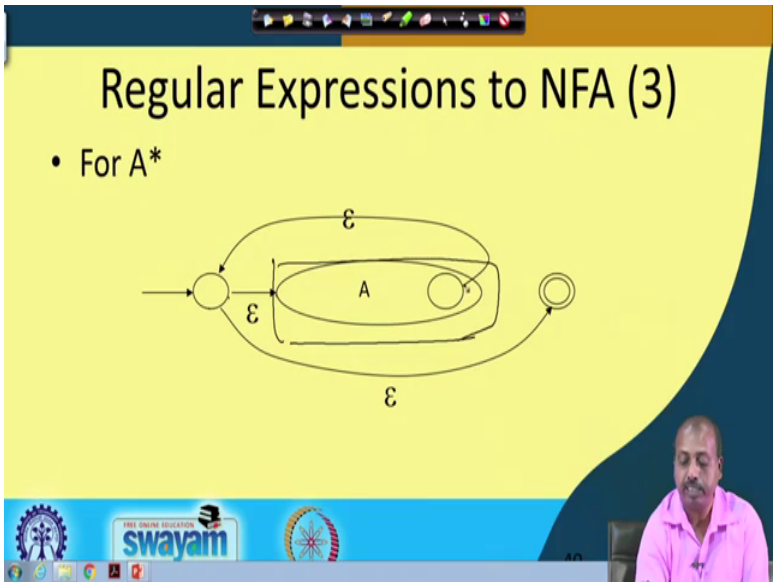
(Refer Slide Time: 27:31)



Now, if you are trying to if you have got two such NFA's constructed for the regular expression A and B, then you can join them together by if the regular expression is say AB, Then, you can join them together by doing it like this. So, the final state of A, from the final state of A put an epsilon transition to the initial state of B; whereas, if the regular expression is A or B so and you have already constructed the NFA's for this regular expressions sub parts A and A and B separately. Then, you can add a few extra states like so you take this these 2 extra states and add epsilon transitions.

From the initial state, so you put an epsilon transition to the initial set of B. Similarly, put an epsilon transition to the initial state of A and from the final state of B, put an epsilon transition to the total final state of the whole expression. Similarly, from the final state of B, put an epsilon transition to the final state. So, this way for A or B, we can make the NFA. So, what I want to mean is that depending upon the regular expressions. So, you can construct it to the small portions of it and then, combined them together for getting the overall regular expression.

(Refer Slide Time: 28:55)



So, this is another very interesting one for A star. So, I assume that I have already constructed the NFA for A. So, this is the NFA for A that is already constructed. So, this part is there. Now, for getting A star I should I should have 0 or more occurrence of A. So, this so, I take one initial state and a final state as extra and put an epsilon transition from the initial state to the final state. So, that way this is this is actually correspond to the 0 occurrence of A.

Now, to make 1 occurrence, so you can go it like this to through an epsilon transition you come to A, make this A and then there is an epsilon transition for the any final state of A back to the initial state of initial state of the regular expressions. So, you go back like this and again by another epsilon you come to the final state. So, if you trace this diagram, then you can understand that this will realize the regular expression A star ok. So, this way we can convert regular expressions to NFA and there is well defined rules like that for doing the conversion.