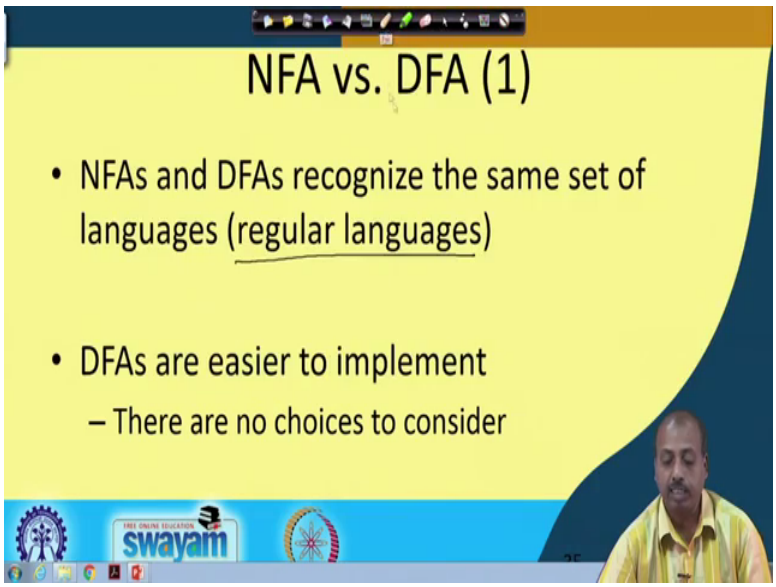
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
Lexical Analysis (Contd.)

(Refer Slide Time: 00:15)



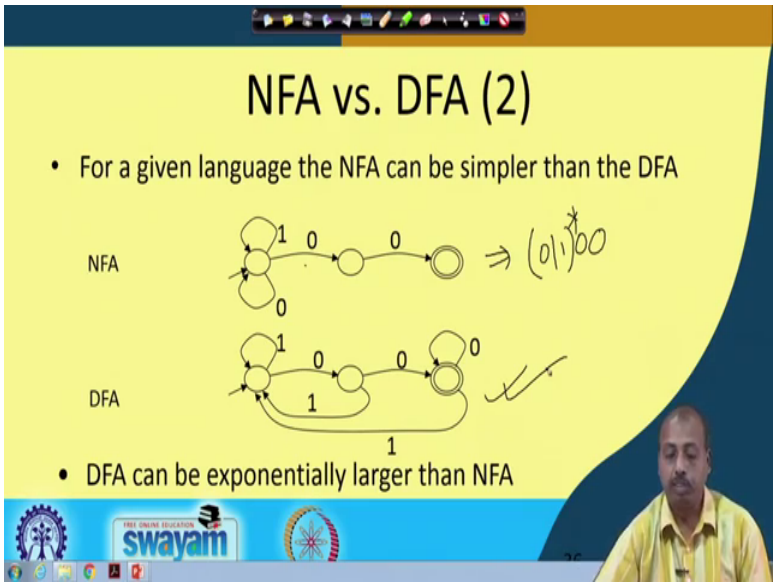
So, in our last class, we have seen two types of finite automata, Non – Deterministic Automata and Deterministic Finite Automata that is NFA and DFA. So, NFA is non-deterministic in the sense that at some state so we have got a choice, whether to make a transition on a particular input to a one state or the other, or also there is there were epsilon transitions. So, without consuming any input, whether the machine should make a transition from one state to the next; whereas, the deterministic finite automata did not have this type of non-determinism. So, there everything is deterministic.

So, from one state on a particular input pattern or input symbol, it will go to one of the next state. So, if there are no transition defined for a particular input symbol, then it is an error. So, the string gets rejected, but if we compare this NFA and DFA, apparently it seems that NFAs may be more powerful, because it has got non-determinism in it. So, or somebody may have say that DFA is better, because DFA there is no non-determinism. But theoretically it can be proved that this NFA and DFA, they are equally powerful as far as the set of languages that they can accept.

So, this set of languages will be called regular languages. So, we will define regular languages later, when we go to the in the grammar portion in the chapter on parser. But, these regular languages are a class of language, where it is a bit restrictive compare to many others, but this NFA and DFA both of them accept the same set of regular languages. So, any regular language will be accepted by a DFA, you can have a corresponding NFA.

And whenever you have got an NFA, so you can get a corresponding DFA; so, there is power wise, there is no difference. However, DFAs are easier to implement, because there is no choice to consider. So, we can have a we can do it very easily. So, we can have some sort of table driven algorithm by which you can make the transitions for the DFA, but there is other issues that we have a like this.

(Refer Slide Time: 02:31)

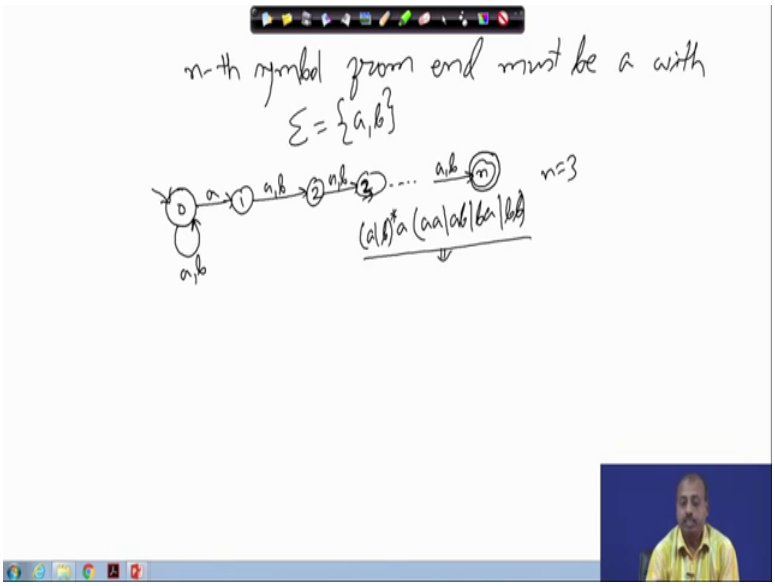


Other issues that we have is a NFA versus DFA. For a given language the NFA can be much simpler compared to DFA. So, here is an example that we see. So, we have got the in the so in the NFA. So, this is accepting the strings which are ending with two 0s. So, in this case it is it is ending with two 0s; so it is 0 0, it is going to the end state or final state, otherwise it is remaining in the initial state.

The same NFA or same regular expression; so, if you want to so this regular expression, corresponding to this is 0 or 1 star followed by 0 0. So, this regular expression so, if you want to realize by means of DFA, you will get a diagram like this. So, you see the

number of states in both the finite automata are same, however the there are more number of transitions. And we will see an example, where we will find that this DFA can be exponentially larger than NFA. So, we will next look into an example which will be doing that.

(Refer Slide Time: 03:45)



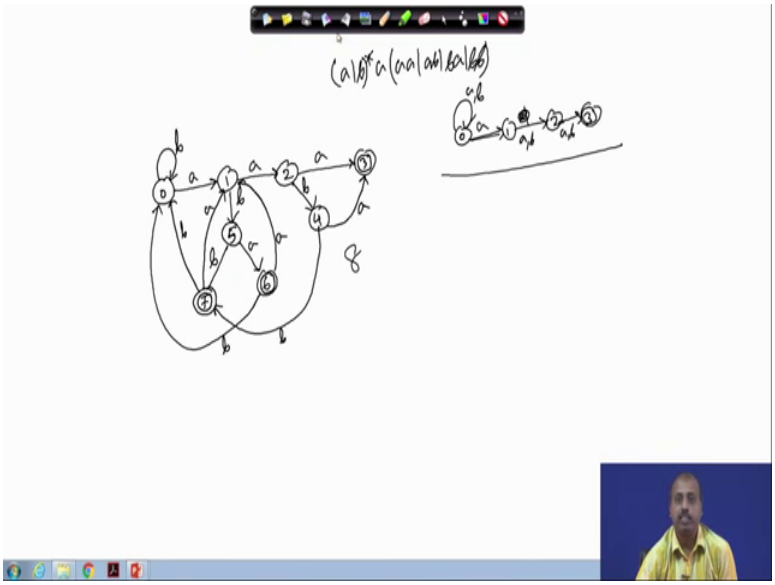
So, consider a regular expression for this particular language. Suppose, I have got it is written like this; so it is the n-th character or n-th symbol from end must be a with alphabet set, with the alphabet set sigma equal to a b. Now, if we try to construct the corresponding finite automata, the NFA construction may be like this. So, this is the start state 0. So, as long as the so, so you get a or b you remain in this state. So, I said that the n-th character must be a. So, if this is the n-th character, so on a; it makes a transition to state 1.

Then either on a or b, it goes to state to 2; then either on a or b, it goes to state 2 state 3 so this way it goes on ok. So, it is may be that the final state. So, this is our state n, this is our state n and on getting a, b it goes here. So, you see that here so, so this is the final state. So, this is a final state. So, you so this if the n-th character is a, then only the strings are going to be accepted, but for the same thing for the same regular expression. So, if we try to draw the corresponding DFA, then the situation will be very different.

So, let us take say n equal to 3 ok, let us take n equal to 3 and for that we try to construct the corresponding DFA. So, if n, n is equal to 3, then the corresponding regular

expression is given by a or b star followed by a; and then the next few characters may be a, a b, next two characters may be b a or b b, so this is the regular expression. In fact, this particular case where this is general that is there that the n-th character from n. So, you cannot have a regular expression for that ok, but when n becomes fixed so like n equal to 3, then you can write down this regular equation. Now, for this regular expression so if I try to draw the corresponding DFA, then it will be something like this.

(Refer Slide Time: 06:23)



So, let, let us take a new page and then do it. So, my regular expression is a or b star, then a, then a or b, a sorry, sorry. So, this a or b star, then a then you have got a or a b, or b a or b b. So, this is the regular expression. So, if you start at state 0, this is my state 0, as long as you get b you remain in this state. So, if you get an a, so basically I am trying to capture this a a a sequence; so you come to state 1, then if you get another a come to state 2, if you get another a come to state 3, which is a final state. So, this captures the sequence a a a.

Now, if you get after so after getting the first a, if you get a b that is you have to define the transition on b. So, one transition on b, you can may be from this from this state, if you get a transition b, you come to a state 4. And from this state 4, there can be several situations like if you get an a, if you get an a, then you see how did you come here is that the so if you get an a here, then you can go to this state, because then the third character from the end is really becoming an a. And if that is end of the string, then this is valid so

this is the thing. Similarly, from this state 1, if you get an, if you get a b, you come to state number 5.

And from state number 5, if you get an a, you come to state 6. And again state 6 is a final state, because the third character from the end is in a. So, to reach state 6, we have come via this path a, b, a. So, this a is the final these are the third, the third most character. So, this 6 is also a final state ok. Now, from this state 5, if you get a b, if you get a b, then also you come to a state 7, but this 7 is also a final state, because here also the last characters are b, b and a so that way it is correct. However, from state 7, if you get a b, so you go back to the first state, state 0 and from here also from state 6 also if you get a b, you go to state 0 ok.

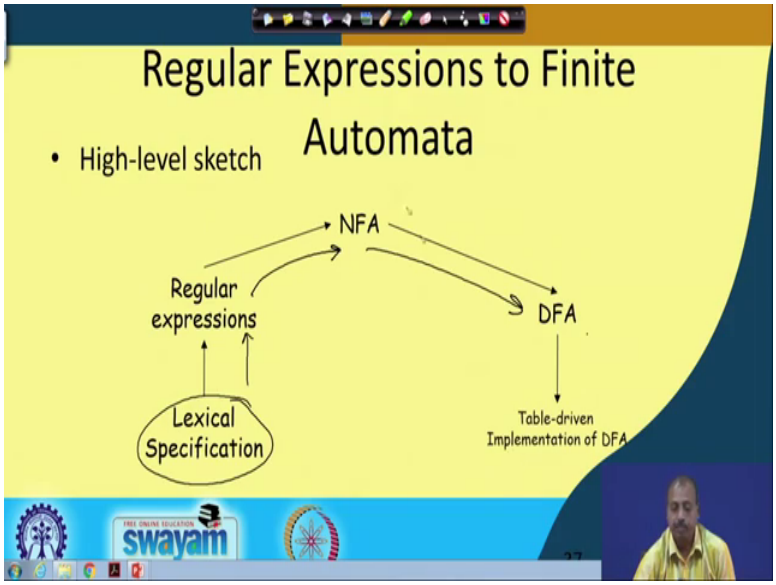
Now, from state 7, if you get an a, then you have to come to state 1, because now a a a, a b a and all those sequences can be managed. So, on a you have to come here. Similarly, from here also on getting a you have to come to state a. And from state 4, if you get a b, state 4, if you get a b, then it is like a b b, so then that is also a final state 7. So, you see that what is happening is the diagram is becoming very complex. Number of states in this case is equal to 8 compared to the previous like, if you are doing it using NFA, then you will do it like this. So, at this state it is a or b then on a you go here and then so this is 0 1, then on a or b you go there a or b so state 2, 3, and 3 being a final state so this was the NFA.

So, getting the third most character as a so it is going to a final state. So, here it was a three of only a 4 state machine, here I have got a 8 state machine. So, that way number of states are increasing significantly. So, that can happen ok. So, in a more pathetic situation so, it may go to the number of states they become exponential in the case of a DFA. So, DFAs are better, because if you are trying to develop a recognizer, then working with DFA is easier, because everything is deterministic. So, that the detection algorithm there will be non-determinism in it, but at the same time the size of the DFA maybe you have very large, so compared to the NFA.

So, for our understanding, we should we will work with NFAs most of the time and then do something so that the NFA can be converted to DFA for the sake of for the sake of implementation. So, this is exactly what is done by the lexical analysis tools. Like given the regular expression, so it first constructs the NFA; and from the NFA, it converts it

into a DFA. So, in the coming few, few slideshow you are going to see how this is going to happen.

(Refer Slide Time: 11:34)

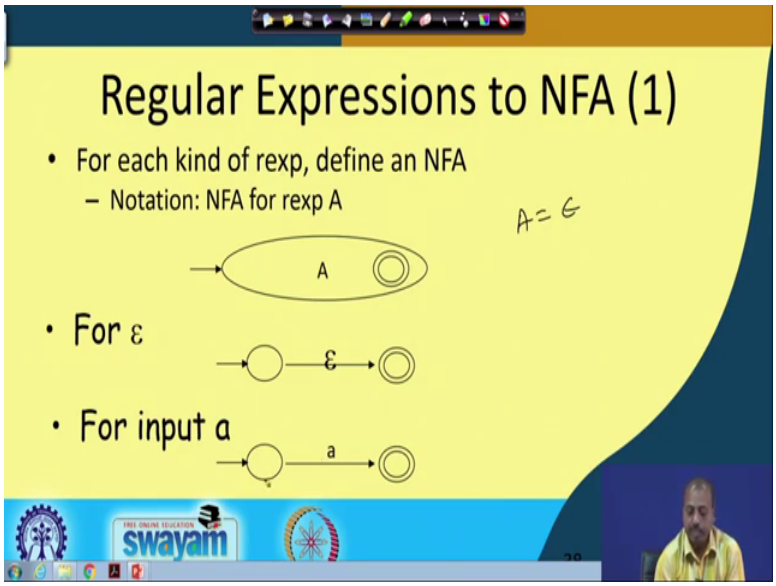


So, regular expression to finite automata the avenue that is taken is like this that is we will start with the lexical specification from the lexical specification, which is an English language description like the third most character from the end should be a like that there is electrical specification, from there we write down the regular expressions.

So, after the regular expressions have been written. So, we convert this regular expressions to NFA and then from this NFA we convert it into DFA. So, there is a conversion algorithm that we will see that can convert a non-deterministic finite automata to a deterministic finite automata. So, basically all the equivalent states of the non-deterministic finite automata. So, they are clubbed together and they have constitute one state in the DFA.

And for regular expression to NFA, so this parts so there are some well defined rules, which will convert, which will do this thing regular expression to NFA. And once we have got the DFA, then we can have a table driven implementation of the DFA, so which will be which will give us the recognizer for the language.

(Refer Slide Time: 12:50)

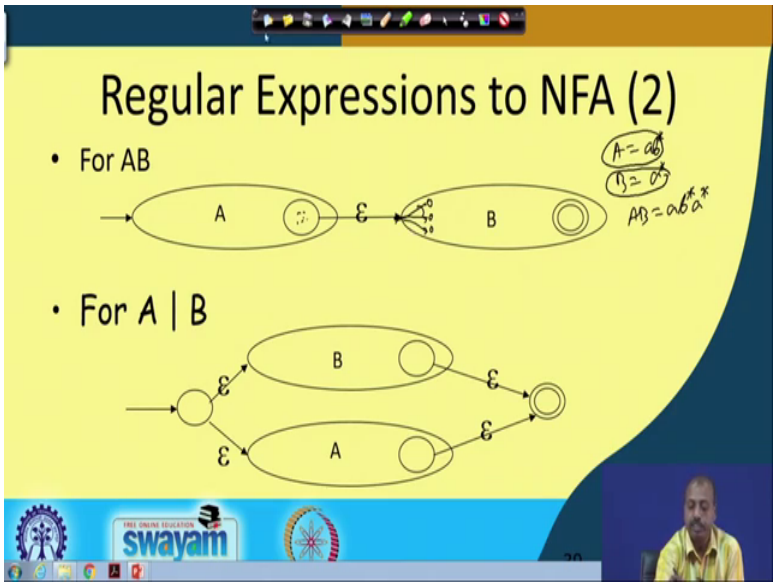


So, next we will slowly go into the portion that regular expression to NFA like say, we will be using this particular notation suppose we have got a regular expression a. So, the corresponding NFA will symbolically represent like this. So, this a is basically there are a number of states, so which a there are number transition between them which constitute the NFA for a for the regular expression a. But for our for our construction purpose, we will represent it in this fashion that is as if there is a final state, and there is an initial state and this arrow means that this will go to the initial state of the NFA for a. So, this notation we will be using quite a, quite a few times in our discussion.

Next we will consider individual symbols like say epsilon. So, if your regular expression a is equal to epsilon, then a regular expression is converted into an NFA like this. So, this is the start state identified by arrow coming to it, then this is the final state identified by two concentric cycles circles. And there is an edge from initial state to final state whose level is epsilon. So, this is the regular NFA corresponding to the regular expression epsilon.

So, this is the regular NFA corresponding to the regular expression epsilon. Similarly, the for the regular expression a the corresponding NFA for corresponding NFA is represented like this that is from the initial state and the final state and then edge between them is labeled by a. So, this way we will be this will be doing the things for individual symbols that we have in the regular expression.

(Refer Slide Time: 14:33)



Now, what about two regular expressions coming together like A is a regular expression and B is a regular expression. Now we have got a new regular expression, which is a followed by b like say your A maybe the regular expression say a b star and say, B maybe the regular expression a star. So, A, B is the regular expression a b star, a star.

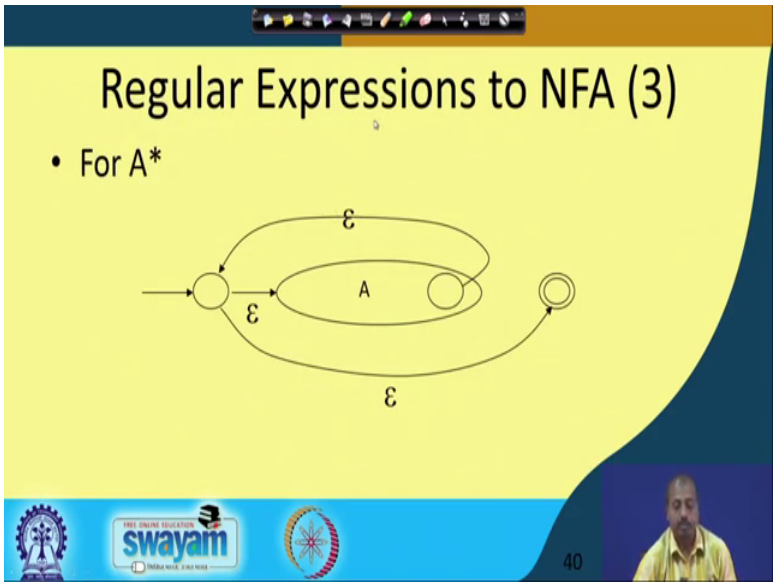
So, how to construct the corresponding NFA with the situation in which you have already constructed the NFA for a and we have already constructed the NFA for B. So, that part is the NFA for a second part is the NFA for B. So, from the final state of A, we add an epsilon transition to the initial state of B. And this state no more remains the final state, because this is not the end of the regular expression. So, this way we do it. So, if there are multiple initial states in this multiple initial states, in this, then there will be multiple such epsilon transitions or we can say of course, normally we have one initial state.

So, the possibility is that there are many final states in A and this is the initial state in B. So, which we have edge coming from all of them and all of them will be labeled epsilon. So, this way we do it ok. So, for A, B so, one regular expression followed by another regular expression. So, we take the final states of the first regular expression from there, we add epsilon transitions to the start state of the next NFA for next regular expression. Next we will see how to do the regular expression for A or B. So, A and B are two regular expressions and the new regular expression formed is the regular expression A or regular expression B.

And as our assumption so, we have already constructed the NFA for regular expression A, which is this part, we have already constructed the NFA for regular expression B, which is this part. Now, for constructing the NFA for A or B what we do, we introduce two new states one initial state and one final state and the initial state or the start state becomes the start state of the whole regular expression A or B.

We add epsilon transitions to the initial states of A and b. So, like this. So, here we add epsilon transitions here and add epsilon transitions to this the initial states. And from the final states of A and B, we add epsilon transitions to the final state of the, the newly created final state the final state for the whole expression. So, this is how we do it for A or B. So, these are quite intuitive, because you can go either by this path or you can go by this path, but and in the first case when it is AB. So, you have to go through like this. So, A followed by B going from A to B, we do not consume any new symbols. Similarly, while taking a decision like whether, we should go to A or B, we do not consume any input ok, so that way this part is realizing the regular expressions AB and A or B.

(Refer Slide Time: 18:02)



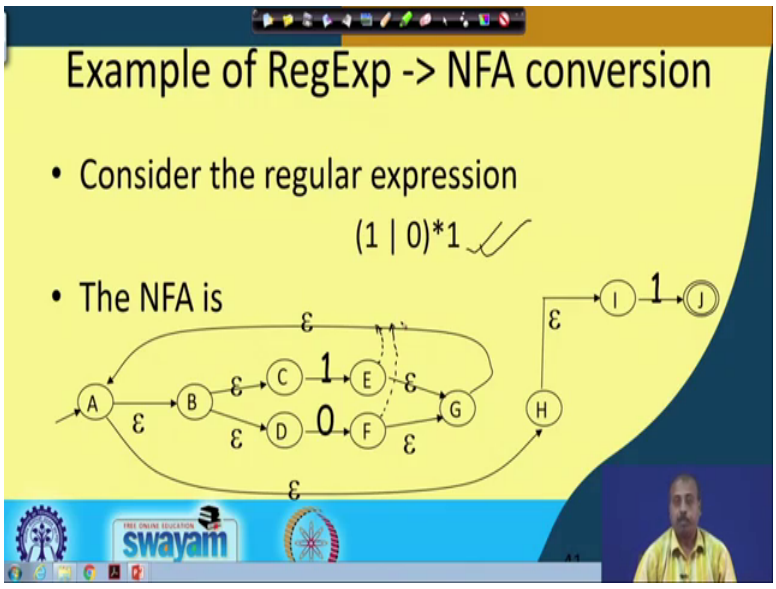
So, the more complex situation comes, when we have got the regular expression A star. So, we already have the NFA for A. So, this is the NFA for A, then we add two new states to the system one is the one is the start state, and one is a one is a start state, and one is a final state, and we add epsilon transition. So, from the start state we add epsilon transition to the start state of A; and from the final state of A, we add epsilon transition to

the newly created start state. And also we add epsilon transition from the start state the newly created start state to the newly created final states. And the final state of A, it is no more the final state ok so, because the transitions are being added ok.

So, you see in this way if we do it, then the advantage that we get is suppose there is A zero occurrence of A because, it is A star. So, suppose we have got zero occurrence of A then it will be going by this epsilon transition. If it is one occurrence of A, then it will go like this with epsilon it will come to A, it will go like this then through epsilon, it will come back. Here then by epsilon it will come to the final state one occurrence. If it, if there are two occurrences of A, then it will go like this, then come back by epsilon like this. And then again go through A one more cycle coming via this epsilon, and again come here and then via this epsilon, it will go to the final state.

So, as many time you want, so depending upon the number of times this A appears, so you can just go on moving around this NFA and finally, when you have reached the end, so you can come by this epsilon to the final state. So, this way you see that we have got a few set of rules for constructing. The NFA for individual types of regular expression individual symbols epsilon, then consecutive appearance of two regular expressions or of two regular expression and this 1 or 0 or more occurrences of the regular expression. So, this way we can construct the NFA.

(Refer Slide Time: 20:13)



Let us take an example suppose, we have got this particular regular expression 1 or 0 star 1. Now, how do we do it? So, if see if you look into individual components, then we have got a regular expression 1, regular expression 0 and regular expression 1. Now, how do you construct a regular expression for 1? So, it is like this. So, this is this is a final state this is the start, start state, and it is labeled with 1. Similarly this 0 regular expression for 0, so you can construct it like this on 0 it goes here.

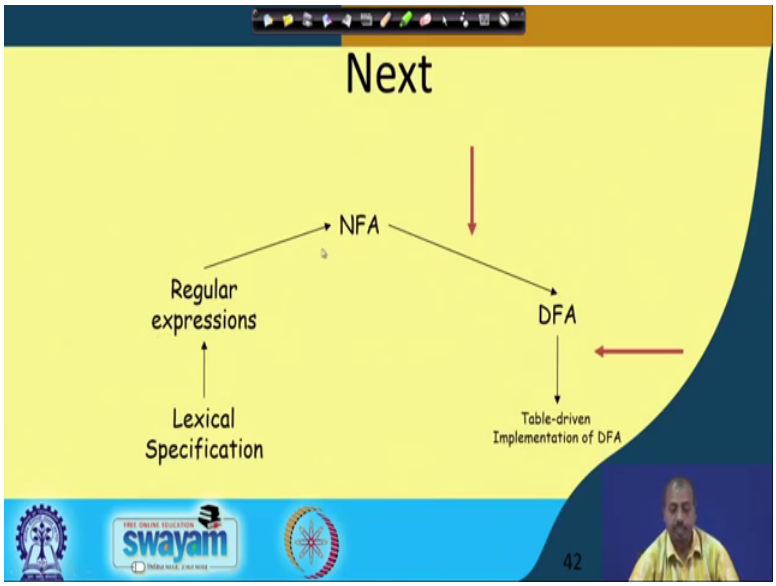
Now, once we have done that the next part is to construct the regular expression for 1 or 0. Now, for 1 or 0 what is we so, so we take the so this is our A ok. So, this is our A this is our B, this is our B. And then we add two new states like B and G there and from B, we add epsilon transition to C and an epsilon transition to D and from E and F, we add epsilon transitions to G, so that way.

So, now, this whole thing it represents the NFA corresponding to the regular expression 1 or 0 after that I have to take a star over this. Now, for taking star over this what we do, we add a new state H here. Now, from and add a new star state a here. And from the star state A, we add an epsilon transition to the initial state of 1 and 0, then from the final state of the NFA for 1 or 0. So, we add a transition epsilon transition back to A and also from A, we add an epsilon transition to the state H the final state H. So, that gives me. So, at this point we have got the regular expression.

So, till this much we have got the regular expression NFA for the regular expression 1 or 0 star with that we have to add the NFA for regular expression one and this is the regular this is the portion of regular expression 1. So, from the final state of the first NFA we add an epsilon transition to the initial state of the, the other NFAs the NFA for 1. So, this is done and epsilon transition is added. So, that way this whole thing becomes the NFA corresponding to this regular expression.

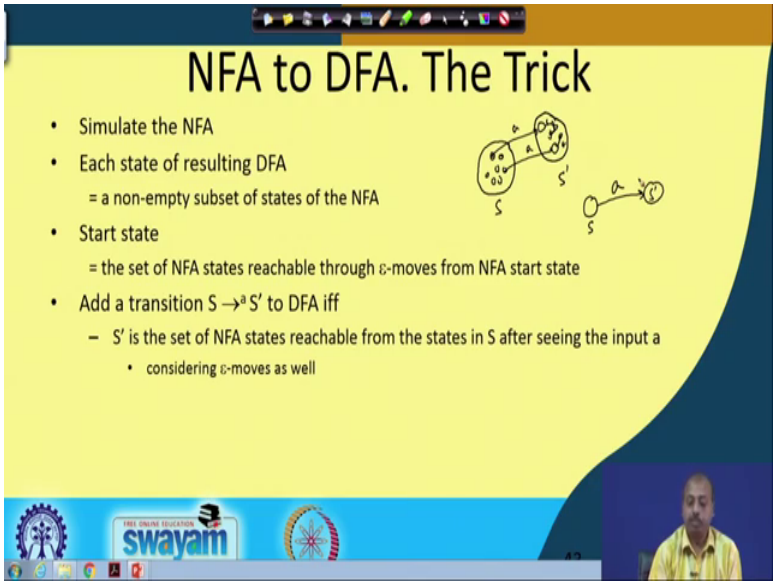
Of course there is lot of scope of optimization, because you see that there are epsilon transitions like from E to G and then again another epsilon transition to A. So, ideally I should be able to do it like this. So, this part this state G can be eliminated and both of them from E and F, I can take them back to a directly. So, this type of optimizations can be done, but going by the set of rules, we get this particular NFA ok.

(Refer Slide Time: 23:40)



Now, once this NFA is there what can we do with this. So, the next thing that we can do is. So, we are at this point. So, we have we started with the lexical specification from there the regular expression from there we got the NFA. The next thing to do is to convert this NFA to DFA and once we are done with that then we will have a table driven implementation of the DFA. Now, how to convert the NFA to DFA, so that part we have to consider see now.

(Refer Slide Time: 24:10)



So, the basic idea is that for NFA to DFA conversion, the idea is to simulate the NFA. So, we simulate the non-deterministic finite automata. And each state of the result of resulting DFA is equivalent to a non-empty subset of states for the NFA. So, you start with the NFA and start simulating for a particular input symbol A and see how far it is going for with without consuming any more input.

Like here, so if you see this one if you, see say this example, so you see if you start with say 0. So, without if you are starting at state A, then without consuming any more input with the just an input 0, you can come from A to B to D to F to G, and then it is not adding anything more, because now it is going to come back to A only. So, this A, B, D, F, G, so all these are all these are can be simulated for the input symbol 0. So, to construct the start state, we consider, we consider the set of NFA states, which are reachable through epsilon moves from the start state.

So, in this particular case, A is the start state. So, you take all the states, which can be reached from a using epsilon transitions only. So, the set of states are A, B, C, D, H and this I. So, all of them can be reached by the epsilon transitions only from state A. So, all of them taken together, so that set will constitute the start state of the DFA; so, this is what is said that this the start state of the DFA is equal to the set of NFA states reachable through epsilon moves from the NFA start state.

Then from any state S, we added transition to transition to state S dash in the DFA on input A if and only if so this condition is satisfied. S dash is the set of NFA states reachable from the states in S, after seeing the input A considering epsilon moves as well. So, what has happened is I have started with the start state. So, a number of NFA states number of NFA states, so they have given me the DFA state, DFA start state S.

Now, from these states so from these individual states on A you see where can you go. So, maybe this state on a it takes here, so this state on a takes here. So, others they do not have transitions on A. In that case these two plus all the states which can be reached from here via epsilon transitions. So, all those states that can be reached from these two states where epsilon transition, so that will constitute the state S dash. So, in that DFA, I will have a state S; I will have a state S dash, and the transition will be labeled with A. So, this is how we will add a transitions to the DFA.