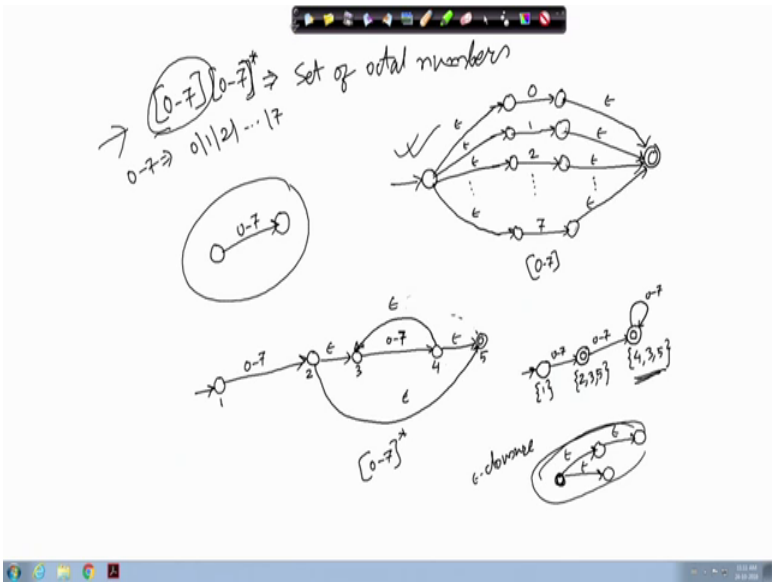
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 15
Lexical Analysis (Contd.)

So, next we will be looking into some more examples of this construction of NFA and DFA from regular expressions.

(Refer Slide Time: 00:29)



So, the next example that we will look into is a regular expression, which is given by 0 to 7 followed by 0 to 7 star. So, basically we have got the digits 0 to 7 followed by 0 or more occurrences of the digit 0 to 7.

So, what is this regular expression? So this is the regular expression that correspond to the set of octal numbers, set of octal numbers, because the numbers can start with any of the digits 0 to 7. And there can be one digit or multiple digit, but the digits are restricted to 7 only, so it cannot be more than that. So, this represents the octal numbers.

So, if you want to construct the corresponding NFA for this one, so for each of the digit we will have NFA like this. So, this is for the 0 ok, so then this is for 1 this is for 1, then we for 2 we will have another such NFA. So, this way we can have up to 7, we can have the NFA's created. Now, for 0 to 7, which actually means that regular expression 0 or 1

or 2 or up to 7, so for that we have to join all of them. So, we have to have one initial state, and one final state. And from this initial state, I have to add epsilon transitions to each of these states ok. So, I have to add epsilon transitions. So, these are all epsilon transitions. And this is the final initial state.

And so these states that we have, so they will no more be final state ok. So, they will be normal state now. And then we will have this, we will have from here we have to add transitions to the final state ok. So, these are again epsilon transitions, and this is a final state.

So, this way we have the NFA corresponding to 0 to 7. Now, for our sake of writing, so for our gravity; so what we will do, we will simply represent it by something like this, we will write here 0 to 7. Essentially, we mean that this whole stuff is actually this one.

Now, from this if we once we have done this, then you can draw the regular expression for 0 to 7 star. So, this part is for 0 to 7, if I take it like that so it is 0 to 7, now for making 0 to 7 star. So, we have to have some epsilon transitions added ok. So, we will have to have some epsilon transitions added.

So, there we have to in introduce one initial state from here, the epsilon transition will come. Similarly, from this one I will have one epsilon transition to the final state ok. And from this state, there should be an epsilon transition back to the there will be an epsilon transition coming back to their; sorry there is a problem here, so this is not correct so ok. So, from this state if I number this state, numbers are 0, 1, 2, 3 etcetera, then from state number 1 I have got an transition on that digits 0 to 7. Now, on epsilon it should come back to the states, so that will capture that 0 to 7 star.

And then I have to have epsilon transition here, and from this first state to the final state there should be an epsilon transition, so that way it will capture 0 to 7 star. So, this part will be 0 to 7 star. Now, before that we have got this 0 to 7. So, for that I will have to add that 0 to 7 portion, so this will be like this 0 to 7 ok. So, then if we number the states, then this is the final this remains the final state ok, this remains the final state.

Now, if we if we want to construct the corresponding DFA, then we have to convert this NFA to DFA by clubbing the states as we did previously. So, let us renumber the states

for this clubbing purpose. So, let us say that the states are numbered as so this is say numbered as 1, this state is 2, this is 3, this is 4, and this is 5 ok.

Now, for the original NFA, this is the start state of. So, when we are trying to construct the DFA, we have to start with state number 1 and take epsilon transitions from state number 1 to have the initial state of the DFA. Since there is no epsilon transition from state number 1. So, in the DFA the initial state is consisting of a single state of NFA, which is 1. Then from 1 on 0 to 7 you can go to state-2, and from state-2 on epsilon transition, we can go to state 5. So, this way it on 0 to 7, it goes to the state, which is consisting of the NFA states 2, 3, and 5; 2, 3, and 5.

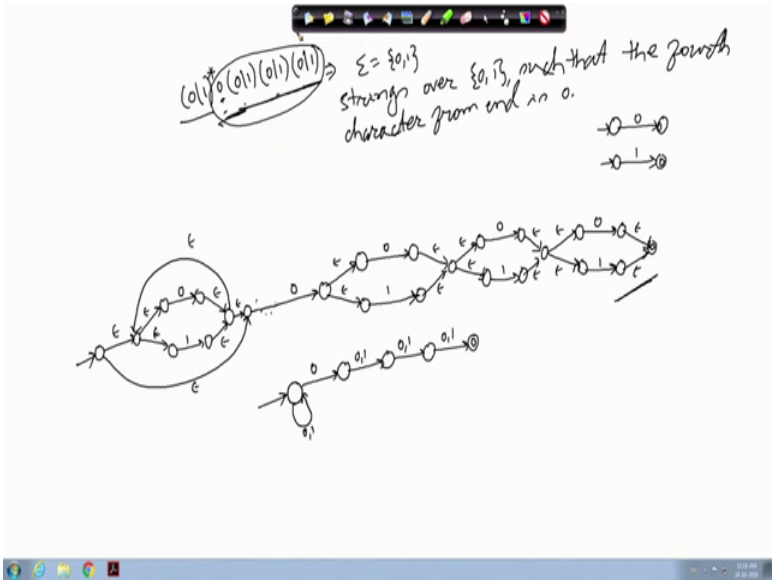
Now, from 3 on epsilon transition on 0 to 7, you go to state number-4. And from state number-4 on epsilon transition, it can go to state-3 and state-5. So, from here on 0 to 7, it can go to a state, which is actually consisting of the NFA states 4 from 2, 3, 5. So, from 3 this is it is going to state 4. And from 4 on epsilon transition, it is going to 3 and 5, so 4, 3, and 5. And you see that here this here this state number-5 is a final state for the NFA. So, they also constitute the final states for the DFA ok.

Now, from state number 3 from state number-3 on getting 0 to 7; so like if I if I consider this particular DFA state, then from state number-4, there is no transition on 0 to 7, for 5 also there is no transition, but for 3 there is a transition to state number-4.

And after coming to 4, so if you take the states reachable, where epsilon transition with which is which is also known as epsilon closer, so this is the technical term. So, from one states, so all the states that are reachable by epsilon transitions only. So, if this is a state, then from this state, where epsilon transition I can reach all the states. So, all the states they will come in the epsilon closure of this state.

So, basically we are trying to compute that epsilon closure. So, from here that epsilon closure will give me the same state, because from 3, it will come to 4. And from 4 if I take the epsilon closure, it will get 3 and 5. So, as a result you will get a loop here for 0 to 7. So, this is the DFA corresponding to the octal numbers in a regular expression that we had ok.

(Refer Slide Time: 09:32)



Next, we will take another example. Say the regular expression, which is given by 0 or 1 star 0, and then followed by 0 or 1, 0 or 1, 0 or 1 fine. So, if we what is this regular expression. So if we look at it a carefully, then here the alphabet set is 0, 1. So, it is actually strings over the symbols 0 and 1.

And then we are if you look into it carefully, then the 4th (Refer Time: 10:06) character is always 0, all the strings where the 4th class character is 0. So, they are the valid strings of this language. So, there is strings over strings over the alphabet set 0, 1 such that the 4th character from end 4th character from end is 0. So, this is the meaning of the regular expression.

So, you can try to construct the NFA for this one. And the NFA for this can be constructed like this. First of all for 0 and 1, so we know that this is the regular this is the NFA. This is 0 and this is 1. Now, for 0 or 1, so we can do it like this, so if this is on 0 and this is on 1. So, we have to add one state before this, so that there are epsilon transitions. And then one state after this; and these are also epsilon transitions. So, this is the part of the NFA corresponding to 0 or 1.

Now, you see that there are 3 such 0 and 1's. So, you can say that as if I have to make it I have to make 3 such structures, and put them one after the other. So, let us put 3 such structures. So, this is on 0 going here, these are epsilon transitions. Then again from here

on epsilon it comes to this state, and then this is the final state. So, this constitutes this part of the regular expression.

Now, we have got before this 0, so for this 0 you see that we should have a transition like this fine. And before that we have got this 0 or 1 star. So, for 0 or 1 is represented like this ok, so this is the 0 or 1. Now, for making a star, what I need to do is that I have to have one more I have to have one more state initial state here. And add one epsilon transition.

And similarly, have another final state here, which will be an epsilon transition. And then add epsilon transition from here to here, and add epsilon transition from here to here. So, this is going to be the case, which is 0 or 1 star. Now, I have to concatenate it with the remaining part of the regular expression. So, I can concatenate it like this, and that is also an epsilon transition.

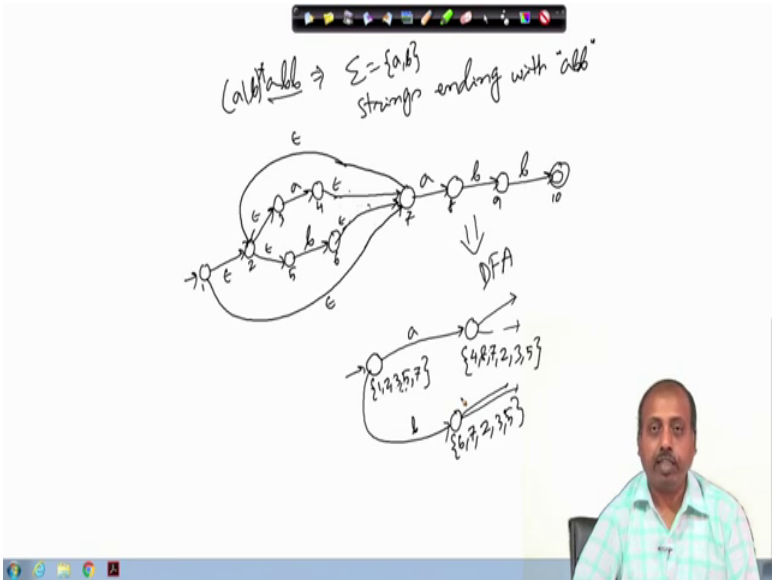
So, you can do it like this or for optimization purpose, since there are two epsilon transitions. So, you may say that I can have only one of them. And you can for go this epsilon this state all together, and you can say that I will make it from here to there by means of 0, so that can be done. Just to simplify the diagram a bit ok. And this is the initial state for the overall NFA.

So, this is one possibility and you can, so that is by the construction the construction rule that we have. So, we can do it like this ok. So, on the other hand, so we can also do it in a different fashion like you can draw an NFA directly, like you can make it like this that as long as it is 0 or 1, it remains in this state. And then when it gets a 0, it makes a non-deterministic move to the next state.

And then you have got this 0 or 1, there are three such states three such transitions 0 or 1 ok. And this is the initial state this is the final state. So, this way also you can draw the NFA. So, this is also NFA, because it has got non-deterministic decision at the first state. So, either it is on getting input 0, either it is remaining in this state or it is going to the next state, so that way we can have a non-deterministic choice. So, this is also an NFA, but you see the NFA that we have constructed here is very complex, but it is but the process is automatic. So, we have followed a rule book for constructing the overall NFA ok.

So, now you can try to convert it into DFA, but convert into DFA we will take good amount of effort because of this reason that at the end you have got something like this, this 4th character from end is 0. So, in our class we have discussed previously that whenever you have got this type of situation, then there is a possibility of state explosion. So, this is a typical example, where the state explosion may occur ok.

(Refer Slide Time: 16:47)



Next we will look into another regular expression, we will be looking into another regular expression say something like this a or b star, then a then b b. So this is a regular expression. So, when the language corresponding to this is that any string, so here this alphabet set is a, b. And this realizes it specifies the words, which are strings ending with a b b, all strings ending with the substring a b b, so that is going to be valid strings for the language.

Now, here also if you try to construct the NFA, then for this a b b this portion, the NFA will be something like this on a it comes to the state, on b it comes to the state, then again on b it goes to this state. And before that we will have to have a or b star for a or b star the NFA is like this, so this is a b. Now, all of them so for that I have to have epsilon transitions add it. So, I have added a epsilon transitions, so this is a or b.

Now, I have to make a or b star. So, for making a or b star, so I have to take another final state. And from here, I have to add epsilon transition. And I have to have one initial state from where there will be an epsilon transition. Now, from this state, it should come back

here that is an epsilon transition and from here it should go there, which is an epsilon transition.

Now, I have to merge these two. So, essentially what happens is that these two states becomes same ok. So, I can just redraw this part, so that this transition. So, this state becomes the final state ok. So, it can come here. And so all these transitions can be merged now sorry so all these epsilon transitions they can be modified, so that they can point to this state epsilon transitions. And from here on epsilon, it can go back there. So, this will also realize this regular expression a or b star a b b. So, you can always convert this regular expression into this NFA into DFA.

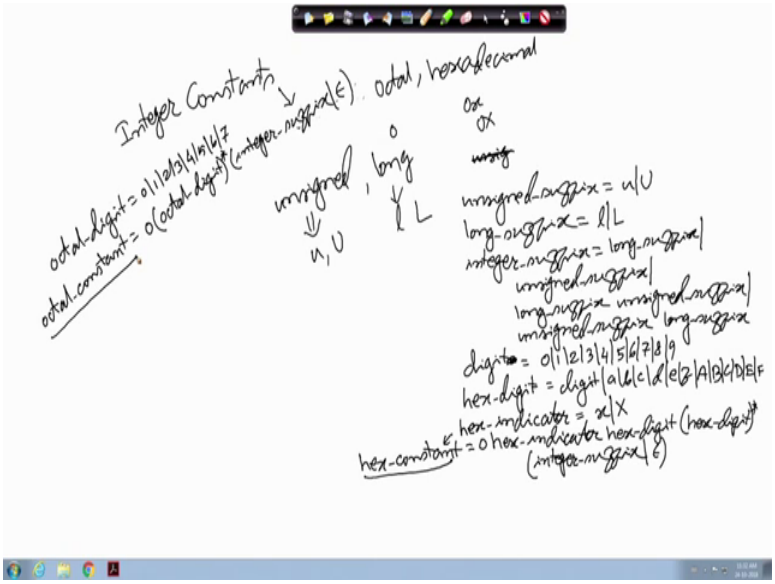
So, you can convert into DFA by following the technique that you can start with say symbol was state number 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; you can number them like this. Then from state number-1 on epsilon transition, it can go to state-2 and state-7. So, you can construct the initial state of the DFA consisting of the states 1, 2, and 7 of the NFA.

Then from this states on 1, 2 sorry 1, 2, 3, 5 will also come. It is not only 1, 2, 7, 1, 2, then 3, 5 and 7 they will come, so that will be the initial state. Now, from here on getting a you can go from 3, you can go to 4 on getting a and from 7, you can go to 8. So, I can say that on a I can go to a state, which is the state 4, 8 and their epsilon closure, so 4, 8 will be there.

Now, from state 4 on epsilon, you can go to 7. And from 7, you can come to 2. From 2 you can go to 3, and you can go to 5. So, this way you can go on constructing the further states on different symbols a and b. Similarly, from state from this state on getting b, you can come to a state on getting b. So, it will come to 6, because from state 5 was there. So, from state 5, 1, b, it will go to state 6. And from state 6 on epsilon transition you can go to 7, then 2 and then from 2, you can go to 3, 5. So, this is the state on this is the transition on state on symbol b.

So, now you can explode further from here what happens on a, what happens on b, so that way you can just go on finding the further transitions, and that will complete this DFA ok. So, you can construct the DFA like that ok. So, this way for from regular is once you have constructed the regular expression from the English language statement of the problem. So, you can make the corresponding NFA, and then convert the NFA to DFA.

(Refer Slide Time: 23:05)



So, once the DFA has been made, you can go for realizing the corresponding lexical analyzer using the tool Lex ok. So, we will take one more example from the lexical analysis portion, so that will be for Lex specification. So, we will consider the constants that you can declare in say C plus plus language; so constant integer constant integer constants.

So integer constants, so they can be decimal can be decimal, can be octal or can be hexadecimal ok. So, these are the similarly so you can have; so this octal is identified by the numbers starting with a 0, and hexadecimal is identified by the number starting with 0 x or 0 x like that.

Similarly, towards the end of the number, you can have some more qualifiers like you can have unsigned, and you can have long. So, these are the other two qualifiers that we have for the integers and unsigned is identified by u or capital U, and long is identified by a l or capital L ok. So, we need to define integer constant some regular expression that will represent integer constants in the C plus plus language.

So, lets us see how we can do this thing. So, we will write like this that first of all we will write, we will write some sub parts of this definition, which is the first one is unsigned suffix unsigned suffix can be equal to u or lowercase u or uppercase U. Then long suffix can be lowercase l or uppercase L.

Now, integer suffix integer suffix so you can have them in any order. So, it may be that it is a long suffix that is the number is ending with a with the symbol l or capital L small l or capital l long suffix or unsigned suffix unsigned suffix or maybe long-long suffix followed by unsigned suffix long suffix followed by unsigned suffix or it may be the other way that is unsigned suffix followed by long suffix. So that is the other way unsigned suffix followed by long suffix. So, these are the way by which the suffix can come after the integer.

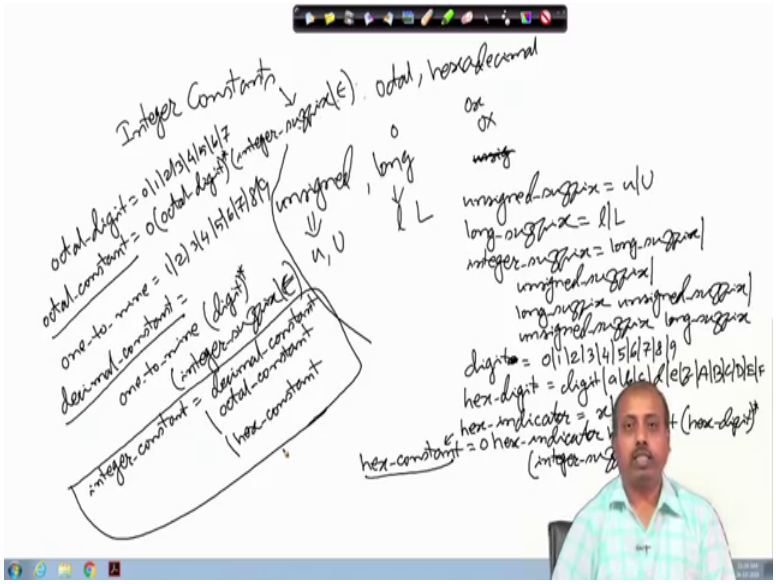
Then I will define digits. So, digits are a single digit is 0 or 1 or 2 or 3 4, 5, 6, 7, 8, 9. So, these are the digits. Now, I will have hexadecimal digits or something more than that. So, hex digit is equal to digit, and there are some more options like a, b, c, d, e, f also the capital versions capital A, capital B, capital C, capital D, capital E and capital ok. So, this is about the hexadecimal digit.

Now, we can have the indicator the hex indicator hex indicator is small x or capital X ok. Now, for the octal numbers, I can have octal digit octal digit. So, it does not include all the digits. So, it is 0 or 1 or 2 or 3 or up to 7, 4, 5, 6, 7. So, I can have eight different digits for that.

And octal constant, now after we have defined this octal digits. So, we can define octal constant octal constant, it can be 0 ok. So, octal constant should start with is 0, then this octal digit any number of octal digits can come now octal digit star. And then I can have an integer suffix after this. I can have some integer suffix integer suffix or epsilon. So, integer suffix may or may not be present, so that way we can have this thing.

Similarly, the hexadecimal constant the hex constants like here. So, after this I can define hex constant hex constant equal to 0 followed by hex indicator 0 followed by hex indicator, then hex digit at least one digit followed by hex digit star. And that should be followed by integer suffix or epsilon integer suffix or epsilon. So, we have defined hex constants, we have to define octal constant.

(Refer Slide Time: 30:38)



Now, I can define the digit the decimal constant, but for decimal constant so we will not take anything that starts with a 0 ok, so 0 it will go to octal. So, to differentiate between them, so we define another the regular definition 1 to 9, which is given by 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9. And now decimal constant decimal constant is defined as this 1 to 9 1 to 9 followed by digit star. And after that I can have integer suffix integer suffix or epsilon integer suffix or epsilon. So, this way I can have decimal constant. So, decimal constant have been defined also, all the constants are been defined.

Now, the overall integer constant is nothing but or of all them. So, integer constant can be defined as decimal constant or octal constant or hex constant ok. So, this is the final definition of integer constant. So, it takes care of all the issues like decimal constant or octal constant, hexadecimal constant ok, so all of them. And then it also takes care of these suffixes a long, unsigned, etcetera ok.

So, this way we can define regular expression for different programming language constructs and accordingly, you can design the corresponding lexical analyzer.