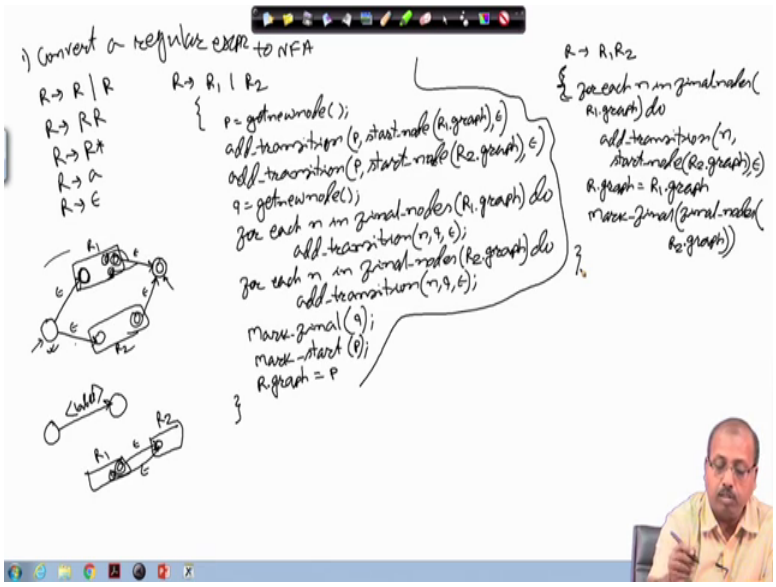
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 38
Parser (Contd.)

Next, we will be taking some examples of syntax directed translation. So, we will be taking an exercise which will be like this.

(Refer Slide Time: 00:23)



So, we want to we know the grammar for regular expressions. So, from the regular expression we need to convert to NFA. So, can we have a compiler assisted technique for doing that? So, the job is to convert a regular expression to NFA ok. So, this if we can do so, we can have an automated tool for that purpose of course, in R lexical analysis chapter we have seen the things to be done. So, you can formalize it and put it into 1 compiler constructions step. So, that you can do this we can have this conversion part automated.

So, for the regular expression the grammar that we consider is like this R producing R or R then R producing R R then R producing R star and also we have got the terminal symbols like say a. So, R producing a and it can give me epsilon. So, these are the rules that we have for the regular expression. Now, for syntax directed translation so, what we will assume is that we have got some attributes. So, each regular expression for that we

have a corresponding NFA and the NFA has got a start state and a number of final states; so, that these are there. So, if we assume that for every regular expression; so, we will maintain its corresponding NFA as an attribute of it.

Then we can combine those attributes as say syntax directed translation actions and so, that I can get the NFA for the complete expression, for the complete regular expression. So, let us take the first rule like say R producing R 1 or R 2. So, I am purposefully writing it as R 1 and R 2 so, that in my action part; so, I can very clearly mean like which attribute whose attribute I am talking about ok. So, otherwise they are all regular expressions R 1 R 2 so, they are all regular expressions. So, on the right-hand side I have got to regular expressions they are concatenated, but they are connected by the OR operator and then that gives me the overall regular expression. So and further rule for this if I have got a regular expression for R 1 and there is a regular expression for R 2.

So, it has got a start state and the final state. So, it has got a starts to the final state and so, this is for R 1 and this is for R 2. Now, if you look into this construction example then it says that you have to have a new state and add some epsilon transitions to them ok. And, then you should have another new state which will be called the call the final node then you will be adding epsilon transitions from here. So, this is the action to be done ok. So, how are you going to write? So, we write it like this. So, p is a pointer which is get new node, we assumed that we have a function get new nodes. So, if you call this function it will return you a new node of the graph ok.

So, this is the so, we will call that function. Then I have to show this is the I have got this new node now, I have now had to add this epsilon transitions. So, I do add the transition add transition what transition? So, a transition when I say so, I have to tell the start state, I have to tell the final state and I have to tell what is the label. So, these are the three things to be saved ok. So, start status this new state that I have taken then this that destination state is the state I am talking about. So, this R 1s start node. So, if I assume that there is a function called start node. So, such that if you pass the graph of R 1 to it so, it will return its start node. So, this is start node of R 1 dot graph and I have to tell what is the label. So, label is epsilon.

So, I assume that there is a function called at transition. So, if you call this function so, it will add a transition between once node 1 to node 2. So, node 1 is p in our case node 2 is

the start node of R 1 and then the transition that is added is labeled with epsilon. So, that is add transition then I can so, this is done. Now, for the second one I have to do another transition has to be added for R 2. So, for that I can do add transition P start node of R 2 dot graph and label is epsilon that add transition can be done. Now so, this part I have made. So, once I have this R 1 and R 2 these graphs available. So, I have made this left part of the diagram.

Now for the right side, for the right side what I have to do is, I have to get another node. So, I say q, q equal to get new node and that will be a terminal node. So, we will come to come later how to make it a terminal node so, that we will do later. Now, this node this graph R 1 may have several final states and for each of these final states are to add this epsilon transitions. So, I write like this that for each n in final nodes of; so, I assume again that there is a function called final nodes that returns the set of final nodes of a graph. So, R 1 dot graph ok. So, for each node n in final nodes of R 1 dot graph do we add transition, add transition from this final this n to q with label as epsilon fine.

And similarly, for R 2 also I need to do the same thing that for each n in final nodes of R 2 dot graph do add transition; add transition n q epsilon. So, once we have done that so, these transitions have been added. Now, what we have not done so, far is, marking this node as the start node marking this node as the final nodes. So, marking p as start node and marking q as the final node of the graph.

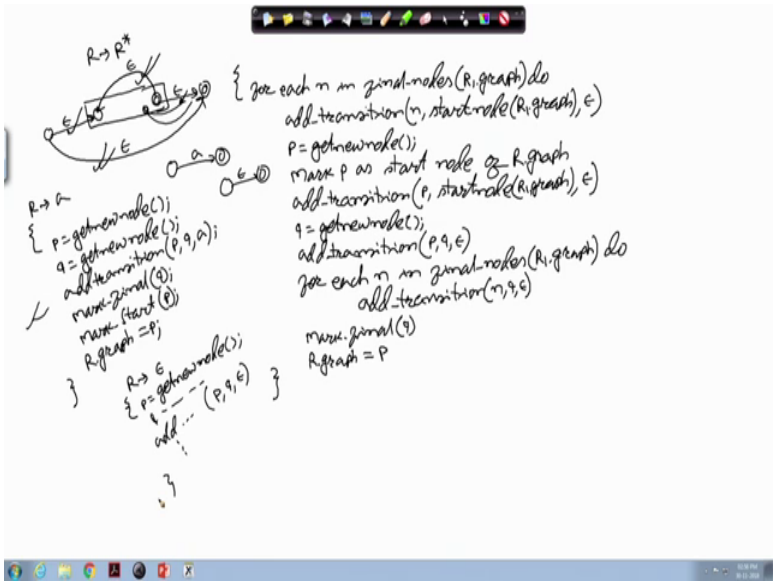
So, we assume that there is a function called mark final which will mark the q node q as a final node and then we say that we are to mark p as they start node. So, I say that as if as if I have got another function that mark start p. So, this is made the start node of p. And, then the attribute R dot graph is made equal to p. So, this way I can write the syntax directed translation scheme.

So, that for R 1 or R 2 I can get the corresponding NFA who by combining the NFA's of R 1 and R 2. So, next we will be looking into the NFA for this R 1 R 2. So, the second rule. So, R producing concatenation R 1 R 2. So, here so, we can say that the rules that will the code that we will have are the actions that we have is that. So, here I have to add transitions like if this is R 1 and this is R 2 then what we do? Is from the final state of R 1 from all final states of R 1; so, we add epsilon transitions to this initial state of R 2. So,

that is the action. So, we write it like this that for each n in final states final nodes of R 1 dot graph final nodes of R 1 dot graph do.

So, what we do? We add transition. So, we add transition n from n to the start node of R 2 dot graph start node of R 2 dot graph and the label being epsilon. So, this is the transition that we have added and then we have to say that as now R dot graph can be simply made equal to R 1 dot graph and then I have to mark the final states. So, mark final for this particular graph R. So, I have to from the final states of R will be the final states of R 2. So, this is mark final. So, from the by applying the function final nodes I get the final nodes of R 2. So, that is R 1 R 2. So, this way we can do it for this R R concatenation of 2 states.

(Refer Slide Time: 11:39)



Now, I need to do it for R star. So, how do we do it for R star? So, R producing R star and this is a bit tricky because, you know that what we need to do is, if this is the situation, then I have to have one epsilon transition from here to here. I have to have an epsilon transition from here to here, I should have an epsilon transition like this and I should have an epsilon transition like this. These are the things to be done for converting R 2 from R2 get the NFA for R star ok. So, what we do? We do it like this. So, I have to add this epsilon transitions. So, first we do the epsilon transition edition from a final states of R to its initial state.

So, that part we do we write it like this for each n in final nodes of R 1 dot graph ok. So, what do we do? We add the epsilon transitions. So, that is add transition n start node of R 1 and the label is epsilon. So, these transitions have been added. So, these transitions are now done. Next I have to add this start state. So, for that I have to get a new node. So, p as get new node and then this is the start node of the of the graph g. So, mark p as start node of R dot graph. Now, I have to add the epsilon transition from this node p to the start of R 1. So, we say add transition p comma start node of R 1 dot graph and the label being epsilon.

So, that this transition has been added now, now I have to add this transition. So, for that I have to get another node so, that is called q. So, that is another get new node. So, after I have got the node so, I do an add transition; add transition p q epsilon. So, this transition is added from p to q epsilon label being epsilon.

Now, I have to add this transition. So, for every final state that I have here in R 1 so, this epsilon transitions are to be added. So, I do it like this for each n in final nodes of R 1 dot graph do add transition n q epsilon ok. So, these epsilon transitions have been added and then I have to mark this q as the final state. So, mark final q and then we have to say that this R dot graph equal p ok.

So, this way we can write down the syntax directed translation scheme for the converting this regular expression R 2 R star. Now, there are two more simple rules that I have got. So, R producing a; so, for R producing a the so, what I have to do is, have to have something like this and this is a, this was the thing.

So, how to do this? So, I right like p equal to get new node, you have first of all create two nodes: one for one will act as the start node other will act as the end node, the terminal road and q. So, the they are the two nodes that I have got p and q and then I have to add transition from p to q; add transition from p to q and the label of the transition is a ok. Then I have to say that q is a final state so, that that can be done by calling the function mark final then to mark p as the start node.

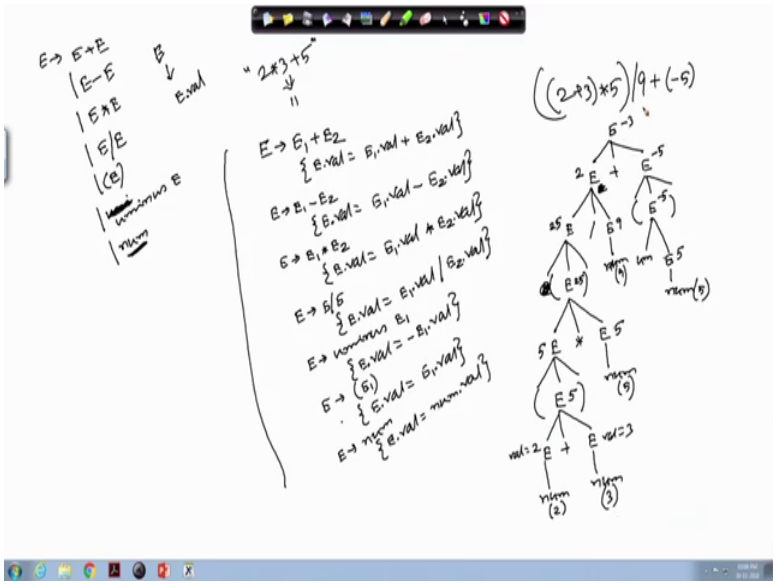
So, that is by calling the function say mark start p and then I have to say that R dot graph equal to p. So, this is for R producing a. So, only the single terminal symbol and what remains is the epsilon transition R producing epsilon. So, R producing epsilon so, here

also it is similar only thing is that this label is epsilon. So, the code for this is the same as this one, but only instead of a we will be adding the epsilon.

So, we will have like p equal to get new node q equal to get new nodes everything will be same only in the add transition part so, this should be p q the label is epsilon. So, rest of the thing will be as it is. So, if you have this set of rules then center so, by syntax directed translation mechanism; so, you can easily generate the corresponding NFA from some regular expression.

So, next we will be using another example for this syntax directed translation which is for evaluating the arithmetic expressions. Suppose, we have got the arithmetic expression grammar which is given by this.

(Refer Slide Time: 19:33)



So, E producing E plus E or E minus E or E star E then E division within bracket E and then unary minus unary minus, we write it as a separate a separate terminal symbol u minus. So, so that to differentiate it from the binary minus and it can be a simple number. Suppose this is the grammar that is given so, we know how to general generate the corresponding parsers. So, that we have already learned. Now, for syntax directed translation like we want to evaluate an expression. So, how to do this? So, we assume that with this with a non-terminal E so, we have got an attribute which is E dot val. So, which will hold the value of the expression.

So, if the if the value of the expression if it is a since this is a number as you note that this is not an id so, this is a number. So, at the compilation stage itself we will be able to compute the value of the expression.

So, it does not need to go to the runtime thing. So, that is why so, it is useful like you maybe while writing some expression we get we have written like 2 into 3 plus 5. So, for the programmers is so, it is written like this, but the compiler can easily convert it into the number that is 11. So, how that conversion from this; so, this is coming in the source text and how it is getting converted to 11. So, that you can do by means of this syntax directed translation mechanism and we assume that we have got an attribute val. So, which will be holding the value of the expression.

So, you can very easily guess like what can be the syntax director translation schemes for this thing like say E producing E 1 or E 2 ok. So, in this case the action is nothing, but so, I have to compute this E dot val equal to E 1 dot val plus E 2 dot val. So, this simply maybe the action. So, because those two values are to be added so, E 1 dot val that attribute contains the value of the expression E 1 and E 2 dot val contains the value of the expression E 2. So, I can just add them by making this E 1 dot val plus E 2 val plus.

Similarly, this E 1 minus E 2 so, E producing E 1 minus E 2. So, there also you can write the sentence directed translation scheme like E dot val equal to E 1 dot val minus E 2 dot val fine. Then E producing say E 1 star E 2; so, there also you can write like E dot val is E 1 dot val multiplied by E 2 dot val. Then E slash E division E slash E so, this E dot val equal to E 1 dot val divided by E 2 dot val; then unary minus E 1. So, this I can say that E dot val equal to minus of E 1 dot val.

And E producing number so, within bracket E, E producing within bracket E E 1. So, this is very simple that E dot val is nothing, but E 1 dot val and then I should have this E producing number. So, here this E dot val will be equal to the lexicon analyzer is, if we assume that it has written the value of this token number on to the into one attribute of it which is which is num dot val. So, this will be equal to num dot val ok. So, you can check like how it will work, like if I have got an expression like say 2 plus 3 into 5 slash 9. So, if we have got something like this then say plus minus 5 ok.

So, how will it look like? The expression the parse tree will be like this. So, this is E plus E, this E gives me within bracket E, this E gives me unary minus then E, then this E

gives me number which is equal to 5. Then this part will be giving me E slash E and this slash E. So, this E gives me an gives me a number whose value is 9. Now, this E will be giving me E sorry within bracket E like that and that will give me E star E. Now, this E will give me a number which is equal to 5; now this E will give me within bracket E. So, that will give me E plus E. So, that will give me number that is 2, this will give me number that is 3. So, this is the parse tree that will be produced by the parser.

Now, this syntax directed translation rules so, they are applied when we are doing the reductions. So, when the shift reduce parser so, it will do the reduction. So, it will first do this reduction. So, this as a result the corresponding val attribute will be equal to 2 by this rule. So, this E dot val equal to num dot val. So, these val equal to 2. So, these val will become equal to 3.

Now this will be doing the addition E 1 dot val plus E 2 dot val. So, this value will become equal to 5, now this is within bracket E so, this will also become equal to 5. Now this will make it 5 so, now this is 5 into 5. So, this will make it this value will be made equal to 25. So, this will also be made equal to 25 because, E within bracket E is E dot val equal to E 1 dot val by this rule this is also 25.

Now, this when this reduction is done so, this E dot val will be equal to 9. Now, when this reduction will happen so, 25 by 9; so, if I assume it is an integer division so, this will give me 2. So, this E is E dot val attribute will be equal to 2. Now this side so, this E dot val equal to number dot val that is 5 then this unary minus so, this will become minus 5.

So, this will give us minus 5 and finally, this E dot val so, it is 2 plus minus 5 that is equal to minus 3. So, it will get that E dot val computed as minus 3. So, in this way you see at the syntax directed translation mechanism so, we can write down a set of actions. So, that if those actions are taken during the parser reduction process; so, it will be able to compute it will be able to do many useful computation.

So, in this particular example so, it is doing it is for finding the value of integer expressions and previously we have seen a technique by which we can convert this regular expression to NFA. So, that conversion so, we can I have appropriate functions written so, that I can easily get the corresponding graphs. So, this way this syntax directed translation can be useful for in the parsing process in in in integrated fashion with the parsing process so, that you can do many useful jobs done by the parser itself.