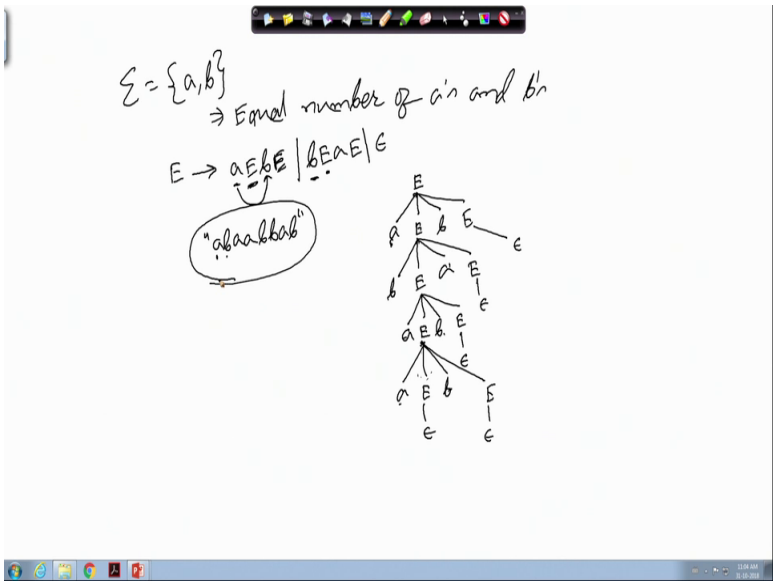
**Compiler Design**
**Prof. Santanu Chattopadhyay**
**Department of E & EC Engineering**
**Indian Institute of Technology, Kharagpur**

**Lecture – 23**
**Parser (Contd.)**

So, next we will be looking into some more examples of this grammars and how to construct the Predictive Parsing table for that.

(Refer Slide Time: 00:28)




So, the next example that we look into is the it is the sigma set the alphabet set is having the symbols a and b. And then we would like to construct language the grammar for the language such that it has got equal number of a's and b's in it. So, the language is having all the strings that have equal number of a's and b's ok

So, for that purpose, so you can write down the grammar like this. So, if I have that non terminal E, it produces a E b E, or b E a E or epsilon. So, here by E we represent the strings that have got equal number of a's and b's. Now intuitively you can understand that the string can start with a, or can start with b. So, if it starts with a then the situation is that there should be a matching b which correspond to that ok.

So, in between we have got some string. So, that have got equal number of a's and b's and then after that the next b coming matches with this a. And after that we can again

have a substring that has got equal number of a's and b's or the string can start with a b and then there can be substring having equal number of a's and b's then the character a. So, that the symbol a so that this b matches with this a and then we again have a substring that has got equal number of a's and b's in it or epsilon. So, this is a possible grammar for that.

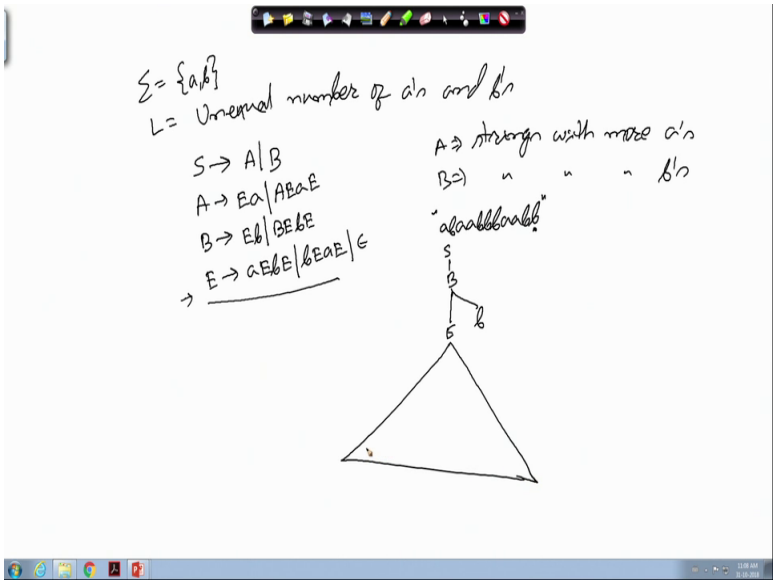
Now, let us see how can we make a string out of that how can we have an example for that. Consider the string say a b a a, then b b a b. So, it is expected that we will have this string should be accepted because there are 4 a's and 4 b's. So, the way the parse tree can be constructed is since its starts with a, so we have to have this left most symbol as a. So, you have to follow this rule b and E.

So, we have got this matching a's and b's. So, this a is here and this b is, this b is this one. So, this way we can have it like this or we can so we can match this b is some so somewhere else also. So, let us see that one possible parse tree may be like this. So, we make it like b E a E. So, that this a matches with this b and this E gives me epsilon and then this E gives me again equal number of a's and b's that part. So, this is a E, and b E where this E gives me epsilon, and this E gives me also epsilon.

So, we have got a b a then again a then. So, this is this E can give me epsilon ok, so number of a's we have is 1, 2, 3, 4. So, here we have got how many a's 1, 2, 3 a's. So, there should be one more a. So in fact, this E so this E should be giving me another derivation should be there. So, they should I have to follow another break here. So, this is a E, b E and then this a. So, this E should give me epsilon this a should give me epsilon.

Now, we have got 4 a's 1, 2, 3, 4 a's and there are 4 b's 1, 2, 3, 4 b's are there. So, that way so this parse tree can give me a proof that this particular string is a belonging to the language having equal number of a's and b's in it. So, that is one example. So, next example that we will take is where it is just the complement of the other one.

(Refer Slide Time: 05:35)




So, the sigma the alphabet set remains like a b, but the language that that will have unequal number of a's and b's; unequal number of a's and b's. So, it is just the complement of the previous one; so, for doing this so the grammar that will construct will be something like this. So, we will take the S as the start symbol of the grammar. So, this gives me S equal to A or B; where A stands for the situation where we have got more a's than b's in the string. So, A this will represent strings with more a's and B will represent the situation with first strings with more b's ok.

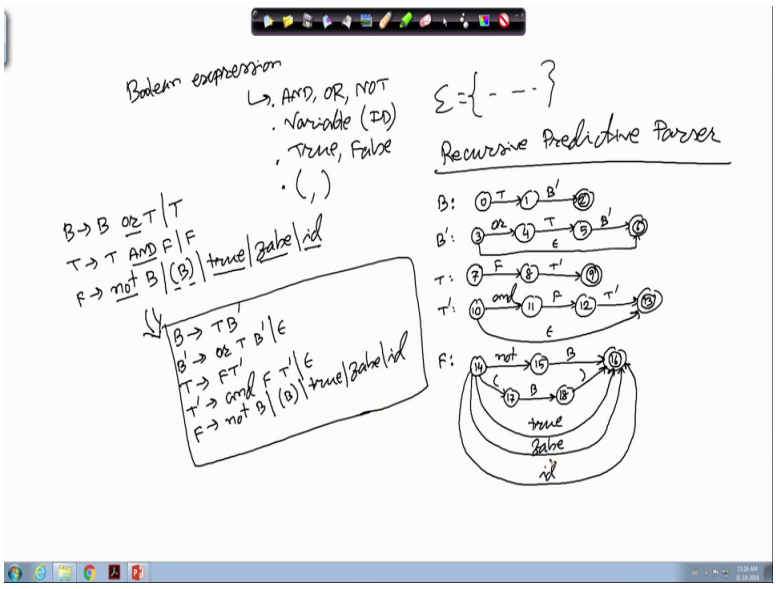
Now, so for A the derivation can be E a, where E is the strings having equal number of a's and b's or it may be A E a E ok. So, it may be equal number A and B followed by A that is we have got one more a here, or it is having more number of a's more number of a's and b's then equal number then one more a and then equal number of a's and b's, so one extra can come.

Similarly the B can be written in a similar fashion like E b or B E b E, where E is what we are written previously having equal number of a's and b's. So, this is a E b E, or b E a E or epsilon ok. So, this is the grammar you can try to derive some string that has got unequal number of a's and b's in it**.** So, let us try to see the derivation for the string say a b, a a, b b b a a, b b. So, this has got a's 1, 2, 3, 4, 5 and there are 1, 2, 3, 4, 5, 6 b's. So, there are 5 a's and 6 b's. So, they should be acceptable by the grammar ok. So, let us see how it is the how the derivation is done. So, S gives S can give a.

So, it has got more number of b's than a's. So, it should follow S producing B and then this S producing B and then we can allow this rule like it can have equal number of a's and then b's. Then this equal number of a's equal number of a's and b's, so this is last b is has been derived. So, this equal number of a's and b's so that can be derive from here. So, this is the E part.

And now we can just follow this particular derivation to get it, so the way we have done in the last example ok. So, you can same set of rule has been used here. So, you can see that we can have a derivation tree below this that will give me the string that has got equal number of a's and b's in it. So, this way we can construct the corresponding parse tree and giving the proof that this is having equal unequal number of a's and b's in them. So, next we will be taking one example which is towards a predictive parsing and it says that it is the you have to construct the grammar for Boolean expressions.

(Refer Slide Time: 10:01)




So, Boolean expression so it can have the operators like AND, OR, and NOT. It can have some variable or identifier and there are some constant values true and false, true and false and they definitely there can be parenthesis. So, there can be open parenthesis and close parenthesis. So, these are the various symbols that we can have in the terminal set for the Boolean expression. So, this sigma has got all these symbols in it.

Now, we can try to construct the corresponding grammar. So, the grammar for Boolean expression will be like this [laughing] B producing B or T or T; T giving T AND F or F;

and F can give me not of B or within bracket B or true, false or I d where these are all terminal symbols, OR, AND, NOT open parenthesis close parenthesis true false id.

So, these are all terminal symbols and B T and F they are non terminals. So, if you try to construct the predictive parsing or say recursive descent parsing in general the top down parsers then the first thing that I have to do is to convert it into a grammar. So, that there is no left recursion.

So, here we have got left recursion. So, that left recursion has to be eliminated. So, the left recursion can be eliminated by applying the that set of rules that we had done previously, B producing T B dash, B dash producing or T B dash or epsilon, T producing F T dash. And then T dash producing and F T dash or epsilon, where F there is no left recursion for F so this remains unaltered not of B within bracket B or true or false or id. So, this is the grammar that we have got finally, for the top down parsing.

So, next we will be look into that recursive version of the predictive parser, recursive predictive parser. And for this recursive predictive parser we know that we need to make a set of transition diagrams. So, the transition diagram for B will be like this. So, this is the state number 0, from here getting T it comes to state number 1, from here getting B dash it goes to state number 2 and that is a final state.

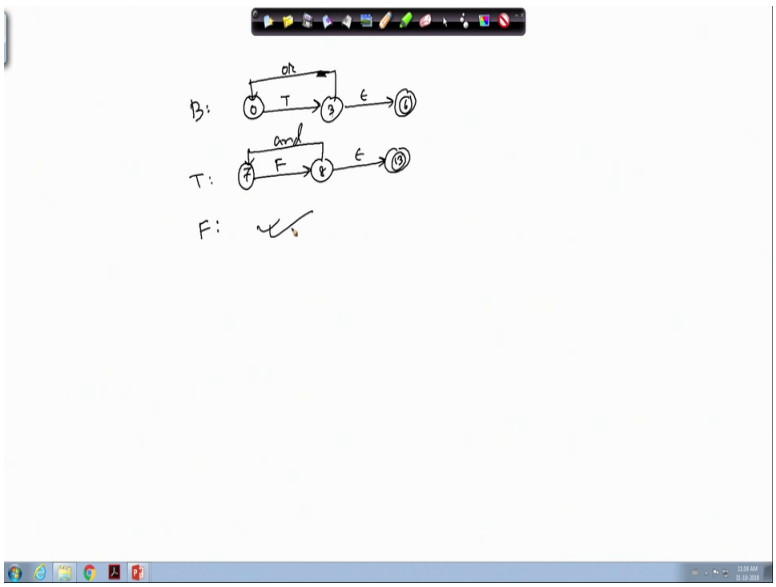
Similarly, for B dash we can have the transition diagram like this. So, up to 2 we have done. So, this will start with 3 on getting or it will come to state 4, and getting T it will go to state 5. And from here getting B dash it will go to state 6 and there is an epsilon production. So, we will have another epsilon transition added like this. So, this is the epsilon transition.

So, that is about these two things that is we have got this B and T next for T dash for T dash we can have F T dash. So, this will start at state number 7 with F it will come to state number 8 and on T dash it will come to state number 9 and that is a final state for T dash. Then we have got so this is for T sorry this is for T. Next we will do it T dash, so T dash is and F T dash. So, that is state number 10 from 10 on and it will go to state 11 and then F it will go to state 12 and then T dash it will go to state 13 and 13 happens to be a final state and then we will have a epsilon transition from 10 to 13.

And finally, we have got F we have got F and then this F will start at state number 14 and from 14 we can have on not it goes to state number 15. And from 15 on B it will go to state 16 and 16 is a final state. Then this open parenthesis so this will take it to state number 17, then B it will go to state number 18 and then close parenthesis. So, it will take it there.

Then true false and id, so these transitions are to be added. So, this is for true, this is for false, and this is for id. So, you have constructed the transition diagrams for this particular grammar. Now we can we can use this transition diagrams for doing the parsing job. However, we can do some simplification also as we have done previously with other examples. So, this after if we do the simplification as we did with the E T F grammar. So, if we do the simplification then we will get a refined set of transition diagrams.

(Refer Slide Time: 17:50)



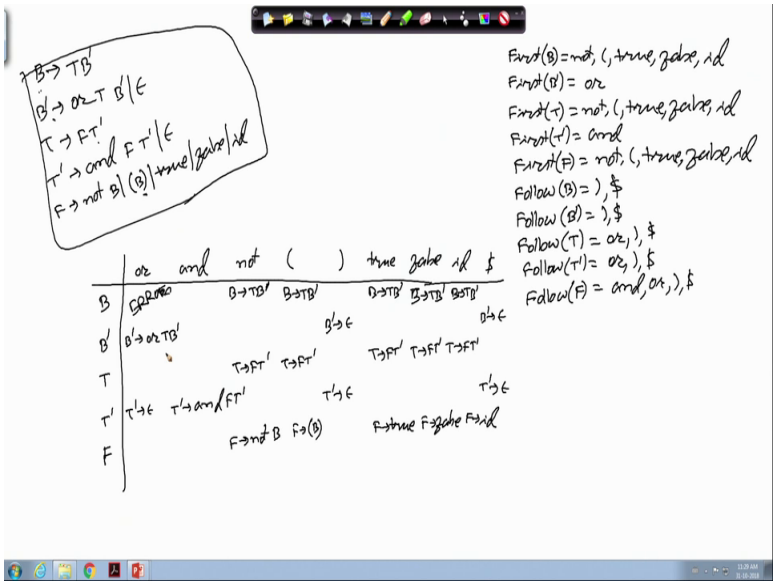

And I am just drawing the final set of transition diagrams you can just check it that is that B transition diagram will be something like this on from on state from 0 on T it will come to state number 1. And from state number 1 so this will this is actually this is a state number 1 will get eliminated and what will remain is the state number 3 ok.

And from here on or it will come to state 0, this is on or and then on epsilon it will go to state 6, for T the transition diagram will be something like this. So, it will start at state number 7, on F it will go to state number 8, from state 8 on getting and it will come back

to state 7. And if it gets epsilon then it will go to state 13 fine. And the F diagram will remain unaltered. So, whatever we had previously so the F diagram will be remaining as it is. So, this is the final set of diagrams that we will have for these things ok.

So, next we will be trying to make the predictive version, non recursive predictive parsing version. So, that will be based on this l l philosophy, so that l l predictive parser. So, we will try to make so for that the first thing that we need to do is to write down the fast and follow sets ok.

(Refer Slide Time: 19:43)




So, for the sake of our understanding; so, I am just writing the grammar once more. So, I have got B producing T B dash, then B dash producing or T B dash, or epsilon, then T produces F T dash, then T dash produces and F T dash, or epsilon. And this F has got not of B within bracket B true false and id. So, this is the grammar that we have. So, first thing that I have to do is to compute the first and follow sets for the individual non terminal symbols. So, the first of B first of B is equal to first of T, the first of T is equal to first of F and first of F has got not then open parenthesis true, false, id.

Now, first of B dash, first of B dash so; this will be having only or then first of T first of T equal to first of F. So, this is the same set that we have written previously true, false, and id. Then first of T dash; first of T dash will have and in it and this first of F will have all these rules all these symbols, not open parenthesis true, false and id. Now, what about the follow sets? So, follow of B equal to so follow of B you see that B can be followed

by close parenthesis. So, it is close parenthesis is there and B is the start symbol, so dollar is there. So, what about follow of B dash follow of B dash?

So, if you have this rule B producing T B dash. So, by the second rule you know whatever is in follow of B will be in follow of B dash. So, follow of B has got close parenthesis and dollar. So, this will also have close parenthesis and dollar then follow of T follow of T. So, by this rule whatever is in first of B dash will be in the follow of T.

So, first of T dash has got or, so or will be there and then follow of t. So, since so by so follow of T by this rule B dash can give me epsilon. So, whatever is in follow of B will be in follow of T, because B dash can give me epsilon. So, follow of B has got open close parenthesis and dollar. So, those two symbols will be there close parenthesis and dollar.

So, similarly you can find out that follow of T dash will be there equal to or close parenthesis and dollar. And follow of F follow of F so it is equal to first of T dash by this rule or this rule. First of T dash has got and so and can be there then follow of F will have. So, whatever is in say not of B, so whatever is in follow of B will be in the follow of whatever is in follow of B will be in the follow of whatever is in follow of F will be in follow of B. So, that is so whatever is in follow of B can be in the follow of F.

So, by the applying the rules so you can see that we can see that this and can be there then this or actually by this rule. So, you can so, whatever is in first of T dash we have taken it here and whatever you have what by this rule whatever will be in the fallow of F will be in the follow of B. So, that way so and by this 1, so this T dash producing and F T dash. Now you see that T dash can give me and so that and has come in the follow of F. And the other symbols that can come in the follow of F is basically this or then this close parenthesis and dollar. So, these symbols can come in the follow set of F.

Now, once we have constructed the first and follow sets. So, we can try to make the corresponding parsing table. So, the parsing table that will be making it will have entries like B, B dash, T, T dash and F on the on the as per as the rows are concerned. And on the columns will have or, and, not, open parenthesis, close parenthesis, true, false, id and dollar.

Now, we have to apply individual table construction rules and see like what can go which, what can go where. So B producing T b dash, so whatever is in follow of T first of T B dash there I have to apply with this particular rule and first of T B dash is equal to first of T. So, not then this open parenthesis, true, false, id, so they will have this rule. So, B producing T B dash, this rule will come here then it will come here also ok. And B producing T B dash it will come there also ok.

So, then we can have say this rule. So, if we if we say the next see the next rule B dash producing or T B dash. So, or T B dash the first is having or, so B dash or will have this rule that B dash produces or B or T B dash. So, this rule will be there now B dash producing epsilon is there. So, whatever is in follow of B dash there I have to add this particular rule. So, B dash follow set has got this close parenthesis. So, here I should have B dash producing epsilon and here I should have B dash producing epsilon ok.

So, if you do this way then for T producing F T dash. So, first of F first of F has got all this symbols. So, there I should put this T producing F T dash. So, T producing F T dash comes here then open parenthesis. So, T producing F T dash comes there, then this true, false, and id. So, T producing F T dash T producing F T dash T producing F T dash. So, all of them come at this place and then you have got T dash producing epsilon.

So, this T dash producing T dash T T dash T dash producing and F T dash. So, this T dash, so T dash and so this should have this rule T dash producing and F T dash and T dash producing epsilon in the follow set of T dash we have got or. So, there I should put this rule T dash producing epsilon then close parenthesis. So, T dash producing epsilon and dollar.

So, there also I should have T dash producing epsilon then F producing not B. So, this rule should be added here F producing not of B, then F producing within bracket B. So, this should be added here because the corresponding first set has got this thing has got the open parenthesis. Similarly, F producing true should be added here, F producing false should be added here in the false and F producing id should be added there ok.

So, this completes the table formulation and whatever entries are undefined. So, they are parsing errors. So, they are all error entry so all these are errors ok. So, you can formulate parsing tables like this and from there we can go to the parsing process to see how this parsing can be done for different symbol not a string.