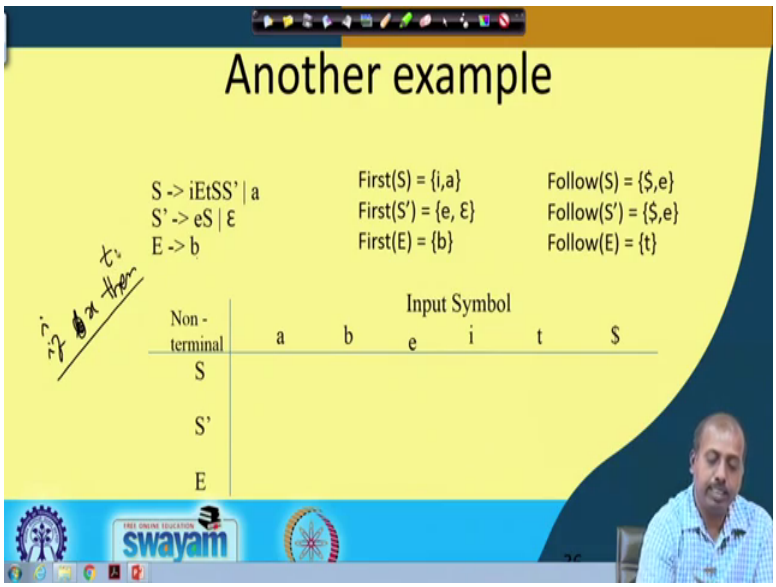
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 22
Parser (Contd.)

(Refer Slide Time: 00:14)



You take another example of this predatory parsing table construction. So, this is the grammar that we have that is S producing this is basically that if then else grammar. So, i for if so, i is a terminal symbol t is a terminal symbol like that i is the, if t is then and E is expression so, like that. So, we have got. So, we the expression part we make it b for the sake of simplicity. So, that is also taken as a terminal. So, to make the parser small. So, we have taken as if I can have only this sort of thing I can have if then the expression is some x then we have to write like this and this if is also i then is t. So, like that anyway so, this is the grammar.

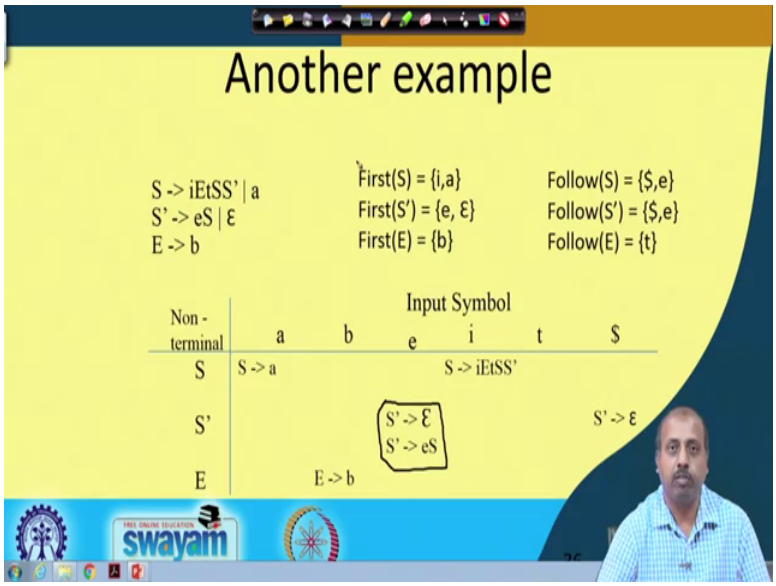
Now, so, first we have to for constructing the predictive parsing table the first thing that we have to do is to compute the first and follow sets. So, first of S, if you go by this rule and so, the first rule. So, it says that I will be there in the first of S by the second rule it says that a should be there in the first of S similarly first of S dash it will have e and this epsilon. So, these two are there and first of E will have b.

So, first computation is straightforward now the follow computation. So, follow S is the start symbol of the grammar. So, dollar must be in the follow of S now in the whatever is so. So, now, this follow of E follow of E is t. So, from this rule you can see that e may be followed by E t. So, that will have that can have t. Now what if so, by now we can now look into this part? So, S S dash. So, S S dash means whatever is in first of S dash can be in the follow of a.

So, first of S dash has got e and epsilon. So, epsilon we cannot take, but e you can take. So, follow of S follow of S can be e. So, follow of S we have added e and dollar was already there and by this rule is producing i S. So, often i E t S S dash so, whatever is in follow of S can be in the follow of S dash. So, follow of S has got dollar and e. So, follow of S dash will also have dollar and e.

Now,. So, this way we can go on arguing like what are the elements that can be there in the follow sets and apply those rules for all the productions and ultimately when the set does not change you stop at that point. So, we so, you can check that after this no more symbol can be added to the follow sets of the non terminals and once we have done that. So, we can try to construct the, this predictive parsing table.

(Refer Slide Time: 03:26)



The first thing that we have is say this rule S producing a. So, this rule says that the, whatever is in first of a they are I have to add this S producing a. So, first of a is a only. So, S producing a is added here similarly from this side it says that whatever is in first of

this i E t S S dash there I have to add this rule. So, first of this is i only so, this is added here S producing i E t S S dash. So, that is done so, first rule is done.

Now, the second rule it says that S dash producing e S. So, S dash producing e S it says that whatever is in first of e first of e S there i have to add this rule. So, S dash produces the first of e S is e. So, S dash producing e S is added here. Now S dash producing epsilon is there. So, that says that whatever is in follow of S dash there I should follow of put this rule S dash producing epsilon. So, follow of S dash has got dollar and e. So, in dollar i put S dash producing epsilon and in e also i put S dash producing epsilon.

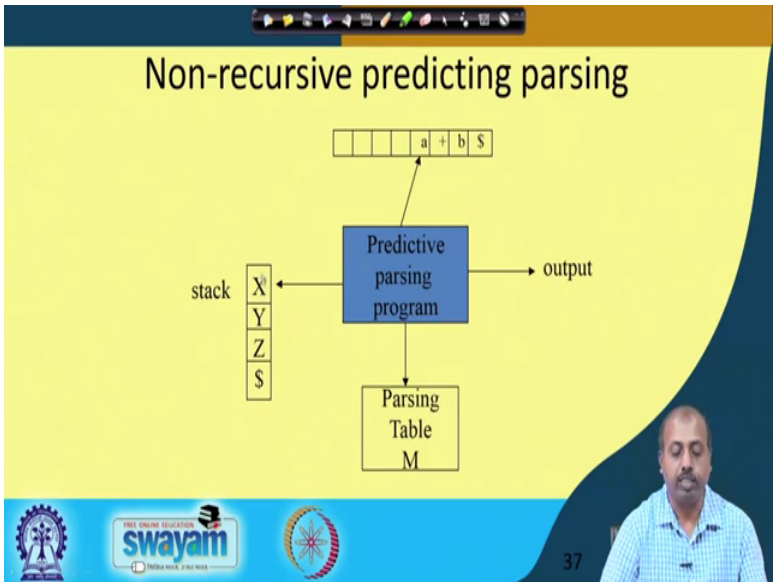
And then this e producing b and b has got those on the first of e contains first of b contains b only. So, there I add this rule the e b has got e producing b this particular rule. Now look into this this situation like for this particular cell. So, what has happened is that there are two rules and by which we can proceed. So, this is actually happening because the grammar that we have taken is that that if then else grammar and this if then else grammar it is ambiguous grammar.

So, that is why. So, this situation has occurred. So, it says that on S dash. So, if you have seen up to S dash then if you if you currently seen S dash now if you see an e part then if you see an e part then one action says that you reduce this S dash by epsilon. So, that this S dash part does not come. So, it is taken as a is the previous statement is taken as if expression then statement.

The second rule is telling you take it you expand it as S dash producing e S. So, that the expression is taken as the statement is taken as e I if condition your, if expression then statement else statement. So, it is telling that you follow the second option. So, both of them are correct as far as the parse tree generation is concerned because this is an ambiguous grammar. So, we can have this type of problem.

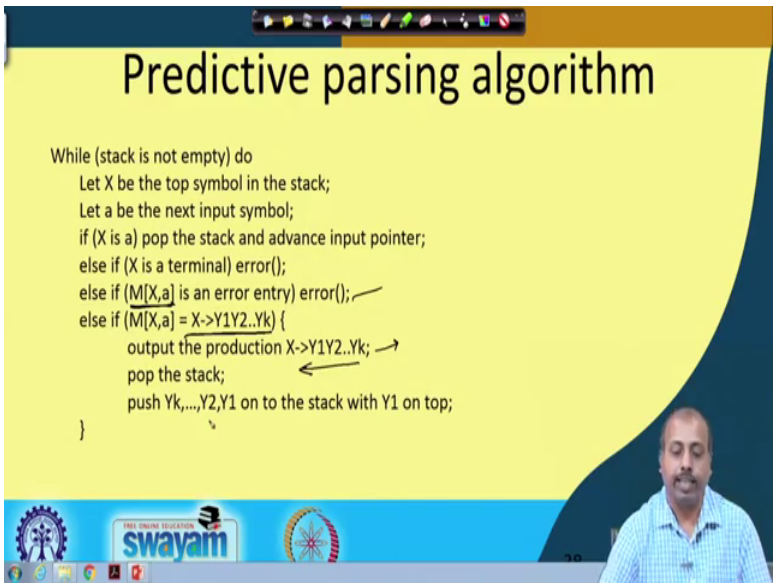
So, so, as I was telling so, once you construct the predictive parsing table. So, if the if some of the entries are multiplied defined, then the grammar is not l l 1 grammar so; however, for many grammar. So, it may be that case that they are l l 1 particularly the expression grammars and also they are l l 1. So, we can design parsers for using this l l 1 policy otherwise the policy is very simple accepting that follow computation part. So, remaining part remaining parts are quite simple.

(Refer Slide Time: 06:47)



So, how do we use it this predictive parse cells? How the how actually the parsing takes place? So, it has got a few components in it one is the stack. So, stack is a stack of non terminal symbols and it there is a dollar here and then we have got these production rules sorry we have got the input and then we have got a parsing table in. So, all of them come to this predictive parsing program and it produces output whether it is correct or not. So, stack can contain both terminals and non terminals. So, this XYZ they are actually grammar symbols. So, they can contain both terminals and non terminals.

(Refer Slide Time: 07:30)



So, how do we proceed? So, so, we look into the top of the symbol at the top of the stack. So, let the symbol be X and we also check the next input symbol suppose it is a. So, like the situation like top of the stack contains X and the next input symbol is a in that situation if X is a. So, if this top of the stack is a terminal and that is equal to the next input symbol then we pop out it from the stack and advance the input pointer.

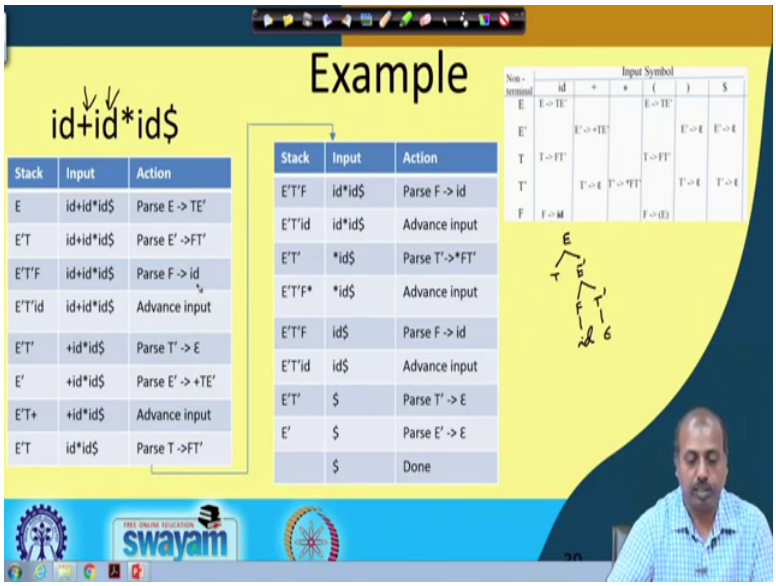
So, if this X happens to be equal to a then this is popped out from the stack and input pointer is advance to the next position. As if there is a match between what we expect on the string so, that is available in the stack and what we actually get on the input string. So, since these two are matching so, we are just adverb we are just consuming that input and adopting out the symbol from the stack and advancing the whole parsing process.

So, that is their else if X is a terminal then error that is if the stack of top of the stack contains a terminal and it does not match with the terminal that you have as the next input ; that means, so, there is an error. So, you are expecting a. So, you are expecting a say b here and you are getting a, at the next input. So, there is an error so, that way it is that we it will be an error condition. So, if so, the so, we have excluded the possibility that the top of the stack contains a terminal symbol. So, if the terminal matches if the input it is fine if the terminal does not match with the next input then that is an error.

So, what remains is the top of the stack contains a non terminal. So, in that case we need to consult this particular entry MX a. So, the parsing table will be consulted and then if this MX a is error is an error entry. So, if the no rule is defined in that case the parser will also tell error. Otherwise if the rule if the entry says that entry says has this particular rule. So, X producing Y 1 Y 2 Y k, then we will output this production rule. So, for the sake of telling that by which rule, we are progressing like that.

So, we will be printing this particular rule will pop out X from the stack. So, X is taken out from the stack and these symbols are put into the stack this it is in it is in the reverse order. So, Y k Y 2. So, Y k Y k minus 1. So, that way they will be pushed so, that y one is on the top of the stack. So, that way this parser will proceed ok. Ultimately when your top of the stack will contain dollar and the input will also pointing to a dollar; that means, we have reached the end of the string and the whole parsing process was correct.

(Refer Slide Time: 10:33)



So, we will see some example by which we can do this parsing consider that that each expression grammar that we have taken. So, this was the parsing table that m table that we have thought about now suppose we have got this particular string to be checked whether it is syntactically correct or not and we have to see how the parsing can proceed. So, we start with i d plus i d star i d so, this is the explained input.

So, that my input pointer is somewhere here and the top of the stack we put the start symbol of the grammar. So, the stack top has got E in it. So, as per our rule is concerned. So, it says that you have to check so, we have to check if X is a terminal do something in this case X is not a terminal because top of the stack contains E which is a non terminal. So, X is not a terminal so, we have to see the corresponding entry in the table MX a.

So, this MX a that is ME i d. So, ME i d is E producing plus TE dash. So, what will happen the action is we have to pop we have to take out this E from the stack. So, we have to take out E from the stack. So, the stack had e. So, this e is taken out of the stack now this TE dash. So, they will be put into the stack. So, that E dash is at the bottom and T is at the top. So, this whatever symbol comes first. So, that will be at the top.

So, this T E dash. So, so, here this stack is shown. So, this is the bottom this side is bottom and this side is top. So, top of the stack now contains T and input pointer is fixed at i d only. Now I have to consult T i d, now if you say T i d. So, it says T producing F T

dash. So, this T will be popped out from the stack and T dash and F they will be put into the stack they will be pushed into the stack.

Now, it contains a from the top of the stack and i d as the next input symbol. So, F i d says that you follow this rule F producing i d. So, F is taken out of the stack and i d is put into the stack. Next time the parser takes that top of the stack contains a terminal which is i d and the input pointer is also at i d. So, these two are matching so, this is taken out of the stack and the input pointer advances. So, input pointer now comes to this one the plus the input pointer now comes to the point plus.

Now, so, this is the situation so, input is like this. Now T dash and plus. So, T dash plus it says that T dash producing epsilon. So, right hand side replacement is done. So, nothing has to be pushed into the stack because it is the epsilon. So, T dash is going out of the stack input remains as it is now E dash and plus. So, E dash and plus says that you go by E dash producing plus T E dash. So, E dash is coming out of the stack and plus T E dash they are put into the stack, next this plus is checked and it matches with the next input symbol plus.

So, these two are matching so, they are taken out of the. So, this is taken out of the stack this is input pointer is advanced. So, that input is now pointing to the i d so, it is now pointing to the i d. So, we have got i d star i d dollar. So, this part is remaining and then this T and i d. So, T and i d it says that you follow T producing F T dash. So, T is going out and F T dash is coming into the stack input remains as it is.

Now, F and i d says that you go back F producing i d. So, this F is taken out an i d is pushed into the stack then i d i d matches. So, it is taken out of the stack and input pointer advances so, that this is left as the input. Now T dash and star so, T dash and star so, it goes by this rule T dash producing star F T dash. So, it is that it is replaced it S is replaced by star F T dash now star and star will match in the next iteration of the loop. So, they will be taken out so, it will have E dash T dash F and i d. So, F and i d says F producing i d. So, it is replaced F is replaced by i d now i d i d they match ok. So, they are i d is taken out of the stack this i d with the input pointer is an advanced.

So, now you have got a situation T dash and dollar. So, T dash dollar says T dash producing epsilon. So, this will be replaced by T dash producing epsilon. So, T dash will go out of the stack now you will have E dash and dollar. So, E dash and dollars is E dash

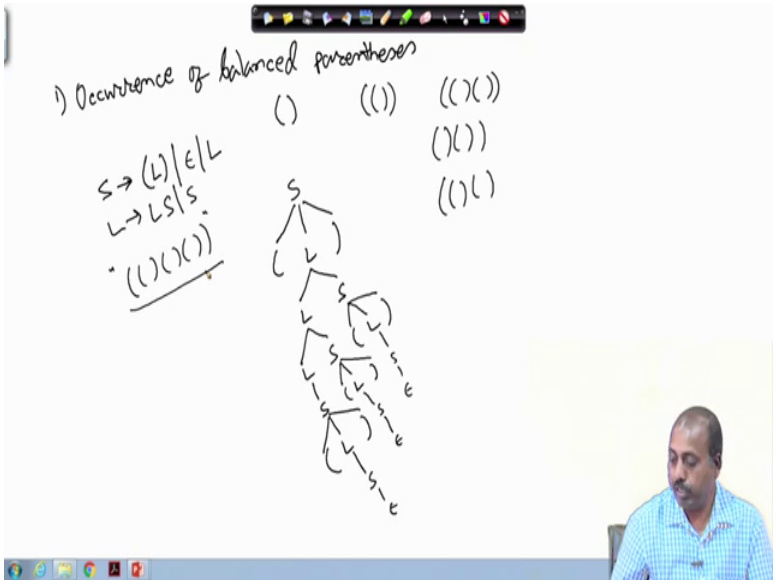
producing epsilon. So, that also goes out and then the input pointer is also a dollar. So, stack is empty and input pointer is also containing in the pointing to dollar symbol. So, that is the end of the string. So, we could successfully parse the input sequence.

So, this is the action in the action part we have purposefully written like how do we proceed and from this you can construct the parts tree. For example, so, you can if you follow these action sets first it says that parts by E producing T E dash. So, if I try to draw the words tree first I will do it like this T E dash next it says the parts using E dash producing F T dash. So, this is this is F T dash then it says that you follow rule F producing i d. So, F producing i d then advance input so, no action then it says T dash producing epsilon.

So, follow this rule T dash producing epsilon. Then it says that parts using this rule that. So, this is no actually from this it is slightly difficult. So, let us take some other example. So, these are the actions. So, these actions will tell us like how it will proceed, but from there constructing the parse tree your directly will be a problem ok. So, constructs we will take some example and then explain it again.

Now, so, we will take a few examples to work out and we will see how can we make a few grammars and we can write a few grammars and see like what we can how we can construct some parsers different types of parsers and all. So, for that purpose so, let us first try to write a few per gram example grammar.

(Refer Slide Time: 18:17)

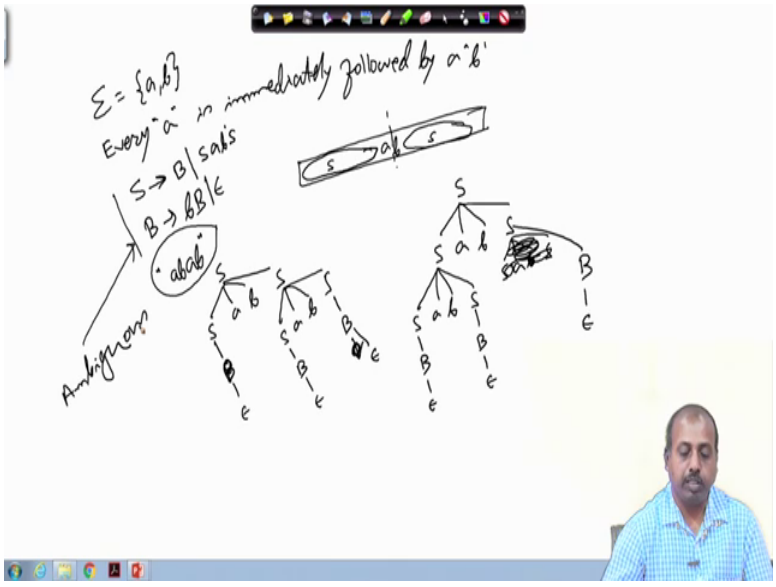


The first example that we write is occurrence of balanced parenthesis occurrence of balanced parentheses. So, balanced parentheses means so, this is a set of balance this is a balanced parenthesis then this is also a balanced parenthesis then this is also a balanced parenthesis but only got unbalanced so, maybe if somebody is writing like this. So, this is unbalanced or somebody may write like this ok. So, this is also unbalanced.

So, this way we can have some unbalanced parenthesis. So, how can I write a grammar for this purpose? So, the grammar for this balanced parenthesis is like this. So, S producing within bracket L or epsilon or L and this L gives L S or S ok. So, for example, suppose we try to derive the string some parentheses parenthesized string like say this one suppose we want to derive this particular string.

So, we can do it like this S producing brackets start L bracket close and then this L giving me L S and from this L S we can say this L can again be made to give L S, this S can give me like this within bracket L. Then this L can give me S and then this S can be made to give S L bracket close then this is S and this can be made to give epsilon similarly this S. So, that gives us the first parenthesis the first part of the parenthesis, then this can give me bracket start L bracket closed and this L can give me S and that can give me epsilon similarly this L can give me S giving me epsilon. So, this way you can draw the parse tree for this particular expression.

(Refer Slide Time: 21:27)



Now, so, next we will be looking into another example where we have got we have got the alphabet set as a b got the alphabet set as a b and every a is immediately followed by a b. In the string in the language every a is immediately followed by a b. So, that is the language that we are looking into.

So, what are the possibilities that the string may be simply a string of B is or it may be S then a is followed by b and then any other string. So, S is the start symbol of the grammar. So, S can derive all strings, where a is immediately followed by b. So, what we can have either the string does not contain any B or I can say the left part of the string I can I can a. So, if this is the whole string as if I can break at a point where one of them is a and other one is b ok.

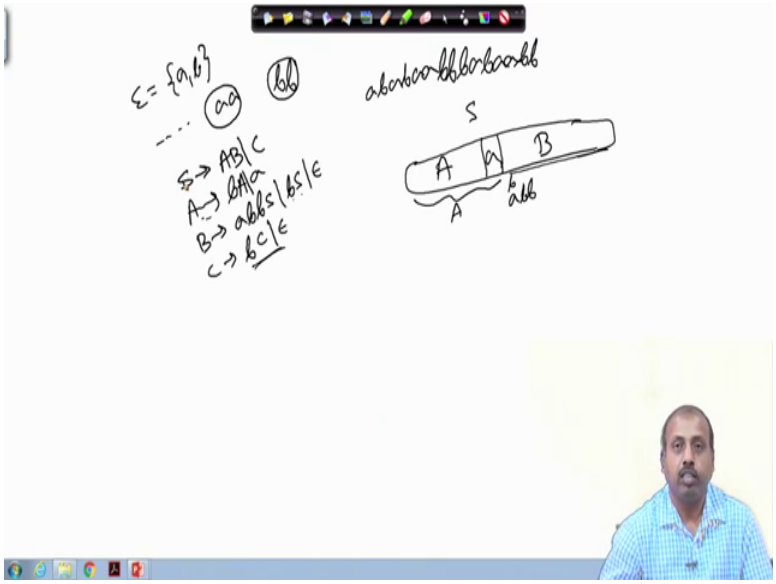
Now, what about this part? So, this part can be any string over a and b such that a is followed by b. Similarly this part also I can have any string of a and b such that a is immediately followed by b. So, we have got this thing that S. So, this is also type of S this is also type of S. So, we can write it like this is producing S a b S and where this B it can be any number of B is. So, that string can be generated by B or epsilon.

Now, let us look into a string a b a b if this is a string. So, one possible derivation for this is like this S a b S then this S gives me epsilon and this S gives me S a b S and then over. So, this S gives me sorry this is cannot give me epsilon. So, this can give me B and B can give me epsilon. So, this gives me B leading to epsilon this is gives me B leading then giving me epsilon.

So, that is one possibility; the other possibility that we have is like this S producing S a b S then this S can give me S a b S and this can give me B then epsilon this can give me B then epsilon and this can give me S a b S or sorry this is already there. So, this can give me B followed by epsilon. So, that way we can have we can have the two different parts trees for the same string a b a b. So, this particular grammar that we have written. So, this is an ambiguous grammar this is an ambiguous grammar.

However, ambiguous grammar. So, if I try to construct the l l 1 parsing table then naturally there will be there will be duplicate entries or duplicate values for some of the table entries. However, many cases it is permitted it is done because the table becomes smaller even if it is ambiguous the table becomes smaller. So, that way we can proceed.

(Refer Slide Time: 25:50)



Now, we will take another example where strings are over a and b. So, we will take an example where we have got this alphabet set as a b and such that any string that contains a a will be followed immediately followed by b b. So, in the previous example, that we took. So, was single a was followed by single b here any pair of a is will be followed by pair of b. So, you have got this type of strings a b a b a, but if you have two as then immediately there will be two bs again you can have a b you can have something like this, but as soon as you have two as then again two bs should appear. So, the language should be like that.

So, the production rules that we have for this particular grammar is like this S producing AB or C or A produces b A or a then B produces a b b S or b s or epsilon and C produces b c or epsilon ok. So, a c is broken up into two parts a and b such that. So, having the first this a part is having the first occurrence of a and then b is now the remaining part and this b must be giving me this thing it must start with the say next a and then b b s or so, this the S is broken also if this is the whole string. So, it is broken at the position where we have got the first a.

Now, the situation may be that after this a a b is occurring. So, then that is fine, but if it happens that an a is occurring then there must be two b. So, this so, this part we are taking as A so, this part we are taking as B. So, up to this much is taken as capital A and this part is taken as capital B. So, if it is after if in capital B we have got this thing. So, if

it if there is an a then this will be b b S on the other end or it may be that there is a single a so, there is no a.

So, there was a single a at this point taken into care of by this A rule and then it is simply another b can come and then S or epsilon and C can give the other part so, b c or epsilon. So, this is basically the strings of bs only bs that can come in. So, that is taken care of by this. So, this way this grammar is slightly complicated it requires some thinking; however, you will be you can see like how this grammars can be written. So, it requires some practicing so, you can just try out some more language specifications and write down the corresponding grammar.