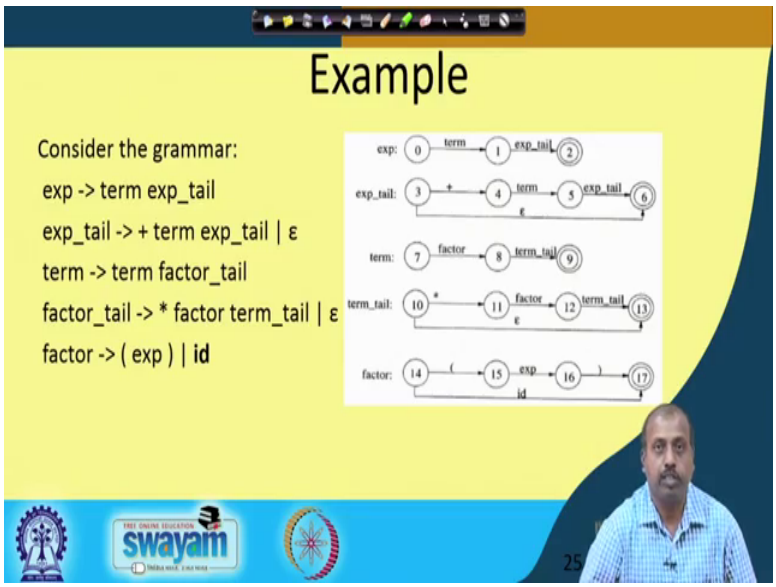
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 21
Parser (Contd.)

(Refer Slide Time: 00:15)



So, today we will first look into another example of grammar from which we can make transition diagrams for predictive parsing. So, the grammar that we considered is like this that it; it is similar to that expression grammar. So, that expression produces a term and expression tail where expression tail is plus then term then expression tail or epsilon. So, it is similar to that ETF grammar for we have eliminated that left recursion and we are writing it explicitly. So, T E dash we are writing like expression tail then T dash we are writing it as a factor term tail or factor tail like that.

So,; this term produces term or followed by factor tail and factor tail is star factor; then term tail or epsilon and factor produces within bracket expression on id. So, as the transition diagrams that we make is like this that first we have got this further expression. So, it is state 0 from there it goes on term it goes to state 1 and from there an expression tail; it will go to state 2 and state 2 is a final state.

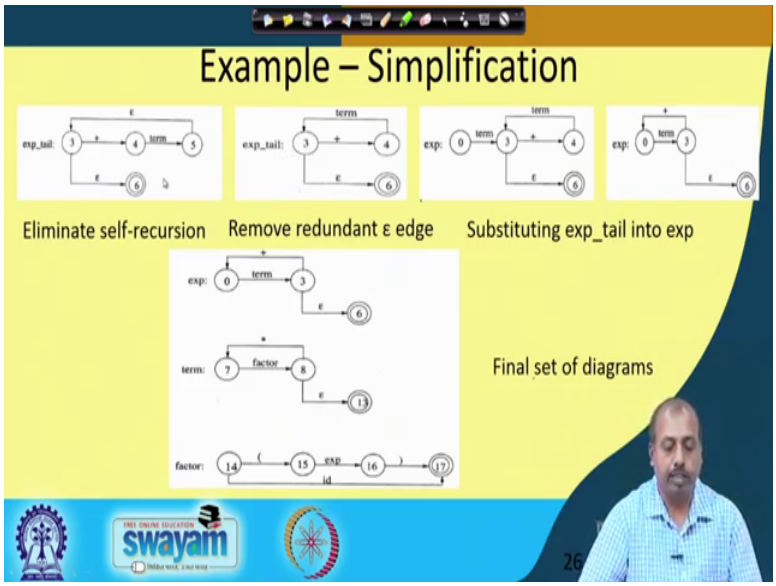
Then these expression tail; so expression tail produces plus term then expression tail epsilon or epsilon. So, it is like this that expression tail. So, it is starting at state 3; so, 1

plus it will go to state 4 and then on term it will come to state 5 and on from 5 on expression tail it will go to 6. And there is an epsilon production, so are from 3 on epsilon it will go to state 6. Similarly term produces term and then factor tail, so it is represented like this.

So, then this factor tail producing star factor term tail; so it is. So, it is term tail producing that is that is basically this is; this should be actually factor tail yeah this should be factor tail. So, this is producing like this the term tail producing star factor and term tail. So, this way this whole like expression is made and this factor produces within bracket expression and id.

So, with bracket start expression then bracket close or this is id. Now we can simplify this expression this set of transition diagrams by like this.

(Refer Slide Time: 02:47)



For example this expression this expression tail we are; we can eliminate the self recursion, like here you see in the expression tail; so it is calling the routine expression tail again; so instead of that from 5, we can come back to state 3.

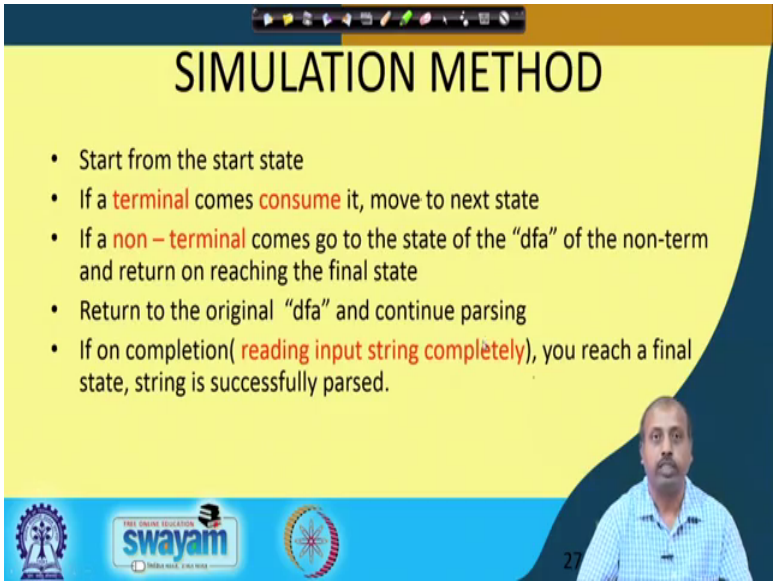
So, that is done here from 5 on epsilons it comes back to state 3. So, that is calling the expression tail. So, by this way we can and simplify the diagram a bit; we can and remove the redundant epsilon adjacent like this is a redundant epsilon edge. So, we can

all together remove this state 5 from state 4 on getting term I can come to state 3; so that is there that is possible.

Similarly, from then if we substitute this expression tail in to the diagram for expression; so in the diagram of expression was like this there we had expression tail now we have our modified the expression tail. So, we just substitute that expression tail diagram there. So, this is giving us expression the; this is giving us that this term then this expression tail diagram is a substituted; so, it is like this after that you can simplify.

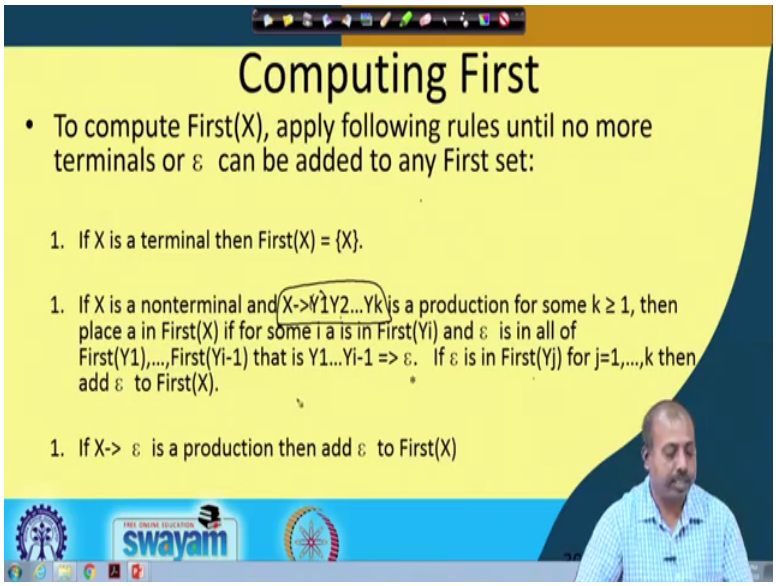
Like say from here on term it is going from 0 to 3 and then on plus and term it is coming back to state 3. So, instead of that on plus I can take it back to stage 0; so it is done like this and then on a epsilon it goes to state 6. So, that we can simplify the expression the; these diagrams. So, this is the final set of diagrams where we have got in the diagram for expression term and factor where we have substituted and eliminated the redundant edges and states and this is the final set of states. So, we can use this diagrams for purpose of predictive parsing; so, as we have seen in the last class.

(Refer Slide Time: 04:43)



So, we can do a simulation and then we can see whether a given string is acceptable by the set of diagrams.

(Refer Slide Time: 04:51)

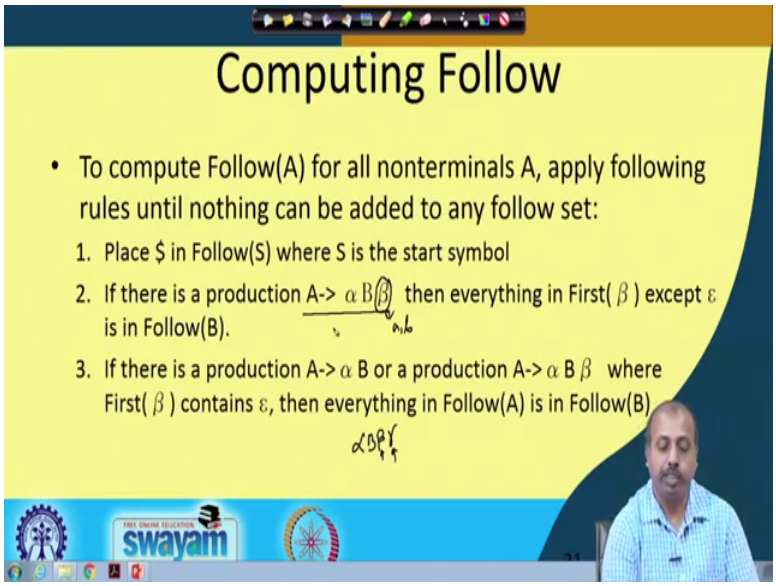


Next we were discussing on this first and follow computation because we want to have a non recursive version of this predictive parsing.

So, the first computation we have seen in the last class where we say that; so, if I have got a general production rule like this X producing Y 1, Y 2 upto Y k then so this whatever is in first of Y 1 will be in first of X accepting epsilon. And then whatever see if first of Y 1 contains epsilon then whatever is in first of Y 2 accepting epsilon will also be in first of X.

And so if all of them have got epsilon in their first Y 1 to Y k, then only epsilon will be there in the first set X. So, this way we can compute the first set and we have seen an example in the last class that shows us how to compute the first set.

(Refer Slide Time: 05:55)



So, today will next look into the computation of follow. So, follow means the terminal symbols that can follow a particular non terminal in any sentential form. So, sentential form we say that thing starting with the start symbol of the grammar.

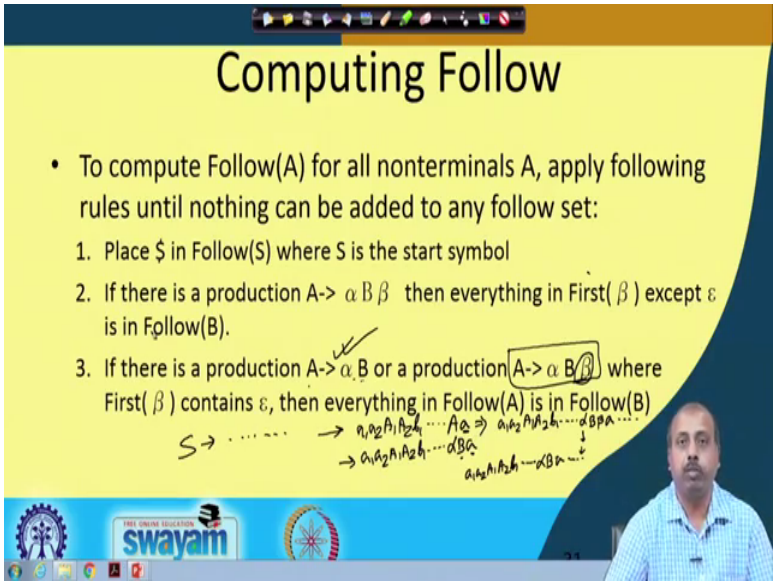
So, if you try to derive strings then all possible have to think about all possible strings of terminals and non terminals that can be derived from the start symbol of S; then in any of those strings if a terminals symbols appears after in non terminal symbols, then we say that the follow set of the non terminal symbol will have this terminal symbol. So, a typical; so the rule the you can understand that the rule that we are going to follow.

Like if this is a production rule A producing alpha B; beta then whatever I can derive from beta and whatever be the first set of that; so that their those symbols will be following B. For example, if first set of beta contains the symbols of a small a and small b, then small a and small b will be in the follow up B.

So, accepting epsilon of course, if epsilon we cannot take because in a may be the actual string that we have is a alpha B beta; then some gamma etcetera. So, even if this give me epsilon; so these first on gamma whatever symbols are there will be coming to the follow up B. So, this way we can we can compute the follow up B; so, whenever we have got a rule do like this; A producing alpha B beta. So, whatever is in first set of beta will be in the follow set of A; so, that is one rule. The other rule that we have is if there is a

production A producing alpha B or A producing alpha B beta. So, I will come to this part later.

(Refer Slide Time: 07:47)



I will come to this, but later; consider this rule A producing alpha B. Then where there then everything in follow of A will be in follow up B.

Why? Suppose starting with the start symbol of the grammar; we are deriving strings at a some point of time, I have got a string where say this is say a 1. So, these symbols are there this time in terminal and non terminal symbols are there ok. Say they after some time there is one A and after that I have got a symbol say small a ok.

Then in the next production what I can do? So this a may be replaced by alpha B. So, what can happen is that the string will get transformed into something like this. So, this is alpha B a so; that means, if a can come in the follow of capital A; if small a can come in the follow of capital A; then small a will also come in the follow of capital B that is obvious because from the in the next stage I can replace this A by alpha B; so, whatever symbol was following A will now follow B.

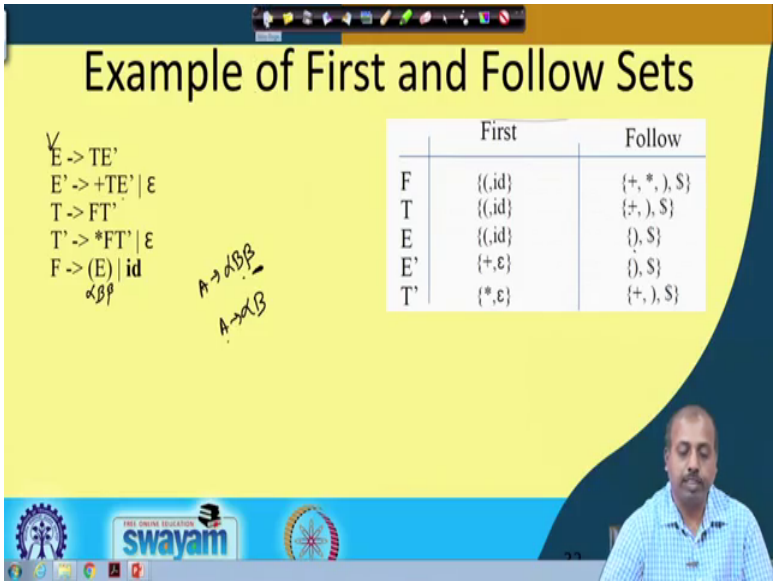
So, this is the; so that is the first part ok. Now if the rule is likely complex like instead of B alpha B; if the rule is alpha B beta in that case; so, what are the symbols in follow of B? So, whatever is in first of beta is in follow of B that we have seen in this rule. Also if this first set of beta contains epsilon if first set of beta contains epsilon; then what will

happen is that say from this I can rewrite this as a 1, a 2 then capital A 1, capital A 2, then b 1 etcetera. And then this A can replaced by alpha B beta then small a and it can go like this.

Now if first of beta contains epsilon; that means, from this beta this of there was through some derivational steps this beta can be replaced by epsilon. So, that the string gets transformed into something like this a 1, a 2, capital A 1, capital A 2, b 1, then alpha B; this beta part reduces to epsilon; so a comes immediately; so that can happen.

So, actually these to rules are similar if you say that the first of beta contains epsilon then these to rules are similar. So, naturally I can whatever is in follow of A will be in follow of B; so that is the second rule. So, these are the 3 rules and the so for the star symbol of the grammar there in a special rule that dollar will be in the follow of S. So, using these 3 rules; so you could able to construct the follow set. So, let us take some example and try to see like how this follow computation can be done; say for this grammar.

(Refer Slide Time: 10:59)



Now, for this grammar; so for applying the first rule that says that the if something is the star symbol of the grammar, then dollar will be in the follow set of E.

Now start symbol of the grammar is E; so the dollar will be in the following set of E. So, dollar is included now you scan through this grammar; try to apply the first the try to apply the rule that you whenever we have got A producing; A producing alpha B beta,

then whatever is in fast of beta is in follow of B. So you if try to apply this; so apply to the first rules alpha is epsilon, B is T and beta is E dash. So, whatever is in first set of E dash can be in the follow of T. So, first of E dash already computed plus and epsilon.

So, out of that epsilon I will not take; so this plus will go to the follow of T; so follow of T has got the plus symbol. Then by apply for second rule also if you have tried to apply that first principle alpha B beta. So, alpha is plus, B is T and beta is E dash as this does not add anything; so it remains as it is. Then if we apply in this rule then whatever is in first of T dash will be in the follow of A. So, first of T dash has got star and epsilon; so follow of F, so follow set of F has got star there.

From this rule nothing new comes because whatever is in follow of T dash will be in the first of T dash will be in the follow of F; so that we have already done. And here you see; if we apply this rules alpha B beta; so this is alpha, this is B and this is beta. So, whatever is in first set of beta is in follow of E; so first set of beta that is close parenthesis. So, first set of it contains close parenthesis only; so in the follow of E will have close parenthesis; the follow of E has got close parenthesis is there.

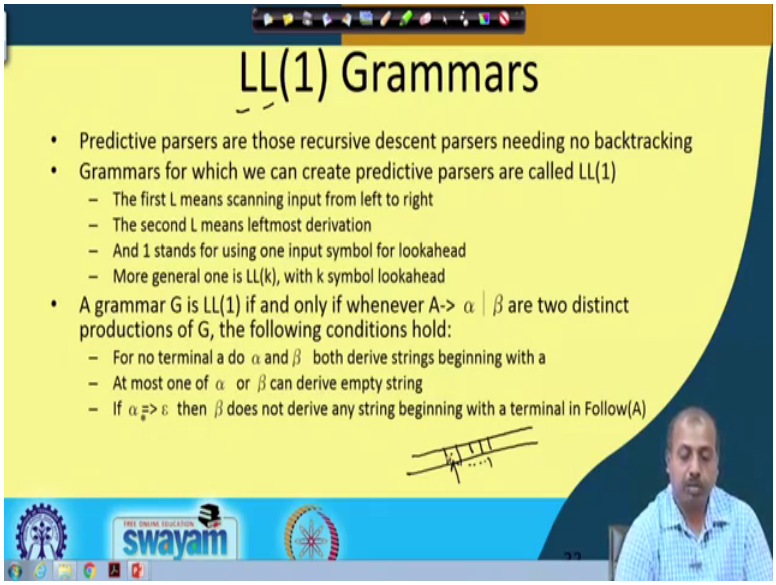
Now, we will apply the second rule; so, whenever you have got A producing alpha B; then whatever is in follow of A is in follow of B. So, whatever is in so follow of E will be in follow of E dash by this rule whatever is in follow up T will be in the. So, follow up E dash. So, follow of E dash follow E has got say close parenthesis and dollar. So, they will be added to the follow of E dash; so close parenthesis and dollar added here.

Similarly, if you apply this rule; so E dash producing plus T E dash for first of E dash contains epsilons. So, by the second rule applies for the follow set and then this whatever is in follow of E dash will be in the follow of T. So, follow of E dash has got close parenthesis and dollar. So, close parenthesis and dollars they are added to the follow of T.

So, coming to the third rule; so whatever is in follow of T will be in follow of T dash. So, follow of T has got plus close parenthesis and dollar. So, they are added to the follow of T was follow of T dash. So, follow of T dash; so, follow of T dash has got plus close parenthesis and dollar. So, this way we have to go on applying the rules again and again till the follow sets do not change.

So, it is slightly involved the process is slightly involve because you have a you really do not know like when you are going to terminate; so you have to go on trying the 3 rules that we have listed for follow computation and whichever rules; when there is no more addition to any of the follow sets, then only the computation will terminates. So, this way it becomes a bit difficult to compute the follow set. But you can very easily write a computer program and for doing this thing iteratively till we are coming to the all the all the sets they have got the maximum possible increase in them. So, this way we can compute the first and follows sets. So, once you have computed the first and follow sets we can go for constructing the parser for this predictive parser which is a non recursive parser and they are known as LL 1 grammars.

(Refer Slide Time: 15:35)



So, this type; so as I said that the it is a left to right the first L, so there are 2 Ls here there are 2 Ls here; the first L the first L will mean that the parser will take a left to right scan of the input. And second L means it will produce a left most derivation.

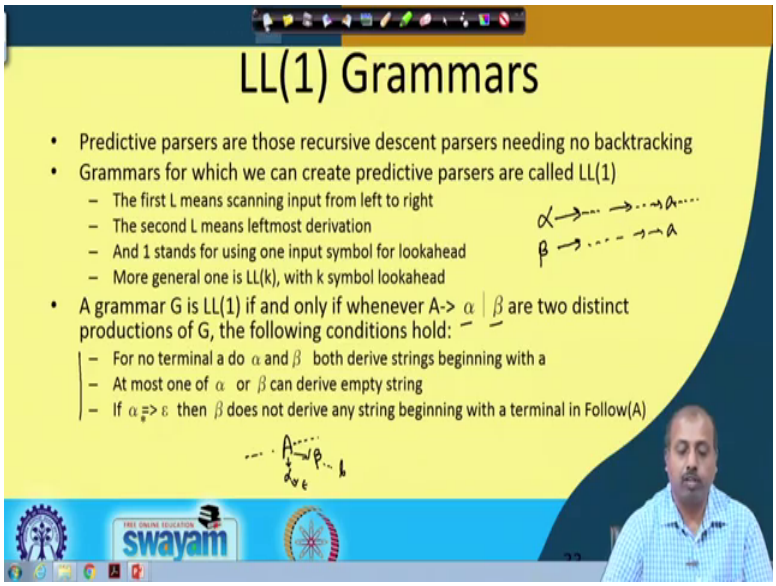
So it will the parser are the compiler that will be designed based on a LL 1 strategy. So, it will be scanning the input from left to right and it will produce a left most derivation of the input strings. And in general we have got LL k; so this in this particular case k is equal to 1; that will tell how many symbol look ahead we use for a; parser to proceed.

So look ahead means that at present so we are looking at the current symbol. So, if it is looking only at the current symbol then will be calling it LL 1 grammar or LL 1 parser. Now if we see that; so now the decision is taken based on the current symbol only.

So, if you need to check k more board symbols ahead to take a decision; in that case the parser will be called and LL k parser. So, the grammar will be called LL k grammar and this is the parser will be called LL k parser. So, there is a rule which says that this they grammar G is LL 1; if and only if whenever A producing alpha or beta are two distinct productions of G; the following conditions hold.

For no terminal a do alpha and beta both derive strings beginning with a. So, it should not be that for both of them. So, if you think about the strings which are derivable from alpha.

(Refer Slide Time: 17:45)



Alpha is a string of terminals and nonterminals; so if you derive further strings from alpha. So, its suppose it starts with symbol a the string starts with symbol a. So, with beta if I do the same thing it should not be the case that is beta also I can come to a situation where the derived string starts with a.

Naturally, in that case there will be confusion. So, if you look into the first of alpha and first of beta if both of them contains a; so you cannot decide that you whether on getting a, whether you should follow the production a producing alpha or the production a

producing beta; so that is one thing. Second point is that at most one of alpha or beta can derive empty string; so that is epsilon production.

So, that is also there like; so if you find that at some point of time I need to consume entire capital a in my sentential form. Without consuming any further input, so I should try to replace it by alpha where alpha can give me epsilon or beta if beta can give me epsilon, but it should not happen that both alpha and beta can give epsilon; so, at most one of them can give epsilon.

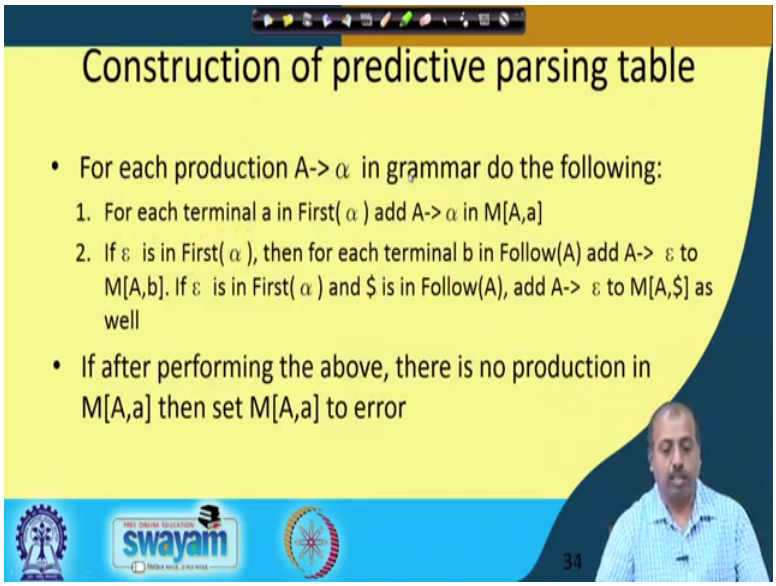
So then we do not have any problem of choice; otherwise there is a choice. So, I can either produced by a producing alpha or a producing beta. So, I really do not know which one will be correct; so that can lead to some erroneous action by the parser. And if a in if alpha in a number of steps produces epsilon then beta does not derived any string beginning with the terminal in the follow of a; so, this is another condition.

So, it says that you; so I have got to something some string at this I have got A and then I have got something else. Now I have got a choice I can replace A by alpha or A by beta; so if I replace A by alpha, if I replace A by alpha and alpha can give me epsilon; alpha can give me epsilon, then from beta I should not be able to derive any string that has got a will terminals in follow of A.

So, that is this if I could have gamma with beta and it can give me a valid string where there is a there is a symbol which is in follow of b that can comes; so b is in follow of capital A; so that can come. So, then there will be confusion again. So, these are the three conditions that have that are to be satisfied for a grammar to be LL 1. So, the detail proof can be obtained in some formal language theory book, but we are not looking into that. So, we would be following these rules to determine whether a grammar is LL 1 or not. Even without doing these checks; so we can decide a like whether is a grammar LL 1 or not by constructing the corresponding parsing table.

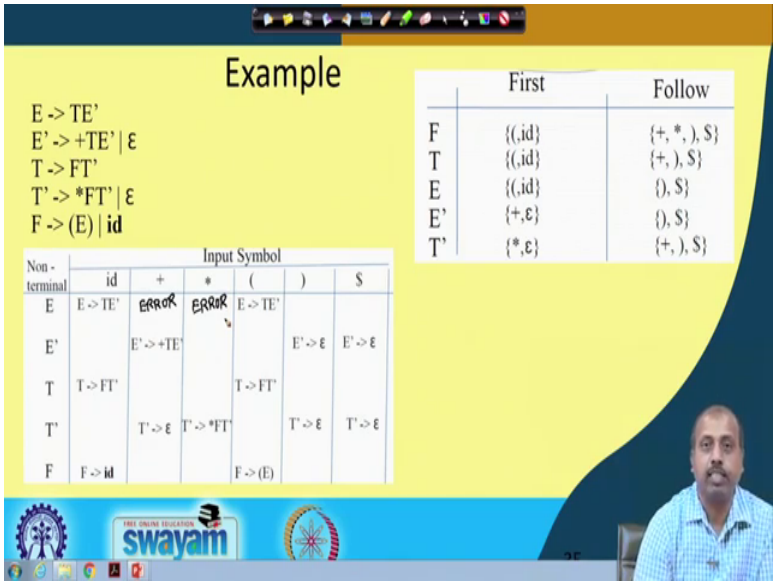
And if the grammar is not LL 1, then there will be multiple entries in the parsing table for a particular value. So, naturally whenever there are multiple values at some entries; so we have got there we say that there is a the grammar has got problems. So, it is not LL 1 grammar; so we will see that.

(Refer Slide Time: 21:21)



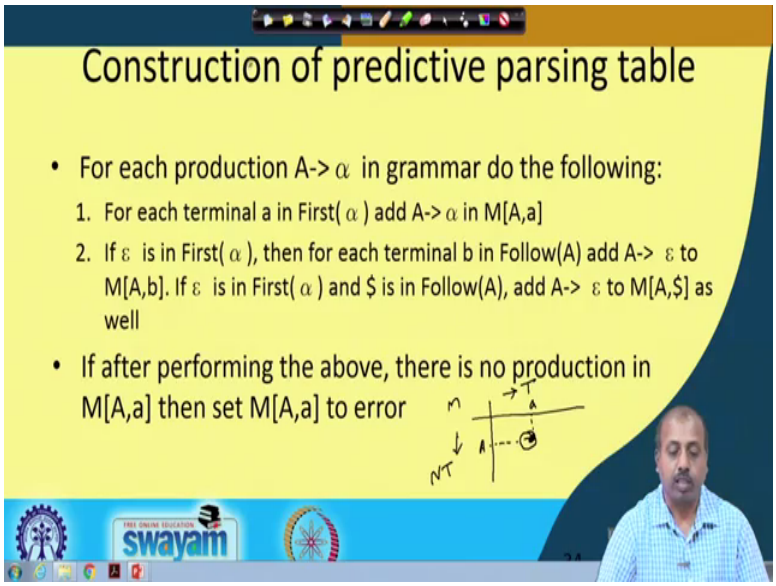
So, how do you construct a predictive parsing table? So, what the predictive parsing strategy will do is that it will have a parsing table; so like this.

(Refer Slide Time: 21:31)



So, it will have a parsing table and looking into the also looking into the; grammar symbol as that is there on some stack and the next input symbols; so it will decide like what action to take. So, is the stack top contains E and the next input symbols is id; then this table says that you go by this rule E producing T E dash, so it will go by this rule ok.

(Refer Slide Time: 22:01)



So, I think there is a so there; so we will see how to construct this parsing table. So, for each production A producing alpha in the grammar so we will do like this.

For each terminal a in first of alpha we add a producing alpha in the table A a. So this table is 2 dimensional table in one dimension we have got all the non terminals; so, this is the table M. So, here I have got all the non terminals from the set M, this set I have got all the terminals ok. So, that way; so this M A a is corresponding to this particular non terminal and this particular terminal. So, what is the entry here; so that entry is decided by this.

So, if A is in first of alpha where if A producing alpha is a rule, if A producing alpha is a grammar rule and first of alpha contains A, then we add this rule A producing alpha to this particular entry. Then it says that if epsilon is in first of alpha then for each terminal b in follow of A, we add a producing alpha to the table M A b. So, if epsilon is in the first of alpha; so this can give me epsilon then we for every terminal which can give me follow of A, which have which there is in follow of A; so, I will like to reduce this by epsilon.

So, this will be added to M A b and finally, if epsilon is in first of alpha and dollar is in follow of A, then we add A producing alpha to M A dollar as well. And after doing this whatever you go whatever entries are undefined; we mark them as error entry. So, how does it also if I look into the example like say how is it constituted? So, this is the

grammar rules that we have ETF grammar modified by eliminating left recursion and all and this is the first and follow sets that we have computed.

Now let us look into this; let us look into this rule E producing TE dash. Now as far our as far our rule is concerned. So, it says that if A producing alpha; so you have to see the first of alpha. So, first of alpha first of TE dash is equal to first of T; so it is bracket start and id. So, in bracket start and id we add this rule. So, E producing T dash is added here and here fine; then we have to have this whether this first of T dash; parser T dash contains epsilon not that has to be seen.

So, as for the second rule it says that if epsilon is in first of alpha. So, whether epsilon is in first of alpha for not; so that has to be seen. So first of TE dash; so first of TE does not contain epsilon. So, there is no question of first TE dash containing epsilon; so second rule is not necessary. Now come to the rule E dash producing the plus E dash; so first of plus TE dash is plus, so I will add this rule E dash producing the plus TE dash in this particular entry.

Now look into this E dash producing epsilon. So, bye that second rule it says that by the second rule; it says that whenever I have got epsilon A producing epsilon; then and whatever we have in the follow of A, there I should add this particular rule. So, follow of E dash the follow E dash is the bracket close and dollar.

So, E dash bracket close I have added E dash producing epsilon and dollar I have [vocalized-noise given E dash producing epsilon; so this is added to these 2 rules. Then this next coming to this rule T producing FT dash; so FT dash first of a FT dash will is equal to first of a; that is bracket start an id. So, bracket id and bracket start; so we have added this T producing FT dash.

Now comes T dash producing star FT dash. So, by the say by the similar rule like T dash produces star; so, the first of the star FT dash is equal to star. So, at the step at T dash star; so we add this particular rule T dash produces star FT dash. And then T dash produces epsilon, so whatever we have in the follow of T dash plus bracket close and dollar; so there I will add T dash producing epsilon. So, plus bracket close and dollar we have added T dash producing epsilon.

Then finally, so this particular rule F producing within bracket E; so first of this contains bracket start. So, F bracket start is F producing within bracket E and this id; so this id part is very very slowly this is applying other rules a producing id. So, first of this a part is id only; so, this F id, so we have added this thing. So, the simple rules for you have got the first look into the first set. And whatever 5 symbols are coming in the first set; so you add the rules at those points and if the first produces epsilon; then whatever is the follow of the point on terminal then for all of them we add the rule. So, that is the straightway rule for producing the predictive parsing table.

So, we will see how can we use it for this passing job. Now we can produce the other entries like all these entries which are left as blank; so they are actually error entries. So, error entries means the; so if you have got a non terminal E and the next input symbol is plus; so on E you cannot get it plus ok; this is an error. So, all the entries that we have here so they are; they can be marked as error; they can be marked as error.

So, whenever the parser comes to the state this particular table entry so; that means, there is an error in the input string; as a result it has come to this. So, the input stream has to be rejected it is not there is some syntax error and the parser needs to recover so that it can continue parsing with the remaining strings.