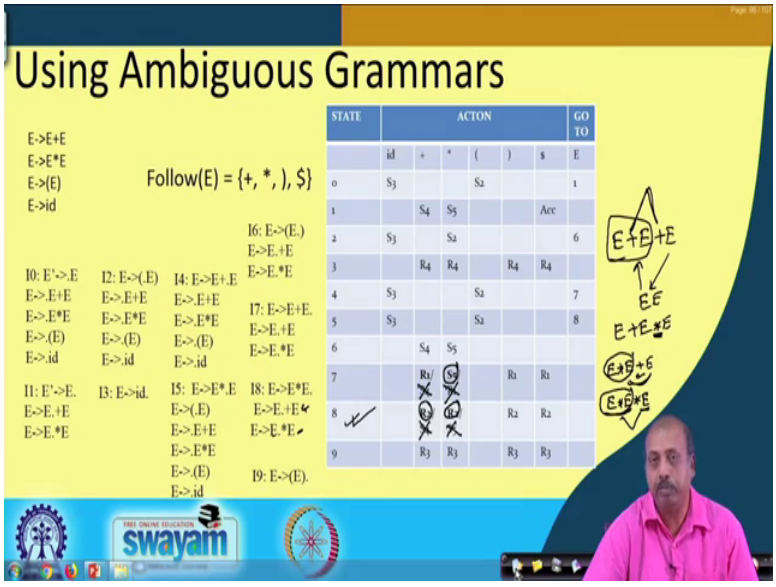
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 31
Parser (Contd.)

So, these ambiguous grammars can be used for reducing the complexity of the parsing table. So, we can have less number of entries in the paring table, but it can give rise to problems. So, let us take an example and try to see how is it going to happen.

(Refer Slide Time: 00:31)



So, let us say that this standard ETF grammar that we have seen previously. So, that is there this T and F. So, these two non-terminals are not used. So, we have got the grammar rewritten using only the non-terminal E. So, E producing E plus E, E producing E star E, E producing within bracket E and E producing id. So, these are the grammar rules.

So, naturally if I have got an expression like say id plus id star id then I have got multiple number of parse trees now. So, if I say I will be doing decomposition like this. So, E plus E and then this E will be giving me id this E will give me E star E. So, if I mark them as id 1, id 2, id 3. So, this is basically id 1. So, and this is this E star E and this giving me id 2 and this giving me id 3, fine or somebody may do it like this E it

gives E star E where this E gives me id 3 and this E gives me E plus E and this E gives me id 1 and this gives me id 2.

So, we have got two different parse trees for the same string and using; so, in one case so, you have so, if you look into it carefully in one case I am doing the multiplication first between id 2 and id 3 and then I am adding to id 1. So, if you look in a bottom up fashion. So, we are doing it like this on the other case I am doing the addition first id 1 plus id 2 and then multiplying it by id 3. So, this is the other case.

Now, syntactically both are correct, but it is giving my giving me two different parse trees. So, it is giving rise to ambiguity. So, the grammar is ambiguous grammar. So, that is that is established. So, this grammar is an ambiguous grammar. So, what happens when we try to construct the corresponding parse tree.

So, let us see what happens. So, let us corresponding parsing table this SLR parsing table. So, this I0 so, this the as per our policy we know that we have to introduce another special non terminal E dash start symbol E dash producing E and then I0 will be E dash producing dot E and closure of that. So, this will give me E producing dot E plus E, E producing dot E star E, E producing dot within bracket E and E producing id. From I0 to on E it will come to I1 this E dash producing E dot then this E producing E plus E is there. So, E producing E dot plus E and similarly from this rule I will get E producing E dot star E.

From I0 on this open parenthesis so, it will be like say open parenthesis dot E close then E producing similarly now it has got dot E, so, all these rules will come now. Similarly, I0 on id it will be I3 which is E producing id dot then from this I from this I1 on plus so, it will come to this I4. So, E producing E plus dot E and again since dot E is there. So, all of them will come. So, I4 will come. So, this way it will be creating all these items.

So, once we have created all these items then we will be trying to construct the parsing table and before that since it is SLR parsing table construction. So, we need to have this follow set computed. So, the follow set in this case is very simple. So, by from this rule you see that E may be followed by plus, from this rule you see that E may be followed by star E may be followed by close parenthesis and dollar is the since E is the start symbol of the grammar. So, dollar is there in the follow set. So, the follow set of E is like this.

Now, let us try to see what happens to the parsing table. So, the parsing table will be something like this say from this state 0 on id it comes to a state 3. So, this is the thing is that we have explained previously from I0 on id it comes to state 3 then from I0 on open parenthesis it comes to S2 I2 then from I1 on from the state I1 on this if it finds a dollar then this E dash producing E dot is there. So, that is the accept state and this on this plus so, this will be a shifting. So, this I1 plus so, this is the state 4. So, that is S4 state and then we have got this S5 on I1 star. So, I1 star is S5. So, this way this table is made.

So, similarly from I3 from this state I3 on this particular rule we see that whatever is in follow of E for them I have to do the reduction follow of E has got this plus, star, close parenthesis and dollar. So, from I3 plus, star, close parenthesis and dollar so, I have got reduced by rule number 4. So, that is fine. Now, what about this state S 7? So, that I7; so, this is the thing. So, in I7 you see that first we will look into say this rule E producing E plus E dot. So, dot is at the end of the production. So, that means, whatever is in the follow of this E, so, I have to do a reduction by this rule and follow of E has got all of them plus, star, close parenthesis, dollar. So, by plus by from for plus, star, close parenthesis and dollar so, for all of them I do a reduction by rule number 1. So, they are doing a reduction by rule number 1.

However, there is also another situation like E E producing E plus E dot plus E. So, that means, so, this says that on plus it should do a shift and shift to what? It should shift to an item where E producing E plus dot E is coming; so, E plus dot E. So, that is basically state number 4. So, state number 4 is this one. So, E plus dot E so, it is coming to state 4. So, this is the other action. So, this is one action and this is the other action. Similarly, on star; so, it is coming to state 5. So, this S 5 is one action and R 1 is another action, fine.

Now, but for this close parenthesis and dollar there is no confusion. So, there is no shifting. So, only I have got some conflicts in this part. Similarly, from state 8 if you try to see then here I have got this E producing E star E and dot. So, whatever is in follow of E there I have to add do a add reduction by this particular rule number 2. So, this by this rule number 2 I am doing the; I am doing the reduction. And, now this rule this particular item is telling me that if you see a plus then you should do a shift and go to the state 4. So, this S 4 is added and similarly for this rule is telling me this item is telling me that if you see a star you should do a shift and go to the state I 5. So, this is the shift 5.

So, this way there will be ambiguities and whenever you have got unambiguous grammar. So, the corresponding SLR parsing table it will have this type of conflicts. And, in this particular case so, even if you construct one LALR parser you will find that this conflicts will persist. So, they will do not get resolved by going to the canonical LR or LALR parser.

However, we know the actions that we have to do like see if we try to resolve this conflicts then you see that. So, at this state number 7 we have a conflict between on seeing a plus whether to do a reduction or do a shifting. So, you know that whenever at state 7, so, you have seen this thing E producing E dot and then you are saying this E producing dot plus E.

So, by associativity property, so, we know that this addition may be done later, so, I can shift this addition. So, this particular addition will be doing later. So, we will we will shift this addition into the stack. So, that way I may forego this reduction I can do a shift operation and similarly if you see a star; if you see a star then you should definitely do a shift. So, with a plus there was an option because I can do the previous addition first and then proceed with this addition, but if you see a star here then definitely you should a shift; you should not do the reduction by E producing E plus E.

Basically, so this E has given me E plus E and then you have got this star and this E. So, this naturally in that case I actually meant the multiplication between this E and this E. So, I should definitely do a shifting. So, this action should definitely be a shift, ok. So, this is not a reduction action, so, this should be a shift action.

However, somebody may say that I will be doing a reduction here I will not do the shift because this addition will be similar. So, I will do a shift I will do a reduction because this is like E plus E plus E something like this. So, you have seen this E plus E and then you have so, this E plus E if I reduce to a new expression E and with that if I add E so, that is good enough; by associativity property of this addition, so, you know that that is they that will not cause any harm. So, possibly this is also an option that I do a reduction and I do not do a shift and this is arbitrary. So, somebody may say I will do shift, somebody may say I will do a reduction because there is no precedence between this plus and this plus. So, they are similar.

However, when this thing becomes a multiplication; so, when it is E plus E star E then this multiplication has to be done first and for that purpose I need to do a shift. So, this action must be a shift action whereas, this action may or so, the I will not do this reduction I will do this shift whereas, this state I can do a reduction or a shift maybe I decide in favour of reduction.

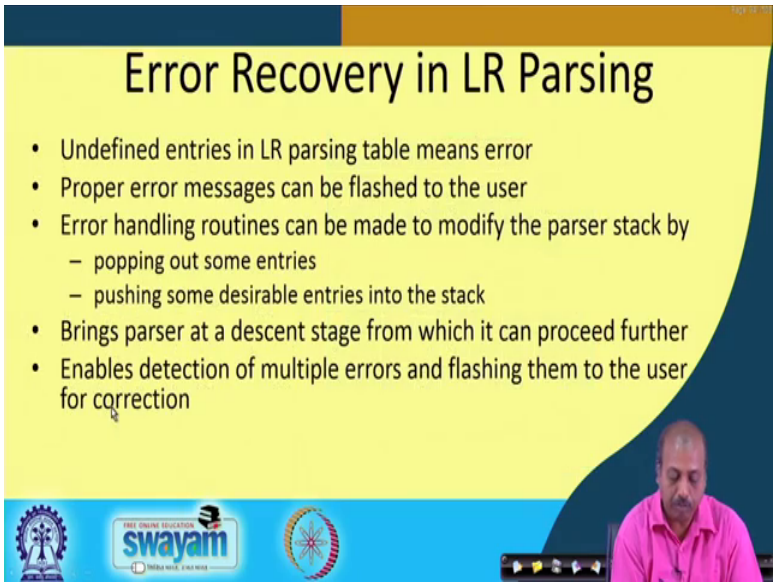
Similarly, from state 8 you see you have already seen a product like say E star E and then this rule is telling me after that if you are seeing something like this plus E, then what are you going to do? Definitely you are doing this you should do this multiplication first and then rather than the shifting this plus. So, if you shift this then that will mean that I will do this addition first and then multiply with this E. But, since you have already seen one star multiplication of two expression so, you should take that as the valid sub expression. So, I should do a reduction by rule number 2, I should not do a shift, ok.

And, similarly if you see this one say second the second one; so, it says that you have seen E star E and then again another star and then E. So, that is the situation. So, if this E is basically one E star E and then the so, this part I have already seen and then I am going to see this thing. So, naturally I should so, here also same thing like previously. So, I can do a shift or I can do a reduce. So, both are valid, but for the sake of simplicity maybe we say that we will do this multiplication first and then with the result I will take this multiplication. So, as a result here also reduce by 2 rule number 2 may be the valid choice and this is not the valid choice.

So, this way you can tell the parser once the parsing table has been constructed and the parser tells you that we have got shift reduce conflicts at this, this, this places or reduce-reduce conflicts at this places so, you can tell the parser that parser generator tool that you can resolve it in this fashion. So, often by telling the precedence between this at this terminal symbols this ambiguities gets resolved. But, or maybe you can tell it explicitly or parsers they generate parser generators they have some default rule goby if there is a shift reduce conflict it will go by shift operation only, but that is not always valid because you have seen in many cases. So, going by shift operation only in case of shift reduce conflicts so, that is not a good choice because at least for this state 8 we have seen that the shift options are not correct, it should be the reduce action. So, so, you can tell the parser like that or you can tell explicitly that do a reduction operation at this point.

So, these way ambiguous grammars can be utilized and once you have used it. So, it will give rise to these conflicts and this conflicts if you can resolve cleverly then you can get a parsing table that will have less complexity then the other the original grammar unambiguous grammar.

(Refer Slide Time: 15:27)



Another important issue with this parsing process is the error recovery. So, in other parsing techniques also we have seen that there is a concept of error recovery because in the recovery process so, we whenever the parser is having a stack so, it is putting those states into the stack. And, it may so happen that parser has gone to a state from where it cannot recover, it cannot come back to a valid state from where it can proceed with the parsing.

So, that is a difficulty. So, so, may be the when the parser is progressing. So, we know that this if the parsing table entry is blank then all those entries are error entries. So, if there are error entries then we should be careful and the parser will just stop telling the syntax error. So, if you do not take care of it at the parsing algorithm itself then of course, this parser will be will not be able to give you any error message and we should be we should be able to give appropriate error message the why the parser has reached an erroneous state. So, that has to be told.

So, this undefined entries in the LR parsing table means error. So, proper error messages can be flashed to the user. So, if we can analyze a particular entry then we can try to we

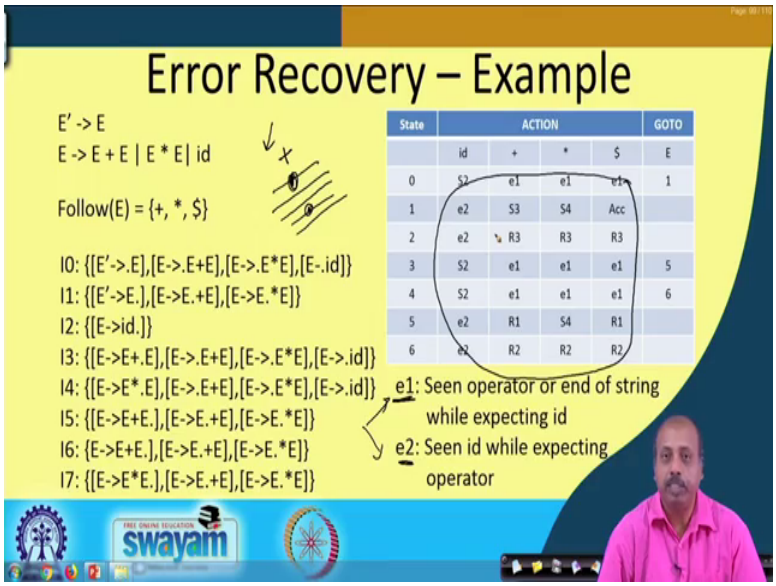
can try to figure out like how we have arrived at that particular entry, what how that state and input combination has arrived and if we know that then appropriately we can flash some error message.

Now, error handling routines can be made to modify the parser stack by popping out some entries from the stack and pushing some desirable entries on into the stack. So, we know that this input string is erroneous. So, we take out some invalid entries from the stack and pushing some entries which the parser designer thinks that that is correct, that will be a correct sequence. And, that way most of the time what it does is that it tries to push some symbols which are going to be the end of string marker and things like that so that one sentence ends or one block ends. So, like that.

It will attempt to bring the parser to a descent stage from which it can proceed further. So, it can identify further errors in the in the input stream. So, that way it can it I needs to come to a descent stage and, enables detection of multiple errors and flashing them to the user for correction. So, in one go you can flash a number of error messages to the user and the user can correct all of them and then come back and resubmit the job of compilation. So, that way this will be useful.

Otherwise what will happen is that the parser will stop at the first entry itself and telling that there is an error and then how to where is the error at least the line number of the source file needs to be told, and possibly what is the error like say semicolon missing or some particular token is missing. So, that type of message, so if we can give then that will be helpful for the parsing further user to rectify the program.

(Refer Slide Time: 18:47)



So, we will take an example. So, suppose we have got a grammar like this E producing this is a simple expression grammar that has got this addition and multiplication E producing E plus E or E star E or id; so, then an E dash. So, to make the corresponding SLR parser, so, we add that this extra production E dash producing E to get the augmented grammar and then we compute the follow set of E follow set of E from this grammar itself you can see that you have got plus, you have got star and since E is the star symbol of the grammar we have got dollar. So, this is the follow set plus star and dollar.

Next thing is that we have to construct the LR 0 items. So, this I0 is E dash producing dot E and then since dot E is there so, closer will take E producing dot E plus E producing dot E star E, E producing dot id into the set I0, then in I1 I will have from go to I0 E. So, this is this will give me E dash producing E dot E producing dot E plus E, E producing E dot star E. So, like that it will give me.

Now, you see that so, in this way we try to construct all the LR 0 items and in this particular case we have got 8 such items I0 to I7. Now, now suppose so, suppose this table that is constructed so, it has got this the it has got this thing. So, this I0 from this state I0, so, this id it says I0 id is I2, so, it is shift 2. Similarly so, but these entries are undefined; these plus, star, dollar so, they are all undefined. So, in state I0, if you find

that the so, so what is this set when are you going to consult this particular entry? So, your state is S 0 and the input stream it has got a plus in it input pointer is here.

So, in state S 0 that is at the very beginning of your parsing process if you see a plus, so, that is the error because your expression can start with an identifier, it cannot start with an operator. So, we can flash this particular message E 1 seen operator end of string is not there seen operator while expecting id. So, it was expecting an identifier, but it has seen an operator. Similarly, if this first symbol itself is a dollar; that means, the end of string. So, there is a null string basically; for null string it is not there. So, we can say that so, that is the if it seens a if it sees a dollar then it can say that it is a seen end of string character while expecting an identifier. So, that is E 1.

Similarly, E 2 so, say so, for all these cases so, if it sees a plus, star or dollar so, it can flash the message E 1. Similarly in state 1, if you are in state 1; that means, you are in this situation. So, this is you have seen an expression and then you can what you can expect is that you have seen some expression say id plus id and you are somewhere here, and then the next thing that you can expect is another; so, so, this does not give anything. So, this is a complete thing. So, if you are at this point; so, next you are expecting a plus or a star that is you are expecting an operator, but you have got an id. So, this entry will be consulted when in state 1 you have got an id. So, in state 1 if you get an id so, the so, you are expecting if you are not seen the dollar then you are you are you are expecting a plus or a star.

So, if you see id that is another error E 2, so, seen id while expecting operator; so, this is by analyzing the entry so, you can find out what may be the problem and accordingly you can populate this table. Similarly, say this state 2. So, state 2 id dollar and again if you are getting an id so; that means, that is an error because you have you have seen an identifier so, you cannot see one identifier following another identifier. So, you cannot the situation cannot be there. So, in between there must be some operator. So, there also I can flash this message E 2 that is you are expecting an operator not an identifier.

State 3; so, in state 3, so, if you find a plus when state 3 you what you are expecting to see is an identifier, ok. So, the so, naturally you can flash this message that E 1 that is E so, you have if you see this plus, star or dollar so, we can say that I was expecting an

identifier and I have got an id. So, that is that is the error. So, I was expecting an identifier and I have got an operator plus, star or end of string dollar. So, that is an error.

Similarly, state 4 so, here also you are expecting an identifier and we have got this one of these operators plus, star or dollar. So, that is also the error E 1. State 5; so, this is this is the situation that you are expecting a plus or a star that is you are expecting some operator and you have got an id. So, that is the error. So, you are expecting an operator there. Then, state 6 I6; so, here also you are the expecting an operator and there you have got an id. So, this is the rule. So, the state 7 is not shown here. So, accordingly you can add that column then it will be at their particular row in this table.

So, this way we can have this error messages. So, we can appropriately flash error message then what may be the action like that is about that is about the situation that this problem has occurred. So, it was expecting an identifier and it has got an operator say. So, say this E 1; say when this E 1 is seen that it was seen an operator while expecting id then one thing that the parser can do is that purposefully in the input stream so, it can push an identifier. So, it can. So, it has a got a plus so, what it can do purposefully it can introduce, it can take the pointer back by one position take it to here and introduce write here one id and then give it to the lexical analyzer as a so, it or the parser that the token is id.

So, that way for this E 1 when this error E 1 occurs apart from flashing the message to take the parser out of the erroneous state for this particular grammar so, it can take the it can push an id token into the input stream and then the parsing process will continue normally. Similarly, for the E 2 situation so, it can it was expecting an operator so, since it was expecting an operator so, it can do like this. So, it can push in one plus into the input stream because plus has the lowest precedence. So, it will not hamper with any other operators. So, I can put a plus.

So, though the parser will generate a will not generate a parse tree because these are erroneous thing, but the parser will be able to continue and it may it may be able to find out more errors in the expression. So, that way so if I have got lines like this then if it has got if it has found an error here. So, it will be in the normal case it will stop at this point, but if you can somehow modify the input stream by introducing the appropriate token, then possibly it can proceed further and it can again detect some error at some later line.

And, on the other thing is that you can also try to modify the stack, this stack that the parser maintains for more complex situation. So, we have to do that, but for this particular example it is fine. So, it is if we can just the there are these are very this is the very simple grammar, but this explains the how can we have this error messages introduced and how this error recovery parts can be introduced.

So, at the end of this process, so, you see that none of the actions in the table that they are undefined. So, there none of them are blank. So, the parser when in the parsing algorithm when it is proceeding so; it will always find some valid entry into this table in the in the action part. So, it may be a normal parsing action in terms of shift and reduce or it may be an error action in terms of these error routines E 1 and E 2. So, that way it will be able to take care of those errors and it can recover the parser will recover from the error.

And, the user will also give a given enough feedback about how this parsing algorithm is going to how this parsing is, what are the errors that has occurred during this parsing; so that the user will be able to correct all those errors and resubmit the program for compilation.