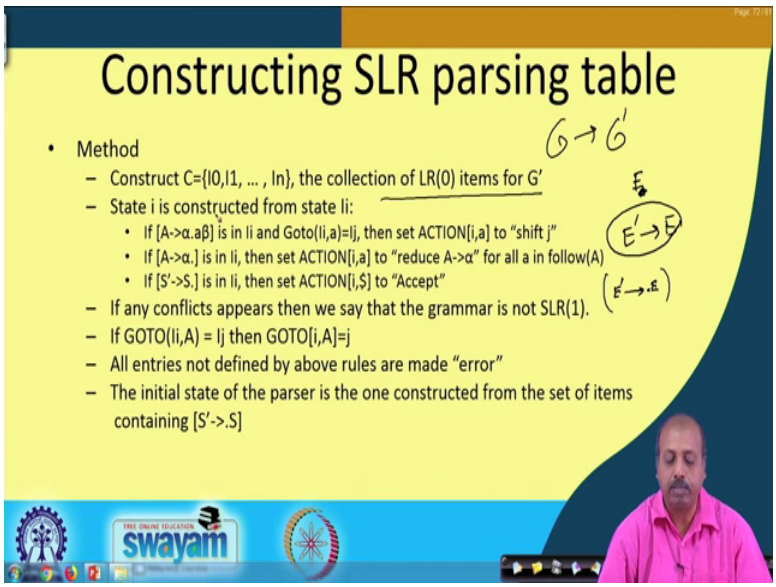
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 29
Parser (Contd.)

(Refer Slide Time: 00:15)

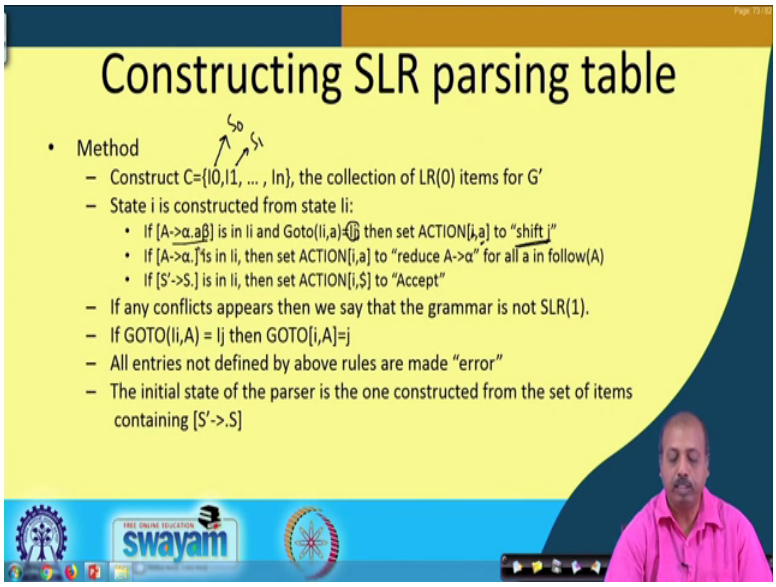


So, next we will be looking into how to construct the SLR parsing table. So, first we have already seen the technique. So, we will formally put it in the form of an algorithm. So, first we construct the LR 0 set of items sets of items. So, this is the first step. So, it first convert the grammar G into an augmented grammar G dash by adding the extra start symbol, like if the grammar start symbol is say E then we add another extra production E dash producing E.

With the idea that when the whole when the parser will try to do a reduction by this particular rule. So, it will mean that the string has been parsed successfully. So, that way we start with the item like E dash producing dot E and that is and we take the closer of this. So, that will make the set I0 and from this set I0. So, we will be seeing on various input symbols input symbols and grammar non-terminals also, so will be having other items produced. So, I0, I1, suppose when we construct the items. So, we get these items like I0 I1 up to In. So, that is there though they that is the collection of LR 0 items for the grammar.

Now, from this, so from the state i will be constructing the set i. So, for each of these i for each of these I0 I1 etcetera, so they will constitute one state. So, this will constitute the state S 0, this will constitute the state S 1 so like that.

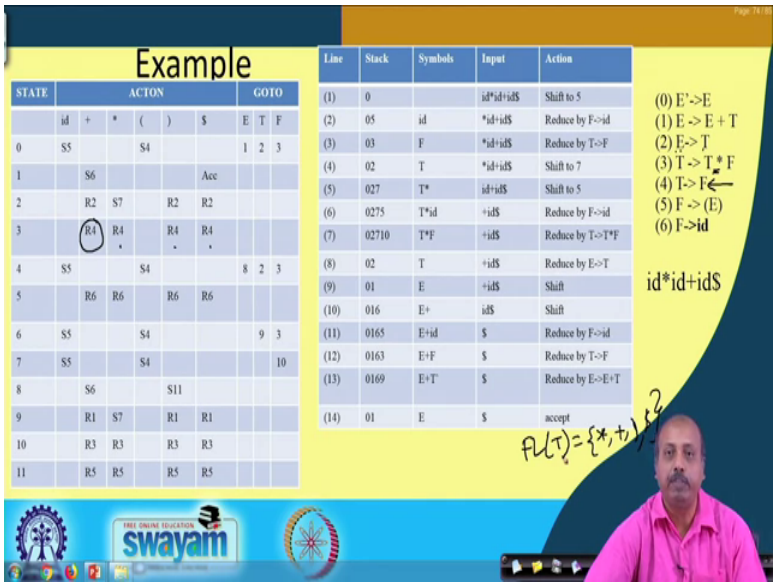
(Refer Slide Time: 01:54)



And we will be have a from this state once we call this the states, so we will be looking into the go to parts. So, we have previously seen how to construct the go to of symbol of a particular state on some terminal and non-terminal symbol and we have also seen like how to make get the closer.

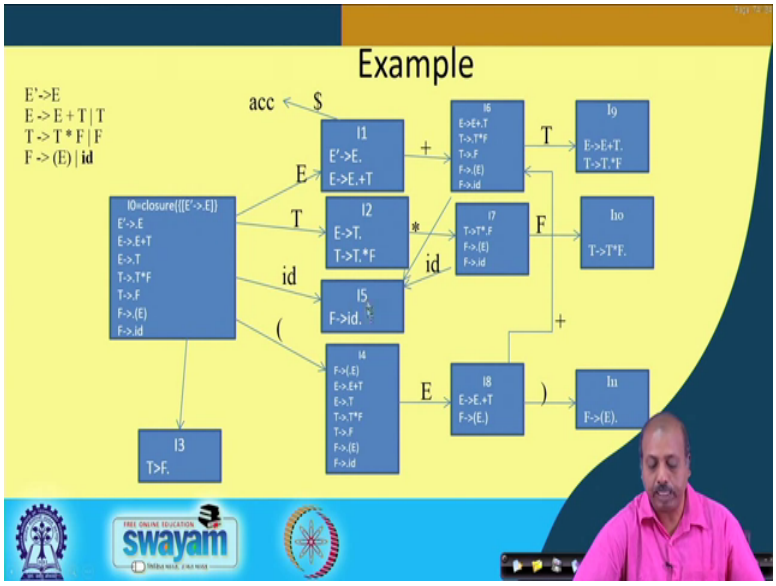
So, if this A producing alpha dot a beta is an item in a particular set of items then that is in Ii. So, if I have got a item like A producing alpha dot a beta and the go to Ii a says it is a new item Ij. Then in the action part of the table at the corresponding to state number i and input a, we will at the action shift j; that means, it will do a shift operation and it will come to a new state which is j. So, that is the meaning of this rule. So, we can, we can look into a, we can look into a particular example like say this one.

(Refer Slide Time: 03:13)



So, while constructing this table from I from the state I 0, so I0 is having E dash producing dot E and the closer of that. So, if from there we will get it is a dot E will have the id also? So, on id it will do a shift and it will come to state 5. So, if we just go back a few slides say on I0 on id it comes to the state I5 comes to the item I5. So, accordingly I will be adding this entry in the table shift and the new state is 5. So, that is the way this table is constructed for the shifting part.

(Refer Slide Time: 03:36)



Now, if you have got a rule like this a producing alpha dot if you have an item in the set Ii like A producing alpha dot then we will first have to see what is the follow of this non-terminal A and for each of the symbols small a in the follow set of A. So, we will be adding this production reduce by A producing alpha. To understand this thing, so let us look into how do you for example, how do you get this particular rule, how do we get this particular reduction.

So, this as I 3 and on plus it is R 4. So, I 3, so it means the reduced by rule number 4. So, rule number 4 is this one, T producing F. So, I have to see what is the follow set of T. So, follow set of T, follow of T is given by. So, T is followed by this star because by this rule, so this is followed by star and by this rule E producing T whatever is in follow of E is in follow of T. So, I will get plus and here E is followed by closing parenthesis, and also since E is the E was the start symbol. So, dollar was there in the follow set of E.

So, as a result the follow of T will have the star plus close parenthesis and dollar. So, you see for this plus star close parenthesis and dollar we have added the action reduced by rule number 4. So, this is done here. So, this way we can we have to we solve for all the symbols that are coming in the follow set of the terminal non-terminal on the left hand side of the production. So, we have to have this we have to add the corresponding rule there.

Next, for particularly for this S dash producing S dot, where S is the actual start symbol of grammar G and S dash is the start symbol of the augmented grammar G dash. So, if S dash producing S dot giving S dot if this item is there; that means, I have I am trying to do a reduction by this particular, we are trying to do a reduction by this rule S dash producing S. So, if in a particular item you have this thing then action i dollar is set to accept, means action i dollar means I am currently in state I, state i and the next input symbol is dollar. So, input is or the input string is also exhausted and I am trying to do this reduction. So, that is naturally the accept state.

So, so the main difficulty here is one thing is that you have to a calculate this items carefully, this item calculations, so that is one job and the other job is to compute the follow of all the all the symbols or the of all the non-terminal symbols that we have in the grammar wherever you will be requiring this A reproducing alpha this type of rule, so you will need to have the follow set of A.

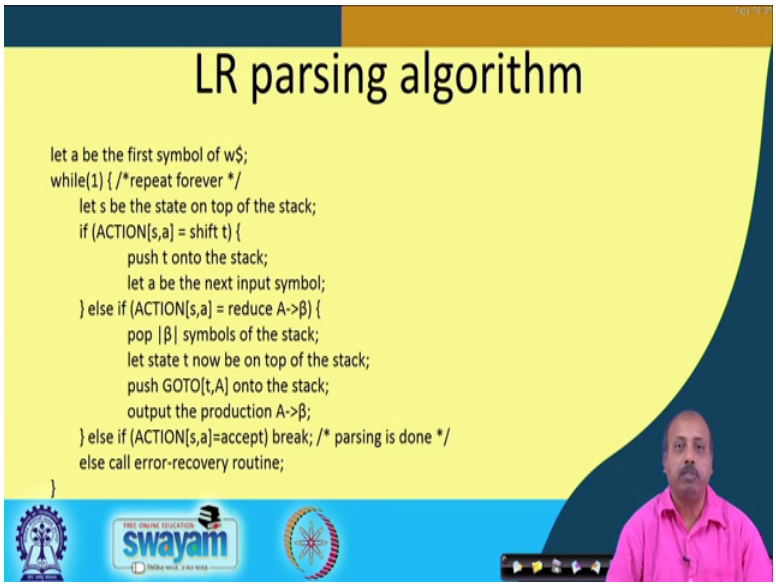
Now, if there is any conflict like while doing this putting this actions shift and reduce. So, if there is a conflict then the grammar is not an SLR grammar. So, like say in the previous example that I that we took. So, if we look into the action part you see there is no entry where there are two actions are defined. So, everywhere it is either a shift operation or a reduce operation or there is no operation. So, no operation means it is an error, but for the action is accept, so those things are there, but there is the no entry in this table where the actions are multiply defined.

So, this table, this particular grammar is an LR SLR grammar because we could construct the SLR table for them without any problem. And then for the go to part. So, if you look into this table. So, there are two parts, one part is the action part and the other part is the go to part. For the go to part I have got the non-terminals E, T and F, and from individual items individual sets of items. So, on the non-terminal with the go tos have been calculated. So, here the corresponding go tos are added.

For example, from state say I have from an item I0 on E, T and F let us see where we are going. So, if you look into this particular diagram for I0 on E it was going to state I1, on T it was coming to this one 12 and here it is not marked the level is not marked, but it has come from this T producing F dot, so this is F. So, this I1, I2 and I3. So, these are the 3 states to which we can go from the state I naught. So, that is done here. So, you see from the state 0 on getting this E T or F. So, it is moving to the states 1, 2 and 3. So, in this way the go to part will be filled up.

So, once this action part and go to part has been met then we can use the algorithm that we have seen previously this parsing algorithm for parsing the from for passing the input string.

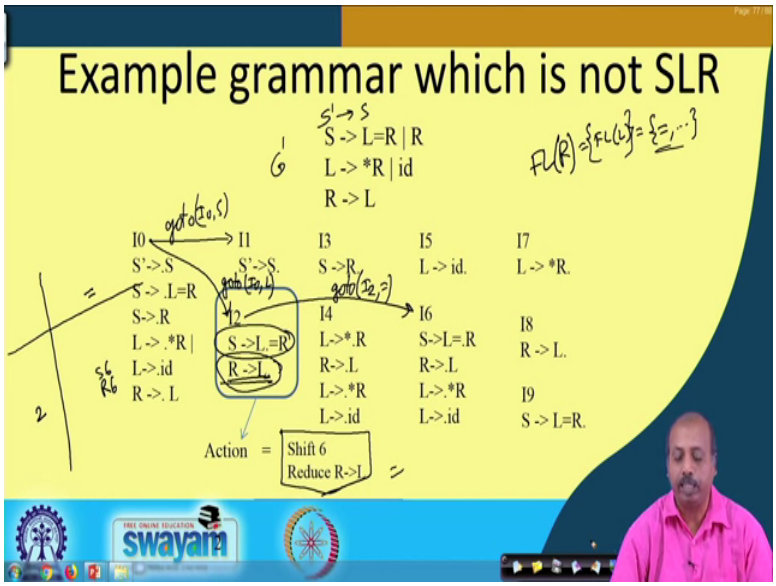
(Refer Slide Time: 09:21)



And at the end; so, if you are finding that ok, we are at a state where the with the action is accept then parsing is over. If there is an error in the input stream that at some point of time you will be landing into one of these entries which is blank in this table and if the entry is blank that means, the input there is some error, ok. And of course, if you come to this dollar and this things, so the accept state then it is then it is accepted.

And if there are multi some integer are multiply defined then there is problem, as I said that the grammar is not a SLR because the parser will be confused whether to do a shift operation or a reduce operation. Or if may it may so happen that two action both the actions are reduce of actions, but they are by two different rules. So, the person will not to know clearly like which rule it should use. So, that way it can give rise to the shift reduce conflict and reduce conflict. So, those conflicts are to be ideally they should not be there in the in the in the parsing table. So, if for a grammar we have a situation where these conflicts are not there then we will tell that the grammar is an SLR grammar.

(Refer Slide Time: 10:39)



So, let us look into an example which is not SLR, ok. So, let us look into this grammar where we have got this thing this S producing L equal to R o R, L producing star R id, R producing L. So, here this star id and equal to so these are terminal symbols, S, L and R they are non-terminal symbols.

So, initially I will have the state. So, with this I will be adding this additional rule is just producing S to get the augmented grammar G dash. Now, on this augmented grammar we try to construct the sets of items. So, this I0 is S dash producing dot S then the closer of that. So, S producing dot L equal to R, S producing dot R they will be coming from this rule, then L producing dot star R or L producing dot id and again since dot R is there, so R producing dot L will come. So, that is I0. I0 to I1, so S dash, so it is go to I0 S. So, go to I0 S is I1, so which is this one.

Similarly, from go to I0 L, so this is the I0 to I1. So, this is go to I0 S. And similarly, from I0 to I2 it is go to I0 on L. So, that way is the others other items are constructed accordingly. So, I am not going to that explanation but let us look into this item more carefully say this item I2. So, S producing L dot equal to R; that means, address of the first item says that I have seen a portion of the string where from which I can derive L and I am expecting the next input symbol to be equal to.

Now, what is the action? Now since the since it is equal to; so the if the next input symbol is equal to I will do a shift. So, the first part we will of will tell us, first item we

will tell me that. Second item telling me know if you even if you see equal to, so you better do a reduction by this rule, a better do a reduction by this rule R producing L dot. So, you have to look into the follow set of R, so you have to look into the follow set of R and for all the terminals in the follow set of R you have to reduce by R producing L. So, what is the follow set of R? So, follow set of R is equal to; so here by this rule L producing star R. So, whatever is in follow of L is in follow of R. So, follow of L is equal to follow of R follow of R will contain follow of L it may contain something more, but it will have this follow of L and follow of L has got this equality symbol. So, you see among the other symbols this equality symbol we will definitely be there. So, follow of L contains equality.

So, when I am doing the action the first one, the first rules says the, so by this rule S producing L dot R L dot equal to R. So, from I2 on equality it goes to I6. So, to from I2 to I6 this is go to I2 equality, ok. So, this is the shift action. So, so those naturally it tells me if I if I get an equality I should do a shift and I the new state will be 6. So, that is coming from this particular rule.

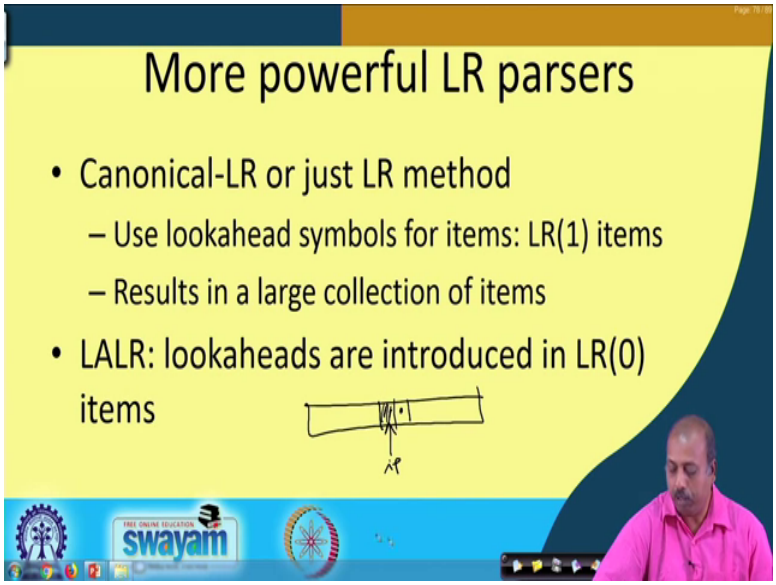
Now, this rule telling me that you should do a reduction, for which symbols? For all the symbols which are coming in the follow set of R and the follow set of R contains equality, so that means, if you see an equality then you should do a reduction by R producing L. So, this is giving rise to a shift reduce conflict on the symbol equality.

So, whenever if you are trying to draw the parses that have a parsing table then for the state 2 and for the input symbol equality, I will have two entries, one is shift 6 another is reduced by rule number 1, 2, 3, 4, 5, 6, reduced by rule number 6 ok, that way I will have two actions. So, so this is the, this particular grammar that is given is not an SLR grammar because it has got multiply defined entries for some of the multiply defined values for some of the entries. So, that has to be taken care of.

So, what is the solution? Like they, so one solution is that you modify the grammar. So, you modify the grammar whether it will catch the language syntax properly or not that is a question. If it if you can modify the grammar and make it SLR, it is well and good. So, if we cannot then we have to go for more powerful parsing strategy like say we have to go for something called say LR parser, called canonical LR parser which is more

powerful than SLR parser, but it will it will have more complexity the complexity of this SLR parser we will see that it is going to be much more compared to this SLR parser.

(Refer Slide Time: 16:35)



So, they are known as canonical LR, sometimes it is called C LR or sometimes it is called just LR. And this SLR parser is often called LR 0 parser, ok. So, in case of LR this canonical LR parser we use some lookahead symbols for items. So, they are called LR 1 items. So, it is they are not LR 0 items, so they are called LR 1 items. So, from the current position you do not only look at the current input symbol. So, what it means is this thing that if this is your, if this is your input stream then at present suppose you are at this position, so your input pointer is here.

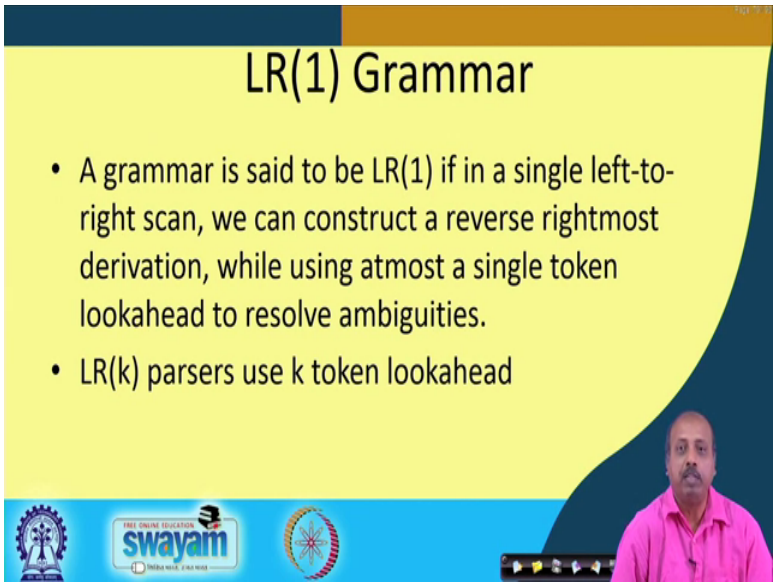
So, in case of LR parser, SLR parser we were taking a decision based on this particular input symbol only but in case of LR 1 parser will be looking ahead by one more symbol. So, we will be looking in to the next symbol also and accordingly take a decision. So, if we do that, then many of these conflicts that we are getting say shift reduce or reduce conflicts, so they may get dissolved as we are doing a lookahead. So, but the difficulty is that it will result in a large collection of items. So, that way the complexity of the parser will be very high.

And there is another variant which is known as LALR parser while lookaheads are introduced in the LR 0 items. So, here also it is there, but here the number of states will be much less compared to this canonical LR parser. So, SLR parser foremost of many

grammar so you can do the parser construction by hand. Canonical LR parser, so it becomes difficult because with as you are increasing the complexity of the grammar, so number of states will become pretty high.

And LALR also since it is depending on these LR 1 items, so it will be again be difficult, it will be difficult to construct by hand, but more, but they can be automated very easily. So, we have we will see that the automated tools for this parser generator, so they will generate LALR parsers because of the reason that LALR parser ultimately the number of states will be same as LR 0 or SLR parser. But power wise it will be as powerful as the canonical LR or LR 1 parser. So, we will see how they are done.

(Refer Slide Time: 19:12)



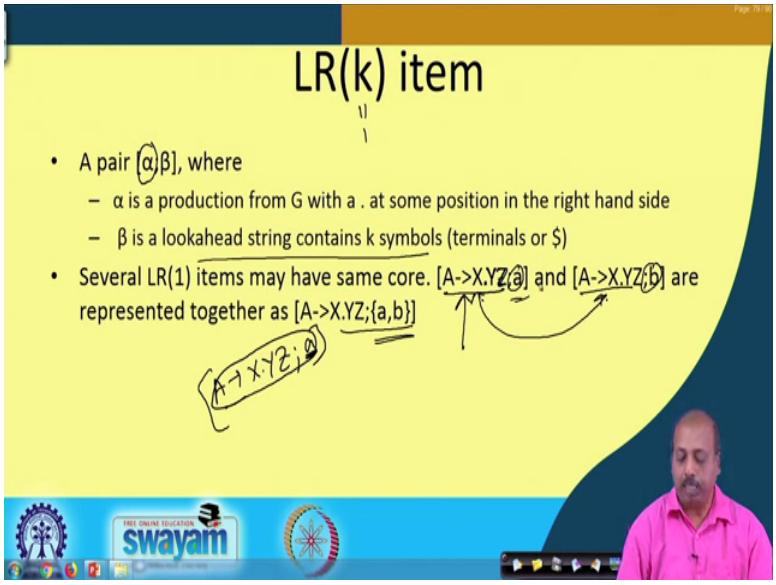
So, let us look into the LR 1 grammar. A grammar will be said to be LR 1 if in a single left right scan we can construct a reverse rightmost derivation while using at most single token lookahead to resolve ambiguities. So, in case of LR 0, so we were not doing any lookahead. In case of LR 1 we are doing lookahead by at most one token. So, you may or may not require to take a your may no may or may not need to base your decision on the next input, on the next input symbol may be at current input symbol only you can take a decision. But in case of confusion, ok. So, you can you are allowed to look into the next token and then take a decision.

So, and it will just like any other this LR parsers, so it will be doing a reverse rightmost derivation. So, it is a bottom up parser and it will be doing a rightmost derivation. So, at

every state the rightmost non-terminal symbol will be replaced by the corresponding right hand side of the production rule.

In general LR k parsers that will they will be using k token lookahead. So, they will be more and more powerful. So, as you are trying to doing by allowing more and more lookahead in the parser becomes more and more powerful. However, the parser complexity we will also increase significantly. So, normally we go with this LR 1 parser, we do not go beyond that because most of the programming languages and the constructs that we have, so LR 1 is good enough. So, we do not need to go beyond that, ok.

(Refer Slide Time: 20:50)



So, what is an LR k items. So, for all our discussion, so we will be taking k equal to 1. So, you can just generalize it to other values of k. So, this LR k item, so it is a sphere alpha semicolon beta, where alpha is a production from G of the grammar G with a dot at some position in the right hand side. So, this alpha part is similar to what we have in the LR 0 item. So, this alpha part is same as that one.

For example, if you have got say a producing X dot Y Z. So, this was an item in case of LR 0 parser. But in case of LR 1 parser there will be another look ahead string that contains k symbols, at most there, so lookahead string that will contains k symbols. So, if I am using k equal to 1; that means, I will have a single set terminal symbols after this. So, this is this is this is an item, A producing X dot YZ semicolon a. So, what is the meaning of that will see.

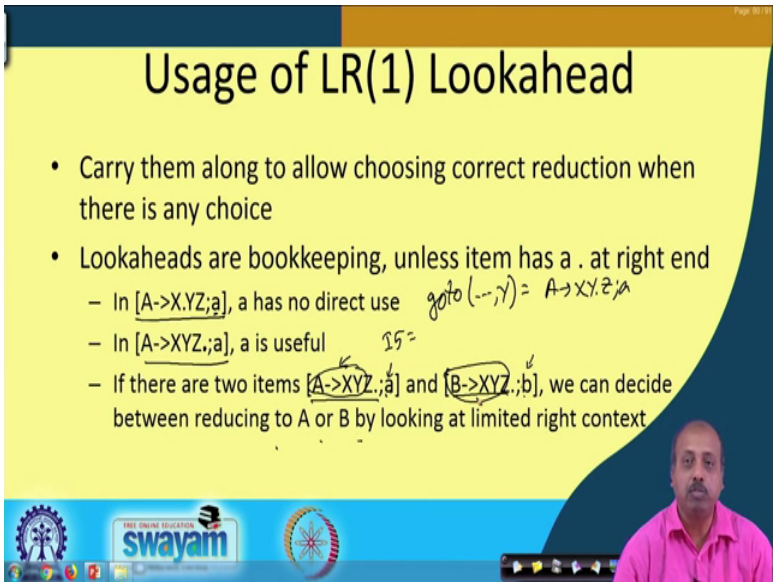
So, basically this part will be giving me the lookahead. So, what is what we are going to do a lookahead. Now, as I am doing this since the items they have got two parts one part is a rule with a dot somewhere and the other part is a terminal symbol or a terminal string then there may be several such items where the rule part is same, ok, rule with dot, so that part is same what differs is the beta part. So, though they will they are said to said to have same core. So, this one, so this is called the core and maybe. So, these two items if you see both of them having are having X producing, A producing X dot Y Z as the core. So, they have got the same core, but the next item that next lookahead token is different here it is A, here it is B.

Now, if I have got say a situation where this thing occurs then many a time will be writing in a short hand form like this A producing X dot Y Z semicolon then within brace. So, we write all the alternative strings for which this is an item. So, a in this particular case, so we have got two such items a producing X dot YZ semicolon a, A producing X dot Y Z semicolon b. So, we combine them together while writing as A producing X dot Y Z semicolon a comma b.

So, this way we will be representing LR k items, where we remember that this part is meaning of the core part is same as what we have done what we have in case of LR 0 item. So, this means that we have for examples say this one A producing X sot Y Z this means we have already seen a string which is derivable from X and after that this dot is there; that means, we are expecting to see a string which is derivable from this Y Z, ok. So, that was the meaning. So, this is fine.

So, this lookahead part, lookahead token, so this will be clear after sometime. So, we will be explaining why this is real, what is this and how is it going to help us.

(Refer Slide Time: 24:43)



So, as such for most of the time. So, this is this is like a baggage that we carry with the items. So, this look ahead item, so we carry them along to carry them along to allow choosing correct reduction when there is any choice. So, this is particularly, this lookahead will be particularly useful when we are going to have this reduction rules. So, in that case when there is a confusion in the reduction process, so then we will be taking help of that. So, we will we will be either using that reduction based on the two or more these tokens, this lookahead tokens; so based on that we will take a decision. Whether to reduce by one rule or the other or whether to do a reduction or not.

Then, so lookaheads are basically bookkeeping, so that way. So, they are bookkeeping. So, unless we have got an item that has got a dot at the writing. So, like if you consider say these two rules these two items, so for this item there is no directive. So, from this from this item, so we can try to generate say go to the item and y. So, we will be simply where may we will be simply getting an item XY dot Z and this part will be carried forward, this a part this token part. So, this will be carried forward. So, it is a lookahead. So, it is just a bookkeeping, whereas, for this particular case where this dot is at the end then it is going to be useful.

How is it going to be useful? Suppose I have got I am at a state say I5 where I have got these two items. So, A producing XYZ dot semicolon a and B producing XYZ dot semicolon b. So, previously what we were doing? So, if we have to do a reduction then

this rule tells me that for all the symbol all the terminal symbols for the in the follow set of A, I have to do reduction by this rule and this rule is telling me that for all the all the terminals in the follow set of B I have to do by this reduction.

Now, if the grammar is such that this follow set of A and follow setup of B, they have got some terminals common between them then this will immediately give rise to a reduce conflict. Now, what this A and B is going to do is that it is trying to modify the context. So, it will see whether the next input symbol is A or not and this will see the next input symbol is B or not, that is a look ahead token. So, the lookahead token is a then only this reduction will be done. If the look ahead token is B then this reduction will be done. And since these two tokens are different they are not same token then we can definitely take a decisions, that read that reduce conflict will get resolved in this fashion.

So, we can decide between reducing to A or B by looking at limited right context. So, we are not doing very large amount of lookahead, we are looking ahead by a single symbol single token and then we are taking a decision. So, that way it is going to be powerful. So, if the if the follow set of A and B are totally disjoined from each other then of course, we were the it would have been an LR 0 grammar, so it would have been an SLR grammar. But if they are not disjoint the A and B, follow set of A and B if they are not disjoint then this grammar will not be an SLR grammar. But with this is if this look if this look ahead tokens are different then this grammar is definitely an LR 1 grammar. So, this way it will be helpful, in this lookahead token they can be utilised for resolving the reduce conflicts between a number of rules.

So, but the problem is that you see that similar items, so they will go on coming only the look ahead part may be different, as a result they will give rise to large number of states or large number of items and that is the thing. So, number of states that you get with this LR 1 parsing policy it will be much larger compared to the LR 0 parsing policy.

And this is exactly the reason why this thing happens, because if we had relied only on this rule part this dot this left side. So, the A producing X dot YZ then number of states in SLR and this canonical LR they would have been same the same, but because of this lookahead token so they are going to be different and they are going to have more number of states in case of this canonical LR parser.