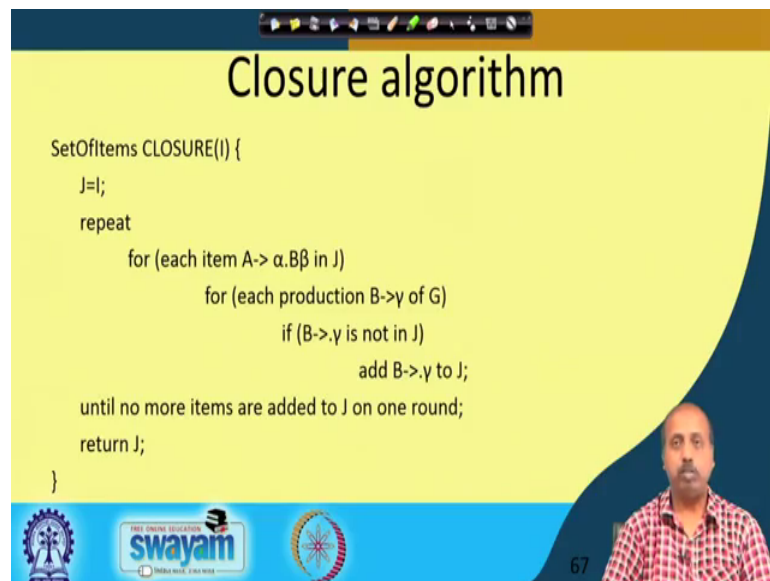


**Compiler Design**  
**Prof. Santanu Chattopadhyay**  
**Department of E & EC Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 28**  
**Parser (Contd.)**

(Refer Slide Time: 00:15)



**Closure algorithm**

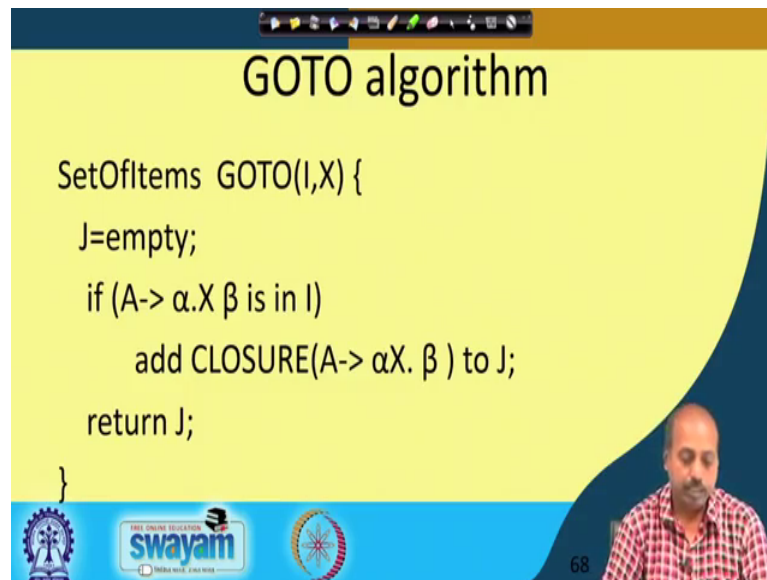
```
SetOfItems CLOSURE(I) {  
    J=I;  
    repeat  
        for (each item A-> α.Bβ in J)  
            for (each production B->γ of G)  
                if (B->γ is not in J)  
                    add B->γ to J;  
    until no more items are added to J on one round;  
    return J;  
}
```

The slide features a yellow background with a blue header and footer. A video inset of a man in a red and white checkered shirt is visible in the bottom right corner. The footer includes logos for IIT Kharagpur, Swayam, and a circular emblem, along with the number 67.

So, we can look into a closer algorithm. So, set of items of closure I. So, it will return a set of items which is the. So, the closure I is the function. So, we start with a set J; J is initialized to I. So, whatever you have in thus item I here. So, J will have their those items and then for each item A producing alpha dot B beta in J and for each production B producing gamma of J. So, if B dot gamma is not in J, then we add B dot gamma to J. So, this algorithm we have already we intuitively seen like while discussing on the examples.

So, this is a formal way of writing the algorithm. So, until no more items can be added to J on one round. So, if you try with all the rules and then find there no more production can be added, then we will be stopping the algorithm and it will be returning the set of items J.

(Refer Slide Time: 01:11)



## GOTO algorithm

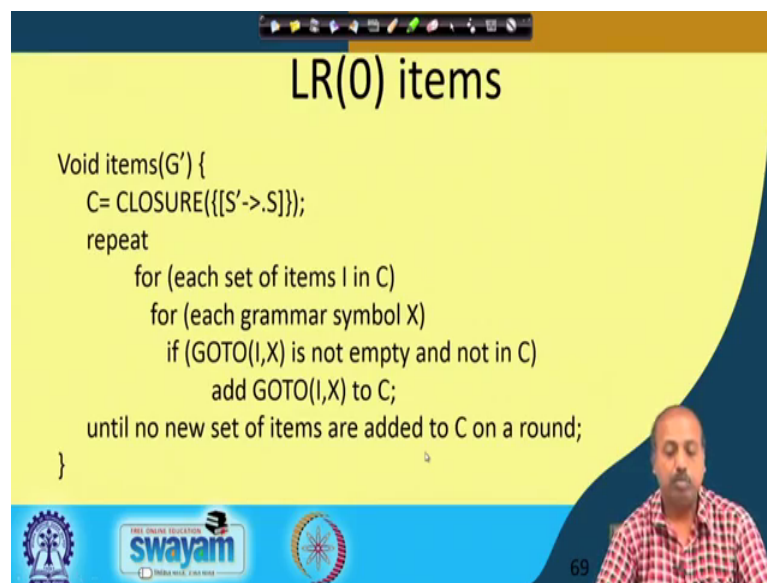
```
SetOfItems GOTO(I,X) {  
    J=empty;  
    if (A->  $\alpha$ .X  $\beta$  is in I)  
        add CLOSURE(A->  $\alpha$ X.  $\beta$  ) to J;  
    return J;  
}
```

68

So, that is the closure algorithm. Similarly, the GOTO algorithm. So, this also we have seen the GOTO operation. So, this is very simple. So, GOTO I, X from item I on input on the input X where do you go? So, input maybe a terminal or a non terminal or a basically some grammar symbols. So, its start with J equal to empty and this if A producing  $\alpha$  dot X  $\beta$  is in I, then we add closure of A producing  $\alpha$  X dot  $\beta$  to J. So, this is  $\alpha$  dot X  $\beta$  is in I. So, on X it will go to  $\alpha$  X dot  $\beta$  and then we take the closure of it and then add that item to J. So, it will be returning J.

So, this is the GOTO part.

(Refer Slide Time: 01:59)



## LR(0) items

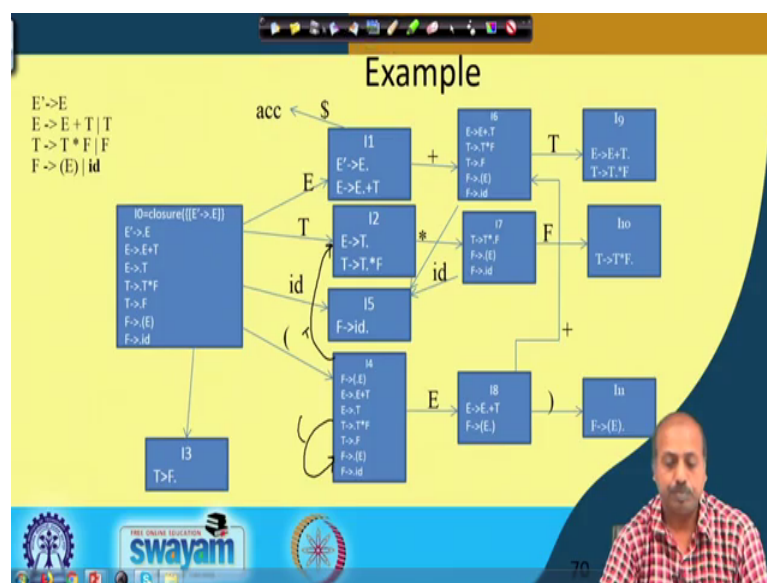
```
Void items(G') {  
    C = CLOSURE({[S' -> .S]});  
    repeat  
        for (each set of items I in C)  
            for (each grammar symbol X)  
                if (GOTO(I, X) is not empty and not in C)  
                    add GOTO(I, X) to C;  
    until no new set of items are added to C on a round;  
}
```

69

Now, the algorithm for constructing the LR 0 items. So, we start with. So, we start with the augmented grammar  $G'$ ; then this is the grammar that is passed. So, we start with the initial set which is the closure of this particular rule  $S' \rightarrow \cdot S$ . So, this as you remember that this  $S' \rightarrow \cdot S$  is the additional rule that has been put into the set into the grammar  $G$  to get the augmented grammar  $G'$ . So,  $C$  equal to closure of  $S' \rightarrow \cdot S$ . Now, for each set of items  $I$  in  $C$  and for each grammar symbol  $X$ . So, if  $GOTO(I, X)$  is not empty and not in  $C$ ; then we add  $GOTO(I, X)$  to  $C$ .

So, we have already seen how to make this  $GOTO(I, X)$  function and the closer function. So, if it is if you if it can give us some new item then will be adding it to  $C$ . until no new set of items can be added to  $C$  on a round. So, this way it will be constructing the LR 0 items. So, this is this is simply the procedure.

(Refer Slide Time: 03:17)



Now, once we have done that. So, let us take an example of grammar and how this items can be constructed. So, if this is the grammar  $E \rightarrow E + T$  producing  $E$ . So, this is the originality of grammar augmented with the rule  $E \rightarrow E + T$  producing  $E$ .

And the initial item is constructed by closure of  $E \rightarrow E + T$  producing  $\cdot E$  this item. So, this  $E \rightarrow E + T$  producing  $\cdot E$ , you will have  $E \rightarrow E + T$  producing  $\cdot E$ . So, this we have already discussed. So, whatever you have got dot before  $E$ . So, all these rules survive  $E \rightarrow E + T$  producing  $\cdot E$  plus  $T$  and  $E$  producing  $\cdot T$  from  $E$  producing  $\cdot T$ . So, if this rule will come  $T \rightarrow T * F$  and all. Now from these, so we take the GOTO. So, you look into the symbol that comes after dot. So, after dot we have got  $E$   $T$   $F$  open parenthesis and  $id$ . So, for each of these cases, so I have to find what is the corresponding GOTO.

So, if I do it on  $E$ . So, I will get  $E \rightarrow E + T$  producing  $E \cdot$  and  $E$  producing  $\cdot E + T$ . So, that is there. Then, you have got this thing  $T$ . So, if you get  $T$ , then this  $E$  producing  $T \cdot$  and  $T$  producing  $T \cdot * F$  they will come; from  $id$ , I will get I will get  $F$  producing  $id \cdot$  and from this open parenthesis, I will get  $F$  producing  $\cdot E$  and since  $\cdot E$  is coming, so all the rules will come. So,  $E$  producing  $\cdot E + T$ , then  $E$  producing  $\cdot T$ ; then  $T$  producing  $\cdot T * F$ . So, all these rules will come. So, this is all these four sets that we have consider seen. So, they are all new sets and also the some  $F$ . So, it will go

to. So, this labeling is missing. So, this should be on F. So, this is on F, it comes to the state 3, I 3. So, this is E produce T producing F dot.

Because nothing more has to be done, so it is like this. Now, from this one from I 1 you see that I have got a dot before plus. So, on plus it will go to the state E producing E plus dot T and from this E plus dot T. So, this dot is there before T. So, again all these rules will come T producing dot T star F T producing dot F producing dot within bracket E and all that. So, all of them will come and from this one there is a dot before star. So, based on that it will come to this rule, T producing star dot F and then the since there is a dot before a F. So, it will be all this F rules will come F producing dot E dot within bracket E and F producing dot id like that. Similarly, from this I cannot have any more production because this is this is no symbol after this dot. From this for this one I will get this dot E. So, dot E it will be adding this thing. So, dot E will be. So, on E it will go to dot so, E plus dot T.

And this F producing E dot within bracket E dot like that and. So, from this one I will have also rules like on T, on T it will go to go to E producing T dot. So, it will have E producing T dot and T producing T dot star F. So, these 2 will come and that will give me the rule I 2 only; this is the set of items I 2 only. So, that does not give us any new set of item, similarly, on F within bracket. So, from this state if we were try to take a transition on F producing, so on this GOTO I 4 open parenthesis, then it will give me this rule F producing open parenthesis dot E close parenthesis. Now since so this is same as I 4 because I 4 was like that only. So, this will be giving us I 4 only. So, if you were try to draw explicitly, then you can you can draw these type of transitions. So, like this. So, this is on open parenthesis.


Similarly, this is on T; this is on T. So, like that you can draw other cases ok. So, this is not they are not showed here, so, apart. So, if you if you explore this algorithm you will find that accepting these set of items no new items are generated. So, all others are just duplicate of the previous ones. So, that way you can just write down the set of items that are generated from the grammar rules.

(Refer Slide Time: 08:19)

## Use of LR(0) automaton

- Example:  $id*id$

| Line | Stack | Symbols  | Input     | Action                         |
|------|-------|----------|-----------|--------------------------------|
| (1)  | 0     | \$       | $id*id\$$ | Shift to 5                     |
| (2)  | 05    | $sid$    | $*id\$$   | Reduce by $F \rightarrow id$   |
| (3)  | 03    | $sF$     | $*id\$$   | Reduce by $T \rightarrow F$    |
| (4)  | 02    | $sT$     | $*id\$$   | Shift to 7                     |
| (5)  | 027   | $sT^*$   | $id\$$    | Shift to 5                     |
| (6)  | 0275  | $sT^*id$ | $\$$      | Reduce by $F \rightarrow id$   |
| (7)  | 02710 | $sT^*F$  | $\$$      | Reduce by $T \rightarrow T^*F$ |
| (8)  | 02    | $sT$     | $\$$      | Reduce by $E \rightarrow T$    |
| (9)  | 01    | $sE$     | $\$$      | accept                         |



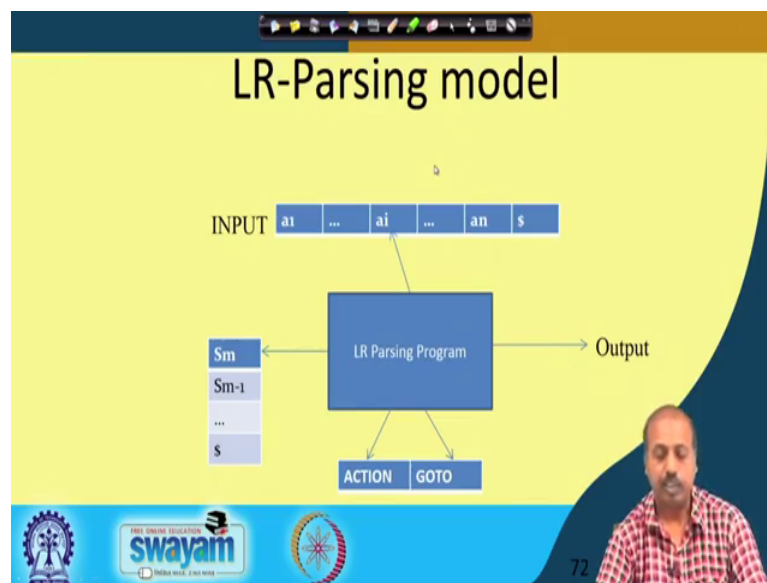
71

Now, so, next we will be looking into how to use this LR 0 automaton for identifying the strings. So, initially the stack contains the state 0 and this as the symbol is dollar and input is  $id\ star\ id\ \$$ . Then, it will say that once it is getting this  $id$ . So, it will be shifting to state 5. So, how does it shift? So, that decision we will see later. Suppose, if the action is the like shift the next input and go to state 5. So, it shifts the next input  $id$  into the stack and go to state 5.

So, this 5 state is put into the stack. Now it says now the next input is  $id$  and the sorry star and the current state is 5. So, current state is 5 and the input is star. So, it finds that there is a rule like  $F$  producing  $id\ dot$ . So, based on that it understands that it has seen an seen a handle. So, it will reduce these by  $F$  producing  $id$ . So, this  $id$  will be taken out from the stack and now, the new state will be this  $F$  will be put into the stack the left hand side and the new state will be determined by the GOTO part.

We will see that algorithm. So, it will come to state 3. So, this way this algorithm will proceed and it will finally, come to this accept state. I am not going to explain it in detail because it is not possible to understand it until unless we have seen in more detail how this parsing algorithm works with the help of this basic set of items and the transition diagrams.

(Refer Slide Time: 10:01)




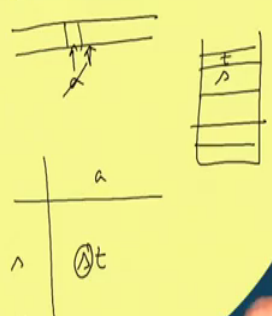
So, this is how the input works that this parsing process works. So, we have got this LR parsing algorithm. So, it has got a stack which has got these states in it. So, this stack will have that state. So, initially this is dollar and it has got the other states in it and then, this is the input  $a_1, a_2$  up to ended with a dollar.

And there is a parsing table. So, which has got two parts in it; one part is called Action part, another is called GOTO part. So, this Action part and GOTO parts; so, these two are there. So, this LR parsing program, so, it will take that it will consider the current input; it will consider the. So, it will consider the current state, the current input and it will consult the corresponding table and based on this table it will take some action whether to shift the next input or whether to do a reduction; whether to accept and all accordingly it will output may be the production rule by which it does the reduction or the declaration that the input is accepted or error. So, like that. So, that is the output part.

(Refer Slide Time: 11:13)

## LR parsing algorithm

```
let a be the first symbol of w$;
while(1) { /*repeat forever */
  let s be the state on top of the stack;
  if (ACTION[s,a] = shift t) {
    push t onto the stack;
    let a be the next input symbol;
  } else if (ACTION[s,a] = reduce A->β) {
    pop |β| symbols of the stack;
    let state t now be on top of the stack;
    push GOTO[t,A] onto the stack;
    output the production A->β;
  } else if (ACTION[s,a]=accept) break; /* parsing is done */
  else call error-recovery routine;
}
```



So, this is the LR parsing algorithm. So, if let  $a$  be the first symbol of the input stream  $w$ . I repeat now what we do and let  $s$  be the state on the top of the stack. Now, if action of  $s$  and  $a$  is shift  $t$ . So, if the action is shift  $t$ , so, it will push  $t$  onto the stack and let, now let  $a$  be the next input symbol. Otherwise if the action is by reduce  $A \rightarrow \beta$ . So, it will pop out  $\beta$  symbols from the stack and now if the state  $t$  is on the top of the stack, then it will go to it will push  $GOTO[t, A]$  onto the stack and output the production  $A \rightarrow \beta$ ; else if the action is accept then it will accept it, otherwise it will call in a recovery routine.

So, what does it mean? So, if this is the input, so, at any point of time suppose this is the symbol  $a$  that we have and the current state  $s$  that is on the stack, we have got the states and the current state is say  $s$ . Now it will consult the table, the parsing table for the state  $S$  and input  $a$  and if the corresponding input is shift  $t$ . So, we will represent it normally by  $s \rightarrow t$ . So, it will be doing. So, part identifies the that it is a shift action. So, it will be a shift operation and the new state will be  $t$ . So, in that case what it will do? It will push  $t$  onto the stack. So, it will be pushing  $t$ , so, this  $s$ . So,  $t$  becomes the state on the stack and the input pointer will be advanced to the next one. So, that way this pointer is advanced.

On the other hand, if action is reduced by  $A \rightarrow \beta$  then it will be popping out  $\beta$  number of entries from the stack and then, it will be if  $t$  is on the top of the stack

now; then it will be finding out this GOTO t, A these state and it will be putting it into the stack. So, we will see some example and then it will be more clear ok.

(Refer Slide Time: 13:35)

**Example**

| STATE | ACTION |    |    |    |    |     | GOTO |   |     |
|-------|--------|----|----|----|----|-----|------|---|-----|
|       | id     | +  | *  | (  | )  | \$  | E    | T | F   |
| 0     | S5     |    |    | S1 |    |     | 1    | 2 | 3   |
| 1     |        | S6 |    |    |    |     |      |   | Acc |
| 2     |        | R2 | S7 |    | R2 |     |      |   |     |
| 3     |        | R1 | R4 |    | R4 |     |      |   |     |
| 4     | S5     |    |    | S4 |    |     | 8    | 2 | 3   |
| 5     |        | R6 | R6 |    | R6 |     |      |   |     |
| 6     | S5     |    |    | S4 |    |     | 9    | 3 |     |
| 7     | S5     |    |    | S4 |    |     |      |   | 10  |
| 8     |        | S6 |    |    |    | S11 |      |   |     |
| 9     |        | R1 | S7 |    | R1 |     |      |   |     |
| 10    |        | R3 | R3 |    | R3 |     |      |   |     |
| 11    |        | R5 | R5 |    | R5 |     |      |   |     |

| Line | Stack   | Symbols | Input    | Action          |
|------|---------|---------|----------|-----------------|
| (1)  | 0       |         | id*id+\$ | Shift to 5      |
| (2)  | 0 5     | id      | *id+\$   | Reduce by F→id  |
| (3)  | 0 5     | F       | *id+\$   | Reduce by T→F   |
| (4)  | 0 2     | T       | *id+\$   | Shift to 7      |
| (5)  | 0 2 7   | T*      | id+\$    | Shift to 5      |
| (6)  | 0 2 7 5 | T*id    | +\$      | Reduce by F→id  |
| (7)  | 0 2 7 5 | T*id    | +\$      | Reduce by T→T*F |
| (8)  | 0 2     |         | +\$      | Reduce by E→T   |
| (9)  | 0 1     | E       | +\$      | Shift           |
| (10) | 0 1 5   | E+      | id\$     | Shift           |
| (11) | 0 1 5   | E+id    | \$       | Reduce by F→id  |
| (12) | 0 1 5   | E+id    | \$       | Reduce by T→F   |
| (13) | 0 1     | E       | \$       | Reduce by E→E+T |
| (14) | 0 1     | E       | \$       | accept          |

Handwritten notes on the right side of the slide include a list of grammar rules: (0) E → E, (1) E → E + T, (2) E → T, (3) T → T \* F, (4) T → F, (5) F → (E), (6) F → id. There are also diagrams of a stack and a state transition arrow.

So, we do not know how do we construct this table ok, this parsing table how do we construct suppose that is not known. So, let us see how this suppose this table is given to us and the meaning the way we read this table is say in these state. So, this state 0, so if you get an input id; then this is the corresponding action. It says that you do a shift operations; so input will be shifted and it will be coming to a state 5 ok. That is the that is how we will read it.

Similarly this is S 4; that means, if the current if the in state 0 if you find open parenthesis. So, you will do a shift operation and it will come to state 4. Now this one. So, this reduce. So, this one. So, R 4, it tells that you have took in state 3 if the input is star; if the input is sorry input is plus, then you have to apply reduction rule number 4 for doing a reduction. So, for that purpose this grammar rules and numbers like 0 1 2 3 4 like that. So, it says that you have to apply a grammar rule and do the reduction by that ok.

So, let us see how this thing proceeds. So, let us taken consider an example say this is the string. So, id star id plus i id. Now initially my input pointer is here and this stack contains state 0 and there is nothing in nothing symbols in the stack. Now, it will be finding. So, this is the input i d. So, this will be from state 0 and id state 0 id it says that it should be state 5. So, 5 the state 5 is pushed into the stack and then, this symbol id is also

put the we have seen the symbol i d. So, this is on the this is also on the stack. Now it says the next is star. So, this is star is. So, now, state 5 star. So, it says reduced by rule number 6 and rule number 6 is F producing id. So, it says. So, it will. So, a reduction. So, this id will be taken out. So, id will be replaced by F and the new state will be identified from this.

So, to identify the new state, what we will do? For a new state whatever if t is now the top of the stack, then GOTO t, A will be the new state. So, now after taking it so 3 is on the top of the stack, so 3 F; so, 3 F sorry. So, sorry this 3 is also popped out because from state 3. So, it has found F. So, this is a we are in at this point. So, this is this is be reduced by F producing id. So, this state 5 has gone out. So, this state 5 has gone out. So, this top of the stack now contains 0 and this 0 F. So, 0 F says that the new state is 3. So, this F is put into the stack and this 3 is put into the stack.

So, we have F is remembered as symbols next symbol and 3 is the stack. Now the next at state 3 I have got this star. So, state 3 star says that you do a go by rule number 7; reduced by rule number reduced by. So, this is basically. So, this is not rule number 7. So, this is rule number 4, there is a mistake here. This should be rule number 4. So, reduce by rule number 4; so, this 3 1 getting star. So, it will be on state 3 on getting star. So, it will be a reducing by T producing f. So, T producing F is rule number 4. So, this should be rule number 4. So, it will reduce by that. So, this F will be going out of the out of the symbol set and this T will be coming.

And the new state will be 0 T. So, 0 T is state number 2. So, this 2 is the new state and now on 2 star; so, 2 star, it says that shift and GOTO state 7. So, this it has done a shift. So, star is shifted and this 7 is there on the stack. Now state 7 is on the stack. Now from there it will find that now it is 7 and I d. So, state 7 id, it says shift 5. So, this id is shifted and this new state becomes 5. So, that is put into the stack. Now 5 and plus. So, 5 plus it says that reduce by rule number 6. So, reduce by rule number a rule F producing I d. So, it will be taking out id from the stack, replace it by F and then F is put into the stack and then from this 5 is also gone.

And from this 7 F; so, 7 F will tell that the GOTO is GOTO 10. So, this 10 is put into the stack. So, new state is the state 10 ok. Now from state 10 on plus from state 10 on plus, it says reduced by rule number 3. So, reduce by rule number 3 is this one, T producing T

star F. So, this T star F will be going out of the stack and then, this state 10 will also be going out. So, the new state is 7 and the left hand side is T. So, 7 T 7 from actually this state 7 will also go. So, it will be the state new state should be 2. So, where is 2? So, this will be actually when it is popping out when it is taking out these T star F. So, all these states will also go out. So, we have popping out 3 in entries from the symbol. So, this 3 states will also go. The 10, 7 and 2 they will go and 0 on F, so it will say that from state 0 on T, it will say that it will go to T; so, 0 T. So, this state two comes here. Actually the point is the here I am popping also here; I have to replace by the right hand side has got 3 star F. So, 3 symbols.

So, 3 symbols are take it be symbols taken from these symbols and 3 states all also to be taken from this stack because for each symbol there is a corresponding states. So, if I am taking out 3 steps also 3 symbols from here. So, 3 states are also to be taken out from here. So, this takes it to 0 state; it makes state 0 at the top and from state 0 on getting on the next input T. So, GOTO 0 T is 2.

So, it goes to state 2. So, it 0 to 2 and T. So, now, from a state 2 on plus state 2 on plus it goes to reduce by rule number 2. So, it will follow rule number 2 that is E producing T. So, it will be removing this symbol from the stack, it will be removing these state from the stack and now this left hand side is E. So, this is replaced by E and this 0 E. So, 0 E is 1. So, this state is 1. Now 1 and plus. So, 1 plus says shift by shift and go to state 6. So, this plus is shifted and it goes to a state 6 and now it will be 6 and i d. So, 6 and id tells it is shift 5. So, it will be it will be shifting. So, it is E plus id and this is 5 and next it will be 5 and dollar ok.

So, 5 and dollar. So, is reduced by rule number 6; so, F producing i d. So, this id goes out this 5 also goes out. So, F comes in and 6 F is you know 6 F is 3. So, accordingly the state becomes 3. Now, 3 and dollar; so, 3 and dollar tells 3 and dollar tells reduce by rule number 4 that is T producing f. So, this F is taken out, T is put into the stack and the now thus, this 3 is also gone; so, 6 and T. So, 6 T is 9. So, it puts the state 9 into the stack and from 9 dollar; so, 9 dollar it says reduce by rule number rule number 1. That is E producing E plus T. So, it will reduce it by this rule. So, E is. So, this E plus T is taken out and so, 1 2 3 symbols. So, 1 2 3 states are taken out.

Now 0 and E, so 0 E says 1. So, it is 1 here; E here and this is dollar. So, this one dollar says accept. So, it will come to the accept state. So, in this particular case; so you can think that as if I have got for understanding this is in a better fashion. So, you can we will we can assume that as if we have got a stack of states. So, this is state stack and we have got a symbol stack. So, whenever we are popping out something from the symbols stack, we have to pop out equal number of entries from the state stack and when whenever we are doing a push operation or shift operation, the state is shifted here and the symbol is shifted on to these stack.

In some sometimes, so we can also visualize it like this that we have got a single stack, we have got a single stack where we push both the symbols and stack; then the another stack states ok. Then alternately I have got a symbol. So, this is this maybe say id and this maybe some state number 0. This maybe plus; then this maybe state number say 2. So, like that. So, I have got alternate between the symbol and the state. So, that can also be done. So, in many books we will find that the convention followed is that way. So, it is a single stack where we have got the alternation of symbols and stack the symbols and states and so that whenever we are talking about popping out, so we pop out twice the number size of the right hand side; twice the size of the production. So, like this.

So, if you are applying say for example, say this rule T producing T star F. So, in the stack you have a situation where you have got this F after that some state number, then star; after that some state number and then T; after that some state number. So, then you are so you have to pop out all these entries from the stack ok. So, that way it is 2 into size of the right hand side. So, 2 it; so, total number of entries popped out is 1 2 3 4 5 6. So, 6 entries are to be popped out. So, many books we will find that it will say you take out 2 into number of symbols on the right hand side of the production from the stack.

So, both are correct. So, either you can consider two different stacks; one for state; one for symbol or you can take it as if there is a single stack where both symbols and stacks are put into symbols and states are put into and it just whenever you are popping out will be taking out twice of that. So, many times we will be following the other convention as well in our later classes for doing this comparison taking while we take some examples maybe we will be doing the other way.

(Refer Slide Time: 27:03)

## Constructing SLR parsing table

- Method
  - Construct  $C = \{I_0, I_1, \dots, I_n\}$ , the collection of LR(0) items for  $G'$
  - State  $i$  is constructed from state  $I_i$ :
    - If  $[A \rightarrow \alpha \cdot a \beta]$  is in  $I_i$  and  $\text{Goto}(I_i, a) = I_j$ , then set  $\text{ACTION}[i, a]$  to "shift  $j$ "
    - If  $[A \rightarrow \alpha \cdot]$  is in  $I_i$ , then set  $\text{ACTION}[i, a]$  to "reduce  $A \rightarrow \alpha$ " for all  $a$  in  $\text{follow}(A)$
    - If  $[S' \rightarrow \cdot S]$  is in  $I_i$ , then set  $\text{ACTION}[i, \$]$  to "Accept"
  - If any conflicts appears then we say that the grammar is not SLR(1).
  - If  $\text{GOTO}(I_i, A) = I_j$  then  $\text{GOTO}[i, A] = j$
  - All entries not defined by above rules are made "error"
  - The initial state of the parser is the one constructed from the set of items containing  $[S' \rightarrow \cdot S]$

So, for constructing SLR parsing table, so first of all we have to construct the collection of LR 0 items for  $G$  dash.

And this state is constructed like this. So, if  $A$  producing  $\alpha \cdot a \beta$  is in a particular item and  $\text{Goto } I_i a$  is  $I_j$ , then we have to set action  $i, a$  to shift  $j$  and now we have if we have got  $A$  producing  $\alpha \cdot$  a production, then is that is in the set of item in the item  $I_i$ , then one action  $i$  so, it will be reduced by  $A$  producing  $\alpha$  for all  $a$  in the follow of  $A$ . So, this basically it is like this that if you have got some in some sentential form where in this portion, you have got other string  $\alpha$ . That is you have already seen something which is derivable from  $\alpha$ . Now if you are attested where  $A$  producing  $\alpha \cdot$  is there; then, after this you are expecting to see something which is which can follow the symbol is.

So, this part whatever comes. So, it is in the follow set of  $A$ . So, that is why this rule that is you look into the follow set of  $A$  and for all those symbols you add the action reduced and this and if this is the case where it is the when you are going to do this particular reduction is  $S$  dash producing  $S \cdot$ . So, it is going to be the accept state. So, we will discuss about this parsing table construction strategy in detail in the next class.