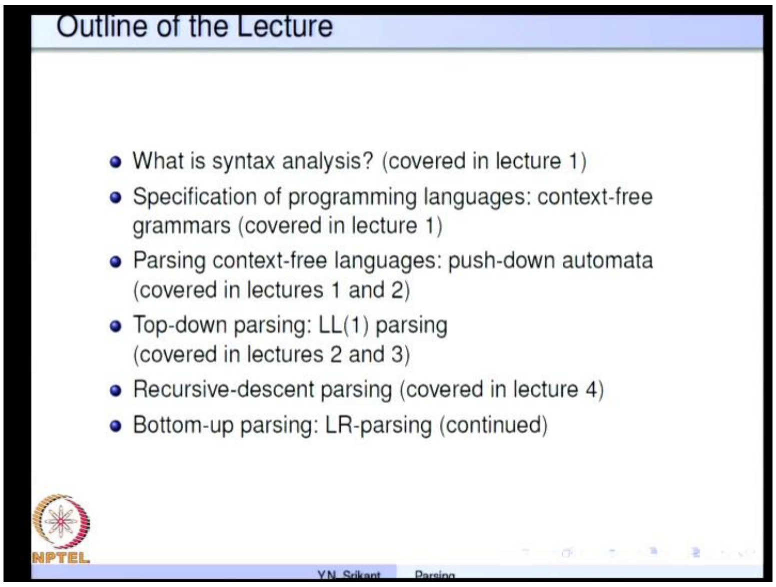
Principles of Complier Design

Prof. Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Lecture - 10
Syntax Analysis: Context-free Grammars, Pushdown Automata and Parsing Part - 6

(Refer Slide Time: 00:21)

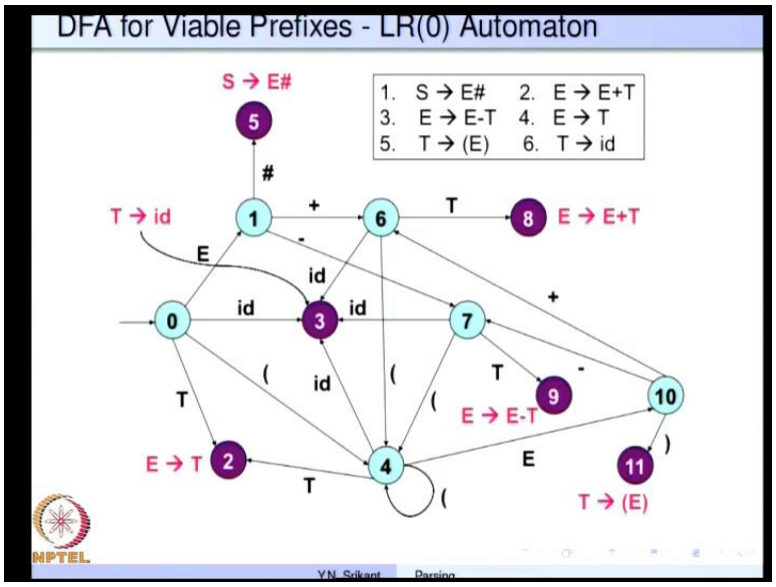


Outline of the Lecture

- What is syntax analysis? (covered in lecture 1)
- Specification of programming languages: context-free grammars (covered in lecture 1)
- Parsing context-free languages: push-down automata (covered in lectures 1 and 2)
- Top-down parsing: LL(1) parsing (covered in lectures 2 and 3)
- Recursive-descent parsing (covered in lecture 4)
- Bottom-up parsing: LR-parsing (continued)

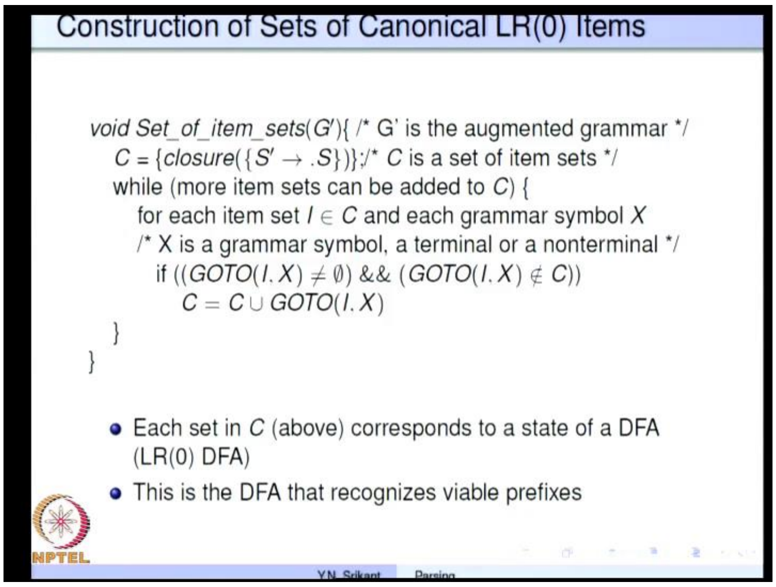
Welcome to the lecture on syntax analysis part 6. So, today we will continue with bottom up parsing study what are the S L R 1 grammars, L R 1 grammars, etcetera, etcetera.

(Refer Slide Time: 00:30)



To do a bit of recap. So, we discuss the concept of viable prefixes items valid items. And then we also discuss the procedure for building the L R 0 automaton, here is the example I showed you the last time.

(Refer Slide Time: 00:54)



Construction of Sets of Canonical LR(0) Items

```
void Set_of_item_sets(G'){ /* G' is the augmented grammar */
   C = {closure({S' → .S})};/* C is a set of item sets */
   while (more item sets can be added to C) {
      for each item set I ∈ C and each grammar symbol X
      /* X is a grammar symbol, a terminal or a nonterminal */
         if ((GOTO(I, X) ≠ ∅) && (GOTO(I, X) ∉ C))
            C = C ∪ GOTO(I, X)
   }
}
```

- Each set in C (above) corresponds to a state of a DFA (LR(0) DFA)
- This is the DFA that recognizes viable prefixes

And the method of construction is very simple, you know they to begin with the item S prime to dot S is the only which we are going to ((Refer Time: 01:08)), and then we take the closure on it. So, that gives you an initial set of items and now we continuously apply the go to function and go on including as many sets of items as possible into the collection C. And when we cannot add any more items we stop. So, each set in the above collection is a state of the LR 0 DFA, and this DFA recognizes viable prefixes. So, each state of the DFA contains only items which are valid for a particular viable prefix or a set of viable prefixes.

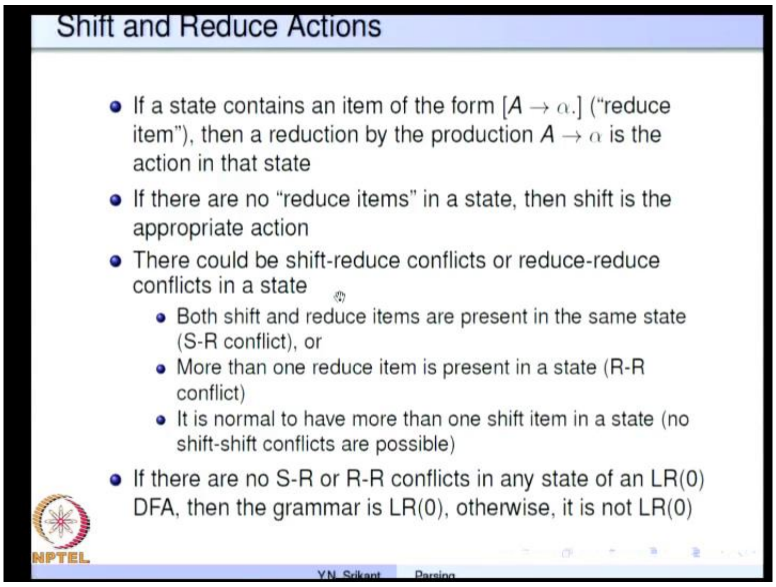
So, let me explain how exactly L R 0 parsing happens with respect to this particular DFA. So, let us consider the simple string i d which is derived from this grammar. So, we start with S going to E hash apply this production And then when we apply the production E going to T and finally, we apply the production T going to I d. So, we get the string i d hash. So, let us see how it is passed using this L R 0 automaton. So, as usual we begin with state 0 which is the initial state and then on the input i d we go to state number 3. So, remember state 0 on the stack now i d is pushed on to the stack when we see and it takes you to state 3 which is again push down to the stack. So, instead three it says it is a reduce state. So, and the reduction is by the production T going to I d. So, the implication of the this would be to pop the top symbol of the stack.

And since the state is also included we pop symbols of the stack and that actually exposes the state 0 on the stack now the non terminal on the left hand side is t. So, we apply the go to function on state 0 and that is takes us to state number 2. So, now, we have pushed the non terminal T on to the stack and a state number 2 is also on to the stack now it says again reduce by E to t. So, we pop state 2 and the non terminal T from the stack again we get state 0 as the top of stack. And now, since the non terminal on the left hand side is E we apply the go to function on the state 0 and the non terminal E which takes us to the state number one. Now, the next input symbol is hash because this is a shift state it is not a reduce state and on hash it goes to state 5 where a reduction by S going to E hash is actually indicated.

So, we again pop items for the stack and you know this state 0 and we push the this non terminal S on to the stack and that is an exception acceptance. So, this is how the terminal string i d is passed. So, let us consider a slightly more complicated terminal you know string i d plus I d. So, i d plus i d is hash derive from this grammar using S going to E hash then E going to E plus T this T will go to i d this E will go to T and then go to I d. So, that is how i d plus i d would hash would be derived again we start from 0 go to state 3 which indicates a reduction go back and on T we go to state 2 which again indicates a reduction. So, go back to state 0 pop symbols and then go to state 1 on E.

So, now, the plus will takes us to state 6 and i d will takes us to state 3 again there is a reduction. So, remember that 3 and 6 are on the stack. So, we pop these two i d you know sorry i d and three are pop from the stack to expose the non terminal 6 and 6 on the non terminal T takes us to state 8. So, here we now have on the stack E plus E 0 E one plus 6 T and eight which indicates a reduction by E to E plus t. So, we go all the way back to state 0 and on E we again go back to state 1. So, and on hash makes it go to state 5 and we accepts the string. So, this is how a parsing happens using this automaton of course, all this made easier once we actually have the table itself.

(Refer Slide Time: 06:16)

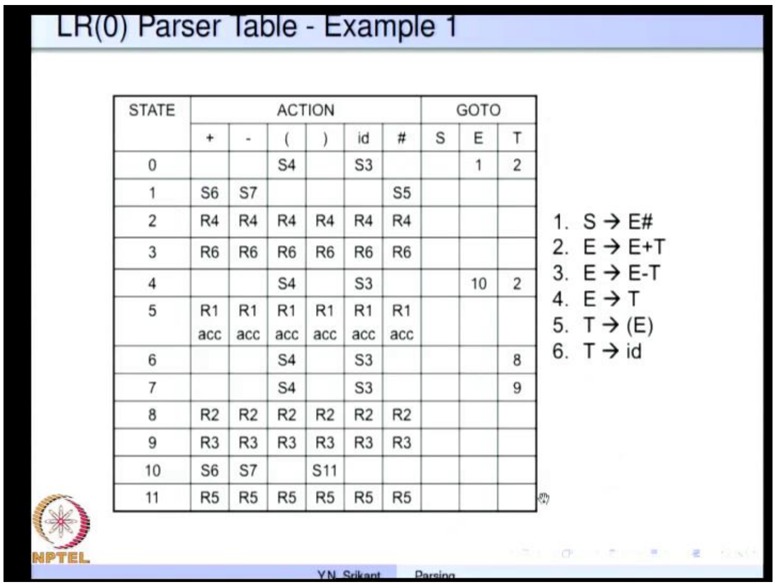


Shift and Reduce Actions

- If a state contains an item of the form $[A \rightarrow \alpha.]$ ("reduce item"), then a reduction by the production $A \rightarrow \alpha$ is the action in that state
- If there are no "reduce items" in a state, then shift is the appropriate action
- There could be shift-reduce conflicts or reduce-reduce conflicts in a state
  - Both shift and reduce items are present in the same state (S-R conflict), or
  - More than one reduce item is present in a state (R-R conflict)
  - It is normal to have more than one shift item in a state (no shift-shift conflicts are possible)
- If there are no S-R or R-R conflicts in any state of an LR(0) DFA, then the grammar is LR(0), otherwise, it is not LR(0)

So, here is the table. So, before going to this table I must explain how exactly the table is filled. So, shift and reduce actions in the state are filled as now we discussed if a state contains an item of the form a going to alpha dot that is called a reduce item. Then a reduction by a to alpha is the action in that state if there are no reduce items in a state. Then it is a shift state and shift is the appropriate action there could be shift reduce conflicts or reduce reduce conflicts. So, when does that happen there is a shift item and there is also a reduce item in the same state or there is a more than one reduce item in the same state. So, it is you know it is is normal to have more one shift item in a state. So, there is no question having a shift shift conflict at all. So, if there are no shift reduce or reduce reduce conflicts in any state of this DFA then the grammar is set to be L R 0 otherwise it is not L R 0.

(Refer Slide Time: 07:29)



LR(0) Parser Table - Example 1

STATE	ACTION						GOTO		
	+	-	(	)	id	#	S	E	T
0			S4		S3			1	2
1	S6	S7				S5			
2	R4	R4	R4	R4	R4	R4			
3	R6	R6	R6	R6	R6	R6			
4			S4		S3			10	2
5	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc			
6			S4		S3				8
7			S4		S3				9
8	R2	R2	R2	R2	R2	R2			
9	R3	R3	R3	R3	R3	R3			
10	S6	S7		S11					
11	R5	R5	R5	R5	R5	R5			

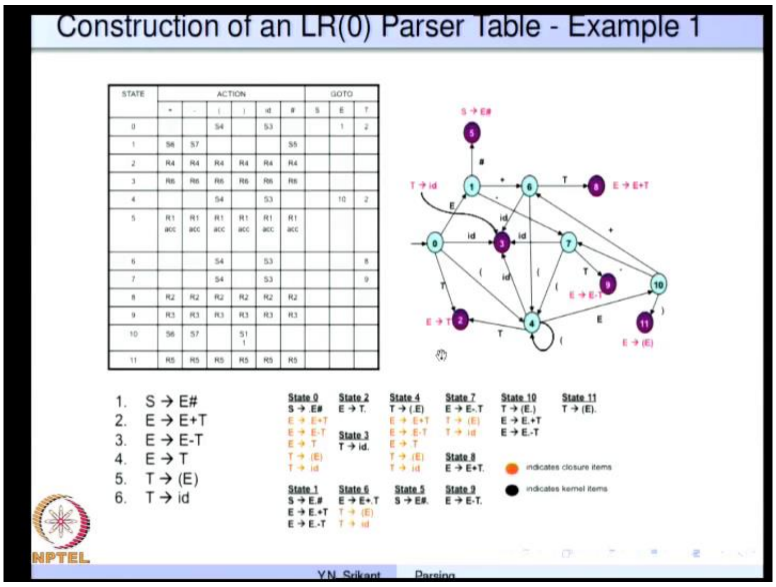
1. S → E#
2. E → E+T
3. E → E-T
4. E → T
5. T → (E)
6. T → id

NPTEL

Y.N. Srikant Parsing

So, this is the, you know table for the set of items that we have already constructed.

(Refer Slide Time: 07:38)



Construction of an LR(0) Parser Table - Example 1

STATE	ACTION						GOTO		
	+	-	(	)	id	#	S	E	T
0			S4		S3			1	2
1	S6	S7				S5			
2	R4	R4	R4	R4	R4	R4			
3	R6	R6	R6	R6	R6	R6			
4			S4		S3			10	2
5	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc	R1 acc			
6			S4		S3				8
7			S4		S3				9
8	R2	R2	R2	R2	R2	R2			
9	R3	R3	R3	R3	R3	R3			
10	S6	S7		S11					
11	R5	R5	R5	R5	R5	R5			

1. S → E#
2. E → E+T
3. E → E-T
4. E → T
5. T → (E)
6. T → id

State 0
S → .E#
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 1
S → E.#
E → E.+T
E → E.-T

State 2
E → T.

State 3
T → id.

State 4
T → (.E)
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 5
S → E#.

State 6
E → E+.T
T → .(E)
T → .id

State 7
E → E-.T
T → .(E)
T → .id

State 8
E → E+T.

State 9
E → E-T.

State 10
T → (E.)
E → E.+T
E → E.-T

State 11
T → (E).

🟠 indicates closure items

⚫ indicates kernel items

NPTEL

Y.N. Srikant Parsing

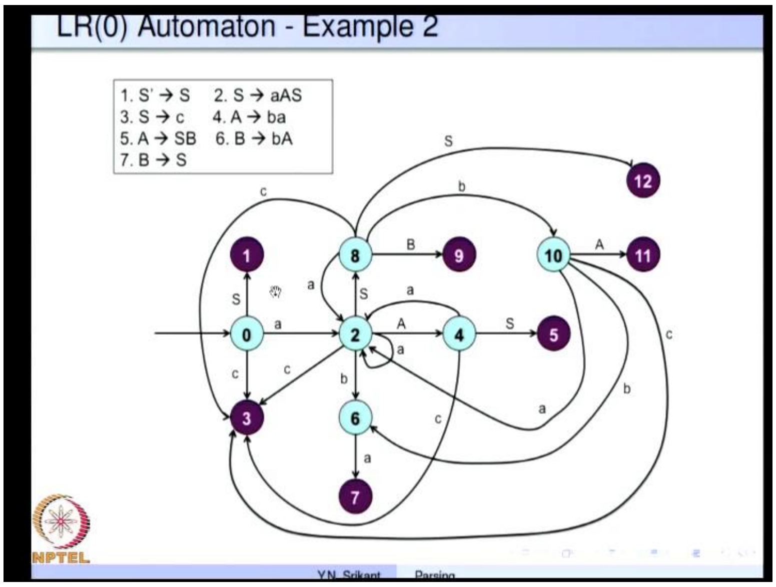
So, let me show you the procedure for filling it. So, we actually look at the items. So, we have state 0 we assume that we are not filled anything here. So, the whole table is empty. So, in the we have state 0 and let us see on which items indicate a shift on a terminals symbol because the non terminal after the dot really indicate a go to. So, only the terminal symbols after the dot indicate a shift on to the stack. So, there is a parenthesis and there is I d as well. So, on state 0 for the left parenthesis there is a shift and for the i d there is another shift. So, when we advance the dot after the left parenthesis you know. So, the state that we get into would be you know, so this state number 4. So, here it is right. So, parenthesis dot E and then the other one on i d you know 0 on i d goes to the state T going to i d dot. So, that is state number 3. So, now, the go to part can also be filled for this particular state.

So, there is a non terminal E after the dot in these items and there is a non terminal T after the dot in this particular item and other two we have already considered. So, on the non terminal E it goes to a state it contains the items E going to E plus dot T and that would be state number 1. So, this is you know on. So, E dot has and then E dot plus T and then E dot minus t. So, of course, S going to dot E hash is also an item with E on the right hand side and after the dot. So, all this 3 items actually are in state 1. So, that we have a goto of 0 on the non terminal E as state number one. So, and similarly 0 go to of 0 on non terminal T is state number 2 that is this particular thing you know.

So, E going to dot T will yields the item E going to T dot. So, this is the manner in which we are going to fill all the entries in this particular table. So, now, again if you consider the same string i d hash and with us with the help of these this table and the set of items let us just make sure that we understand how it is passed. So, we start from state 0 and on i d it says shift and go to state 3. So, that is the state number three here and we see that it is a reduce items that is the reason why this state indicates you know this state indicates a reduction by the non by the production T going to i d and then you know so; that means, we must pop the stack. So, i d is removed state 3 is removed and we state 0 again on the stack. So, the non terminal T from state takes us to state number 2. So, that is here. So, we would have push the non terminal T and the state number 2 on to the stack. So, so we have gone to state number 2 here and it has only a reduce item.

So, there is a reduction E to T which brings us back to the state 0 and state 0 on E takes us to state 1; that is this particular state and state 1 contains a you know S going to E dot hash E going to E dot plus T and E going to E dot minus T, and then we read hash we go to state number 5. So, that the this is state number 5 which is an accept state. So, in state number 5 the entire row says accepts on any symbol. So, that is a how the string is passed. So, and you can also see that there is exactly one entry in every one of these you know slots right here of course, accepts is nothing, but a comment you know. So, reduction by one and the it says it is an accept state. So, it is not that there are two entries of reduce you know nature or something like that. So, this is a grammar which definitely L R 0, because there are non conflicts in any state.

(Refer Slide Time: 12:32)



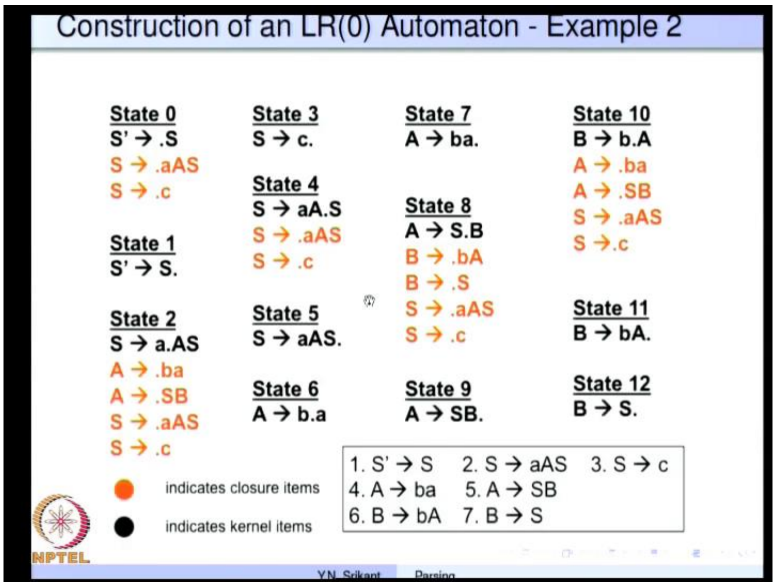
LR(0) Automaton - Example 2

1. S' → S   2. S → aAS
3. S → c   4. A → ba
5. A → SB   6. B → bA
7. B → S

Y.N. Srikant   Parsing

So, let us go further, take another example how the L R 0 automaton is constructed. So, this is the grammar this is the automaton, we will study the how to construct it as well.

(Refer Slide Time: 12:43)



Construction of an LR(0) Automaton - Example 2

State 0
S' → .S
S → .aAS
S → .c

State 1
S' → S.

State 2
S → a.AS
A → .ba
A → .SB
S → .aAS
S → .c

State 3
S → c.

State 4
S → aA.S
S → .aAS
S → .c

State 5
S → aAS.

State 6
A → b.a

State 7
A → ba.

State 8
A → S.B
B → .bA
B → .S
S → .aAS
S → .c

State 9
A → SB.

State 10
B → b.A
A → .ba
A → .SB
S → .aAS
S → .c

State 11
B → bA.

State 12
B → S.

● indicates closure items
● indicates kernel items

1. S' → S   2. S → aAS   3. S → c
4. A → ba   5. A → SB
6. B → bA   7. B → S

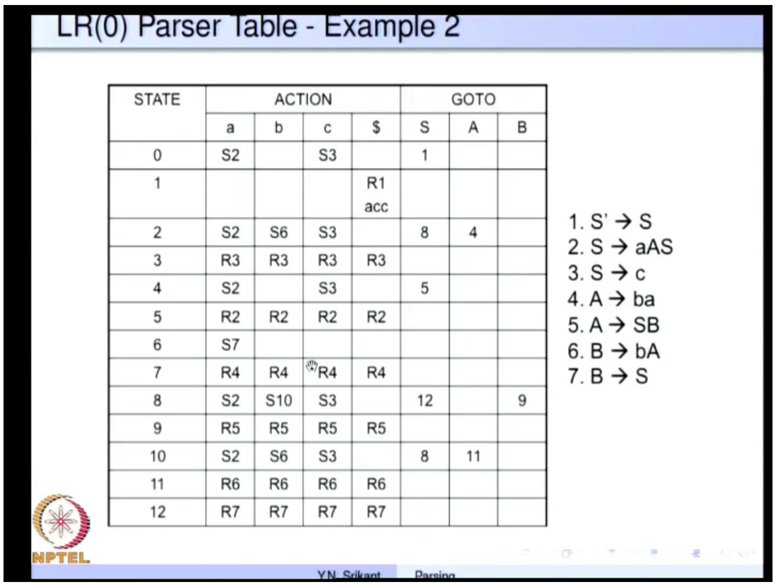
Y.N. Srikant   Parsing

So, we start from state 0. So, this is the augmented grammar S prime going to dot S s going to S going to S prime going to S prime S going to a A S and S going to c. And initial item we begin with is always S prime going to dot s. So, applying the closure we add these two item S going to dot a A S and S going to dot c there are no more items to be added. So, once state 0 is constructed we you know construct the goto state for this particular item S prime going to dot S becomes S prime going to S dot no more items can be added here that gives a state 1. So, when we advace the dot after the a here. So, we get a dot a A S. So, that gives us this item and now the closure includes all these items you know starting with a and then there is a item with S after the dot. So, the S items are also again added. So, remember there are two S items here S going to a dot a A S and S going to dot a A S.

But dots are in the different positions. So, there is nothing incorporate about this. So, now, S going to c dot gives us state 3. So, we have covered all the items in state 0 this does not gives us any more here it gives us S going a a dot S And then we add these two items by the closure operation and then this gives us a going to ash b dot a. So, that is the state and then a going to S dot p that would be this state with this closure items these closure items and S going to a dot a S is the same state number 2 and S going to c dot is already covered it is state number three. So, this gives us S going to A A S dot that is state number 5 and this items gives us S going to a dot a A S which is already state number 2.

So, no new state are added here and this of course, S going to dot c again gives us nothing more extra. So, this state again does not give us any states b dot a going to b dot a gives us a going to b a dot again this is a reduce state and nothing more can be added to this. So, this of course, gives us some more it gives us a state a going to S b dot and b going to b dot a is state number ten with extra closure items here and this and this do not give us any extra states b going to S dot is a state which state number twelve. So, this b a dot b going to b dot a gives us b going to b a dot and others do not generate any more new states. So, you can see that this particular grammar does not have any shift reduce or reduce conflicts in any states there is exactly one reduce state item in a state whenever there is 1. So, and there does not any shift item as well.

(Refer Slide Time: 16:05)



LR(0) Parser Table - Example 2

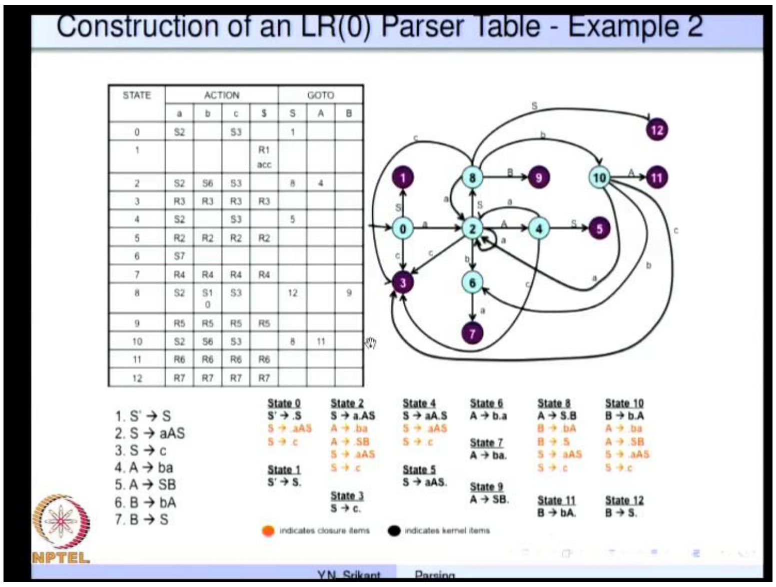
STATE	ACTION				GOTO		
	a	b	c	$	S	A	B
0	S2		S3		1		
1				R1 acc			
2	S2	S6	S3		8	4	
3	R3	R3	R3	R3			
4	S2		S3		5		
5	R2	R2	R2	R2			
6	S7						
7	R4	R4	R4	R4			
8	S2	S10	S3		12		9
9	R5	R5	R5	R5			
10	S2	S6	S3		8	11	
11	R6	R6	R6	R6			
12	R7	R7	R7	R7			

1. $S' \rightarrow S$
2. $S \rightarrow aAS$
3. $S \rightarrow c$
4. $A \rightarrow ba$
5. $A \rightarrow SB$
6. $B \rightarrow bA$
7. $B \rightarrow S$

Y.N. Srikant Parsing

So, this is a grammar which is L R 0 here is the you know a table for it.

(Refer Slide Time: 16:11)



Construction of an LR(0) Parser Table - Example 2

STATE	ACTION				GOTO		
	a	b	c	$	S	A	B
0	S2		S3		1		
1				R1 acc			
2	S2	S6	S3		8	4	
3	R3	R3	R3	R3			
4	S2		S3		5		
5	R2	R2	R2	R2			
6	S7						
7	R4	R4	R4	R4			
8	S2	S1 0	S3		12		9
9	R5	R5	R5	R5			
10	S2	S6	S3		8	11	
11	R6	R6	R6	R6			
12	R7	R7	R7	R7			

1. $S' \rightarrow S$
2. $S \rightarrow aAS$
3. $S \rightarrow c$
4. $A \rightarrow ba$
5. $A \rightarrow SB$
6. $B \rightarrow bA$
7. $B \rightarrow S$

State 0
$S' \rightarrow .S$
$S \rightarrow .aAS$
$S \rightarrow .c$

State 1
$S' \rightarrow S.$

State 2
$S \rightarrow a.AS$
$A \rightarrow .ba$
$A \rightarrow .SB$
$S \rightarrow .aAS$
$S \rightarrow .c$

State 3
$S \rightarrow c.$

State 4
$S \rightarrow aA.S$
$S \rightarrow .aAS$
$S \rightarrow .c$

State 5
$S \rightarrow aAS.$

State 6
$A \rightarrow b.a$

State 7
$A \rightarrow ba.$

State 9
$A \rightarrow SB.$

State 8
$A \rightarrow S.B$
$B \rightarrow .bA$
$B \rightarrow .S$
$S \rightarrow .aAS$
$S \rightarrow .c$

State 11
$B \rightarrow bA.$

State 10
$B \rightarrow b.A$
$A \rightarrow .ba$
$A \rightarrow .SB$
$S \rightarrow .aAS$
$S \rightarrow .c$

State 12
$B \rightarrow S.$

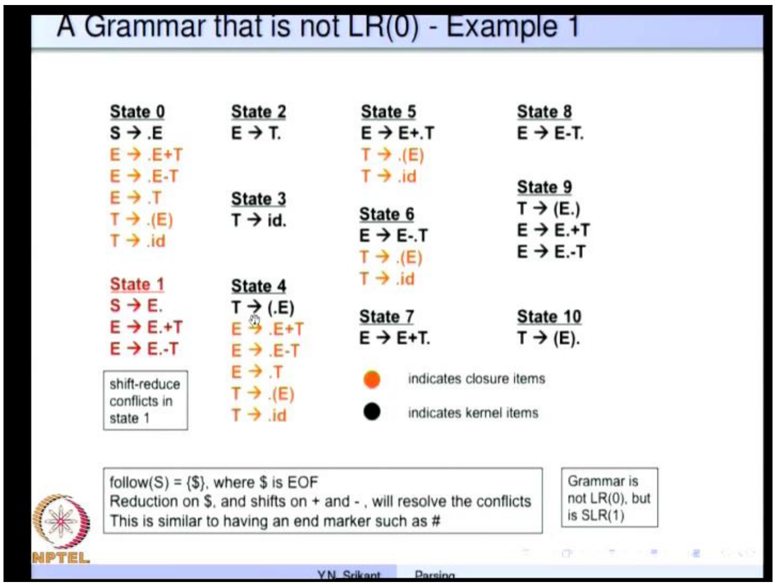
● indicates closure items ● indicates kernel items

Y.N. Srikant Parsing

And here is the tables; the sets of items; the grammar and the DFA, all put together. So, let us take one state and understand again how to fill the table. So, state number 2. So, from here state number 2 on b. So, we go to state number 6 right. So, b dot a. So, state number 6 is here b dot a. So, state number 2 on says shift 6 right and then there is a c here. So, on c it should go to state number 3. So, 2 on c says go to state number three and on a of course, this remains you know dot a A S this is a going to dot a A s. So, this generates a state which is state number itself.

So, it remains in state number 2 and operation is shift. So, you can see the self directed arc here right regarding goto part of the table. So, 2 and S you know. So, 2 on S goes to state number 8. So, the reason being there is a symbol S after the dot a going to dot S B. So, this goes to state containing a going to S dot b. So, a going to S dot b is here right state number eight and on the non terminal eight goes to state number 4. So, that is because we have S going to a dot a A S. So, this gives us state number 4 which contains a a dot s. So, this is the way in which we fill up the entire table and since there is exactly one entry in each of the slots this grammar is indeed L R 0.

(Refer Slide Time: 18:15)



A Grammar that is not LR(0) - Example 1

State 0
S → .E
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 1
S → E.
E → E.+T
E → E.-T

shift-reduce conflicts in state 1

State 2
E → T.

State 3
T → id.

State 4
T → (.E)
E → .E+T
E → .E-T
E → .T
T → .(E)
T → .id

State 5
E → E+.T
T → .(E)
T → .id

State 6
E → E-.T
T → .(E)
T → .id

State 7
E → E+T.

State 8
E → E-T.

State 9
T → (E.)
E → E.+T
E → E.-T

State 10
T → (E).

● indicates closure items
● indicates kernel items

follow(S) = {$}, where $ is EOF
Reduction on $, and shifts on + and - , will resolve the conflicts
This is similar to having an end marker such as #

Grammar is not LR(0), but is SLR(1)

Y.N. Srikant Parsing

So, we so far we saw examples where the grammar is L R 0. So, it is time to move on and find out you know whether there are grammar which are not L R 0 at all they violet the L R 0 property indeed. So, we take the old grammar with plus and minus, but remember the old grammar had S going to E hash we really removed the hash. So, now, it is time to explain why we remove and what happen if we remove it. So, we lets construct the set of items for this I am not going to explain it in greater detail again. But we have this first item and then the closure then the items S going to dot E S going to dot E plus T and S E E going to dot E minus T. They give raise to state number 1, which contains d going to E dot E going E dot plus T and E going to E dot minus T.

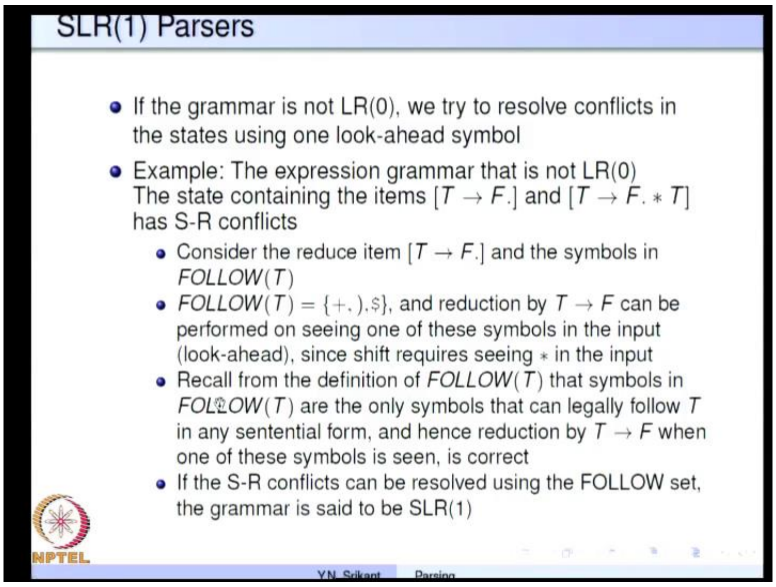
So, it is easy to see that there is one reduce item and there are two shift items right. So, this is the reduce item and these two are the shift items. So, there is a shift reduce conflict in this particular state. So, once we reach this particular state on a particular input symbol. You know we are not able to say whether the state should reduce by the production S going to E or shift the next item on to the stack. Remember the table when we actually operate using the table the parser does not see what exactly is the item and then decide the action you know it it is there is exactly one action in each of the slots. So, on 2 on a becomes S 2 etcetera. So, whereas, for the reduce there we always had exactly one item R 4 and this was S 7. So, in the entire row we always set it is a reduce by 4 see for state number 5 it is is always a reduce by production by number 2.

And for the state number 3 it is always a reduce by production number 3 and so on. So, in other words for the shift item, you know we see what exactly is a symbol and push it on to the stack, but for the reduce item we should be able to say that it is a reduction by a particular production without even bothering about the next input symbol. So, this is drawback of the L R 0 parsing strategy because we do not consider the you know what exactly is a next input symbol in order to say whether it is a reduction or it is a shift. So, in this case the grammar fails the L R test, but this is an reduction to the next class of grammars called S L R 1 or simple L R 1 grammars. So, it happens the next top is S L R 1. So, let me mention what happens there.

So, we actually compute what is known as the follow set of the non terminal S whenever there is a reduction we compute the the follow of the left hand side non terminal which happens to be a dollar. So, if we want to do a reduction then we do it only on dollar and on plus and minus we do a shift this actually resolves the conflict and makes the grammar go ahead with the same parsing strategy. So, doing this is similarly having an end marker such as hash that we had here. So, if we had a hash here S going E hash then this state wouldn’t have been a reduce state it still have been a shift state. So, the reduce state would have been the next one you know S going to E hash dot. So, the assumption here is the input no matter which syntax error you know occurs in the input there will always be 1 hash symbol coming in.

So, if the has symbol itself making missing then the parsers stop with an error somewhere in the middle, but if the hash symbol definitely exist then it can go on you know. So, it will definitely read that and if there is an error it provides an error otherwise it goes on. So, the hash symbol makes this grammar L R 0 and removing the hash symbol which is a marker in the input makes this non L R 0. But fortunately there is another way of breaking this tie and that is the using the L R S L R 1 strategy.

(Refer Slide Time: 23:33)



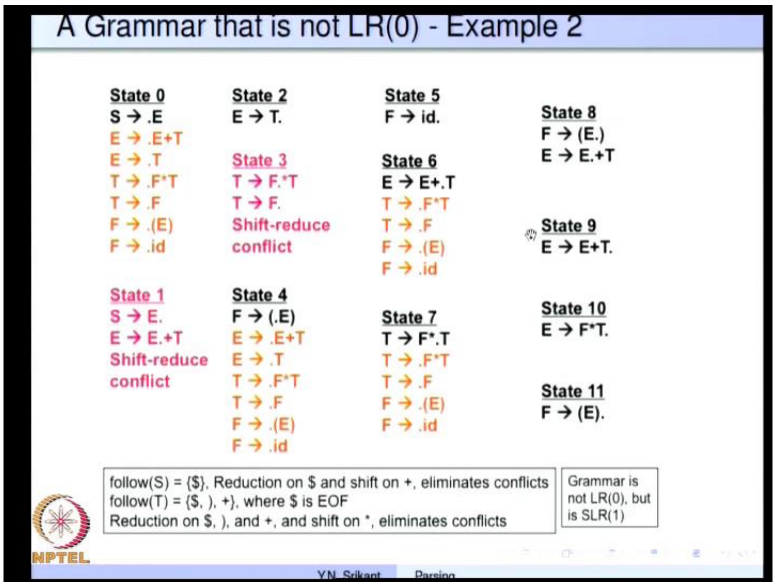
SLR(1) Parsers

- If the grammar is not LR(0), we try to resolve conflicts in the states using one look-ahead symbol
- Example: The expression grammar that is not LR(0) The state containing the items $[T \to F.]$ and $[T \to F. * T]$ has S-R conflicts
  - Consider the reduce item $[T \to F.]$ and the symbols in $FOLLOW(T)$
  - $FOLLOW(T) = \{+,), \$\}$, and reduction by $T \to F$ can be performed on seeing one of these symbols in the input (look-ahead), since shift requires seeing $*$ in the input
  - Recall from the definition of $FOLLOW(T)$ that symbols in $FOLLOW(T)$ are the only symbols that can legally follow T in any sentential form, and hence reduction by $T \to F$ when one of these symbols is seen, is correct
  - If the S-R conflicts can be resolved using the FOLLOW set, the grammar is said to be SLR(1)

Y.N. Srikant Parsing

If the grammar is not L R 0, we try to resolve conflicts in the state using one look a head symbol L R 0. We never used any look a head to resolve the conflict here we try to use the look a head symbol. So, how do we do that we saw that the you know expression grammar of course, even the simplified expression grammar is not L R 0.

(Refer Slide Time: 24:04)



A Grammar that is not LR(0) - Example 2

State 0
S → .E
E → .E+T
E → .T
T → .F*T
T → .F
F → .(E)
F → .id

State 1
S → E.
E → E.+T
Shift-reduce conflict

State 2
E → T.

State 3
T → F.*T
T → F.
Shift-reduce conflict

State 4
F → (.E)
E → .E+T
E → .T
T → .F*T
T → .F
F → .(E)
F → .id

State 5
F → id.

State 6
E → E+.T
T → .F*T
T → .F
F → .(E)
F → .id

State 7
T → F*.T
T → .F*T
T → .F
F → .(E)
F → .id

State 8
F → (E.)
E → E.+T

State 9
E → E+T.

State 10
E → F*T.

State 11
F → (E).

follow(S) = {\$}, Reduction on \$ and shift on +, eliminates conflicts
follow(T) = {\$,), +}, where \$ is EOF
Reduction on \$,), and +, and shift on *, eliminates conflicts

Grammar is not LR(0), but is SLR(1)

Y.N. Srikant Parsing

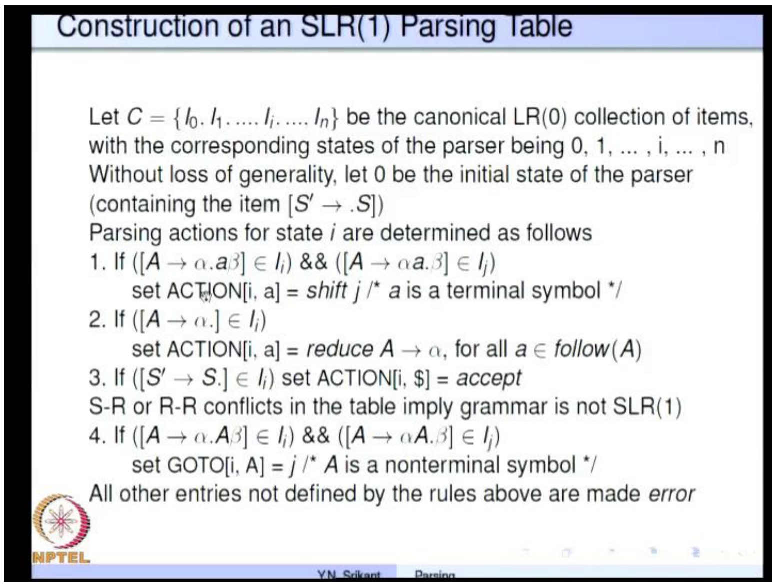
So, let me show you the grammar here. So, this is S going to dot E etcetera, etcetera. So, this is full expression grammar S going to T E E going to E plus T E going T T going to F star T T going to F T going to F going to paraenesis E paraenesis and F going to i d. So, to begin with we have a S going to dot E and then we had the closure items, we get state 1 by advancing the dot for this particular item and for this items as well. So, get these two items in the state number one and there are no more items to be added by closure. Because there is nothing after the dot here and there is only terminal symbol after the dot here. So, it is easy to see that there is a you know shift reduce conflict here right. So, similarly you know this thing going further we get this state T going to F dot star T and we also have which has the item T going to f dot as well. So, again there is a shift reduce conflict this is a reduce item this is a shift item.

So, in both these states there are shift reduce conflicts and how to resolve it you know that is the discussion we right now are going to have the expression grammar is not L R 0. That is what we saw just now there are the state contains the items T going to f dot and T going F dot star T. This has shift reduce conflicts not only that this state S going to E dot and E going to E dot plus T also has shift reduce conflicts. So, what we do now is consider the reduce item T going to f dot. So, this has the symbol T the non terminal T on the left hand side of the production T going to F. So, we consider the symbols in the follow set of T. So, recall that follow. So, recall that follow set of T says these are the only symbols which can legally follow T in any sentential form.

So, it is fair to consider the symbols which follow this particular non terminal and say for those symbols. So, in this case plus right paraenesis and dollar these are the symbols which can be in the follow of T. So, we say reduction by T to F can be performed on seeing on of these symbols otherwise the sentential form is you know in correct. So, we cannot make a reduction by T T to F. So, seeing one of the symbols in the input and you know since shift requires seeing star in the input that is here. So, there is no conflict between the reduction and the shift shift says c star in the input. And our new strategy says reduction by T to F provided the next input symbol is either plus or right paraenesis or dollar right.

So, these are the only symbols with the follow T symbols are the only once which can legally follow t's and any sentential form. So, the reduction by T to F in one of the symbol is seen is; obviously, correct. So, when shift you know reduce conflicts are resolved using the follow set the grammar uses a S L R 1 strategy and if we can indeed resolve all the conflicts in this manner then the grammar is set to be S L R 1. So, we saw how to you now resolve conflict in this case right. So, in this case also we can do that. So, we consider the follow of d follow of S is just dollar. So, reduction on dollar and shift on plus again the conflicts can be very easily eliminated. So, this grammar happens to be S L R 1 even though it is not L R 0.

(Refer Slide Time: 28:07)



Construction of an SLR(1) Parsing Table

Let $C = \{I_0, I_1, \ldots, I_i, \ldots, I_n\}$ be the canonical LR(0) collection of items, with the corresponding states of the parser being 0, 1, ... , i, ... , n
Without loss of generality, let 0 be the initial state of the parser (containing the item $[S' \to .S]$)
Parsing actions for state *i* are determined as follows

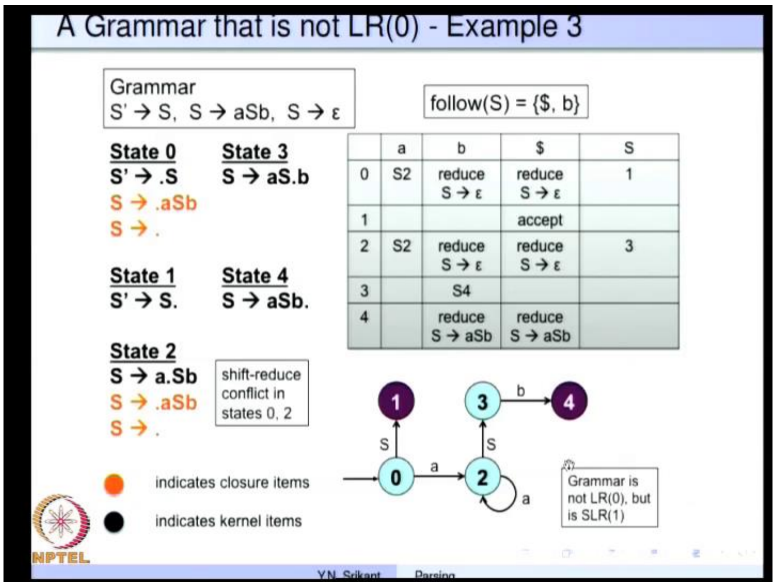
```
1. If ($[A \to \alpha.a\beta] \in I_i$) && ($[A \to \alpha a.\beta] \in I_j$)
      set ACTION[i, a] = shift j /* a is a terminal symbol */
2. If ($[A \to \alpha.] \in I_i$)
      set ACTION[i, a] = reduce $A \to \alpha$, for all $a \in follow(A)$
3. If ($[S' \to S.] \in I_i$) set ACTION[i, $] = accept
S-R or R-R conflicts in the table imply grammar is not SLR(1)
4. If ($[A \to \alpha.A\beta] \in I_i$) && ($[A \to \alpha A.\beta] \in I_j$)
      set GOTO[i, A] = j /* A is a nonterminal symbol */
```

All other entries not defined by the rules above are made *error*

Y.N. Srikant Parsing

So, now, how do we construct the S L R 1 parsing table, it is fairly straight forward. So, we again build the we build the l S L R 0 automaton as usual and once it is built parsing actions for straight I or determined using follow symbol. So, one; if there is a shift item alpha A going to alpha dot a beta and alpha dot a dot beta is in some other state I j. So, this is state I this is state j right. So, we add the action I comma a equal to shift j to the table when. So, we are looking at the next symbol and then making it a shift whereas, in the case of a reduction we set action I comma a equal to reduce by a to alpha for all the symbols in the follow of a that is it. So, for the item S prime going to S dot we set it as accept on dollar and any shit reduce or reduce conflicts in the table. After filling all this implies that the grammar is not S L R 1 the goto part is as usual. So, we have a going to alpha dot a beta in the state I and a going to alpha a dot beta in the state j. So, we set go to I comma a equal to j. So, this is and any other you know entry is made as error.

(Refer Slide Time: 29:44)



A Grammar that is not LR(0) - Example 3

Grammar
S' → S, S → aSb, S → ε

follow(S) = {$, b}

State 0
S' → .S
S → .aSb
S → .

State 1
S' → S.

State 2
S → a.Sb
S → .aSb
S → .

State 3
S → aS.b

State 4
S → aSb.

shift-reduce conflict in states 0, 2

	a	b	$	S
0	S2	reduce S → ε	reduce S → ε	1
1			accept	
2	S2	reduce S → ε	reduce S → ε	3
3		S4		
4		reduce S → aSb	reduce S → aSb	

$$\begin{tikzcd}
1 & 3 \arrow[r, "b"] & 4 \\
\arrow[r] & 0 \arrow[ul, "S"] \arrow[r, "a"] & 2 \arrow[u, "S"] \arrow[loop below, "a"]
\end{tikzcd}$$

● indicates closure items

● indicates kernel items

Grammar is not LR(0), but is SLR(1)

Y.N. Srikant Parsing

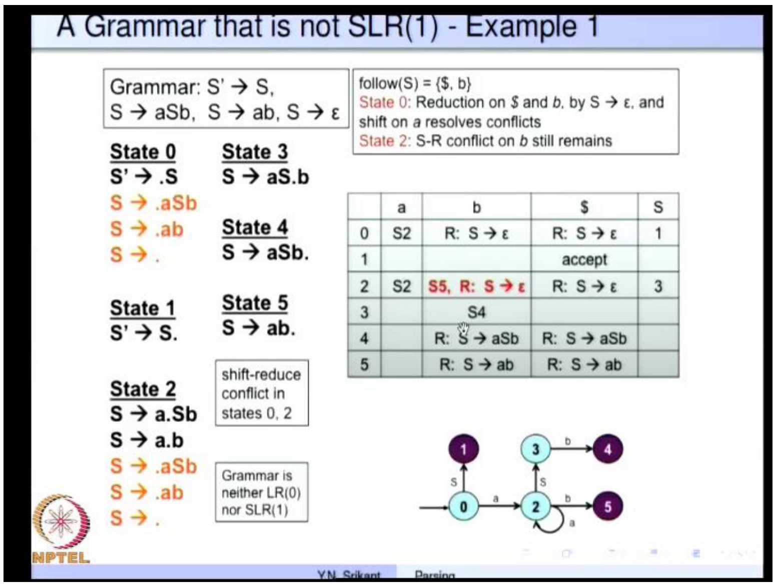
So, here is an example of a grammar that is not L R 0 and the grammar is definitely S L R 1. So, again the grammar is S prime going to S, S going to a S b and S going to epsilon. So, I have chosen this example because of the production S going to epsilon. So, state 0 contains the first item S prime going to dot S and then S prime going to dot a A S b and S prime going to dot are added to it. So, in some text books instead of adding the items S going to dot it is indicated as S going to epsilon dot which is perfectly, but be careful you one should not you now generate the next item S going epsilon dot again.

Now, S going to dot is the only item that can be generated which of course, can be written as a S going to dot epsilon, but we should not regard epsilon as another symbol and generate one more state S going to epsilon dot that would be an error. Then we generate these states S prime going to S dot S prime going to dot a S b you know a dot S b dot a S b and dot S going to a S dot b and S going to a S b dot. So, if you look at the state number 0 this epsilon production and epsilon item S going to dot it colitis with these two you know. So, this is a reduction where as these two are shifts. So, there is a shift reduce conflict in state number 0 and again in state number 2 as well this colitis with these two. So, here we have a reduce item and these two are shift.

So, now suppose you know in both the cases the concern non terminal is S on the left hand side. So, let us consider the follow of the non terminal S that happens to be dollar and b that is very easy to see because after this you know after this S only be a dollar and here after the S there is a b. So, dollar and b are in the follow of s. So, if we say that on dollar and b it is a reduction and on a it is a shift it resolves the conflict in this state. And similarly in this case again dollar and b indicate a reduce and S indicates a shift that again you know resolves the conflict in this state as well. So, one can fill up the S L R 1 table using these states now. So, on 0 on a it goes to state number 2 and it is a shift on b it the symbol is in the follow. Therefore, it is a reduce item and again this is also a reduce action.

So, for state 1 it is always accept on dollar and for state 2 for a we have a shift and for the other two symbols we have a reduce. So, state number three on b as only a shift and state number 4 again you know has a reduce on S to a S b and reduce on a S b. So, you compute the follow symbol of S follow symbols of S we get dollar and b and therefore, it is a reduction in both these cases. So, we can see that there is exactly 1 and 3 in each of these slots therefore, the grammar is indeed S L R 1.

(Refer Slide Time: 33:32)



A Grammar that is not SLR(1) - Example 1

Grammar: S' → S,
S → aSb, S → ab, S → ε

follow(S) = {\$, b}
State 0: Reduction on *\$* and *b*, by S → ε, and shift on *a* resolves conflicts
State 2: S-R conflict on *b* still remains

State 0
S' → .S
S → .aSb
S → .ab
S → .

State 1
S' → S.

State 2
S → a.Sb
S → a.b
S → .aSb
S → .ab
S → .

State 3
S → aS.b

State 4
S → aSb.

State 5
S → ab.

shift-reduce conflict in states 0, 2

Grammar is neither LR(0) nor SLR(1)

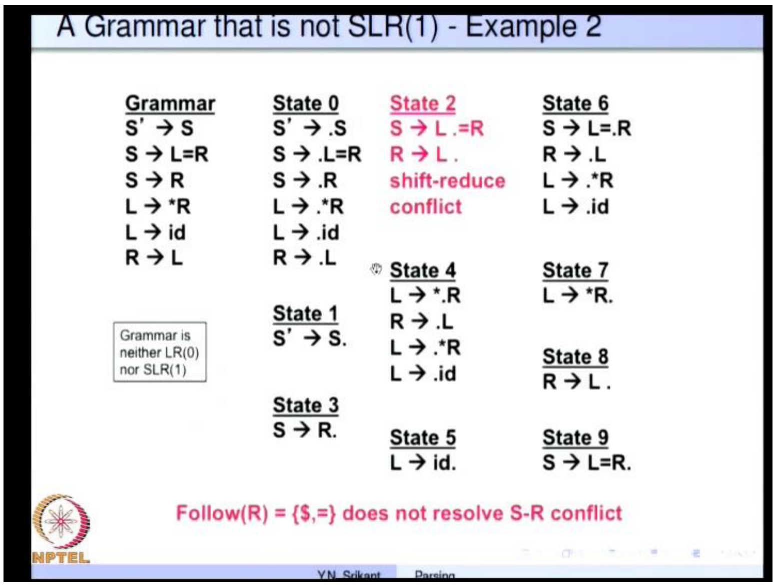
	a	b	\$	S
0	S2	R: S → ε	R: S → ε	1
1			accept	
2	S2	S5, R: S → ε	R: S → ε	3
3		S4		
4		R: S → aSb	R: S → aSb	
5		R: S → ab	R: S → ab	

$$\begin{tikzcd}
 & 1 & 3 \arrow[r, "b"] & 4 \\
 \arrow[r] & 0 \arrow[u, "S"] \arrow[r, "a"] & 2 \arrow[u, "S"] \arrow[r, "b"] \arrow[loop below, "a"] & 5
\end{tikzcd}$$

Y.N. Srikant Parsing

So, now we saw L R 0 then we showed an example of non L R 0 now we did not saw examples of S L R 1. Because it uses a look a head symbol 1 now it is time to see a grammar which is definitely not S L R 1. So, if the grammar is not S L R 1; obviously, it will not be L R 0 either. So, here is the same grammar as we saw before. So, this grammar, but we added the production S going to a b the rest the other three productions are the same by adding S going to a b we actually make this grammar ambiguous. So, all ambiguous grammars are not L R k for any k. So, it cannot be L R 0 or even L R 1. So, little one S L R 1, but it tells us a lot about the nature of non S L R 1 grammars. So, state 0 contains all this state 1 is here state 2 is here and so on and because of that dot S going to dot item. There is a shift reduce conflict in this state and shift reduce conflict in this state as well and the symbol that is involved is S. So, it follow is dollar comma b and in state number 0 a reduce on dollar on b and shift on a indeed resolves the conflicts. So, there is no problem as far as state number is 0 concerned and in state number 2 unfortunately there is already an item S going to S dot b. So, this means it is a shift on b and this says it must be a reduce on dollar on b. So, the conflict between this item and S going to dot still remains there is a shift reduce conflict even after using the you know follow symbols. So, this grammars fails to be S L R 1.

(Refer Slide Time: 35:51)



A Grammar that is not SLR(1) - Example 2

Grammar	State 0	State 2	State 6
S' → S	S' → .S	S → L .=R	S → L=.R
S → L=R	S → .L=R	R → L .	R → .L
S → R	S → .R	shift-reduce	L → .*R
L → *R	L → .*R	conflict	L → .id
L → id	L → .id		
R → L	R → .L		

State 1
S' → S.

State 3
S → R.

State 4
L → *.R
R → .L
L → .*R
L → .id

State 5
L → id.

State 7
L → *R.

State 8
R → L .

State 9
S → L=R.

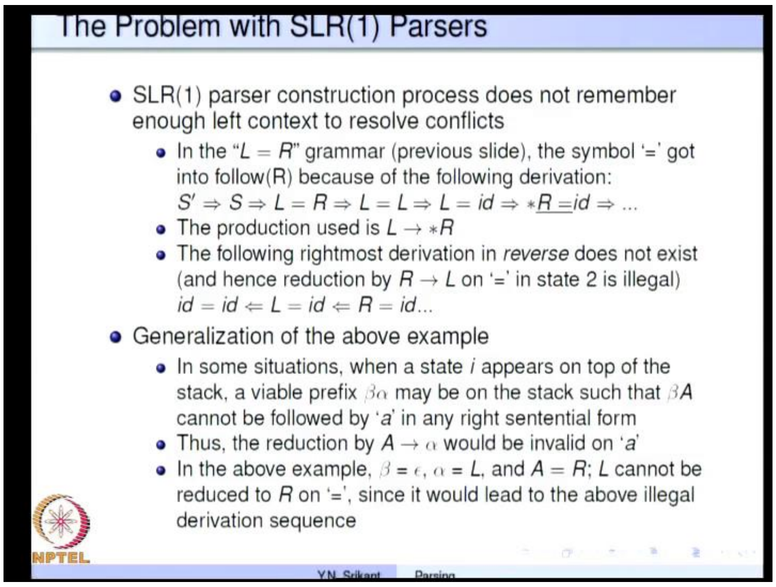
Grammar is neither LR(0) nor SLR(1)

Follow(R) = {$,=} does not resolve S-R conflict

Y.N. Srikant Parsing

So, let us take another example which is not S L R 1, this example is from the book by ohio lemmes at annulment the grammar is here. And it is a very concocted grammar and we constructs the states state 0 1 2 3 etcetera this is fairly straight forward, what is of importance to us is state number 2. So, state number 2 contains the item S going to l dot equal to R and it also contains the item R going to l dot. So, now, the symbol R is on the left hand side of the reduce item. So, if you compute the follow of R it gives us dollar and equal to. So, again since dollar dollar is, but equal to is also here right. So, this is a shift item which is shifted the symbol is shifted on the input symbol being equal to and this is a reduce item and our S L R 1 do’s a it must be reduction on both dollar and equal to. So, there is a shift reduce conflict even if you try to apply the S L R 1 strategy. So, therefore, this grammar is not definitely not S L R 1 it was not L R 0 now it is not S L R 1 either

(Refer Slide Time: 37:24)



The Problem with SLR(1) Parsers

- SLR(1) parser construction process does not remember enough left context to resolve conflicts
  - In the "$L = R$" grammar (previous slide), the symbol '=' got into follow(R) because of the following derivation: $S' \Rightarrow S \Rightarrow L = R \Rightarrow L = L \Rightarrow L = id \Rightarrow *\underline{R =}id \Rightarrow \dots$
  - The production used is $L \rightarrow *R$
  - The following rightmost derivation in *reverse* does not exist (and hence reduction by $R \rightarrow L$ on '=' in state 2 is illegal) $id = id \Leftarrow L = id \Leftarrow R = id \dots$
- Generalization of the above example
  - In some situations, when a state i appears on top of the stack, a viable prefix $\beta\alpha$ may be on the stack such that βA cannot be followed by 'a' in any right sentential form
  - Thus, the reduction by $A \rightarrow \alpha$ would be invalid on 'a'
  - In the above example, $\beta = \epsilon$, $\alpha = L$, and $A = R$; L cannot be reduced to R on '=', since it would lead to the above illegal derivation sequence

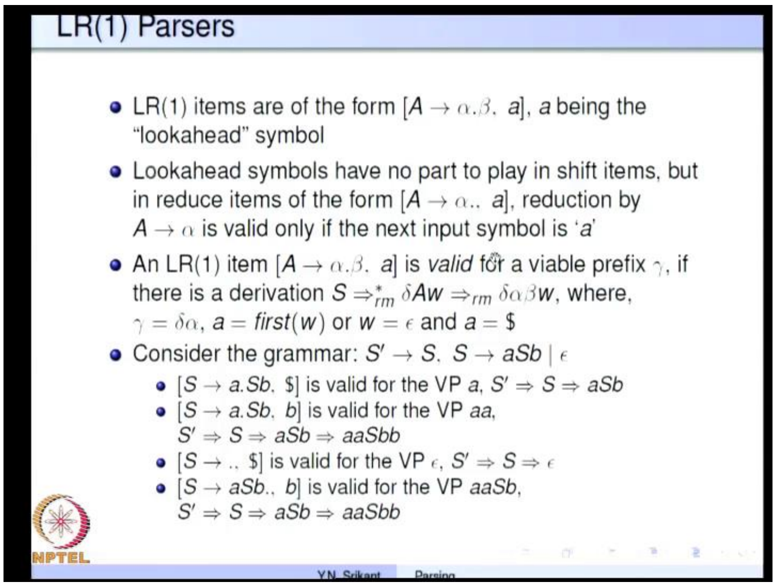
Y.N. Srikant Parsing

So, let us understand why this particular failure occurred in the last example or the one before the problem is the S L R 1 parsers considers only you know considered every symbol in the follow. The construction process does not remember enough left context to resolve the conflicts. So, let us take the l equal to R grammar that is this particular grammar this let us see how the symbol equal to got into the follow set. So, we start with S prime derive S then derive l equal to R. Now, this derives l equal to l that derives l equal to i d this is the rightmost derivation. Therefore, this must be derive first and then l to star R is apply and we get star R equal to i d. So, now, equal to you know is in the follow set of R that is easy to see, because this is sentential form the problem is the. You know it is not possible to reduce the R alone to l if at all we reduce it must be star R reduce to l and it is not possible to have any reduction of R to l.

Because you know we really do not have a production of that kind the only production we have is star R being reduce to l l going to star R is the only production whereas in this case it says you know let l be reduce to R right. So, the reduction of l to R or R to l can never happen in any sentential form. So, this is the basic difficulty the following rightmost derivation reverse does not exist at all there is a reduction of l to R right. So, so that is we have let us say R equal to i d which gives us l equal to i d that gives us i d equal to I d. So, if we had this then we would have reduce l 2 r. So, similarly reduction of R to l on its own cannot happen. So, on equal to the moral of the story is on equal to the reduction should have been star R to l. And it is never correct to reduce you know l to R just because you know the here is the reduction from l to R, but it was not because of the equal to symbol, but because of a different reason. So, we will never get a a sentential form and a derivation in which we can reduce l to R seeing on equal to that would be in correct. So, this is the reason why the S L R 1 strategy failed in the previous example.

So, if we try to generalize the above whatever I said here the point is in some situations when a state I appears on the top of the stack. A viable prefix beta alpha may be on the stack such that it is not possible for beta a to be followed by a in any right sentential form. So, in other words we if we actually have you know a very limited follow then the situation can be rectified. So, here we have a general follow and equal to gets into the follow set right. So, it is get into the follow set of r, but otherwise if we have a limited follow then we would actually store enough of the left context to make sure that such you know things do not happen. So, this is a basic idea.

(Refer Slide Time: 41:43)



LR(1) Parsers

- LR(1) items are of the form $[A \rightarrow \alpha.\beta,\ a]$, a being the "lookahead" symbol
- Lookahead symbols have no part to play in shift items, but in reduce items of the form $[A \rightarrow \alpha.,\ a]$, reduction by $A \rightarrow \alpha$ is valid only if the next input symbol is 'a'
- An LR(1) item $[A \rightarrow \alpha.\beta,\ a]$ is *valid* for a viable prefix γ, if there is a derivation $S \Rightarrow^*_{rm} \delta A w \Rightarrow_{rm} \delta\alpha\beta w$, where, $\gamma = \delta\alpha$, $a = first(w)$ or $w = \epsilon$ and $a = \$$
- Consider the grammar: $S' \rightarrow S,\ S \rightarrow aSb \mid \epsilon$
    - $[S \rightarrow a.Sb,\ \$]$ is valid for the VP a, $S' \Rightarrow S \Rightarrow aSb$
    - $[S \rightarrow a.Sb,\ b]$ is valid for the VP aa, $S' \Rightarrow S \Rightarrow aSb \Rightarrow aaSbb$
    - $[S \rightarrow .,\ \$]$ is valid for the VP ϵ, $S' \Rightarrow S \Rightarrow \epsilon$
    - $[S \rightarrow aSb.,\ b]$ is valid for the VP $aaSb$, $S' \Rightarrow S \Rightarrow aSb \Rightarrow aaSbb$

Y.N. Srikant Parsing

Let us see how to implement it in the next class of grammars called L R 1 grammars and L R 1 parsers. So, what we really try to do here is we form items which are the form a going to alpha dot beta comma a and a is of course, called as the look ahead symbol. So, the, this is nothing greater on its own, but the way we actually derive states using such items is very different in this case. So, look ahead symbols have no rule to play ion shift items, but the reduce items of the form a going to alpha dot comma a a reduction is valid only if the input symbol is a. So, this you know input symbol actually is some kind of limited follow that we mentioned in the last slide. So, then lets also finish of this definition of validity an L R 1 item a going to alpha dot beta comma a is valid for a viable prefix gamma. If there is a derivation S deriving delta a W by rightmost derivations then we apply the production a going to alpha beta.

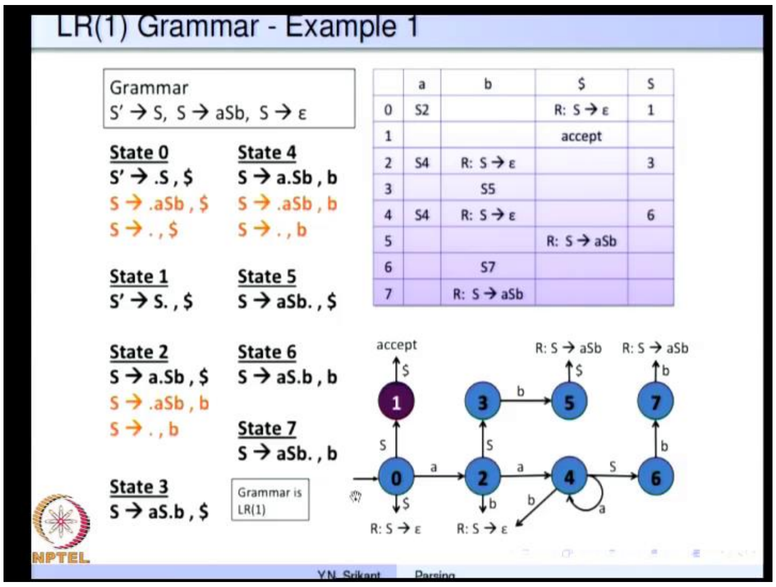
So, in this case we consider this entire derivation and a to alpha beta has been applied now if the first set of w that is if first symbol of w is actually a, then this particular item is valid for the sentential form gamma equal to with the viable prefix gamma equal to delta alpha if w is empty that is a null then a must be dollar. So, in both these cases this item is valid for the viable prefix gamma which is equal to delta alpha. So, let us take an example S prime going to S and S going to a S b or epsilon. So, consider the derivation S prime going to S and S giving a S b. So, and now you know the production applied at this point is S going to a S b. So, the handle ends at this point. So, this is dollar right and of the terminal string. So, a a s and a S b these are all the viable prefixes. So, the item S going to a dot S b comma dollar is valid for the viable prefix a.

So, a is a viable prefix that I already mentioned now the production which has been applied is a S b. So, if you a S going to a S b. So, here the production applied was S going to alpha beta. So, the w part you can see is empty. So, a is must be dollar which is indeed the case for the second one we consider the item S going to a dot S b comma b and the derivation is S prime giving S s giving a S b and that giving us a a S b b. So, the production which has been applied at this point is the S going to a S b in the middle. So, the w part is this w b right and the viable prefixes are a a a A A S and a a S b. So, we look at the viable prefix a a and since w is once symbol b and the symbol b here also matches we know that this is valid for the viable prefix a a.

So, how about the item S going to dot with a look a head dollar. So, derive the null string S prime going to S going to epsilon. So, w is empty again. So, the symbol look a head symbol must be dollar the only possible viable prefix is epsilon. So, everything is here and the last example S going to a S b dot comma b is valid for the viable prefix a a S b. So, again the same derivation. So, we have applied the production S going to a S b here.

So, the last b is the W and that is symbol here as well. So, that is take care of viable prefixes and lets proceed further.

(Refer Slide Time: 46:26)



LR(1) Grammar - Example 1

Grammar
$S' \rightarrow S,\ S \rightarrow aSb,\ S \rightarrow \varepsilon$

State 0
$S' \rightarrow .S\ ,\ \$$
$S \rightarrow .aSb\ ,\ \$$
$S \rightarrow .\ ,\ \$$

State 1
$S' \rightarrow S.\ ,\ \$$

State 2
$S \rightarrow a.Sb\ ,\ \$$
$S \rightarrow .aSb\ ,\ b$
$S \rightarrow .\ ,\ b$

State 3
$S \rightarrow aS.b\ ,\ \$$

State 4
$S \rightarrow a.Sb\ ,\ b$
$S \rightarrow .aSb\ ,\ b$
$S \rightarrow .\ ,\ b$

State 5
$S \rightarrow aSb.\ ,\ \$$

State 6
$S \rightarrow aS.b\ ,\ b$

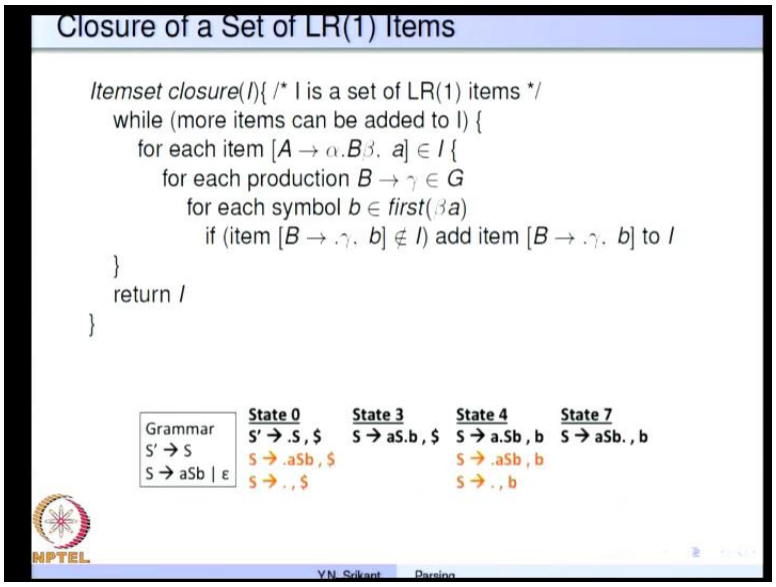
State 7
$S \rightarrow aSb.\ ,\ b$

Grammar is LR(1)

	a	b	$	S
0	S2		R: $S \rightarrow \varepsilon$	1
1			accept	
2	S4	R: $S \rightarrow \varepsilon$		3
3		S5		
4	S4	R: $S \rightarrow \varepsilon$		6
5			R: $S \rightarrow aSb$	
6		S7		
7		R: $S \rightarrow aSb$		

So, here we have an L R 1 grammar example and the set of items the automaton and the table are all shown here. So, let us understand the procedure for constructing the set of items first and then move on to filling the table and so on. But in this slide what I want to show you is that each state contains many types of items. So, here is reduce item for example, S going to dot and here we have shift items this state also has reduce item and shift item this state also has reduce item and shift item. But the manner in which we are going to fill the table is going to be slightly different. So, even though we have many reduce items and shift items in each state this grammar is still L R 1. You know if you look at this particular table you can easily see that there is a exactly one entry in each slot. So, the grammar is indeed. So, let us now understand how to construct the sets of items.

(Refer Slide Time: 47:54)



Closure of a Set of LR(1) Items

```
Itemset closure(I){ /* I is a set of LR(1) items */
    while (more items can be added to I) {
        for each item $[A \to \alpha.B\beta,\ a] \in I$ {
            for each production $B \to \gamma \in G$
                for each symbol $b \in first(\beta a)$
                    if (item $[B \to .\gamma,\ b] \notin I$) add item $[B \to .\gamma,\ b]$ to $I$
    }
    return I
}
```

Grammar
$S' \to S$
$S \to aSb \mid \varepsilon$

State 0	State 3	State 4	State 7
$S' \to .S, \$$	$S \to aS.b, \$$	$S \to a.Sb, b$	$S \to aSb., b$
$S \to .aSb, \$$		$S \to .aSb, b$	
$S \to ., \$$		$S \to ., b$	

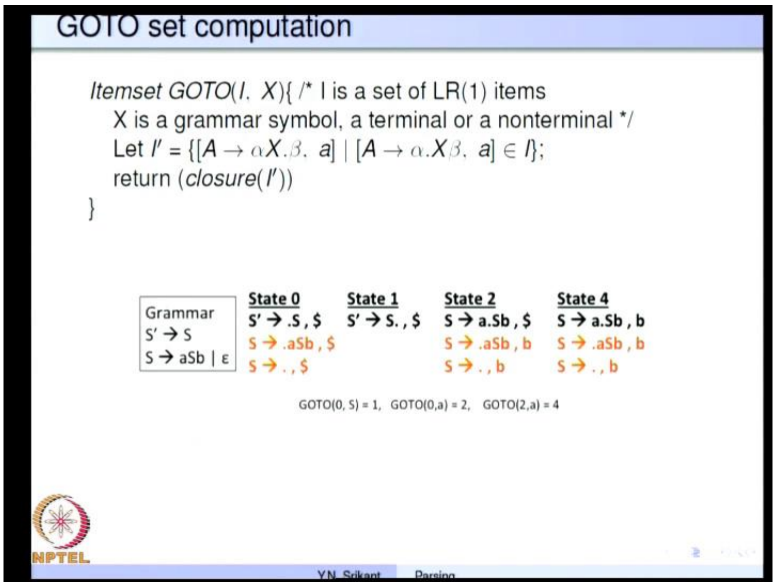
Y.N. Srikant Parsing

The procedure is similar, what we have is a closure procedure here also. So, we always begin, let us say in the state we have the item S prime going to dot S comma dollar. So, the first symbol after the dot is a non terminal. So, now, we must add you know for the closure operation we must add items with starting from this non terminal S there are two of them S going to dot a S b and S going to dot now the closure operation is very simple. So, what we consider is the symbols after the you know non terminal S and concatenated with the look ahead symbol here. So, after the non terminal S there is no symbol that is null string, but there is a dollar here. So, the concatenation of these two gives us dollar and we take the first of this string.

And those symbols are actually listed as the look ahead symbols for the items that follow in the closure. So, here for example, the procedures is for each item a going to alpha dot b beta comma a in the item set I. And we have a production b to gamma then consider each symbol in the first of beta a. So, here is the rest of the production after b and concatenated with the symbol a. So, that is the first of beta a the reason we mention first of beta a and not a particular set of symbols is beta could consist of non terminals. So, they it could begin with non terminals in which there I will be more than one item in the first set and it is possible that beta is null. So, this a which is definitely non null takes care producing at least one item for the first set. So, we form the items of the form B going to dot gamma comma b. So, that is done for each of these you know eliminates in the first of beta a.

And add the item B going to dot a gamma dot gamma comma b to the set of items b. So, that is how we add these two now this state you know does not grow because of the glow here whereas, here we have a S going to a dot S b comma b now after the S we have b and the look ahead is b. So, first of you know the beta a S string is b b first of that would be little symbol b little b. So, we have little b as the look a head in these two and the first part a S s going to dot a S b and S going to dot. So, this is how the closure operation is applied in the case of L R 1 items. So, basically we are trying to send the restricted follow as part of the items itself in the form of a look a head.

(Refer Slide Time: 51:15)



GOTO set computation

```
Itemset GOTO(I, X){ /* I is a set of LR(1) items
    X is a grammar symbol, a terminal or a nonterminal */
    Let $I' = \{[A \to \alpha X.\beta,\ a] \mid [A \to \alpha.X\beta,\ a] \in I\}$;
    return (closure(I'))
}
```

Grammar	State 0	State 1	State 2	State 4
$S' \to S$	$S' \to .S, \$$	$S' \to S., \$$	$S \to a.Sb, \$$	$S \to a.Sb, b$
$S \to aSb \mid \epsilon$	$S \to .aSb, \$$		$S \to .aSb, b$	$S \to .aSb, b$
	$S \to ., \$$		$S \to ., b$	$S \to ., b$

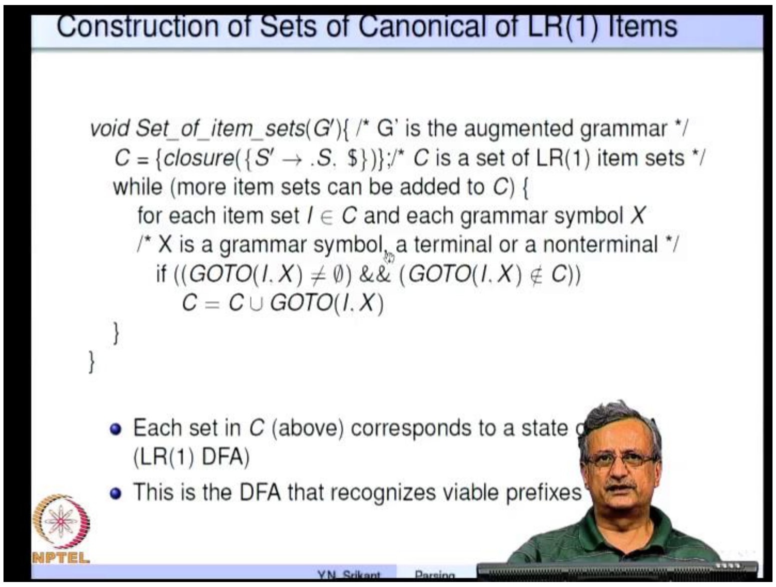
GOTO(0, S) = 1, GOTO(0,a) = 2, GOTO(2,a) = 4

Y.N. Srikant Parsing

The go to is as it was before. So, if there is a item a going to alpha dot x beta with a lookahead a in the item set I. Then we form a new set I prime which contains the items a going to alpha X dot beta comma a. So, this part does not change in the goto then we form the closure of this particular goto. So, this particular I prime is actually made to undergo a closure operation and that is gives us the goto set. So, let us see that in this example. So, state 0 we applied the closure and we got this and now to get state 1 we advance the dot by one position. So, that is gives S dot comma dollar. So, that is it here and a state number 2 is obtain by advancing the dot after the a.

So, we we have S going to a dot S b comma dollar. So, that is the item here and then we get these items because of the closure operation. So, observe that after the dot after the S we have a b and here is a a dollar. So, this is b dollar and the first of b dollar is just b. So, that is why these two symbols are b and from this item we have a S going to a dot S you know sorry this item S going a dot a S b comma b gives us S going to a dot S b comma b. So, and then we had a few more items, because of the closure operation. So, this is the this example for creating the go to sets.

(Refer Slide Time: 53:14)



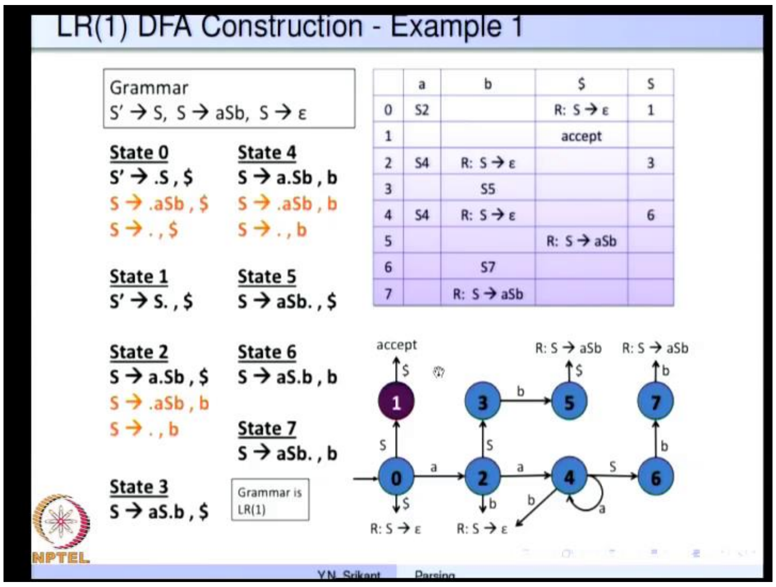
Construction of Sets of Canonical of LR(1) Items

```
void Set_of_item_sets(G'){ /* G' is the augmented grammar */
   C = {closure({$S' \rightarrow .S, \$$})};/* C is a set of LR(1) item sets */
   while (more item sets can be added to C) {
      for each item set $I \in C$ and each grammar symbol X
      /* X is a grammar symbol, a terminal or a nonterminal */
        if ((GOTO(I, X) $\neq \emptyset$) && (GOTO(I, X) $\notin$ C))
           C = C $\cup$ GOTO(I, X)
   }
}
```

- Each set in *C* (above) corresponds to a state o (LR(1) DFA)
- This is the DFA that recognizes viable prefixes

And now let us see how these two can be put together this procedure is very similar to what we had before for each item set in C. And each grammar symbol X if goto of I comma C is not equal to phi and goto of I comma X is not in c a ago the item sets c rather this set of items C and add goto I X to it. So, becomes better a bigger set of item sets rather. So, each set in c above corresponds to a state of the DFA in this case it is called the L R 1 DFA. And again this DFA recognizes viable prefixes and contains only those items which are valid for the viable prefixes that each that particular state.

(Refer Slide Time: 54:06)



LR(1) DFA Construction - Example 1

Grammar
S′ → S, S → aSb, S → ε

State 0
S′ → .S , $
S → .aSb , $
S → . , $

State 1
S′ → S. , $

State 2
S → a.Sb , $
S → .aSb , b
S → . , b

State 3
S → aS.b , $

State 4
S → a.Sb , b
S → .aSb , b
S → . , b

State 5
S → aSb. , $

State 6
S → aS.b , b

State 7
S → aSb. , b

Grammar is LR(1)

	a	b	$	S
0	S2		R: S → ε	1
1			accept	
2	S4	R: S → ε		3
3		S5		
4	S4	R: S → ε		6
5			R: S → aSb	
6		S7		
7		R: S → aSb		

So, getting back to this example. So, we have seen parts of this example already. So, here is state zero. So, we start with this item S prime going to dot S these two are added by closure then we get state 1 state 2 state there then state 4 state 5 state 6 and state seven. So, here is the DFA corresponding to it and here is the you know table. So, let us how the table is filled for state 0 on little a is a shift and there is no item with a little b after the dot and this is a reduce item and the reduction happens only on dollar. So, we add reduce by S 2 epsilon and because of this dot S we have a go to go to entry as well. So, similarly for state number one we have just an accept you know there is nothing more possible. This is S prime going to S dot and dollar on dollar it accepts and for state number 2 on a on the symbol a we have a shift to state number 4.

So, that is because of this right and on a this is a reduction and on a b the reduction is possible via S 2 epsilon and this item gives rise to a go to that is go to number go to state number 3. So, in this manner you know we fill 3 as well which is just a shift. So, on b and for state number 4 we have a go to that is state number 6 and then we have a 6 because of this a and we have a reduce because of this. So, that is on b. So, you observe that even though there are many shift and reduce items in each state. We really do not have more than one reduce or shift action in each of these slots in this particular table and therefore, this grammar happens to be L R 1. So, we will end the lecture here and continue with more examples in the next lecture.

Thank you.