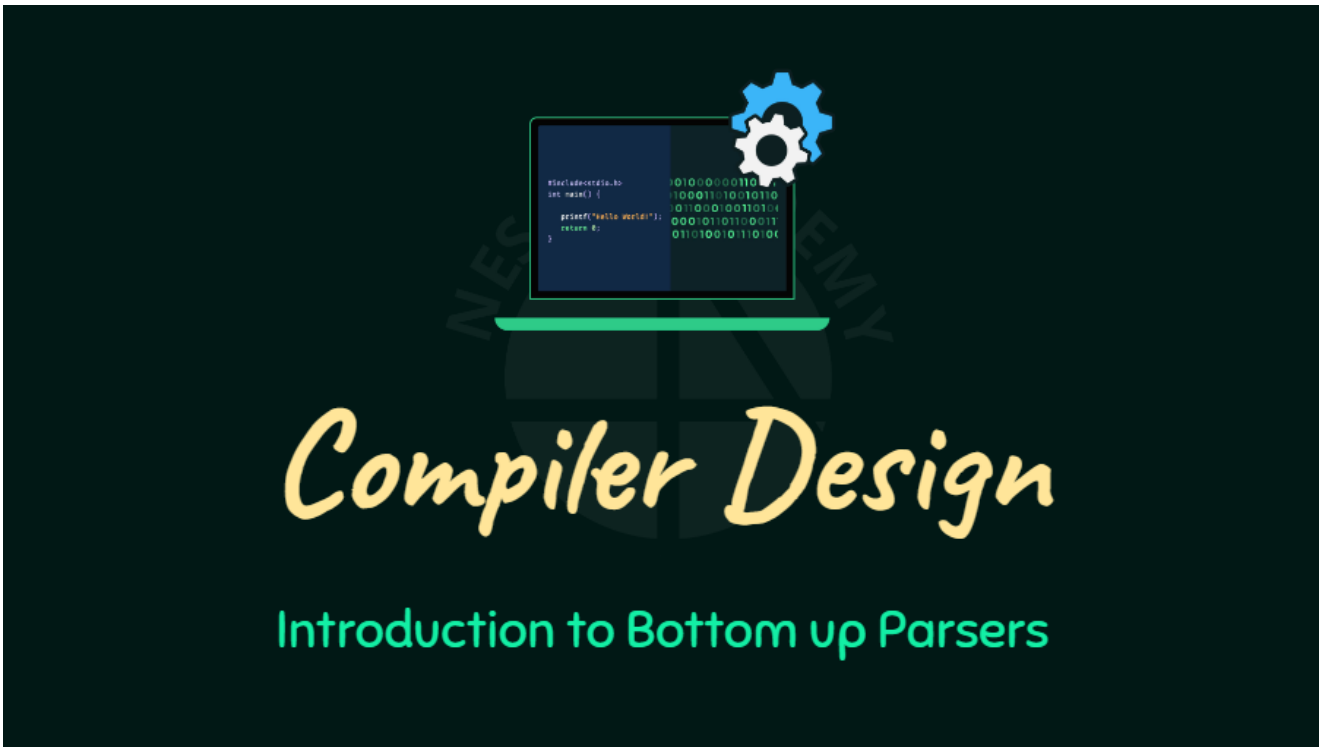
Bottom-Up Parsers | Neso Academy

nesoacademy.org/cs/12-compiler-design/ppts/04-bottom-upparsers

Bottom-up ParsersNeso AcademyCHAPTER-4



Compiler DesignIntroduction to Bottom up Parsers

Outcome

- ☆ Understanding the Bottom up approach of parsing.
- ☆ Revisiting the Classification of Parsers.

Outcome☆Understanding the Bottom up approach of parsing.☆Revisiting the Classification of Parsers.

Generation of Parse Tree – Bottom up approach:

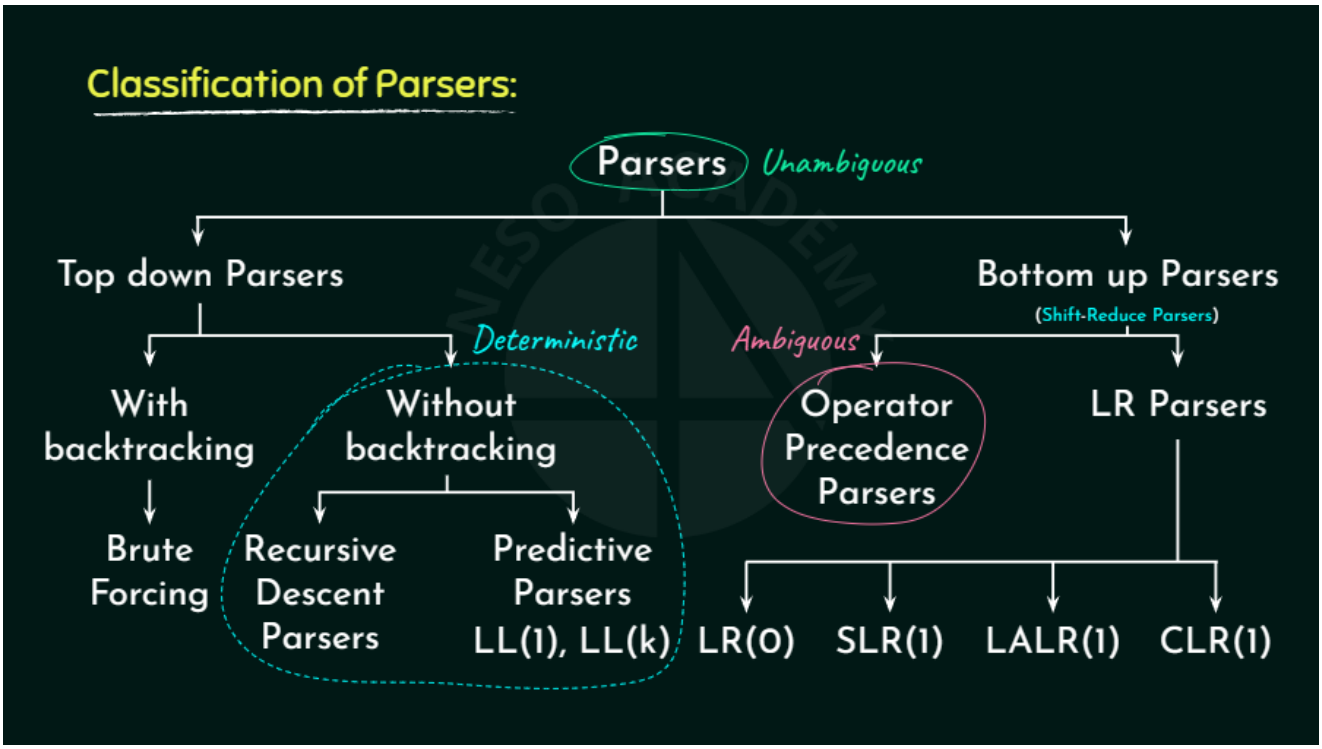
S → aABe
A → Abc I b
B → d

Decision: When to reduce.

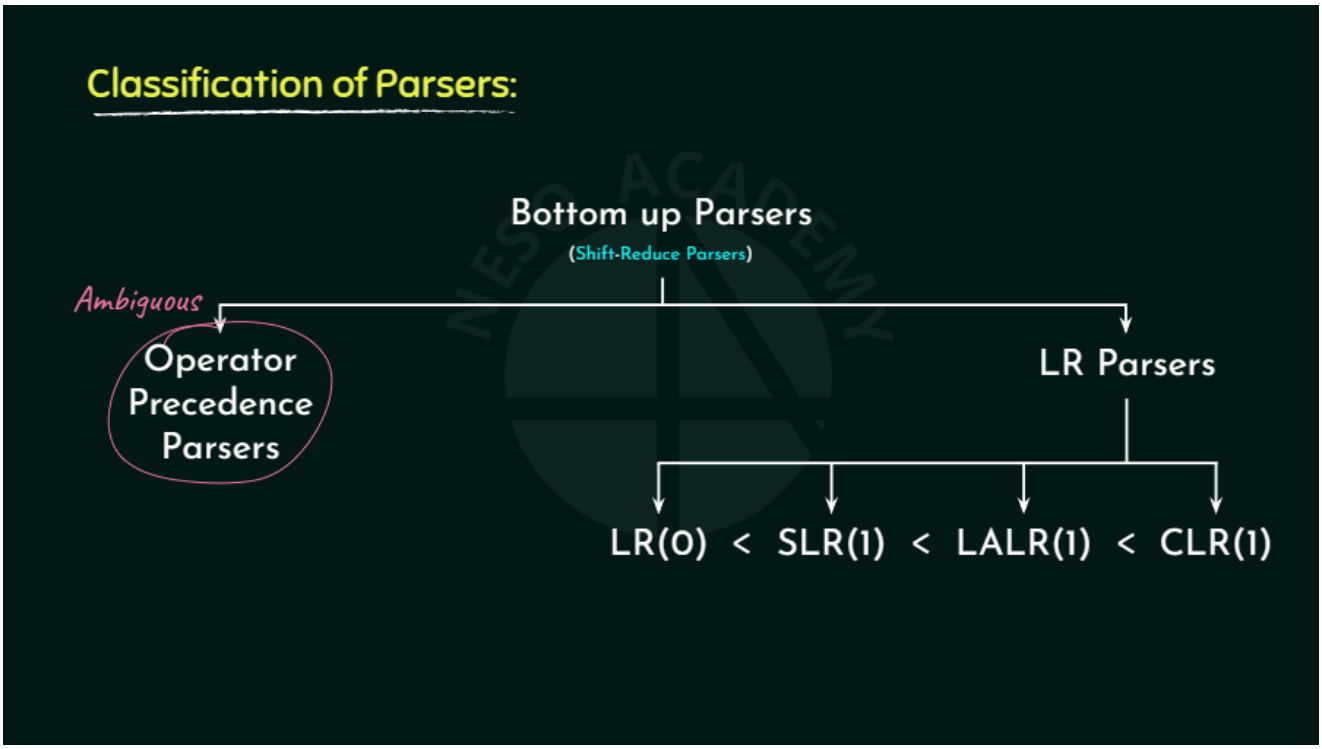
S ⇒ aABe
⇒ aAde
⇒ aAbcde
⇒ aabcde

(Right most Derivation) - In reverse.

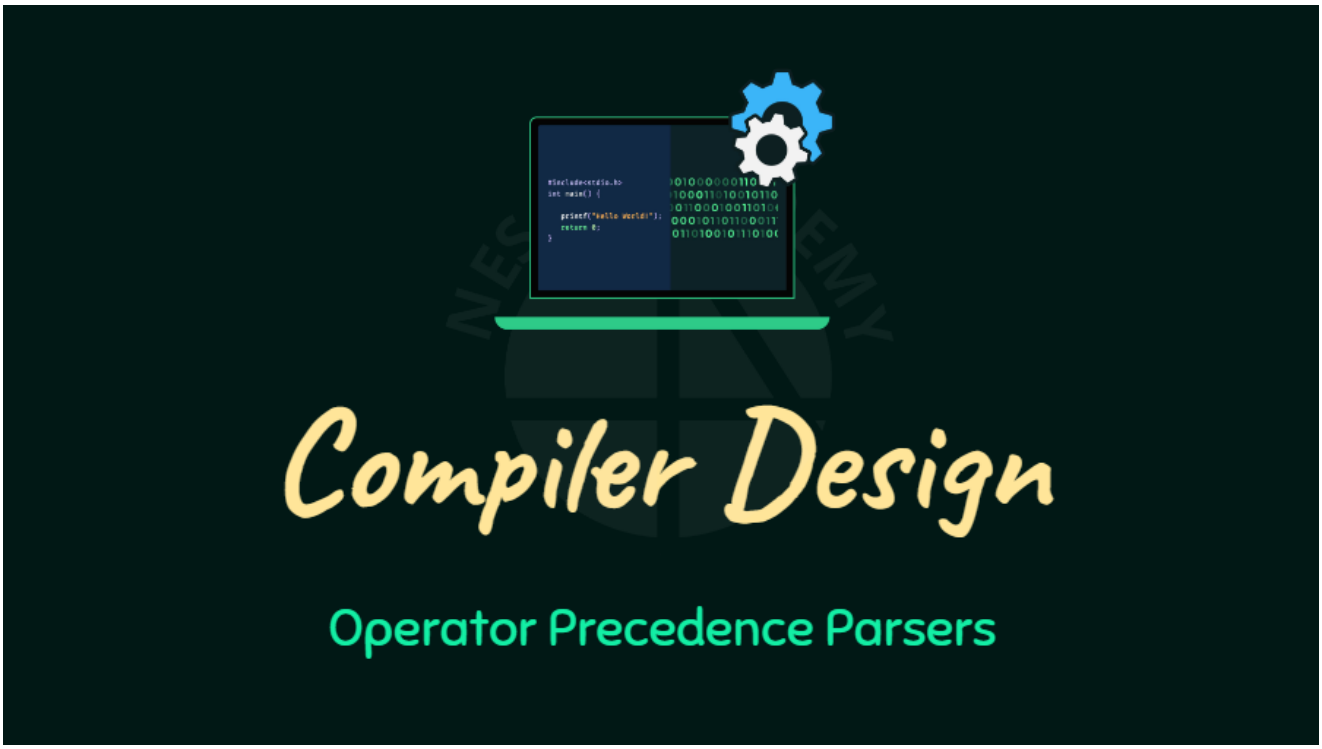
Generation of Parse Tree - Bottom up approach:S → aABe A → Abc I bB → dS ⇒ aABe⇒ aAde⇒ aAbcde⇒ aabcde(Right most Derivation)- In reverse.Decision:When to reduce.abbcdeAASB



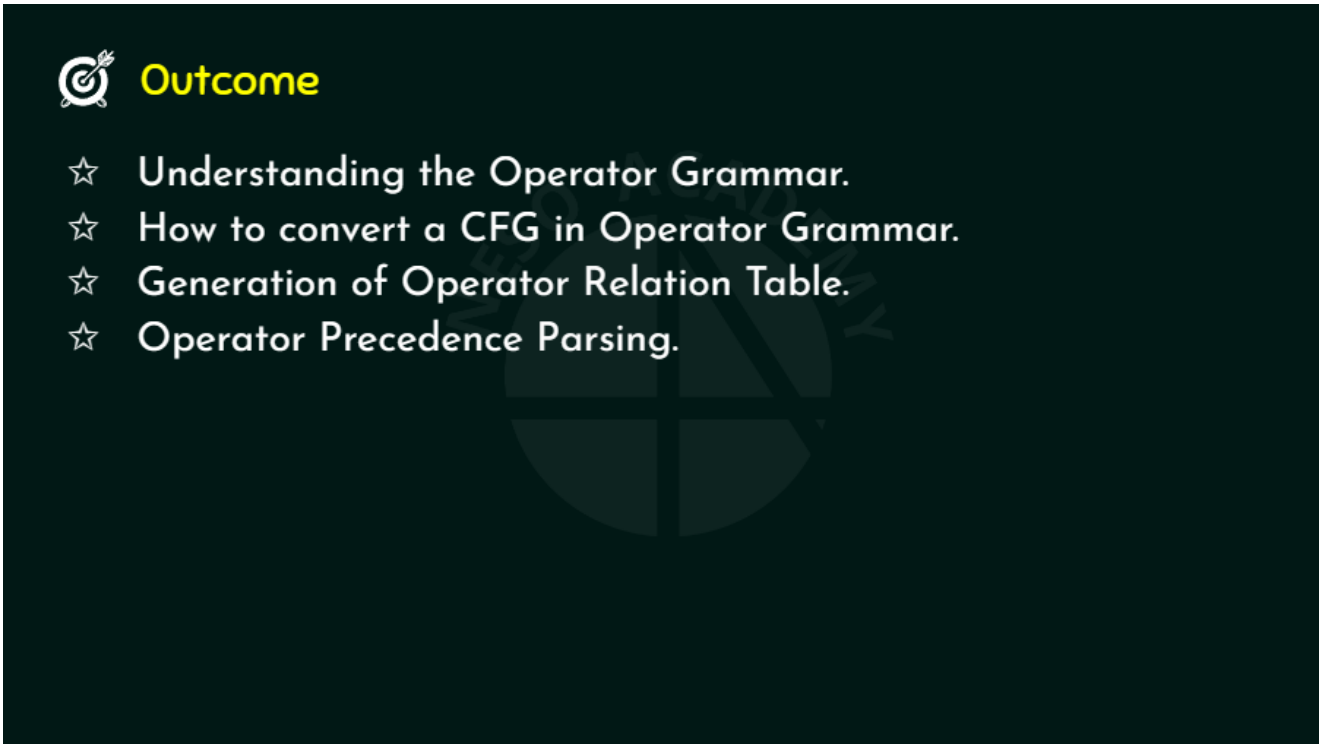
Classification of Parsers:ParsersTop down ParsersBottom up Parsers(Shift-Reduce Parsers)With backtrackingWithout backtrackingBrute ForcingLR ParsersOperator Precedence ParsersLR(0)CLR(1)LALR(1)SLR(1)Recursive Descent ParsersPredictive ParsersLL(1), LL(k)UnambiguousAmbiguousDeterministic



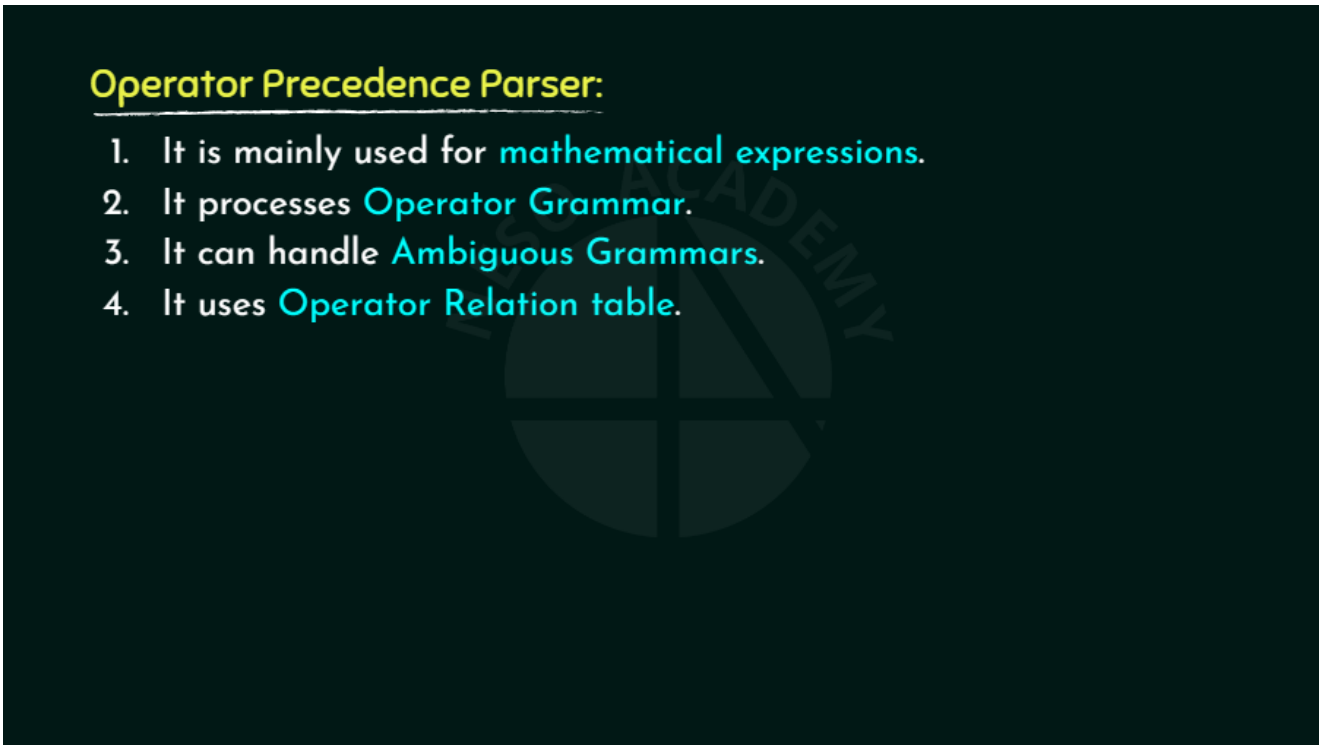
AmbiguousClassification of Parsers:Bottom up Parsers(Shift-Reduce Parsers)LR ParsersOperator Precedence ParsersLR(0)CLR(1)LALR(1)SLR(1)<<<



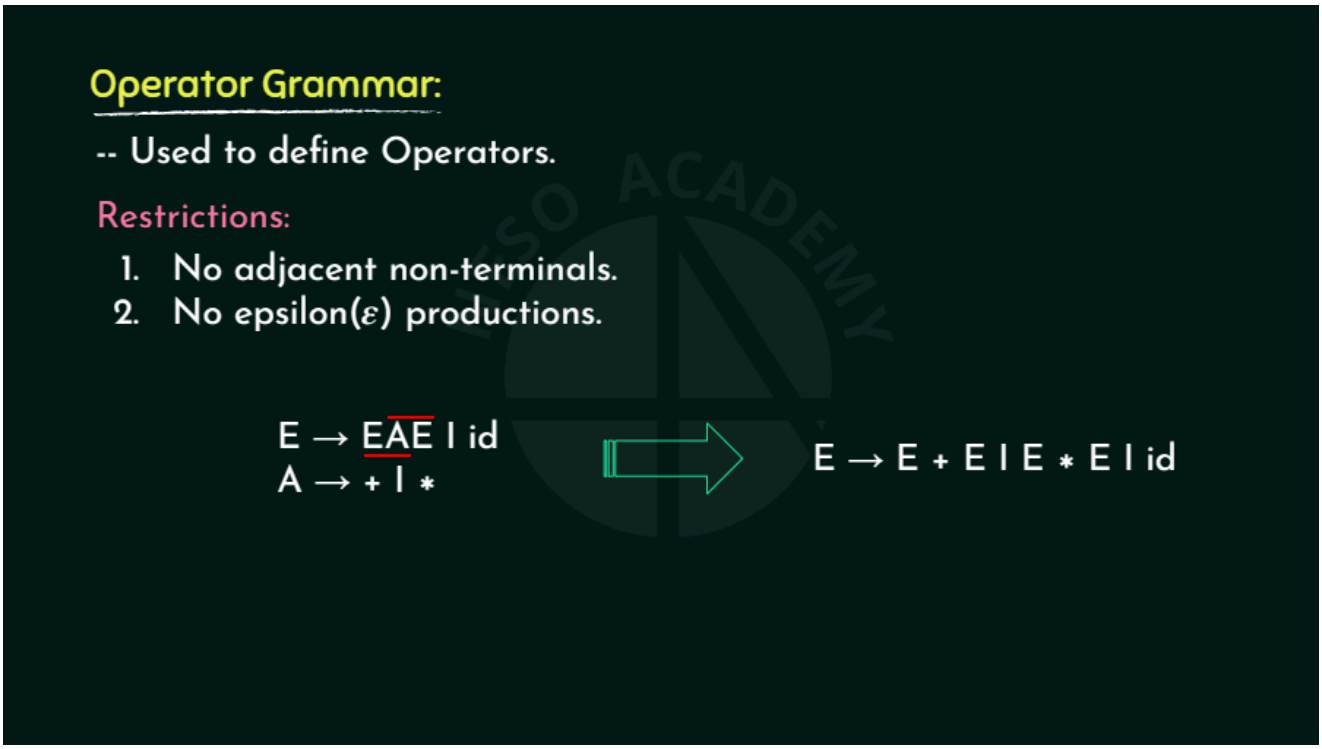
Compiler DesignOperator Precedence Parsers



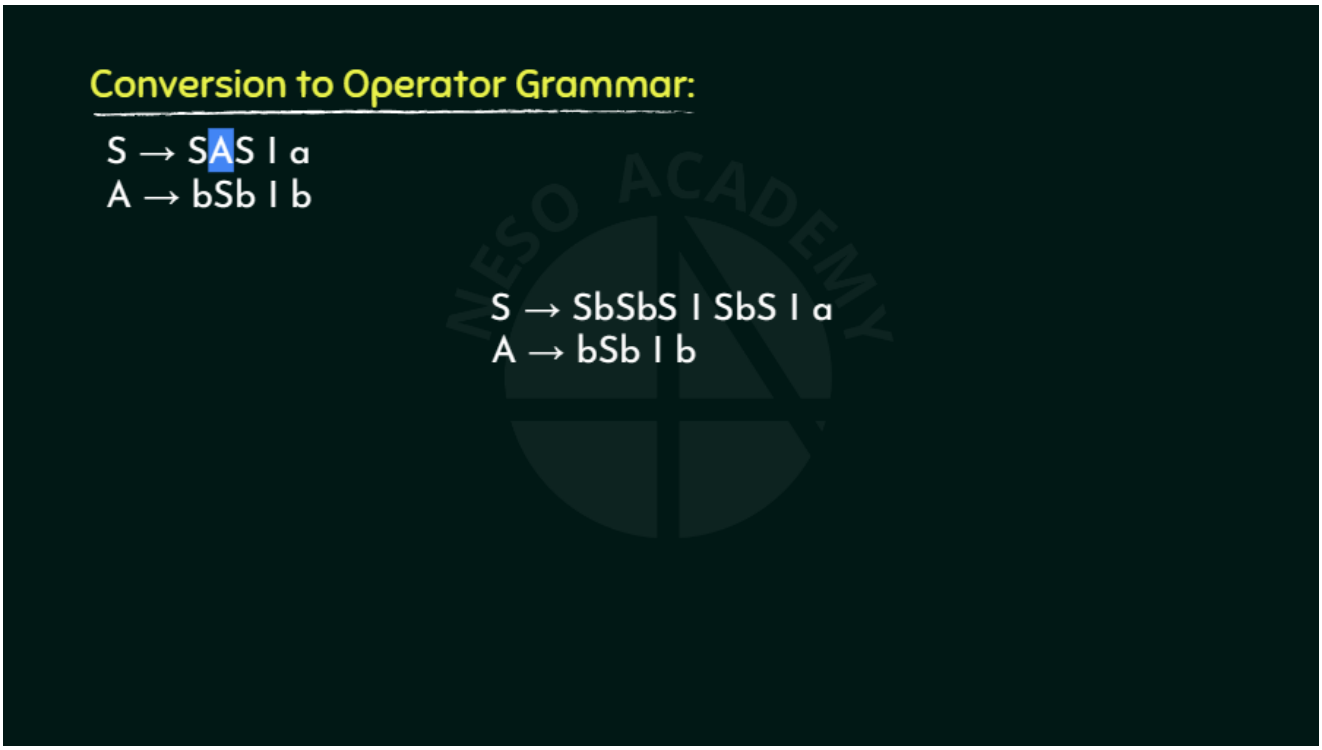
Outcome☆Understanding the Operator Grammar. ☆How to convert a CFG in Operator Grammar.☆Generation of Operator Relation Table.☆Operator Precedence Parsing.



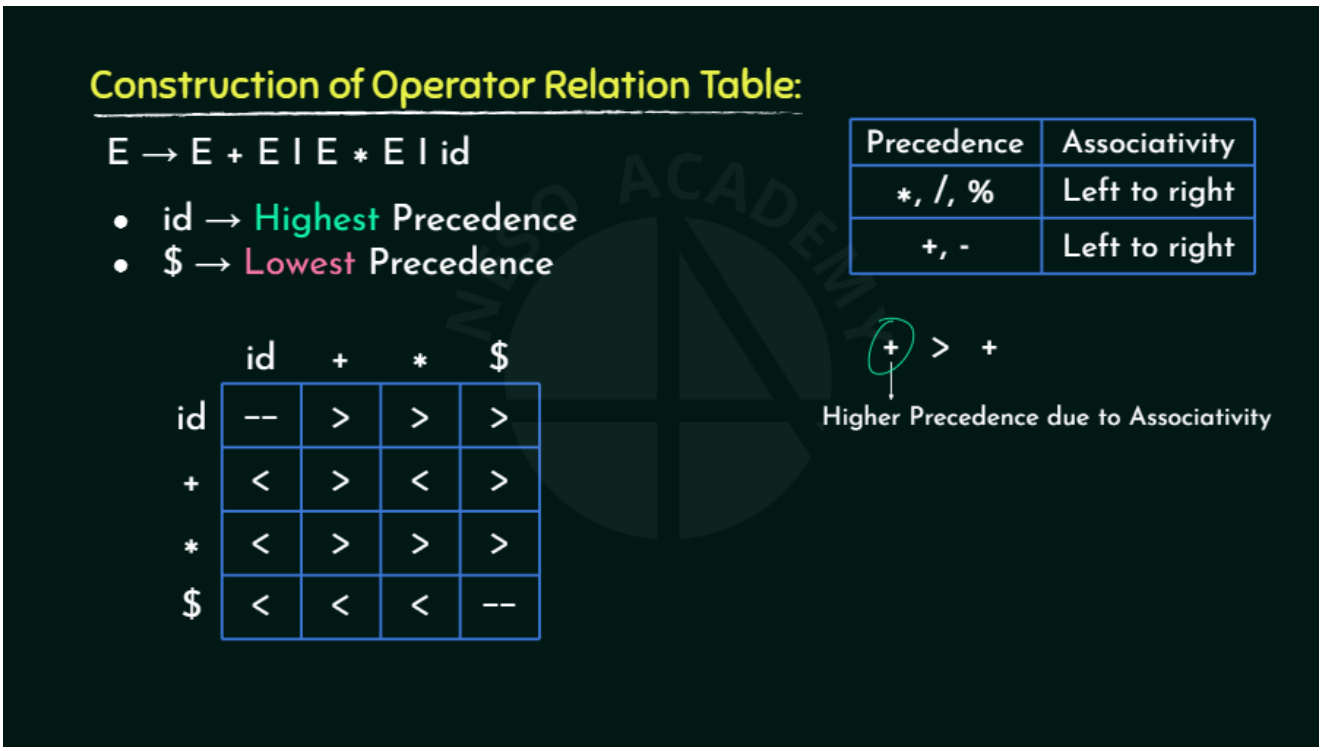
Operator Precedence Parser:1.It is mainly used for mathematical expressions. 2.It processes Operator Grammar.3.It can handle Ambiguous Grammars.4.It uses Operator Relation table.



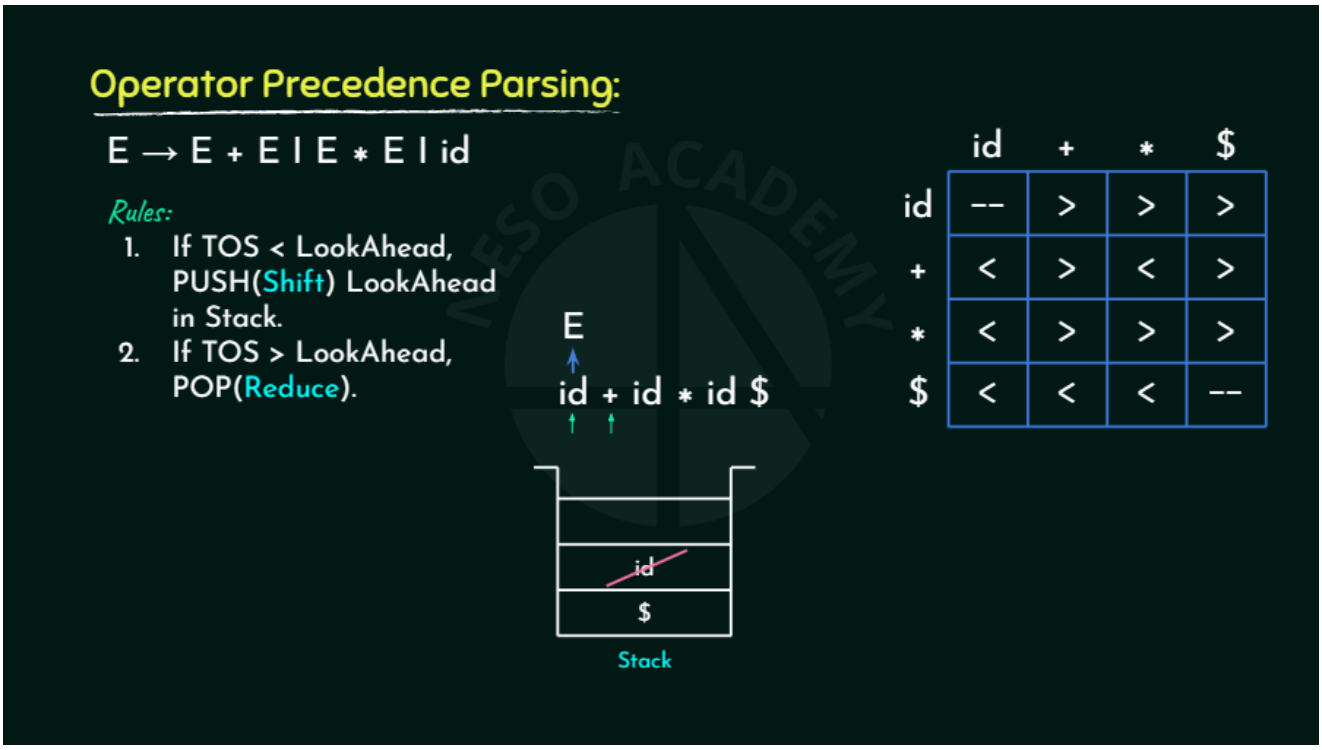
Operator Grammar:-- Used to define Operators.Restrictions:1.No adjacent non-terminals.2.No epsilon(ε) productions.E → EAE I id A → + I +*E → E + E I E + E I id*



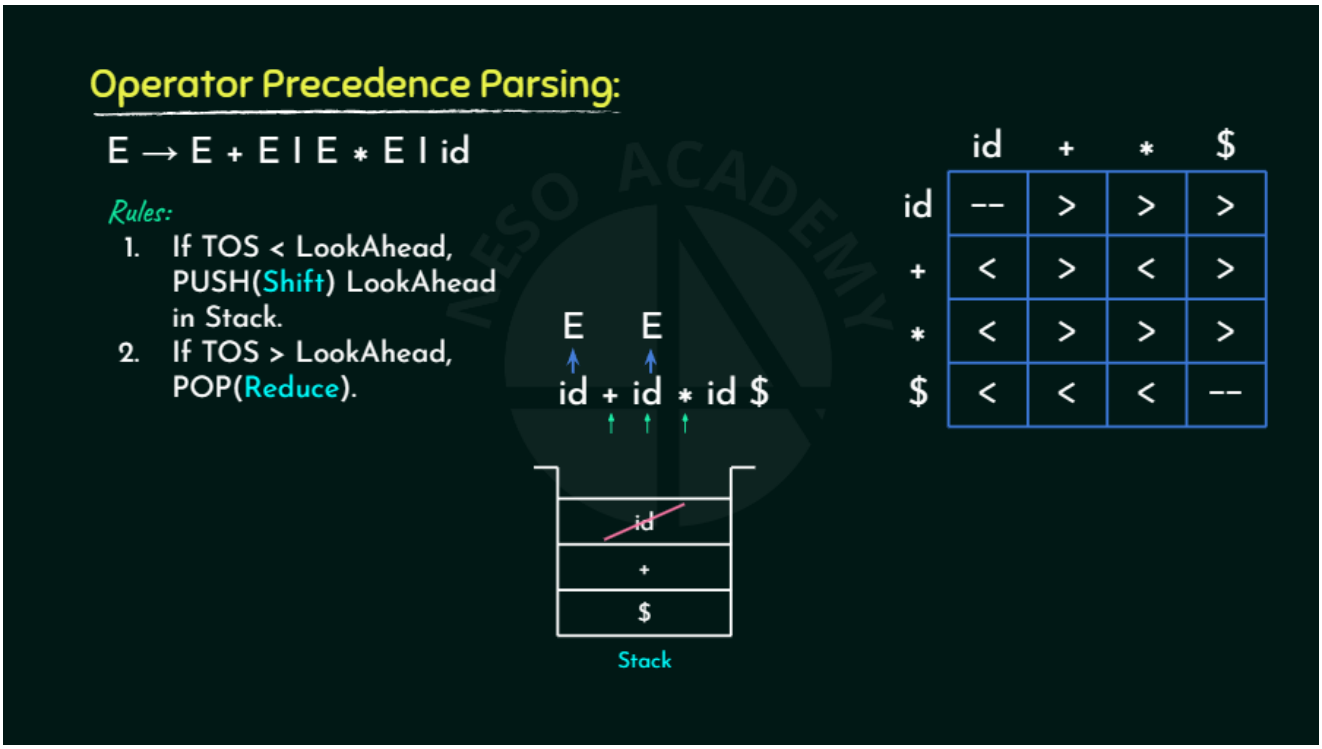
S → SAS I a A → bSb I b Conversion to Operator Grammar:bSbSISbSI a → SA → bSb I bS



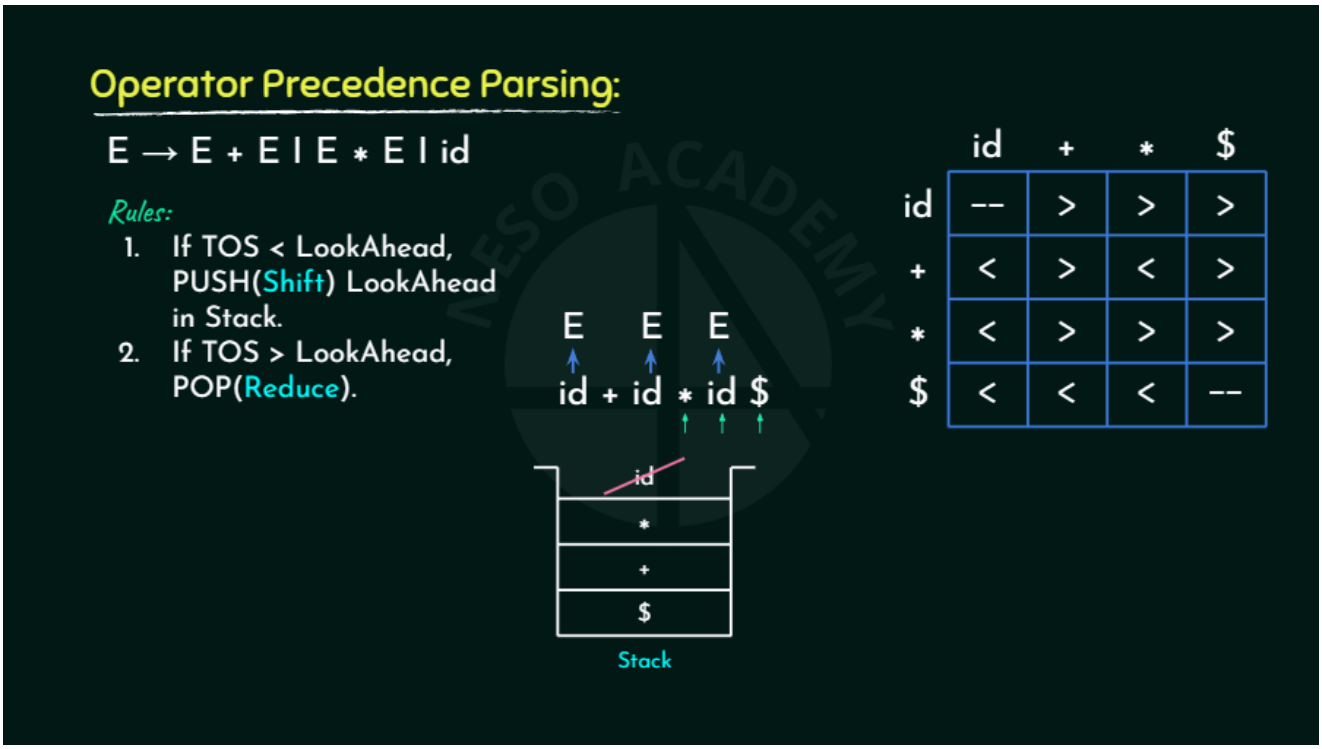
++Higher Precedence due to Associativity Construction of Operator Relation Table:E → E + E I E + E I id*+, /, %*PrecedenceAssociativityLeft to right+, -Left to right●id → Highest Precedence●$ → Lowest Precedence-->>>>><<<>>><<<--idid+$+$**>



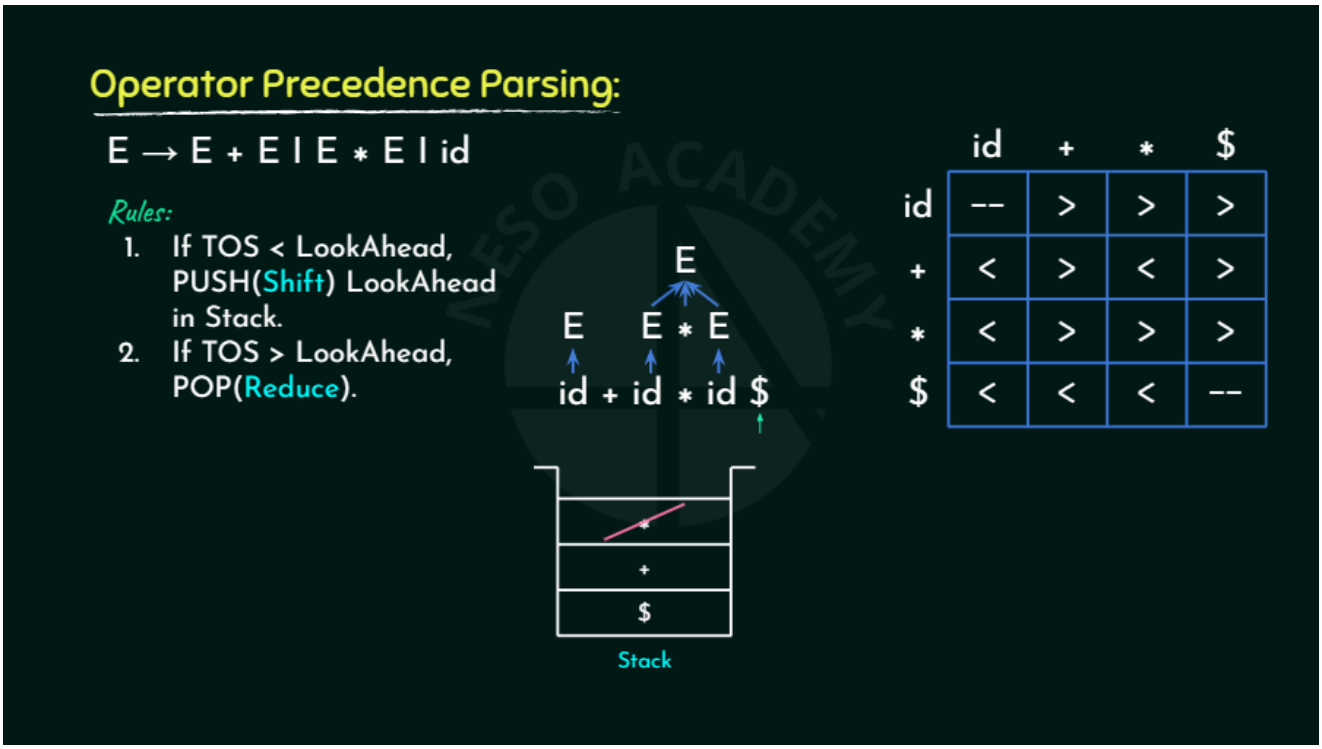
Stack Operator Precedence Parsing:E → E + E I E + E I id*idid+$+$**-->>>>><<<>>><<<--id + id + id $*ERules:1.If TOS < LookAhead, PUSH(Shift) LookAhead in Stack.2.If TOS > LookAhead, POP(Reduce).$id



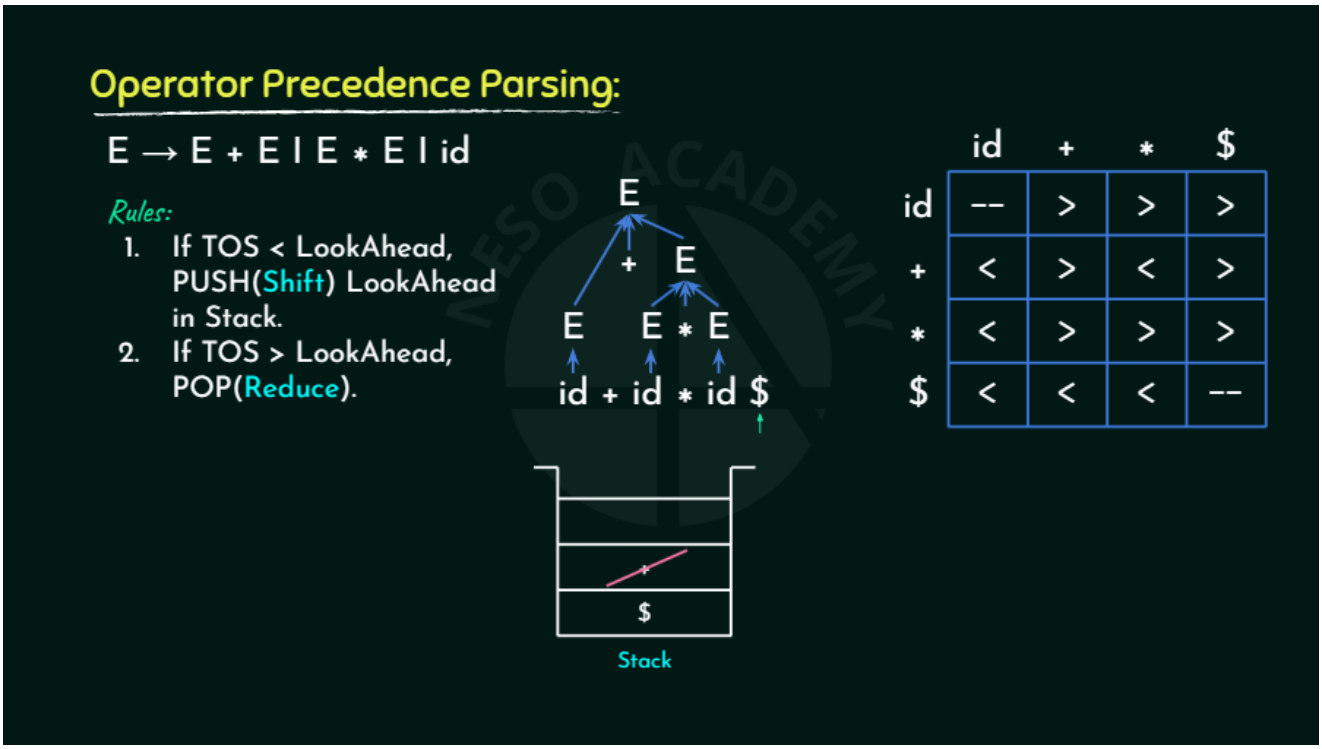
Operator Precedence Parsing:E → E + E I E + E I id*idid+$+$**-->>>>><<<>>><<<--id + id + id $*ERules:1.If TOS < LookAhead, PUSH(Shift) LookAhead in Stack.2.If TOS > LookAhead, POP(Reduce).$Stack+idE



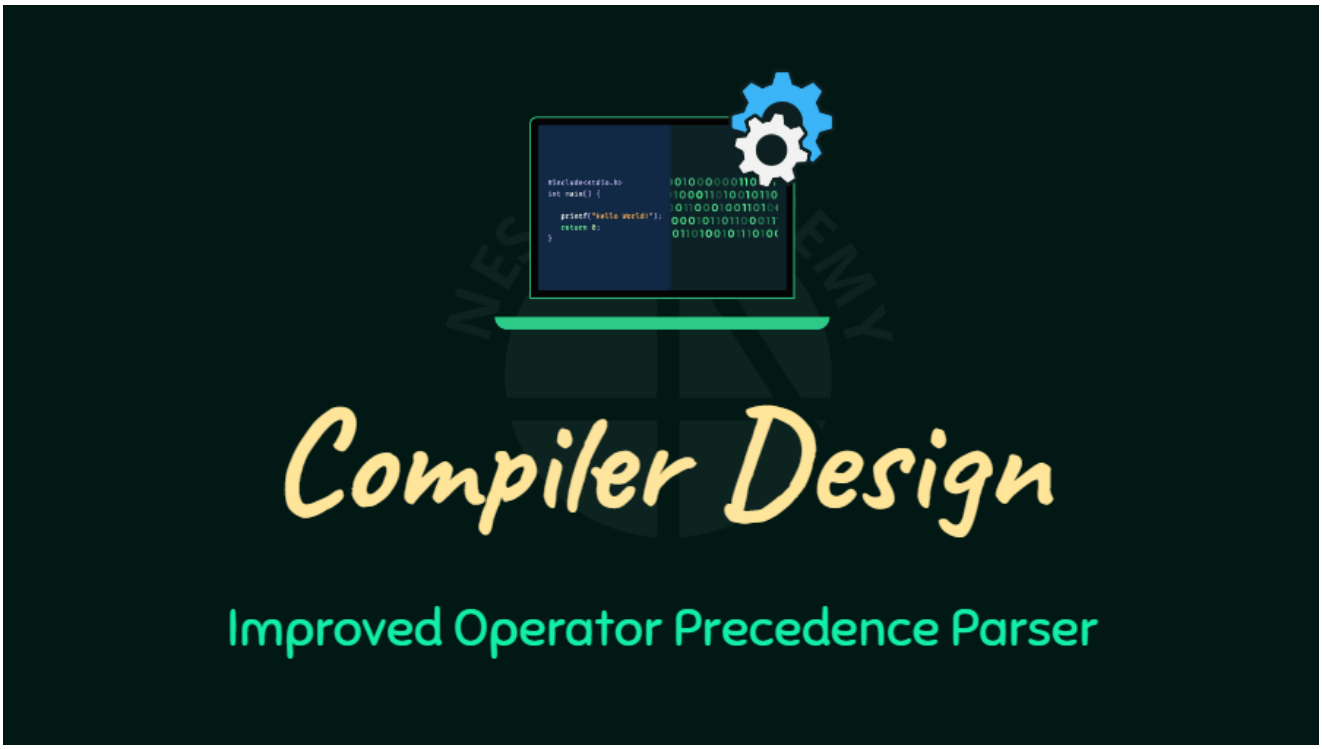
Operator Precedence Parsing:E → E + E I E + E I id*idid+$+$**-->>>>>><<<>>>><<<--id + id + id $*ERules:1.If TOS < LookAhead, PUSH(Shift) LookAhead in Stack.2.If TOS > LookAhead, POP(Reduce).$Stack+*EidE



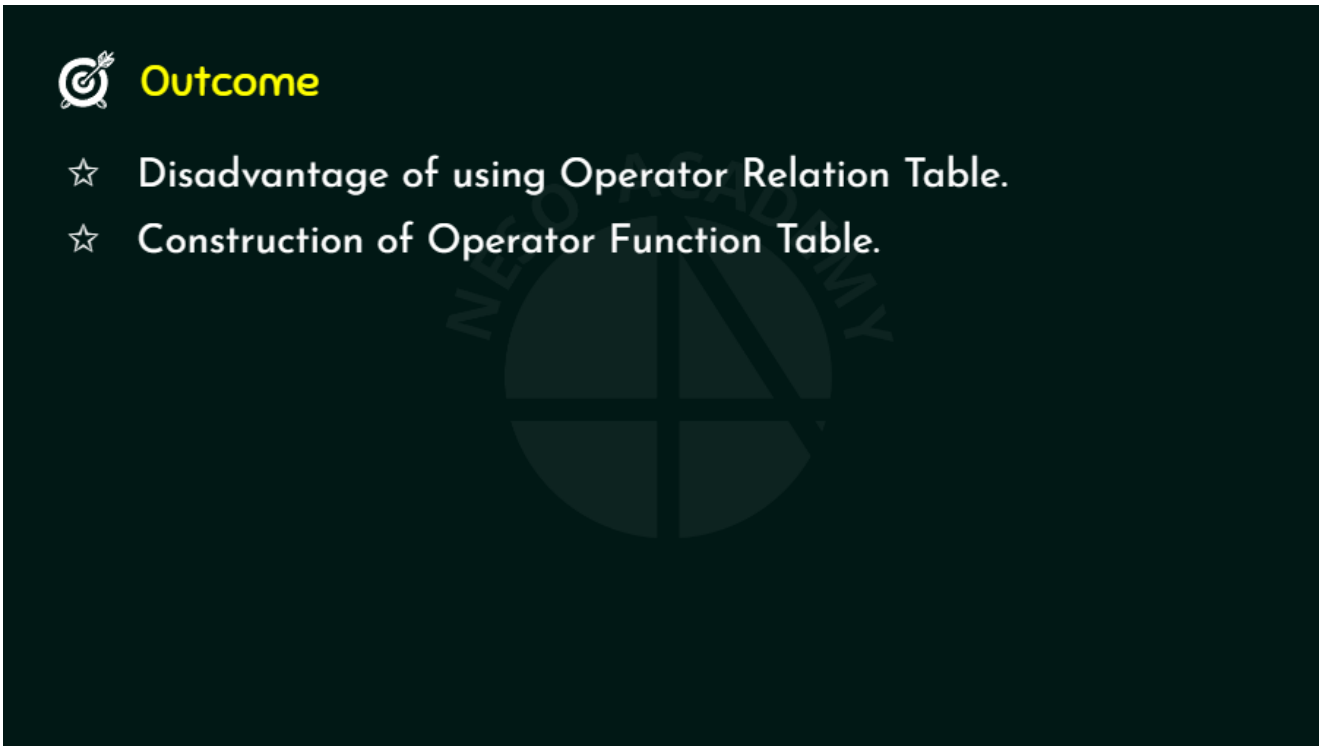
Operator Precedence Parsing:E → E + E I E + E I id*idid+$+$**-->>>>>><<<>>>><<<--id + id + id $*ERules:1.If TOS < LookAhead, PUSH(Shift) LookAhead in Stack.2.If TOS > LookAhead, POP(Reduce).$Stack+*EE*E



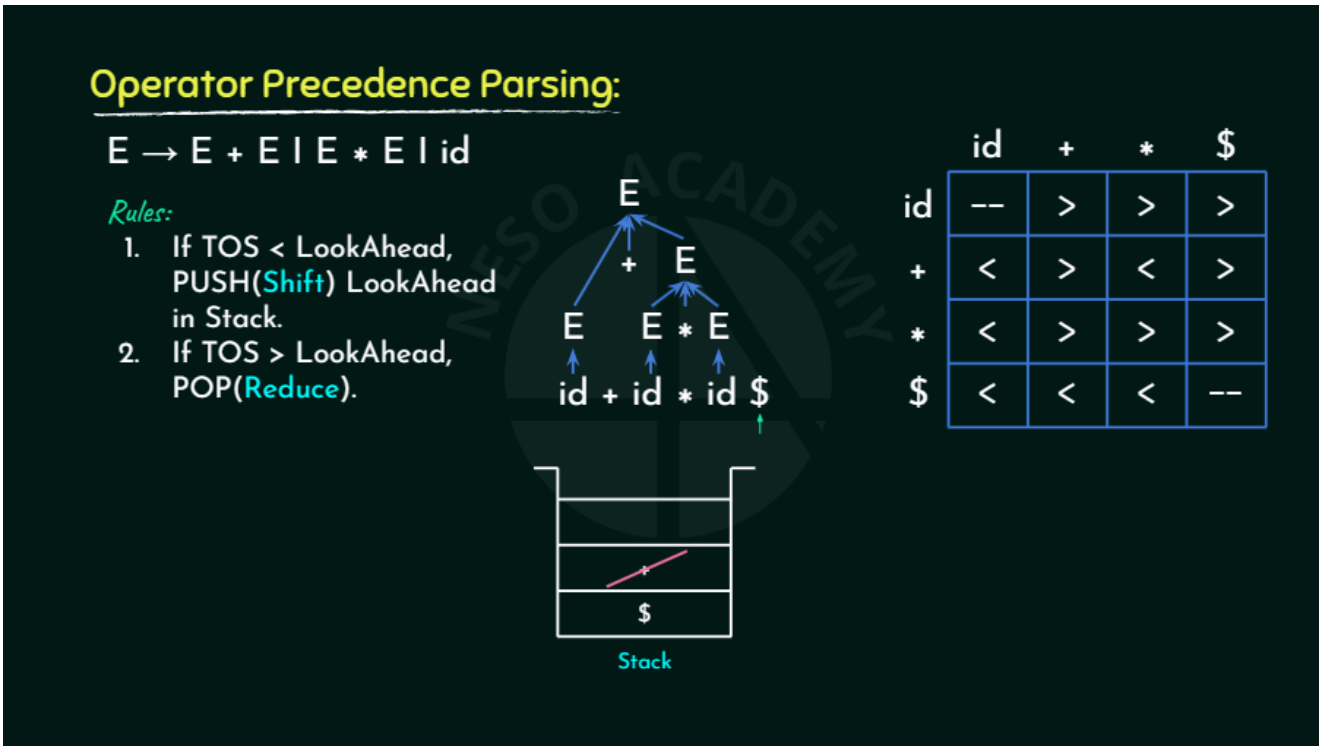
Operator Precedence Parsing:E → E + E I E + E I id*idid+$+$**-->>>>>><<<>>><<<--id + id + id $*ERules:1.If TOS < LookAhead, PUSH(Shift) LookAhead in Stack.2.If TOS > LookAhead, POP(Reduce).$Stack+EE*E+ E



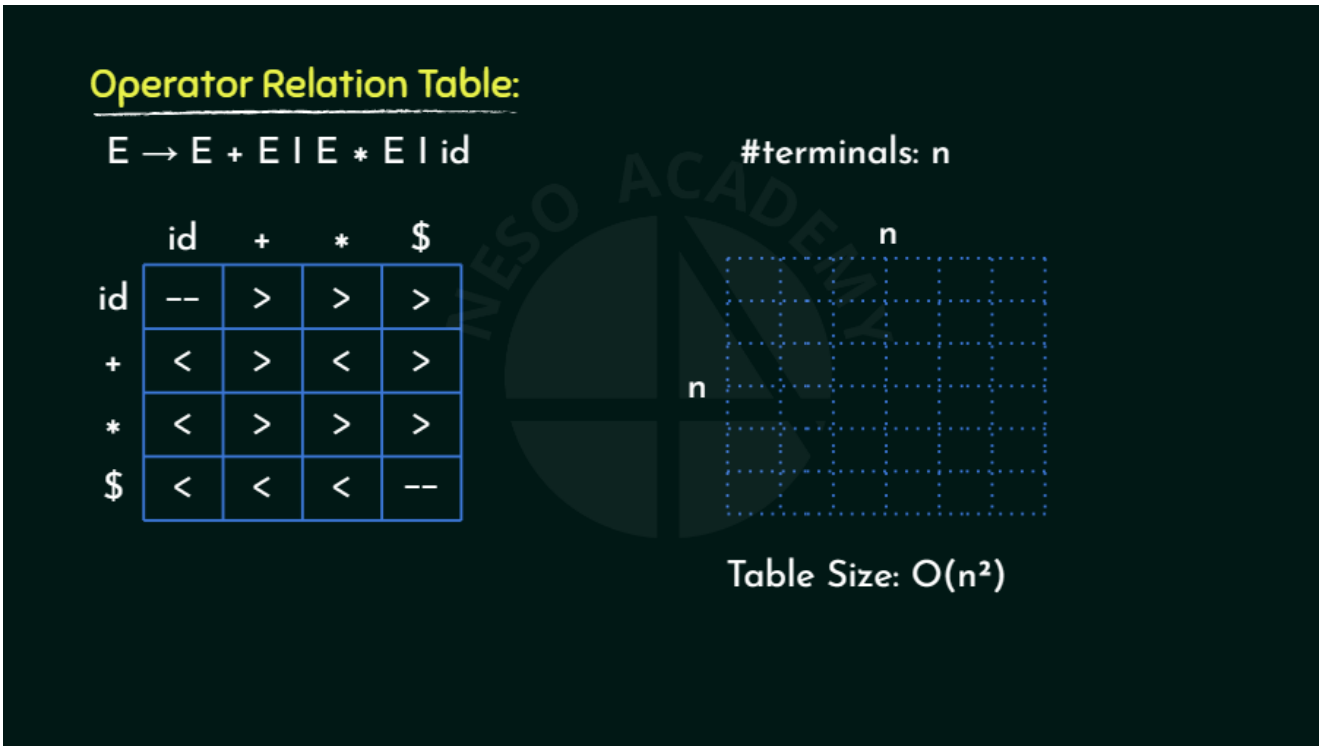
Compiler DesignImproved Operator Precedence Parser



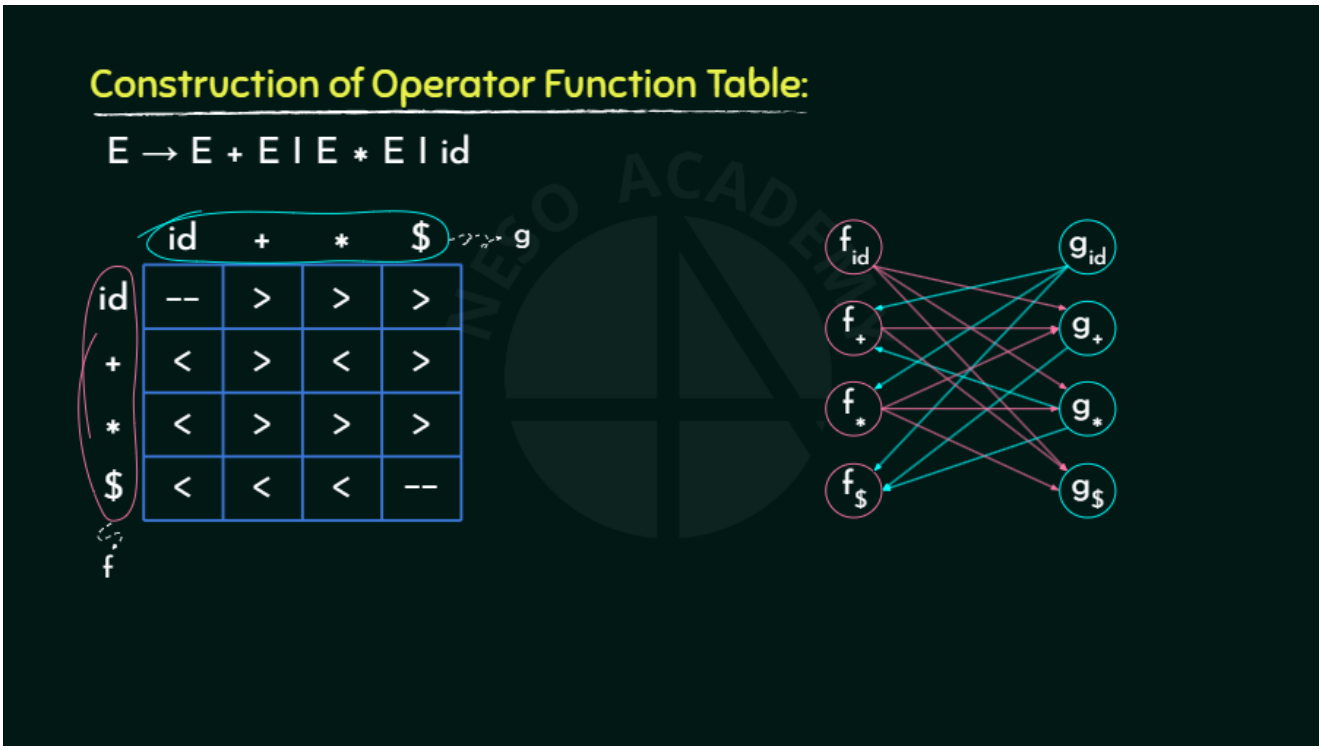
Outcome☆Disadvantage of using Operator Relation Table. ☆Construction of Operator Function Table.



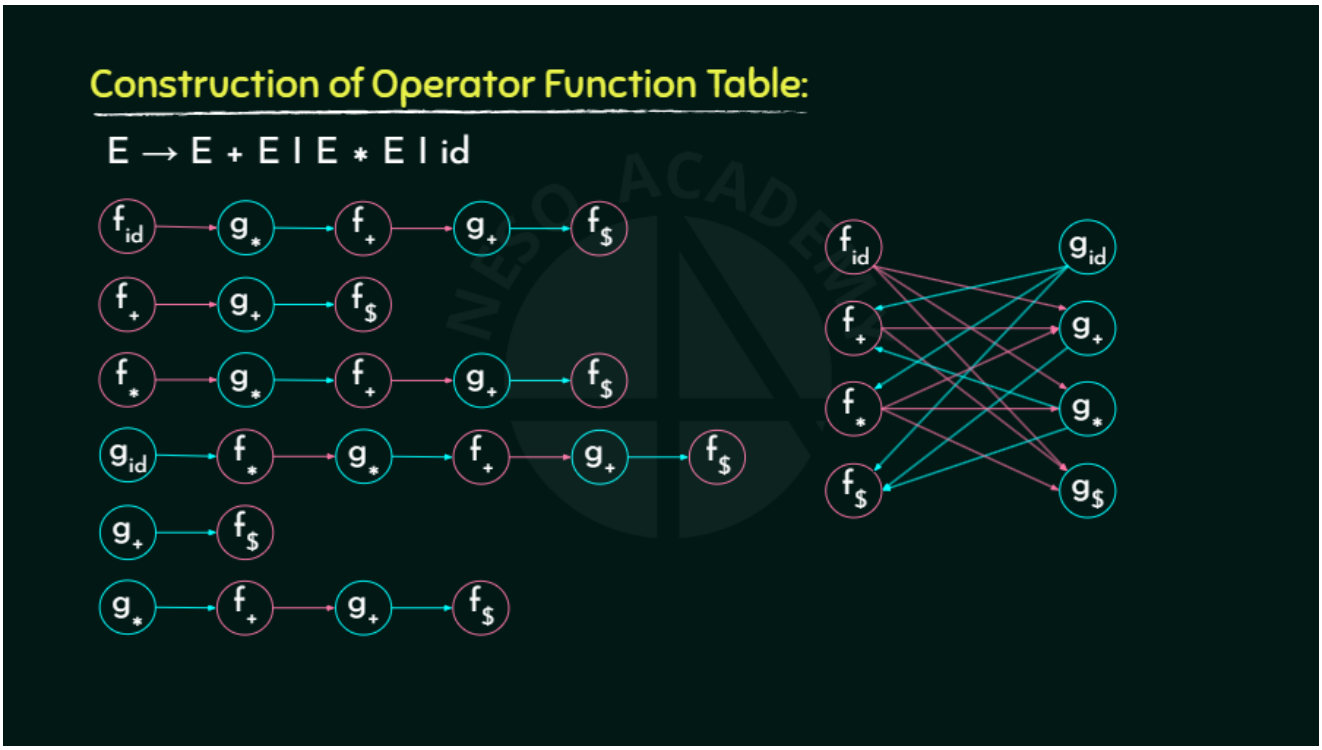
Operator Precedence Parsing:E → E + E I E + E I id*idid+$+$**-->>>>><<<>>><<<--id + id + id $*ERules:1.If TOS < LookAhead, PUSH(Shift) LookAhead in Stack.2.If TOS > LookAhead, POP(Reduce).$Stack+EE*E+ E



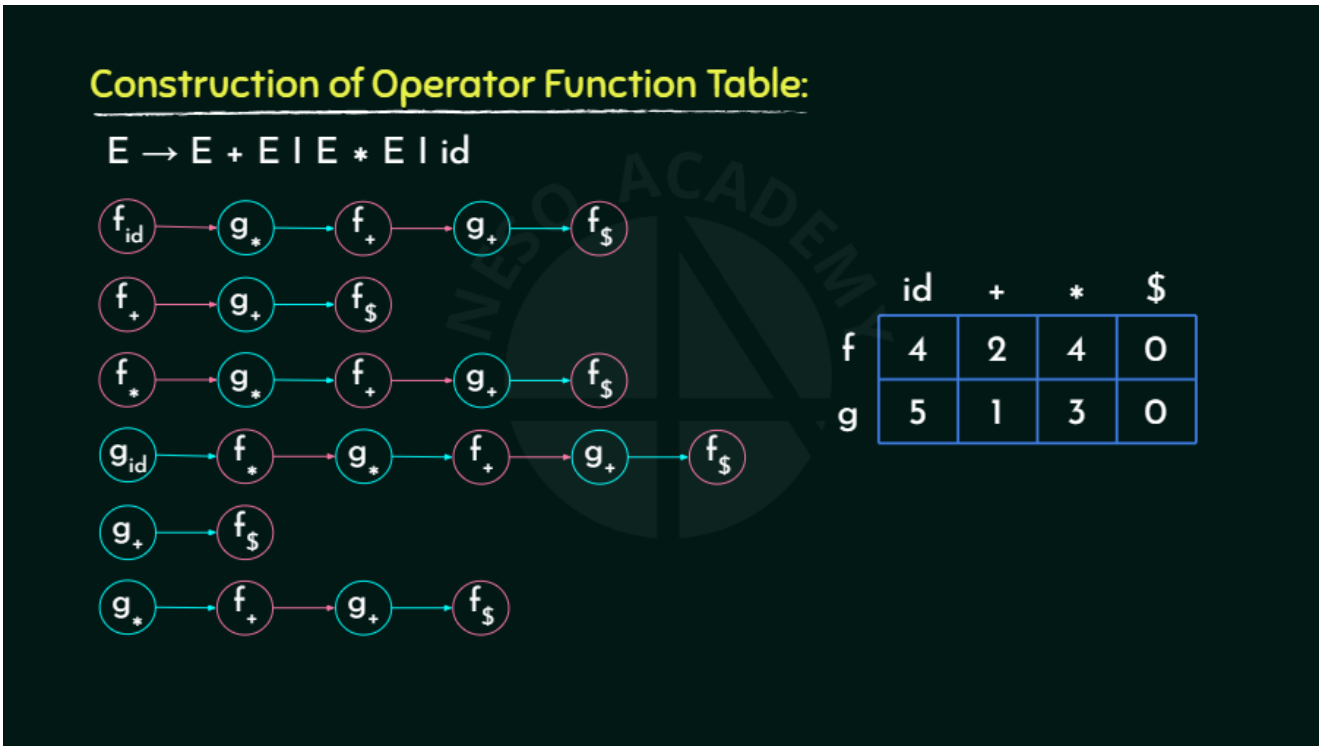
Operator Relation Table:E → E + E I E + E I id*idid+$+$**-->>>>><<<>>><<<--#terminals: n nnTable Size: O(n²)



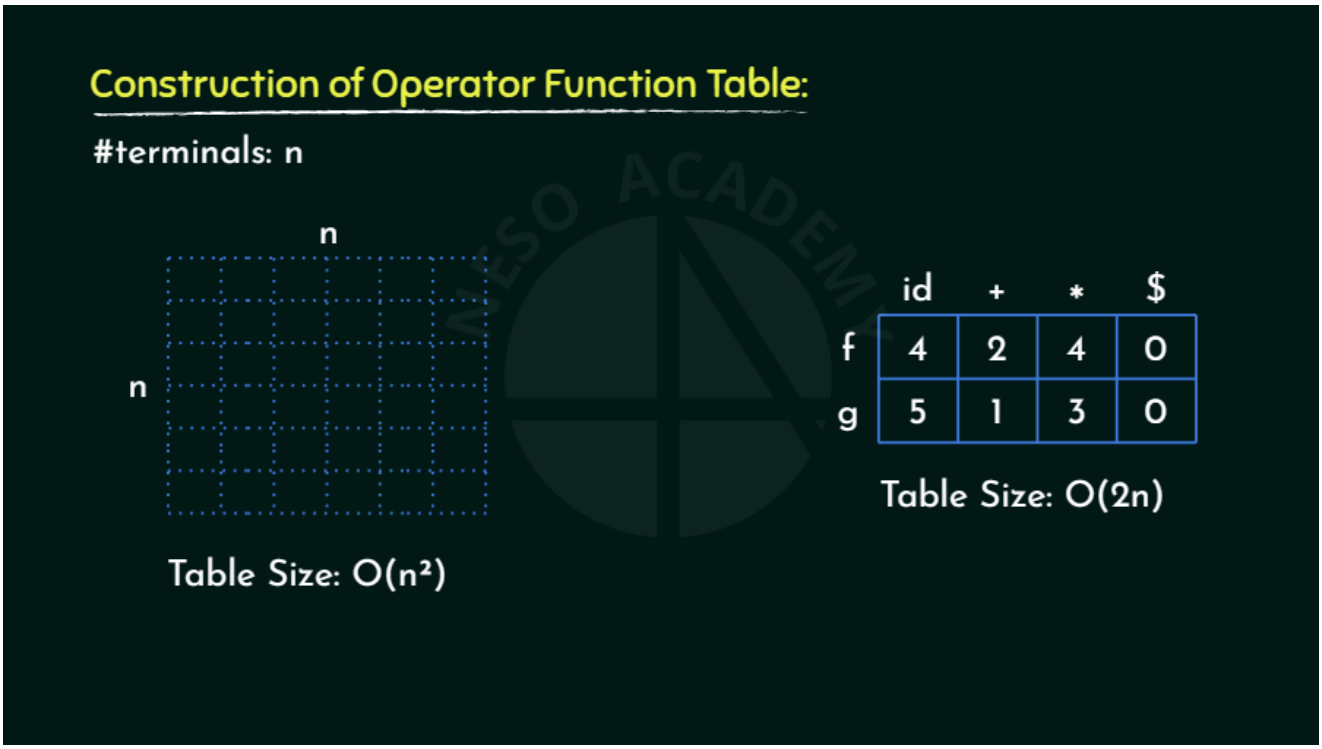
Construction of Operator Function Table:E → E + E I E + E I id*idid+$+$**-->>>>><<<>>><<<--g ffidf+f+*f$g+gidg+*g$



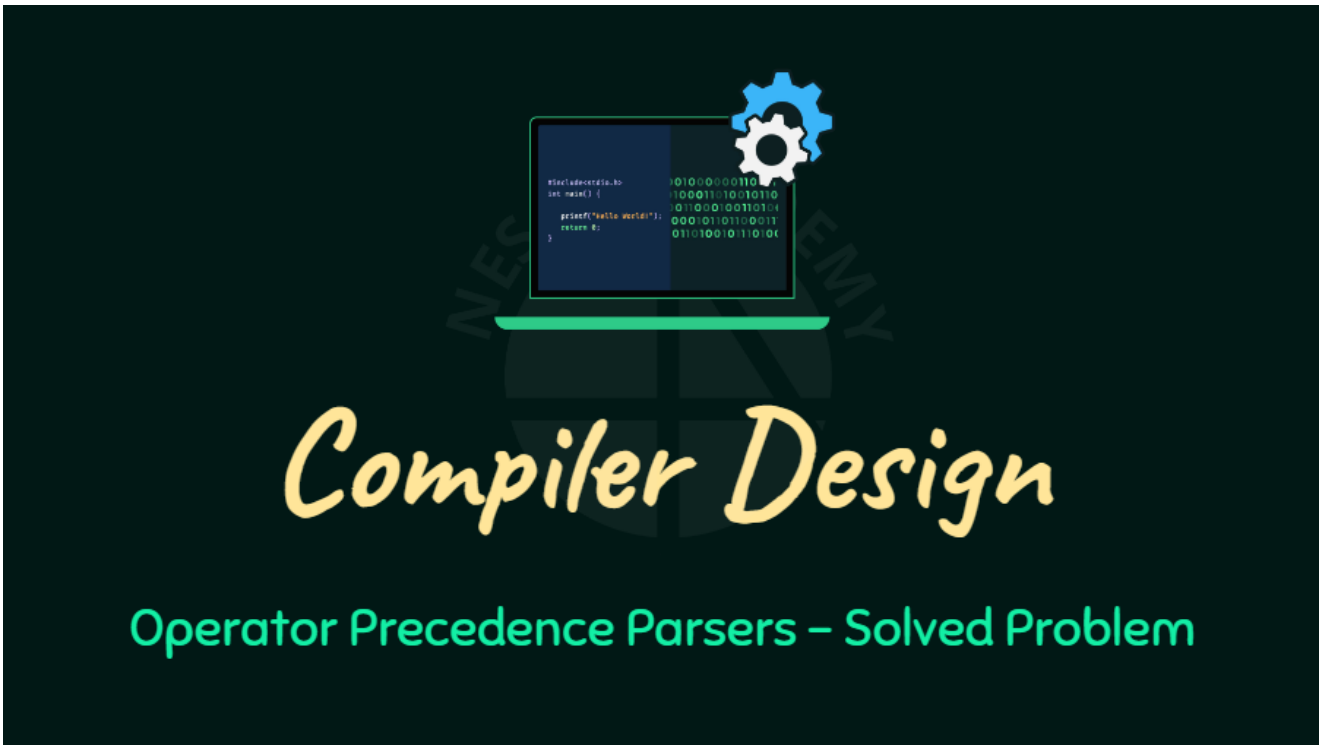
Construction of Operator Function Table:E → E + E I E + E I id*fidf+g+gidf+*f$g+*g$fidg+*f+g+f$f+g+f$f+*g+*f+g+f$gidg+*f+g+f$f+*g+*f+g+f$g+f$



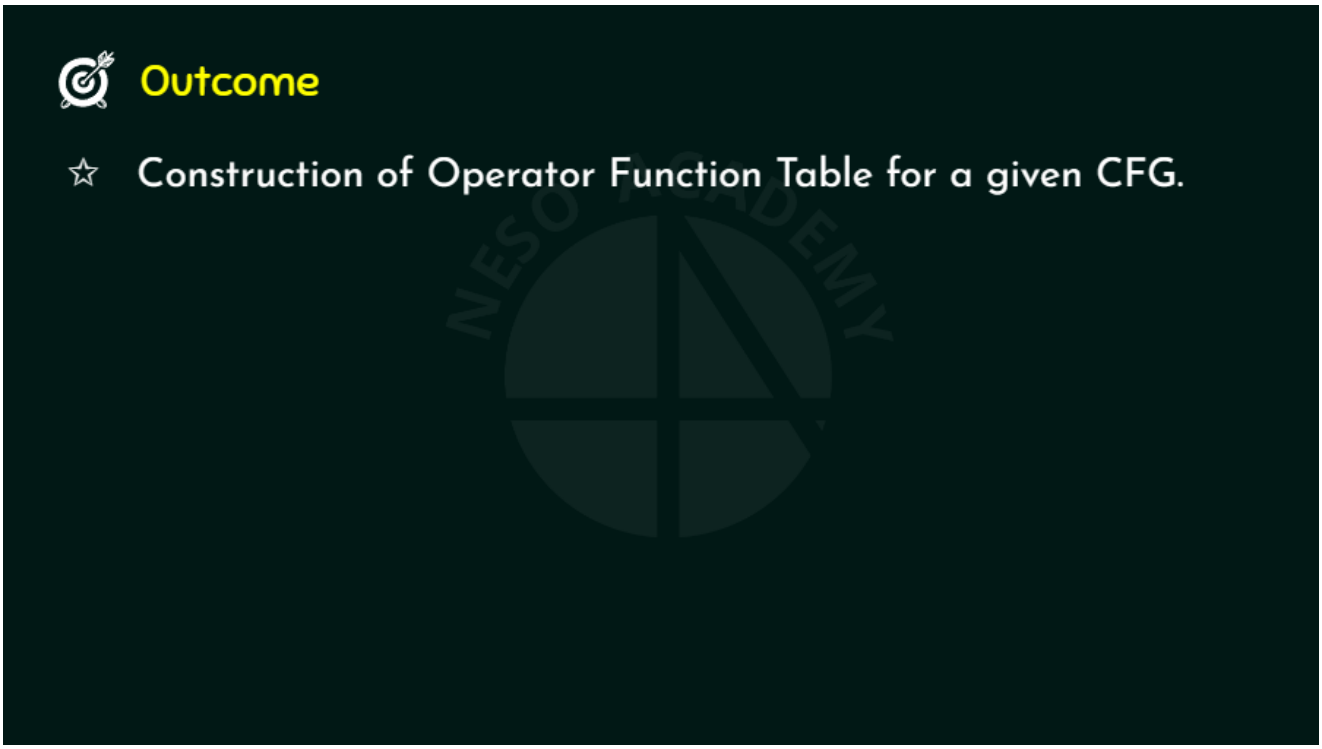
Construction of Operator Function Table:E → E + E I E + E I id*fidg+*f+g+f$f+g+f$f+*g+*f+g+f$gidg+*f+g+f$f+*g+*f+g+f$g+f$42400135fid+$g*



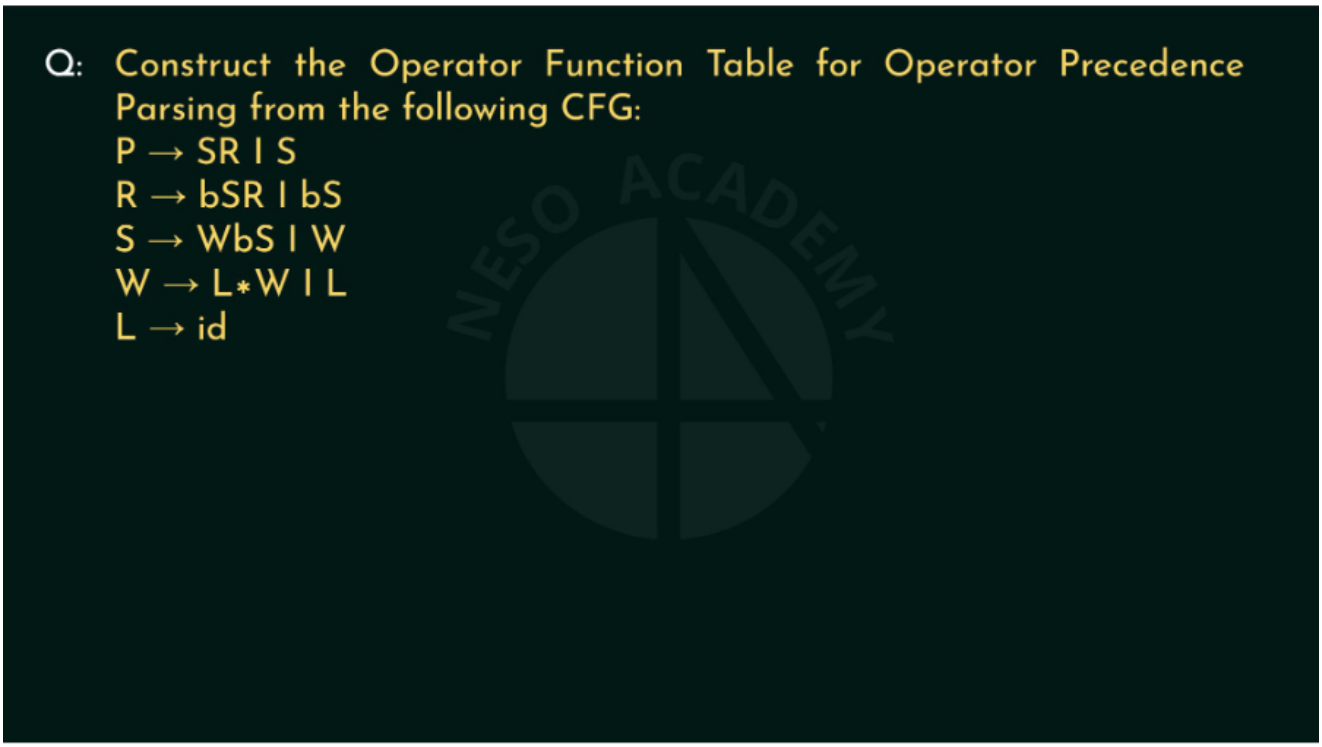
Construction of Operator Function Table:#terminals: n nnTable Size: O(n²)
fid+$g*42400135Table Size: O(2n)

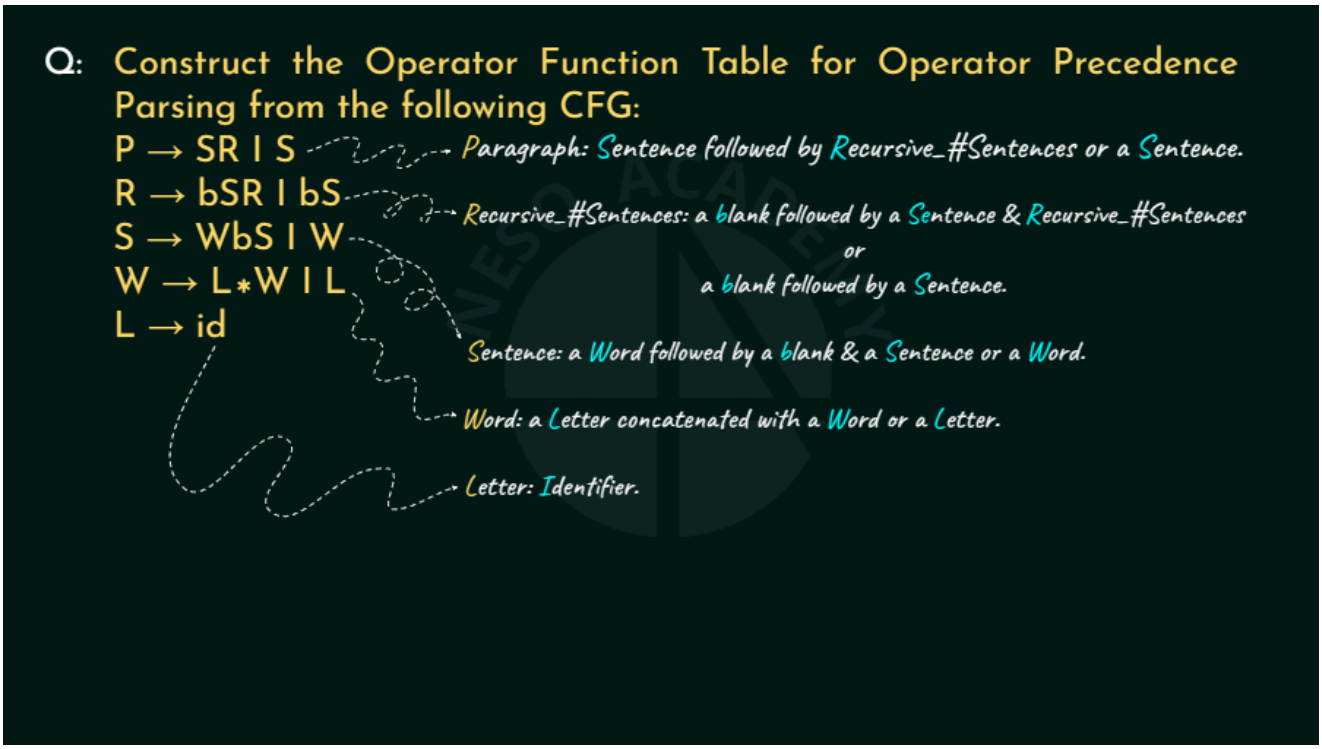


Compiler DesignOperator Precedence Parsers - Solved Problem

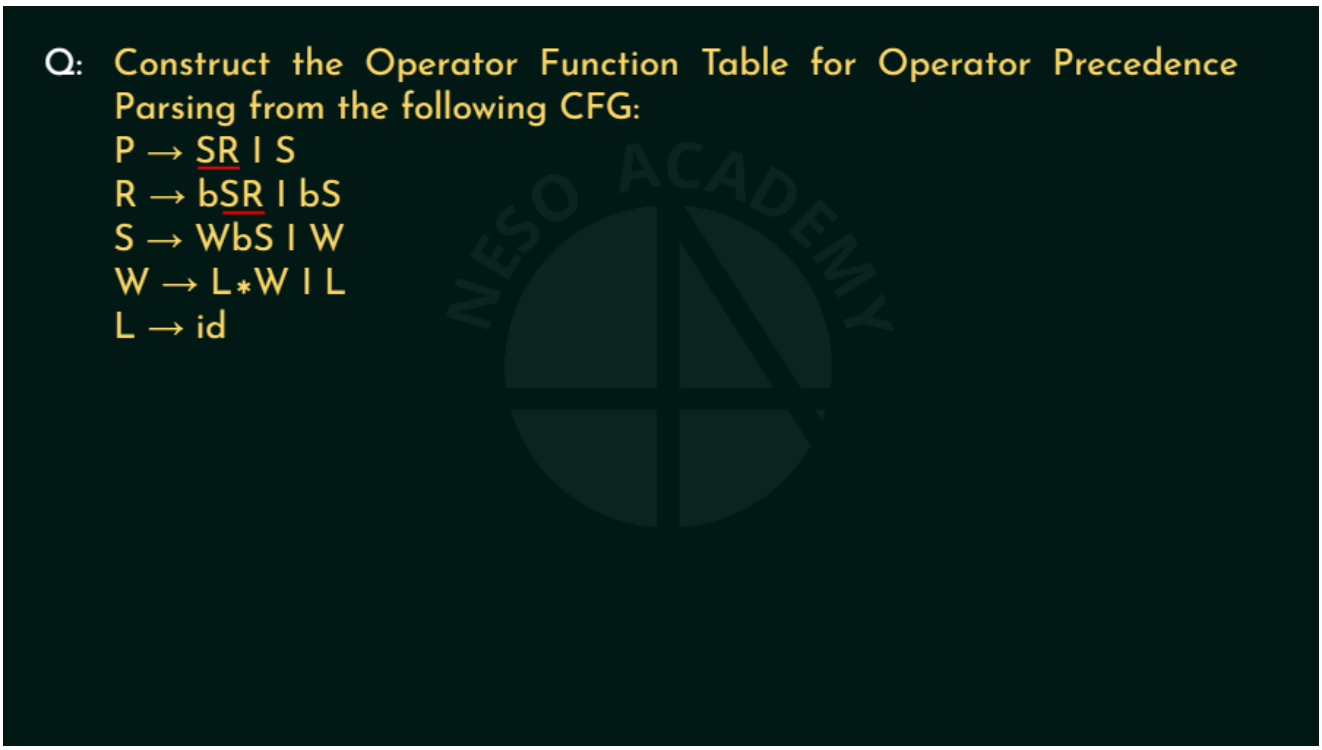


Outcome☆Construction of Operator Function Table for a given CFG.

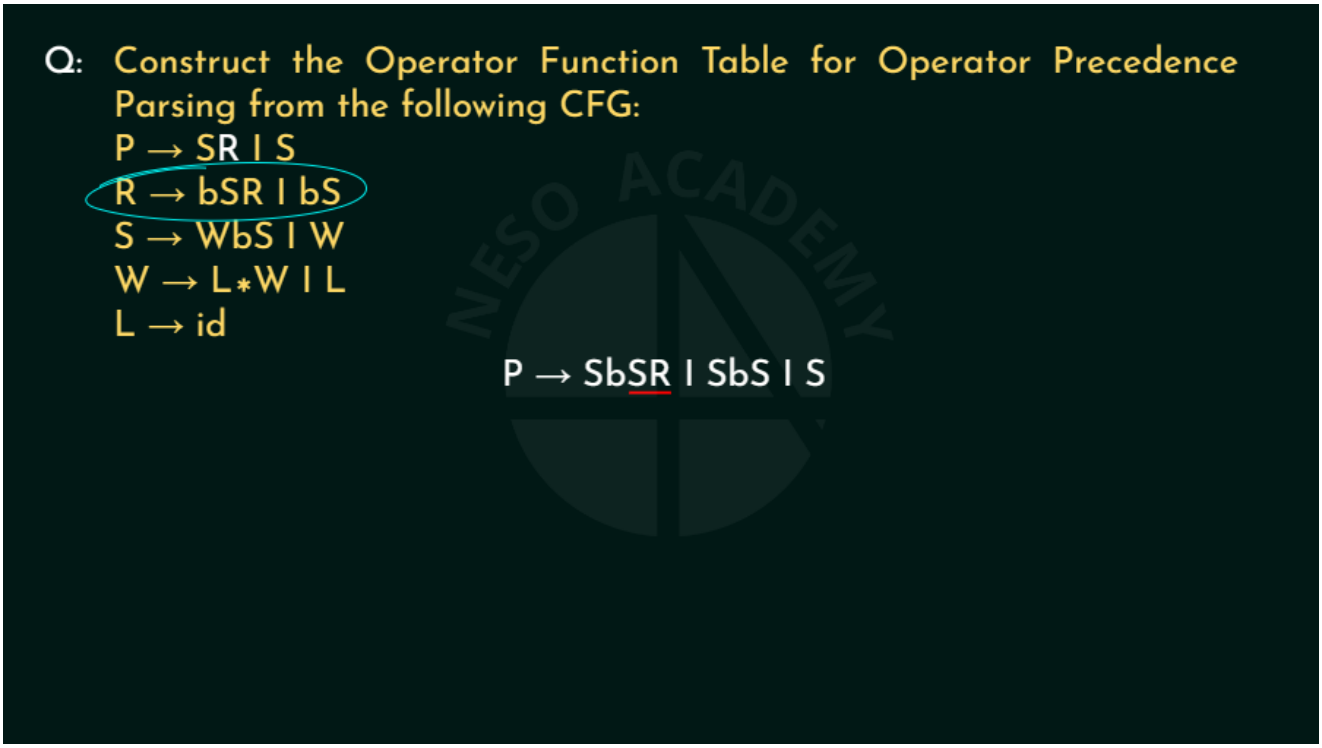




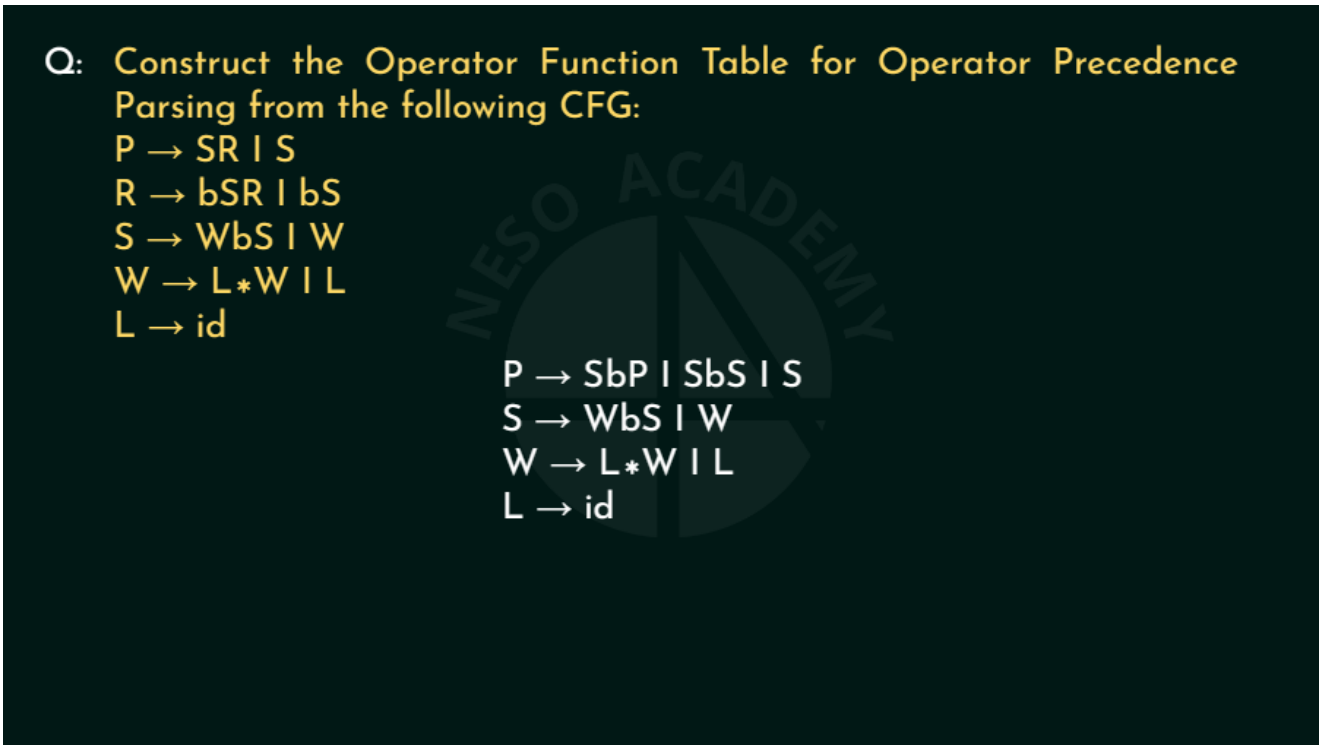
Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:P → SR I S R → bSR I bSS → WbS I WW → L+W I LL → id*Paragraph: Sentence followed by Recursive_#Sentences or a Sentence.Recursive_#Sentences: a blank followed by a Sentence & Recursive_#Sentences or a blank followed by a Sentence.Sentence: a Word followed by a blank & a Sentence or a Word.Word: a Letter concatenated with a Word or a Letter.Letter: Identifier.



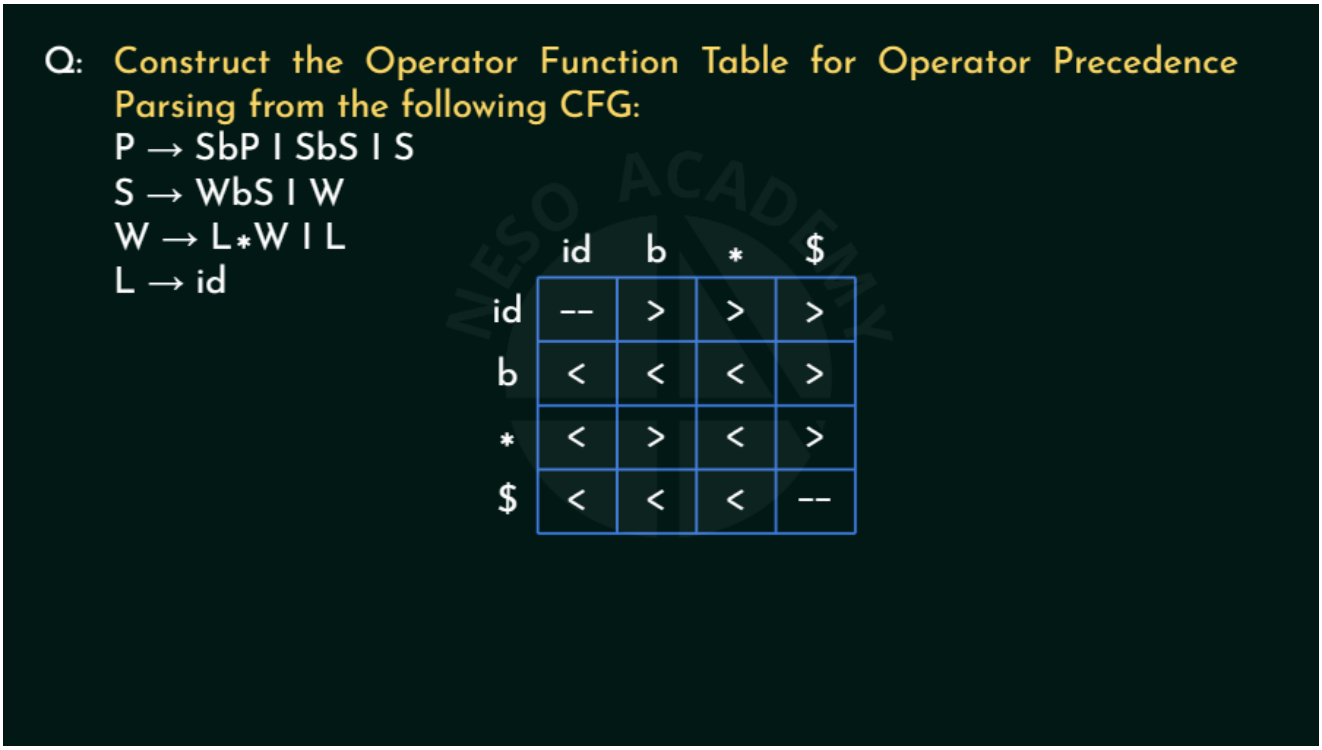
Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:P → SR I S R → bSR I bSS → WbS I WW → L+W I LL → id*



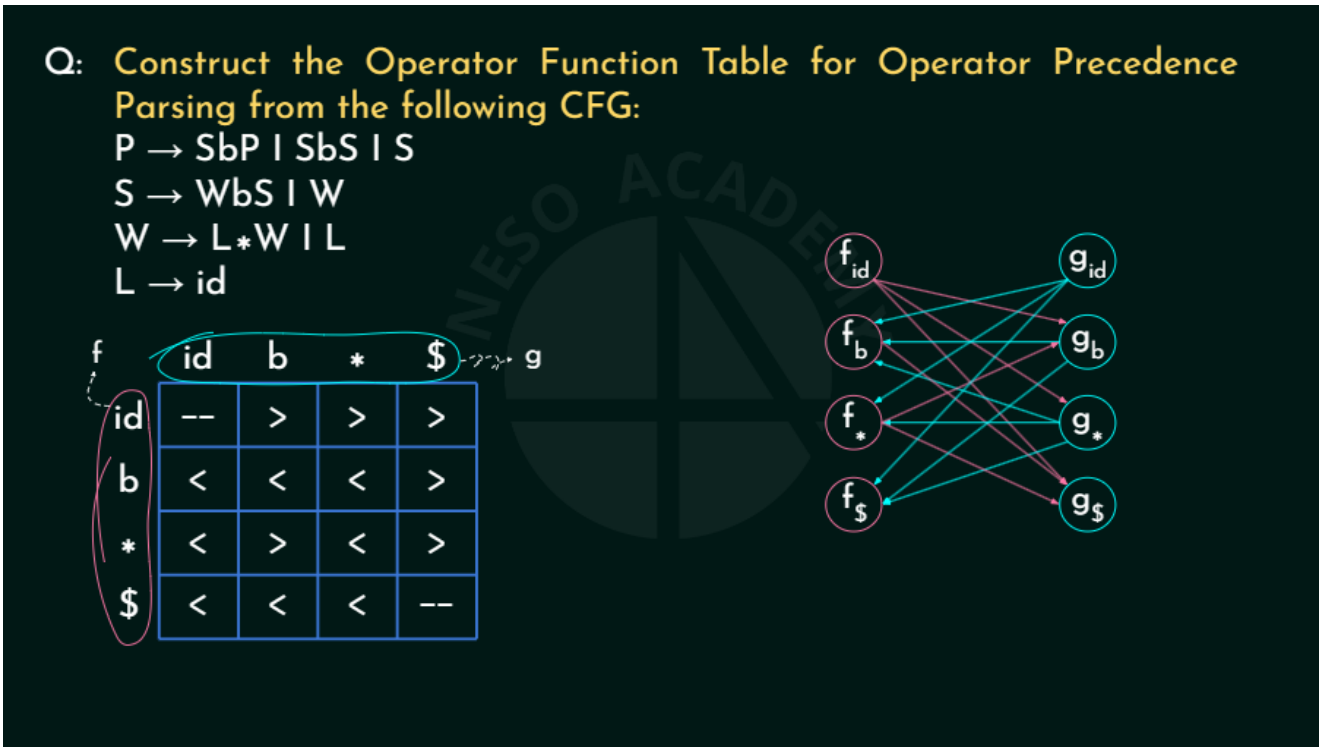
Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:P → SR I S R → bSR I bSS → WbS I WW → L+W I LL → id*P→ SbSRI SbSI S



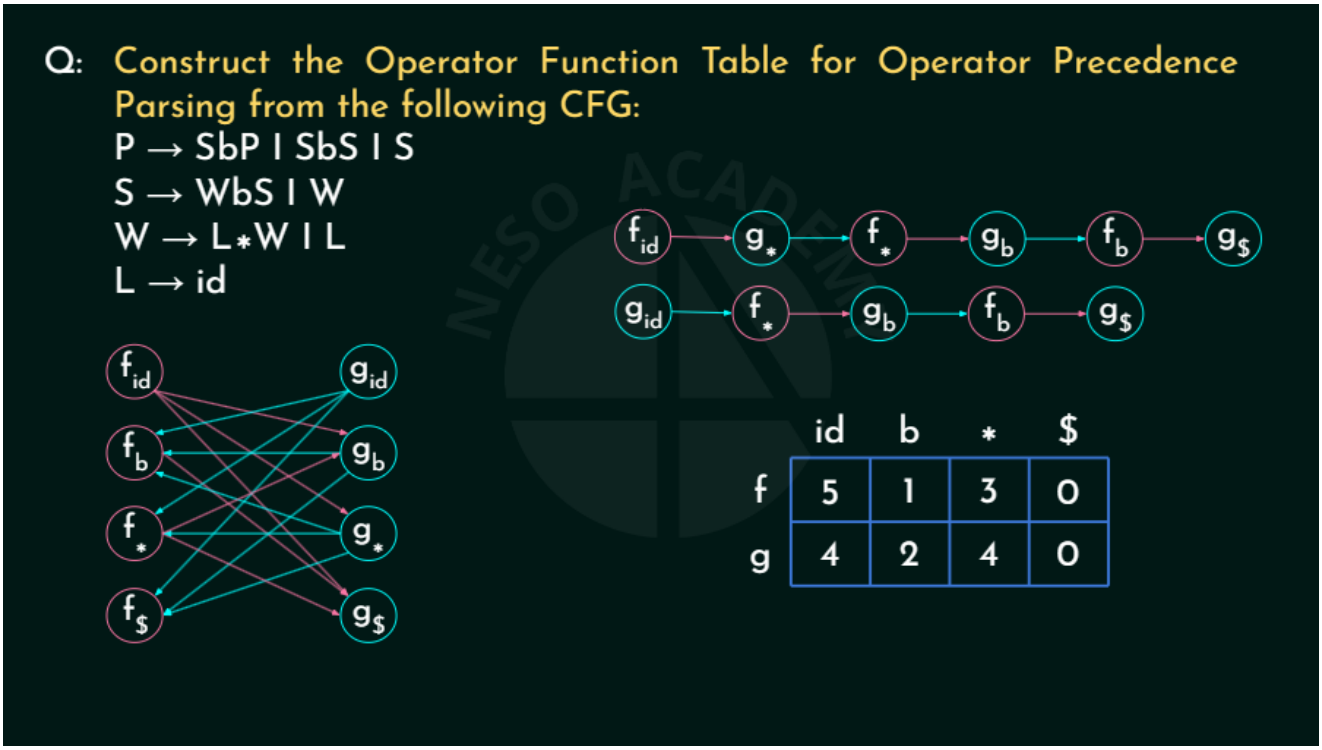
Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:P → SR I S R → bSR I bSS → WbS I WW → L+W I LL → id*P→ SbPI SbSI SS → WbS I WW → L+W I L*L → id



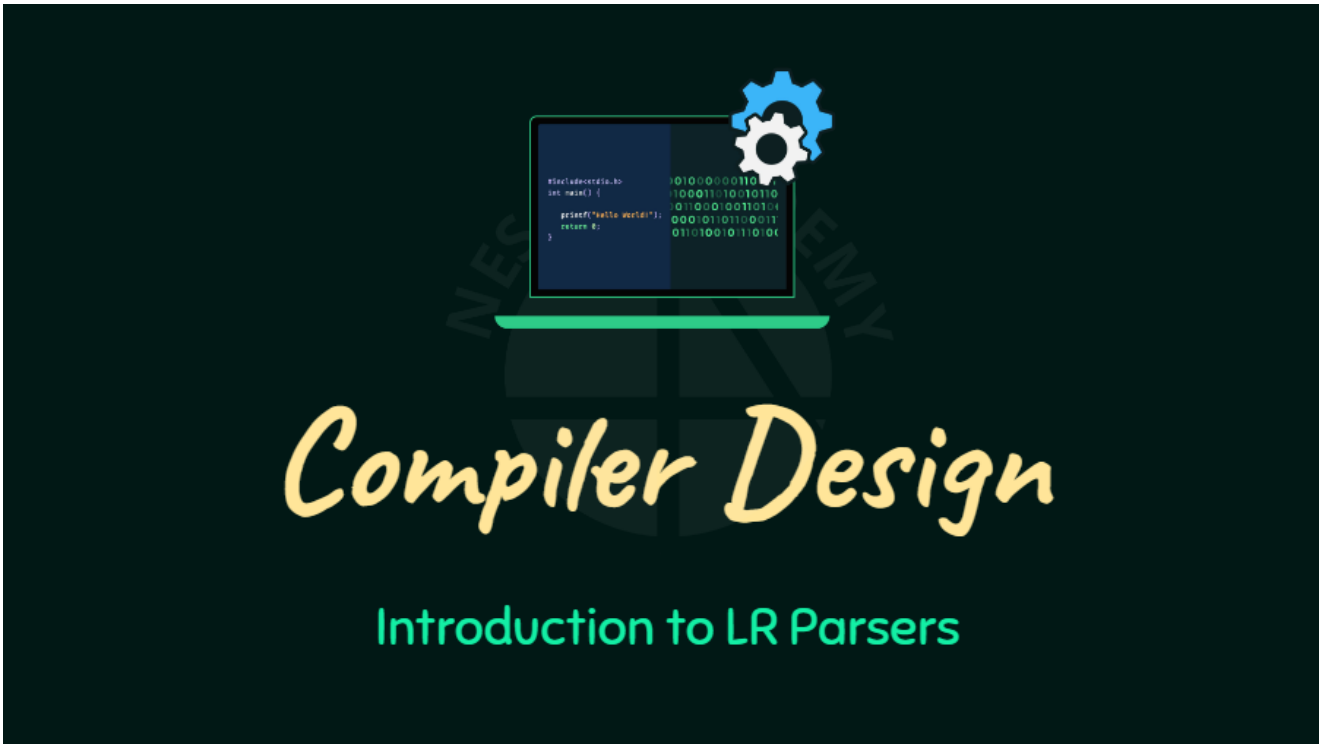
Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:W → L+W I LP→ SbPI SbSI SS → WbS I W*L → idididb$b$**-->>>><<<<><><<<--



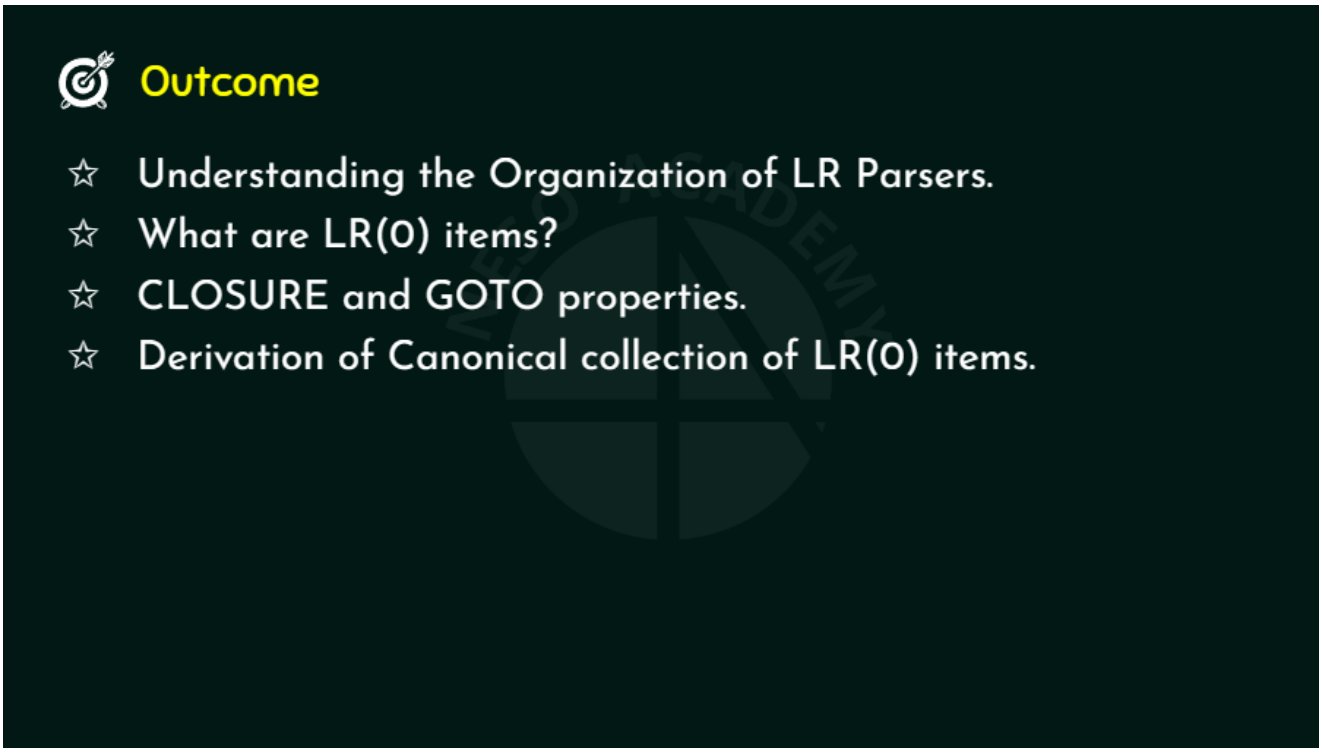
Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:W → L+W I LP→ SbPI SbSI SS → WbS I W*L → idididb$b$**-->>>><<<<><><<<-- fidfbgbgidf+*f$g+*g$g f



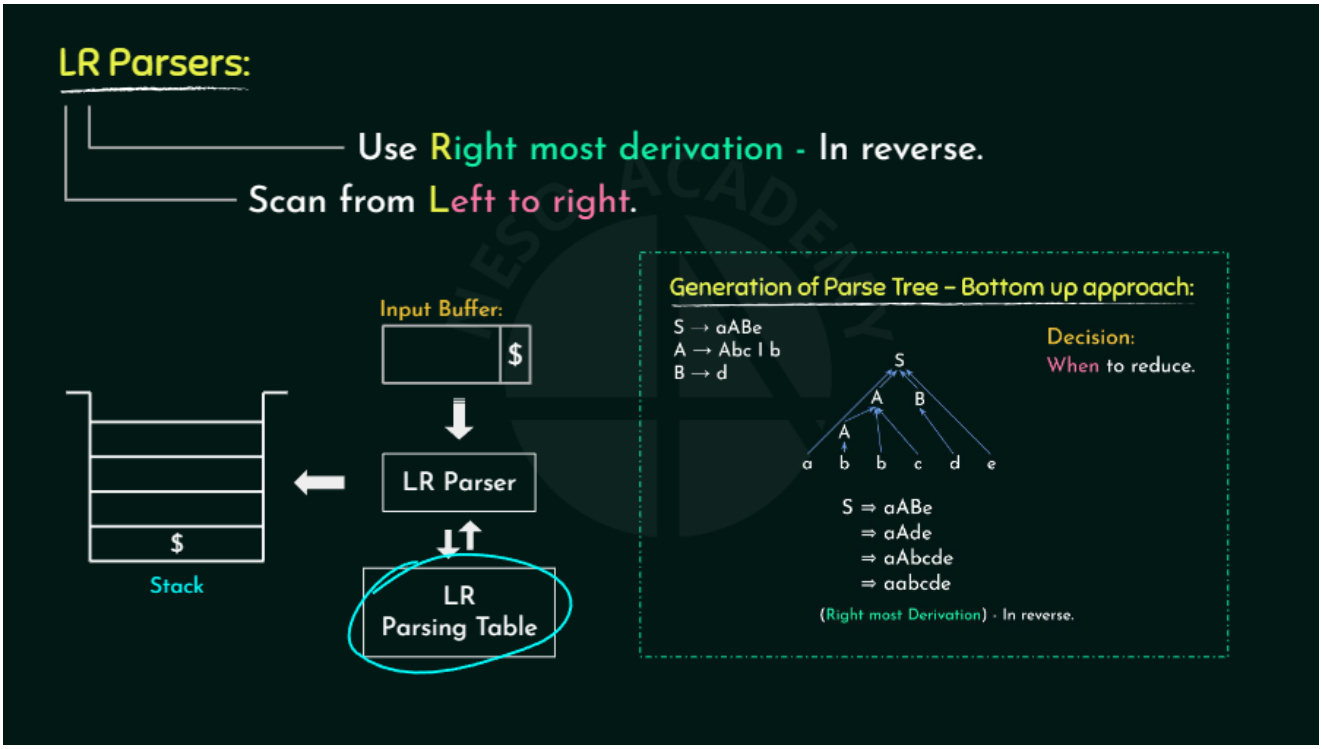
fidb$g*Q:Construct the Operator Function Table for Operator Precedence Parsing from the following CFG:W → L+W I LP→ SbPI SbSI SS → WbS I W*L → idfidfbgbgidf+*f$g+*g$fidg+*gbfbgidf+*f+*g$gbfbg$51300244



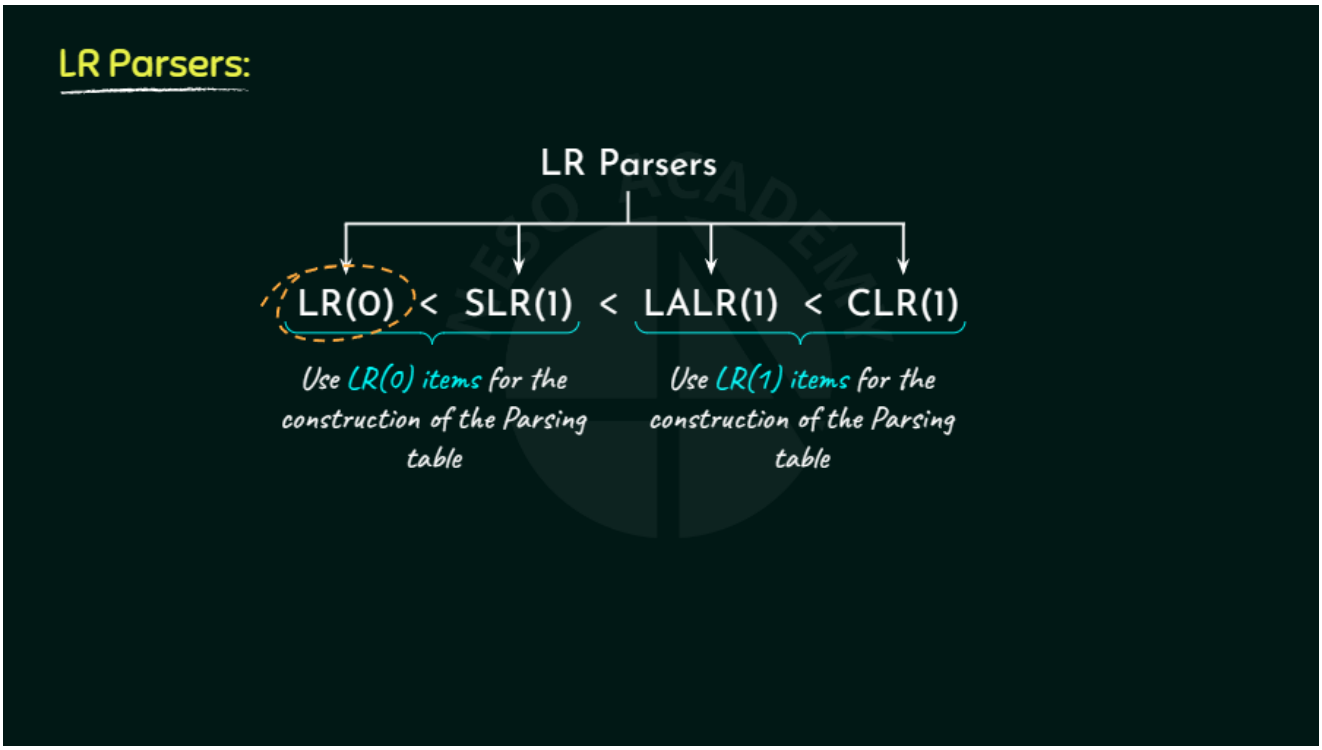
Compiler Design Introduction to LR Parsers



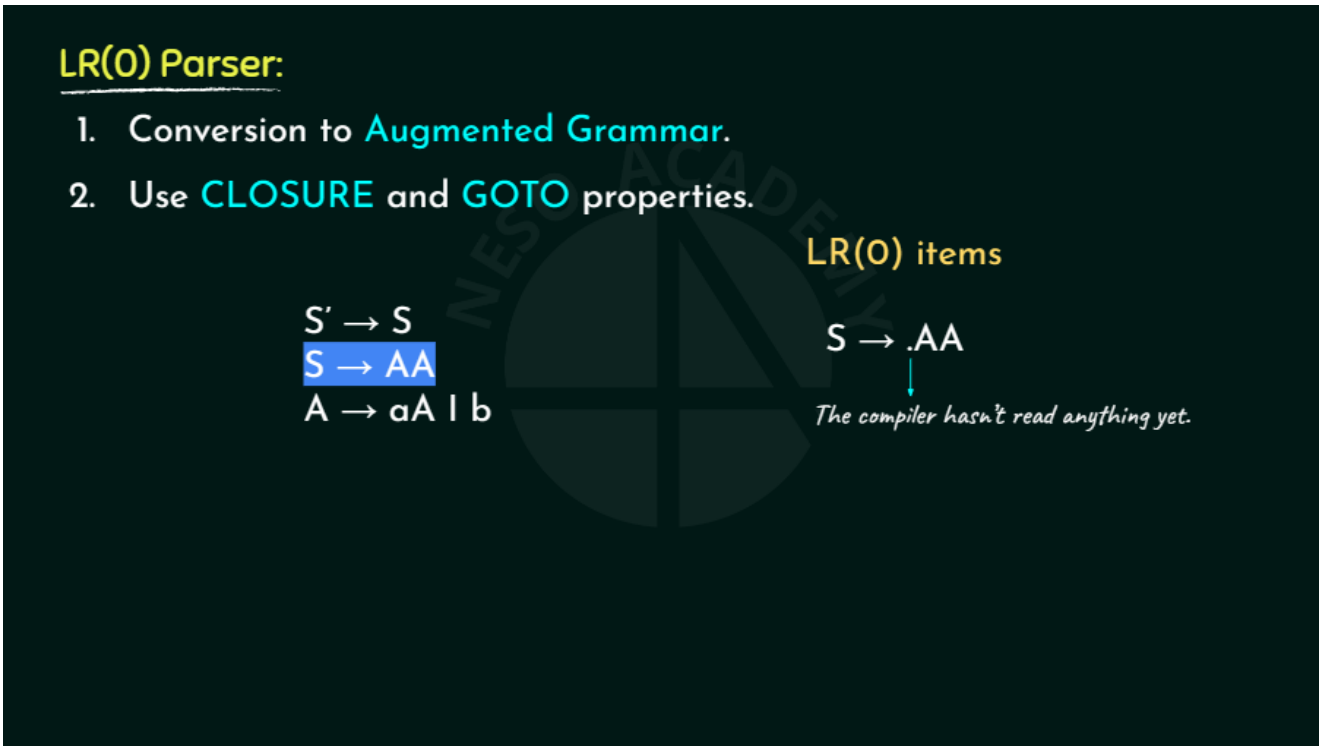
Outcome☆Understanding the Organization of LR Parsers.☆What are LR(0) items? ☆CLOSURE and GOTO properties.☆Derivation of Canonical collection of LR(0) items.



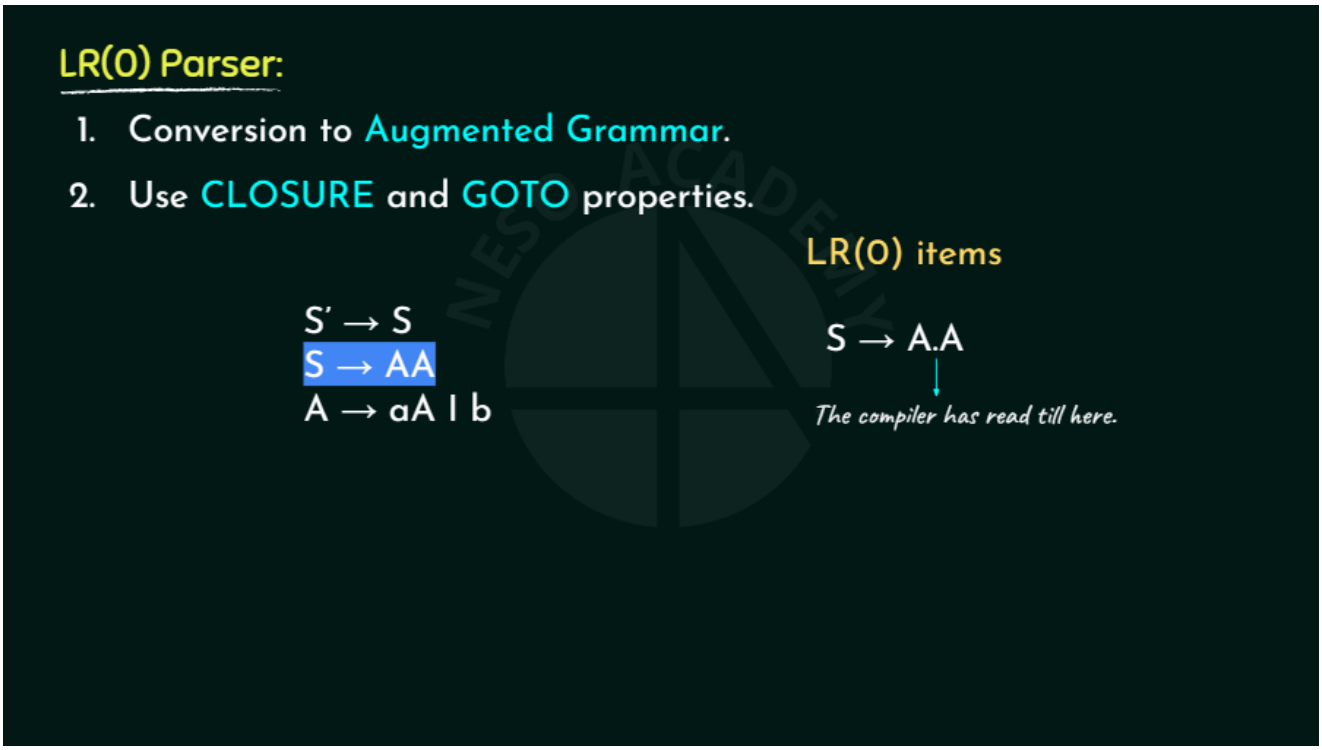
LR Parsers:$$LR ParserLR Parsing TableInput Buffer:StackScan from Left to right.Use Right most derivation - In reverse.



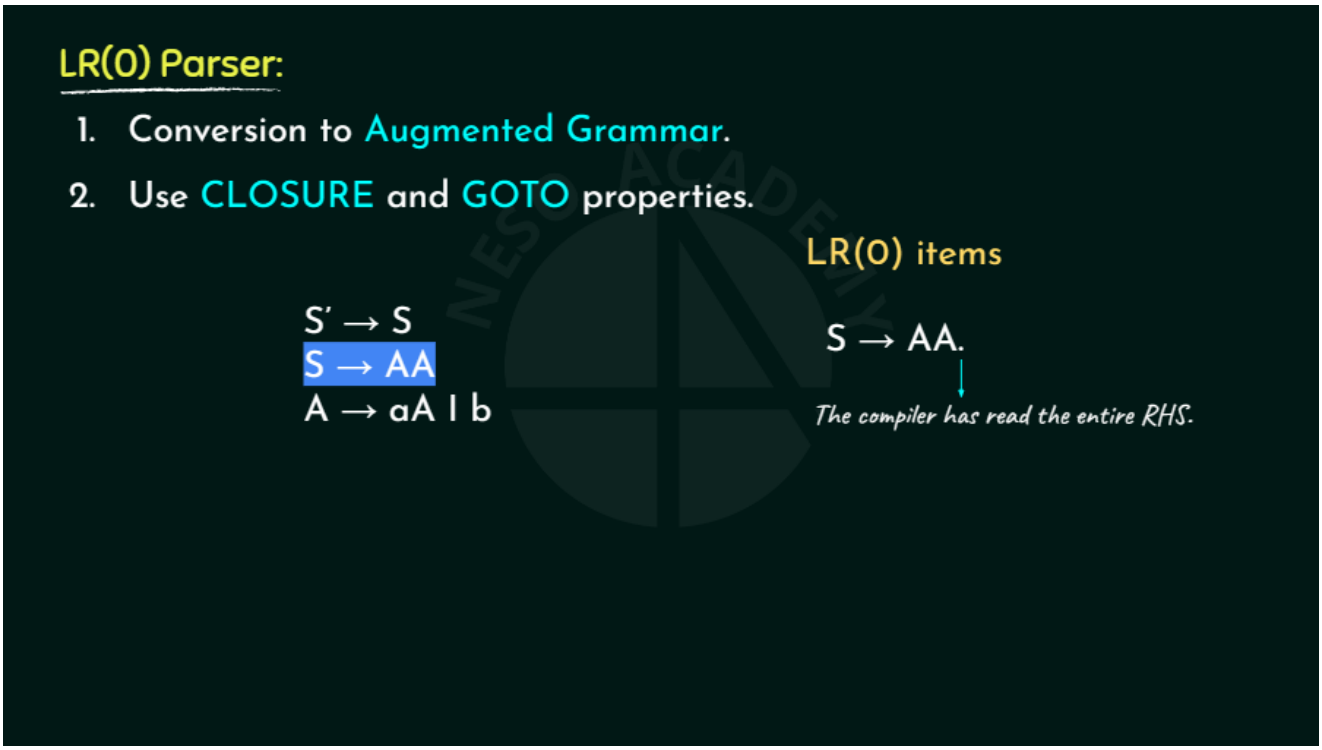
LR Parsers:LR ParsersLR(0)CLR(1)LALR(1)SLR(1)<<<Use LR(0) items for the construction of the Parsing tableUse LR(1) items for the construction of the Parsing table



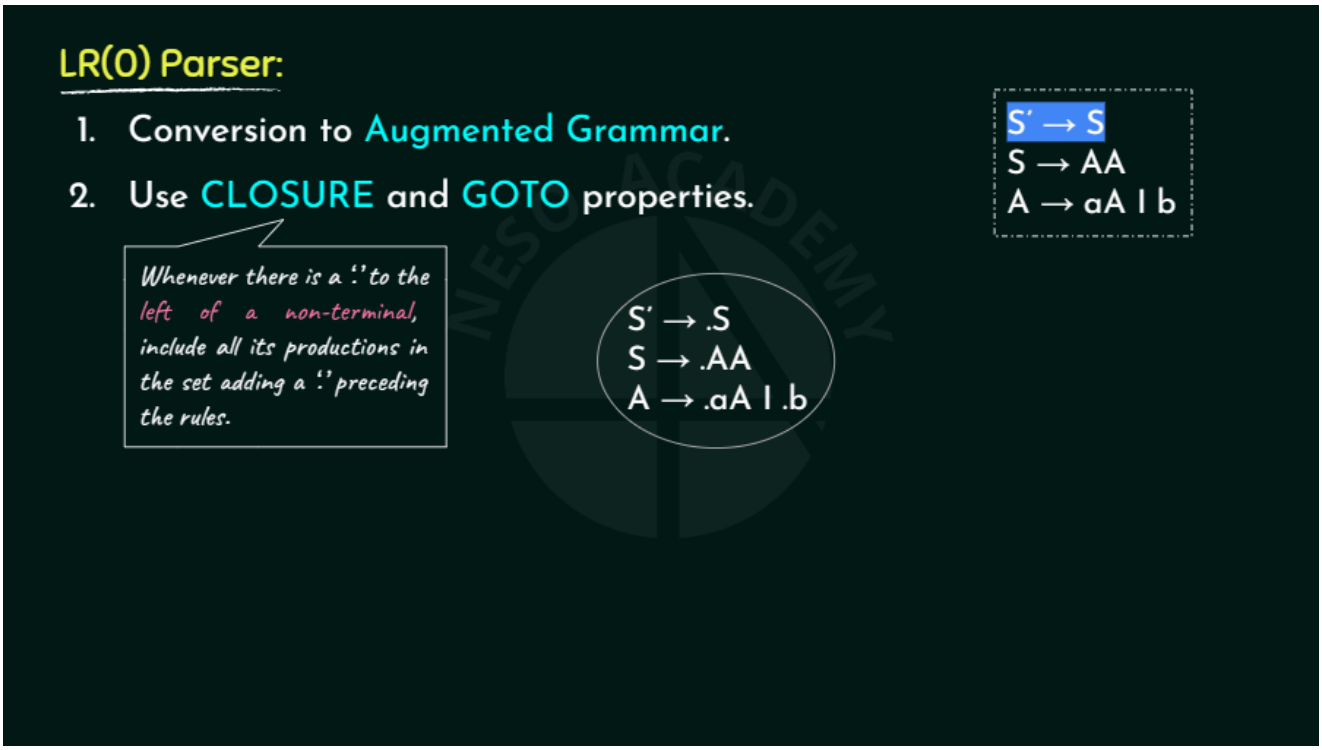
1.Conversion to Augmented Grammar.2.Use CLOSURE and GOTO properties.S' → SS → AA A → aA I bLR(0) itemsLR(0) Parser:S → .AAThe compiler hasn't read anything yet.



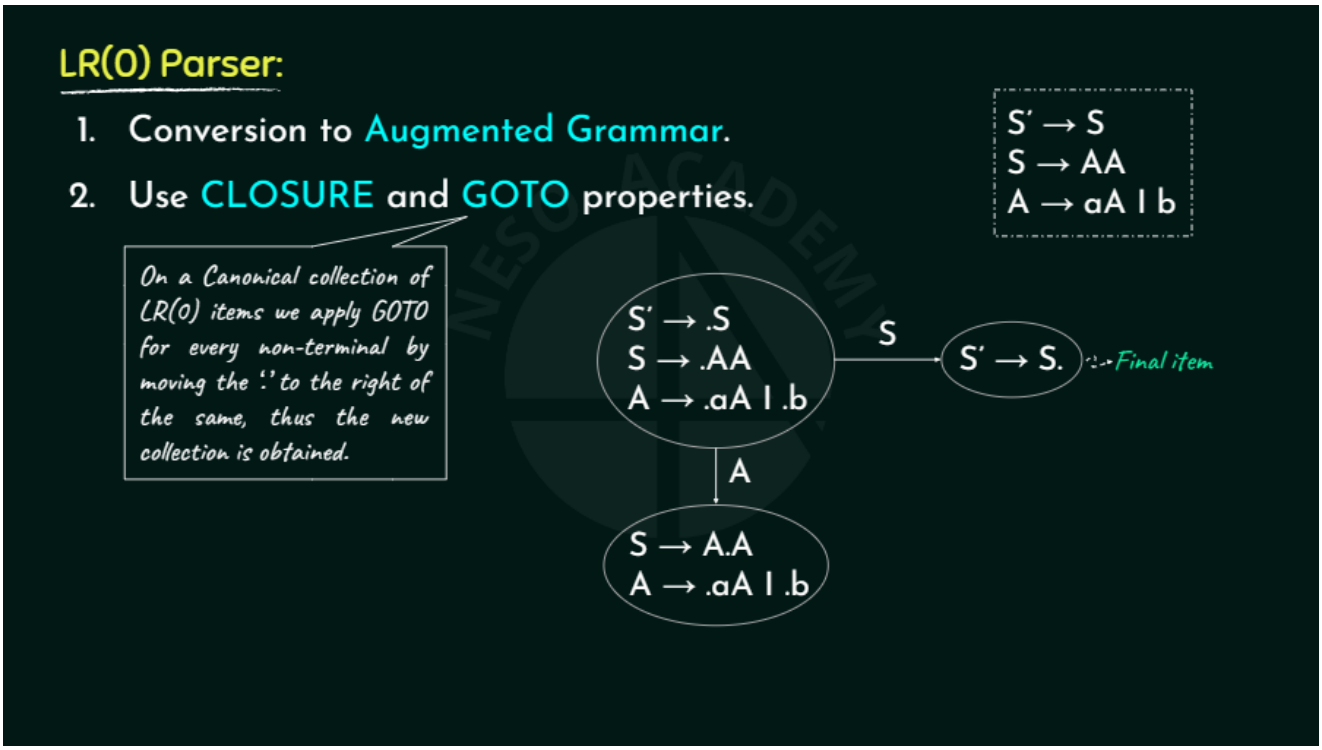
1.Conversion to Augmented Grammar.2.Use CLOSURE and GOTO properties.S' → SS → AA A → aA I bLR(0) itemsLR(0) Parser:S → A.AThe compiler has read till here.



1.Conversion to Augmented Grammar.2.Use CLOSURE and GOTO properties.S' → SS → AA A → aA I bLR(0) itemsLR(0) Parser:S → AA.The compiler has read the entire RHS.

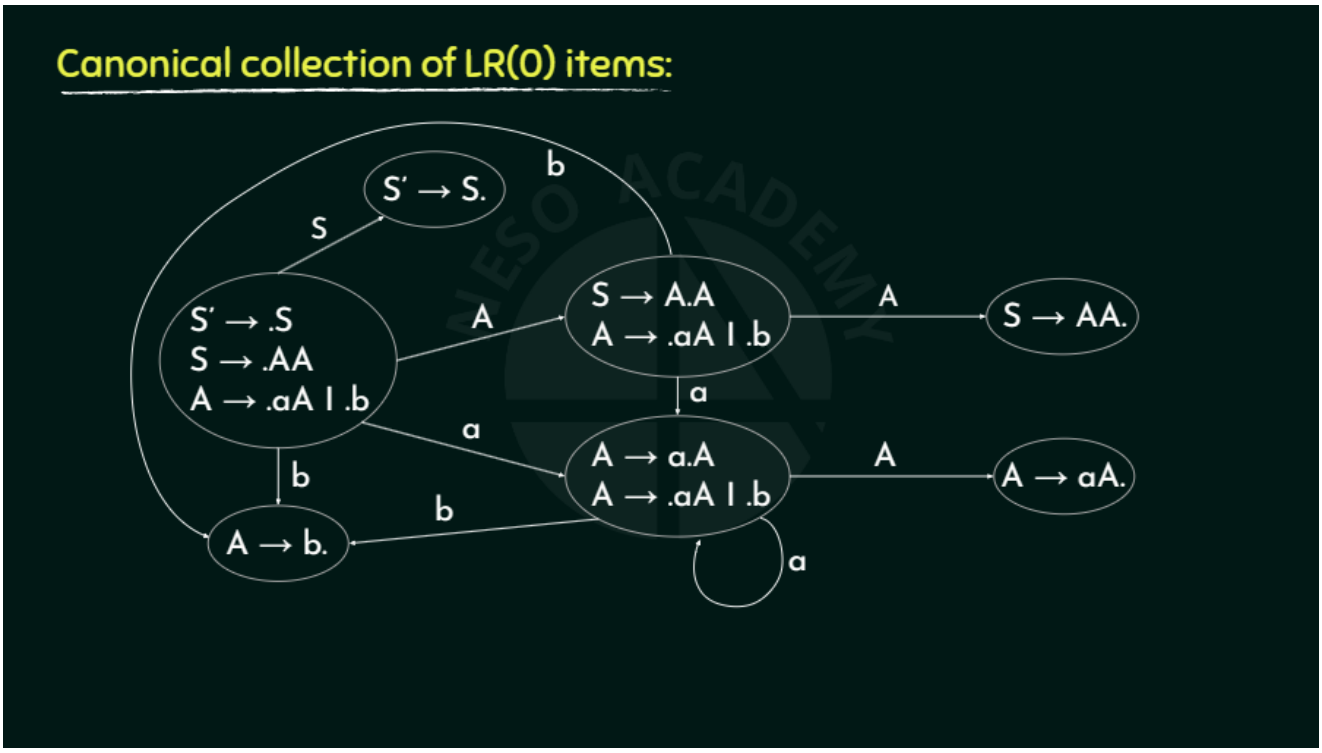


1.Conversion to Augmented Grammar.2.Use CLOSURE and GOTO properties.S' → SS → AA A → aA I bLR(0) Parser:Whenever there is a '.' to the left of a non-terminal, include all its productions in the set adding a '.' preceding the rules. S' → .SS → .AA A → .aA I .b



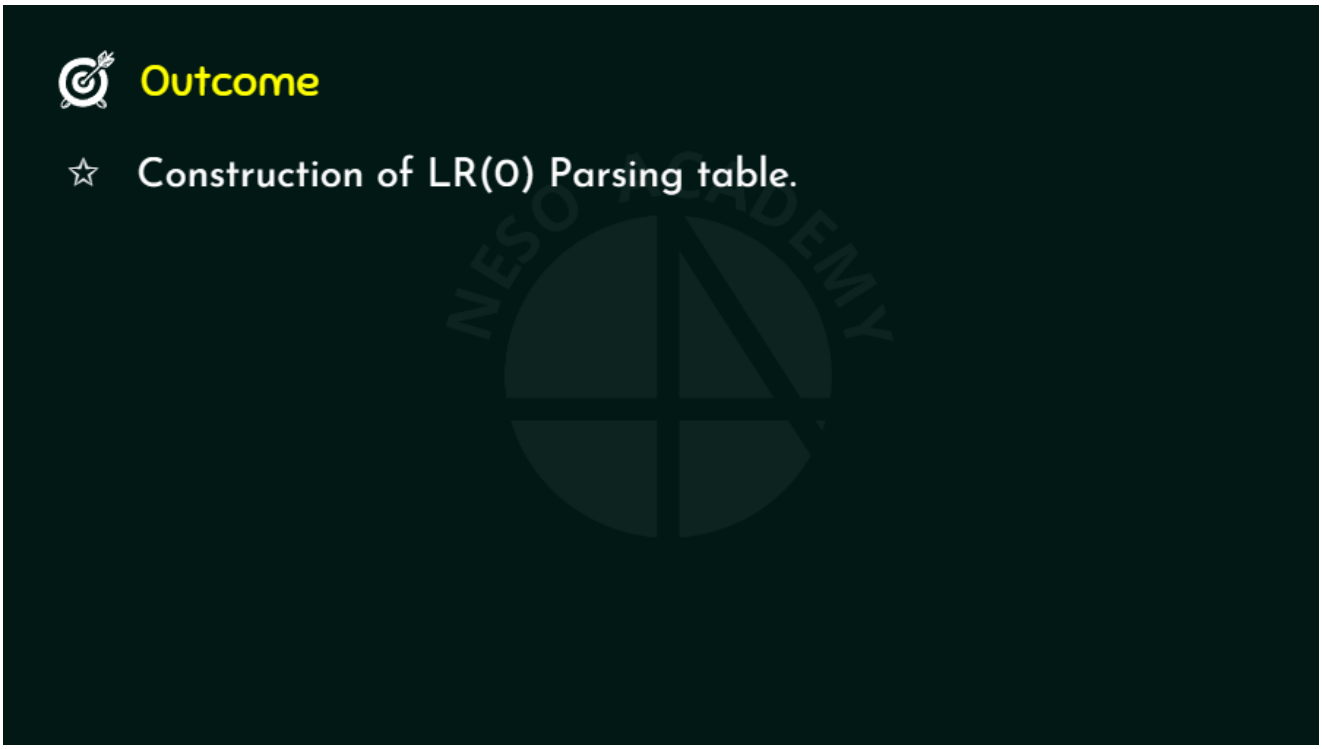
1.Conversion to Augmented Grammar.2.Use CLOSURE and GOTO properties.S' → SS → AA A → aA I bLR(0) Parser:On a Canonical collection of LR(0) items we apply GOTO for every non-terminal by moving the '.' to the right of the same, thus the new collection is

obtained. S’ → .SS → .AA A → .aA I .bS’ → S.SFinal itemS → A.A A → .aA I .bA

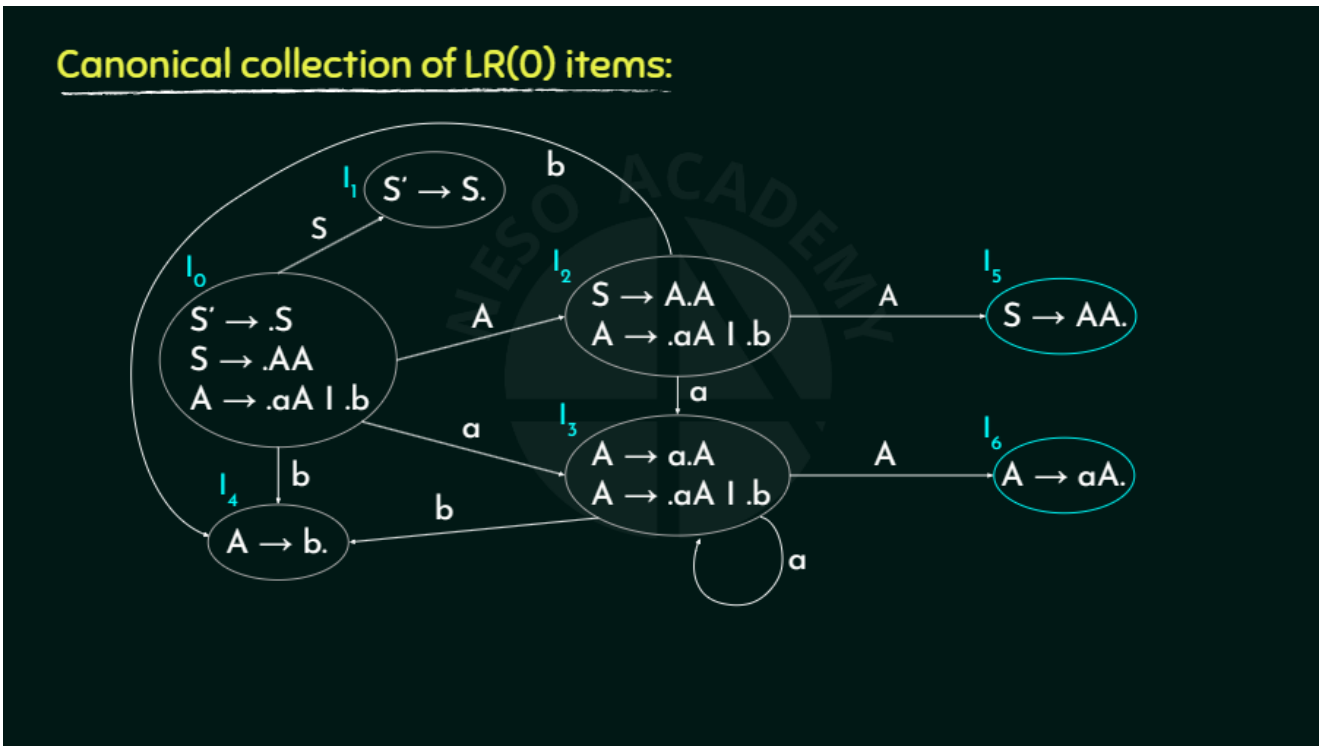


Canonical collection of LR(0) items:A → b.A → a.A A → .aA I .bS → AA.A → aA.S’ → .SS → .AA A → .aA I .bS’ → S.SS → A.A A → .aA I .bAAAababba

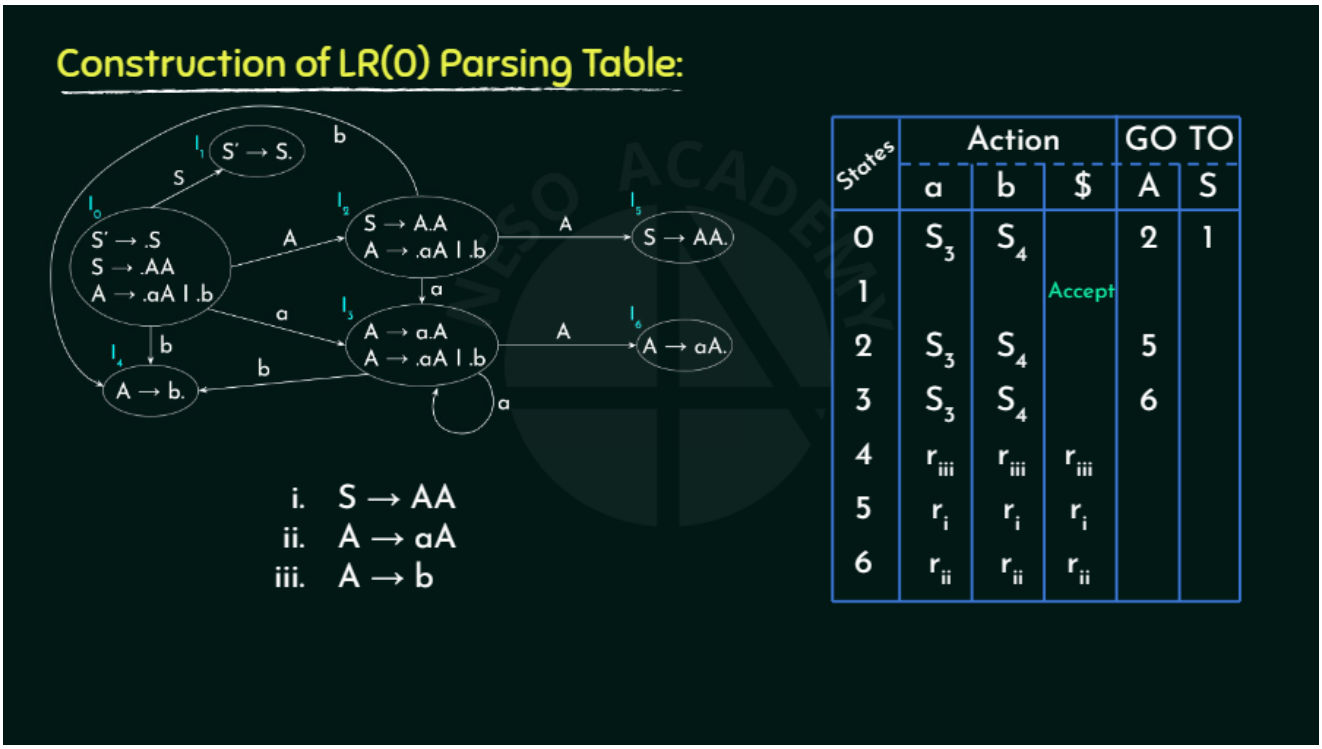
Compiler Design LR(0) Parsers - Part 1



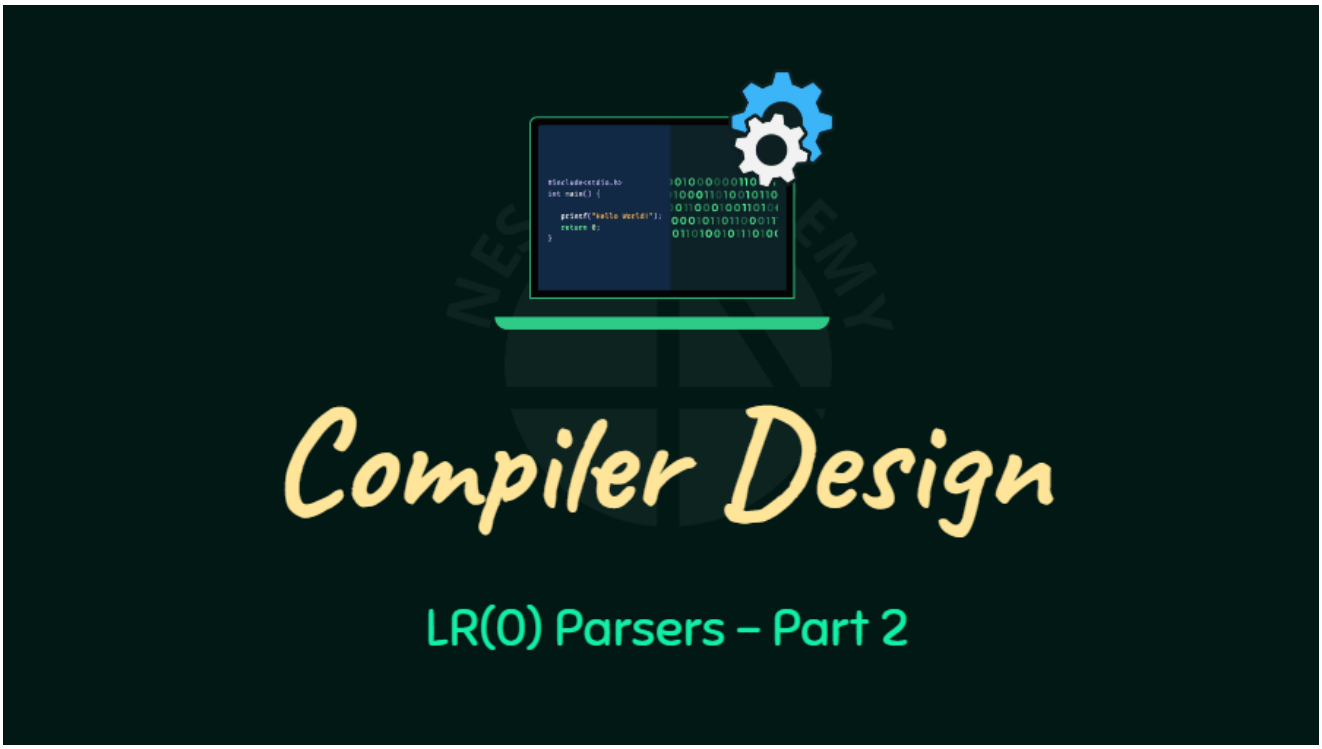
Outcome☆Construction of LR(0) Parsing table.



Canonical collection of LR(0) items:A → b.A → a.A A → .aA I .bS → AA.A → aA.S' → .SS → .AA A → .aA I .bS' → S.SS → A.A A → .aA I .bAAAababbI0I1I2I3I4I5I6a



i.S → AA ii.A → aAiii.A → b1ActionGO TOStatesConstruction of LR(0) Parsing Table:0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriiri56ab$SA



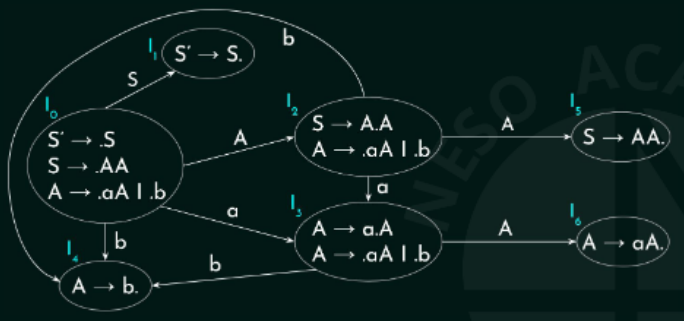
Compiler Design LR(0) Parsers - Part 2

Outcome

☆ LR(0) Parsing Procedure.

Outcome☆LR(0) Parsing Procedure.

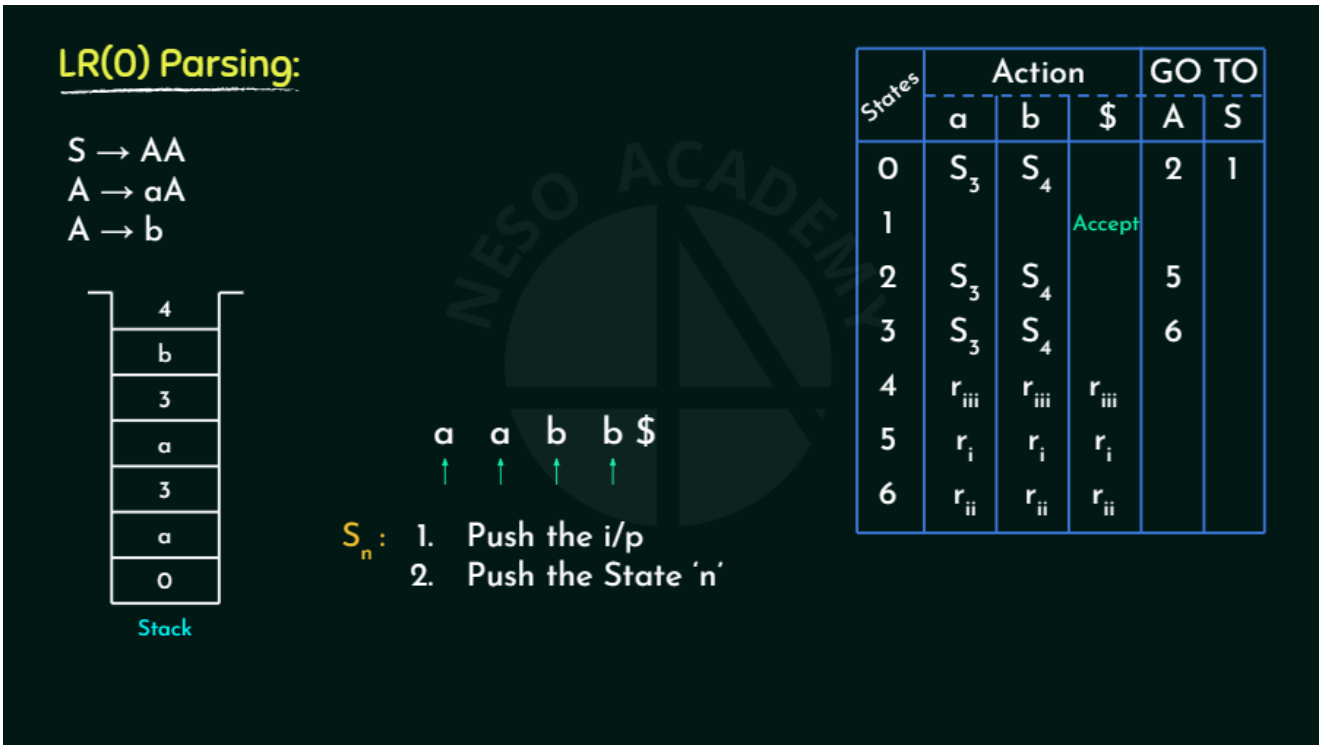
Construction of LR(0) Parsing Table:



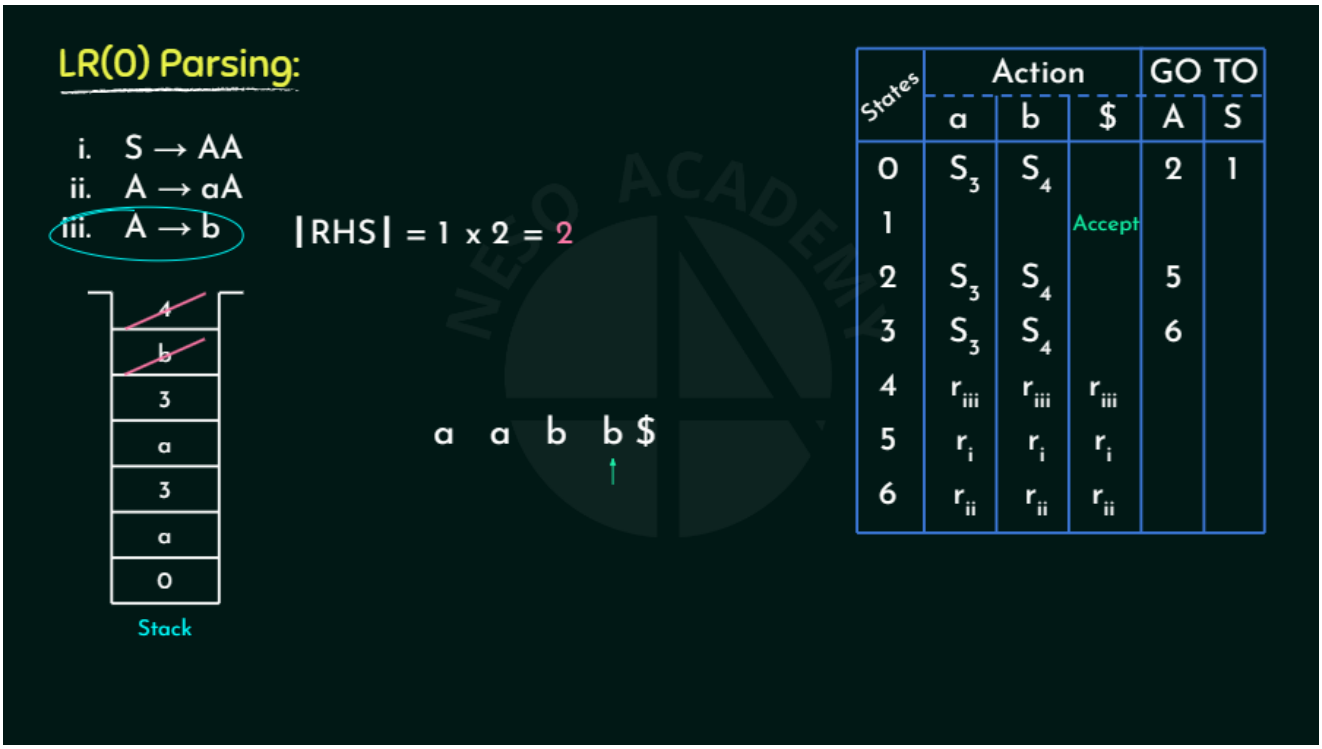
i. $S \rightarrow AA$
ii. $A \rightarrow aA$
iii. $A \rightarrow b$

States	Action			GO TO	
	a	b	$	A	S
0	S_3	S_4		2	1
1			Accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_{iii}	r_{iii}	r_{iii}		
5	r_i	r_i	r_i		
6	r_{ii}	r_{ii}	r_{ii}		

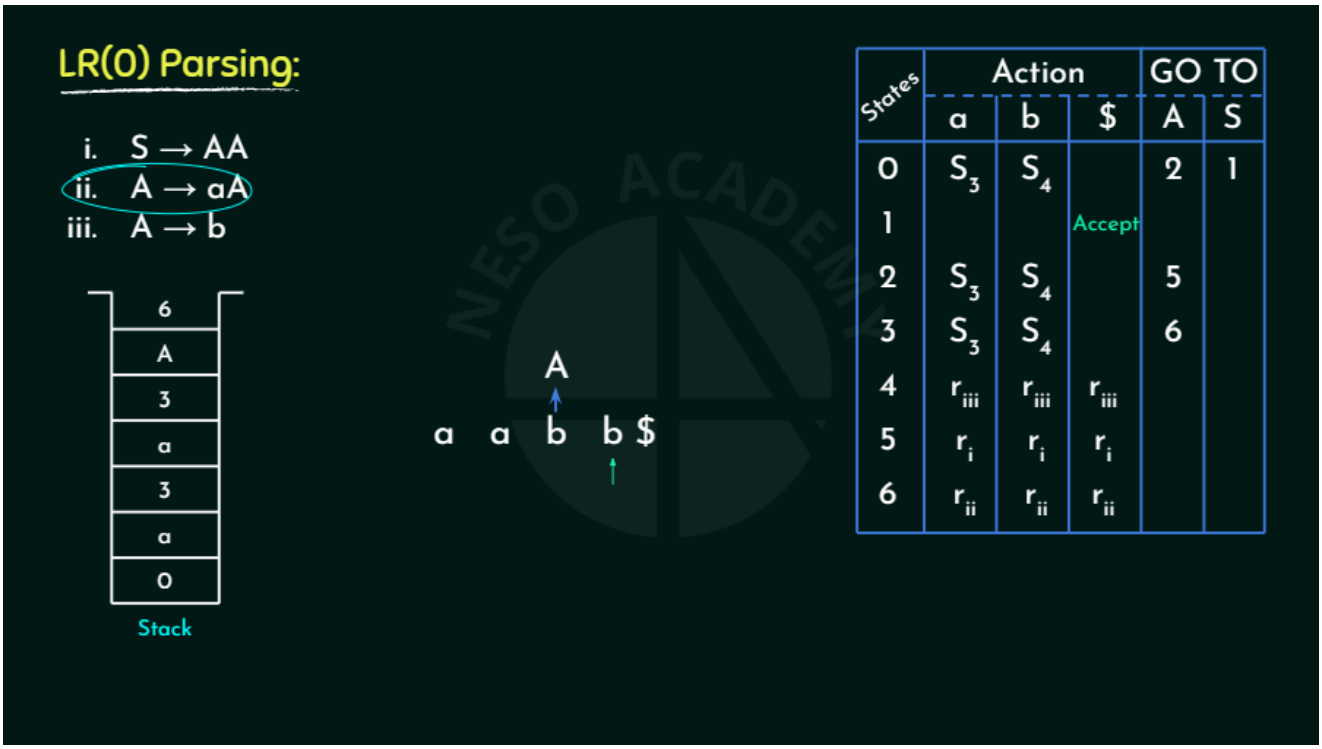
i.S → AA ii.A → aAiii.A → b1ActionGO TOStatesConstruction of LR(0) Parsing Table:0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriii56ab$SA



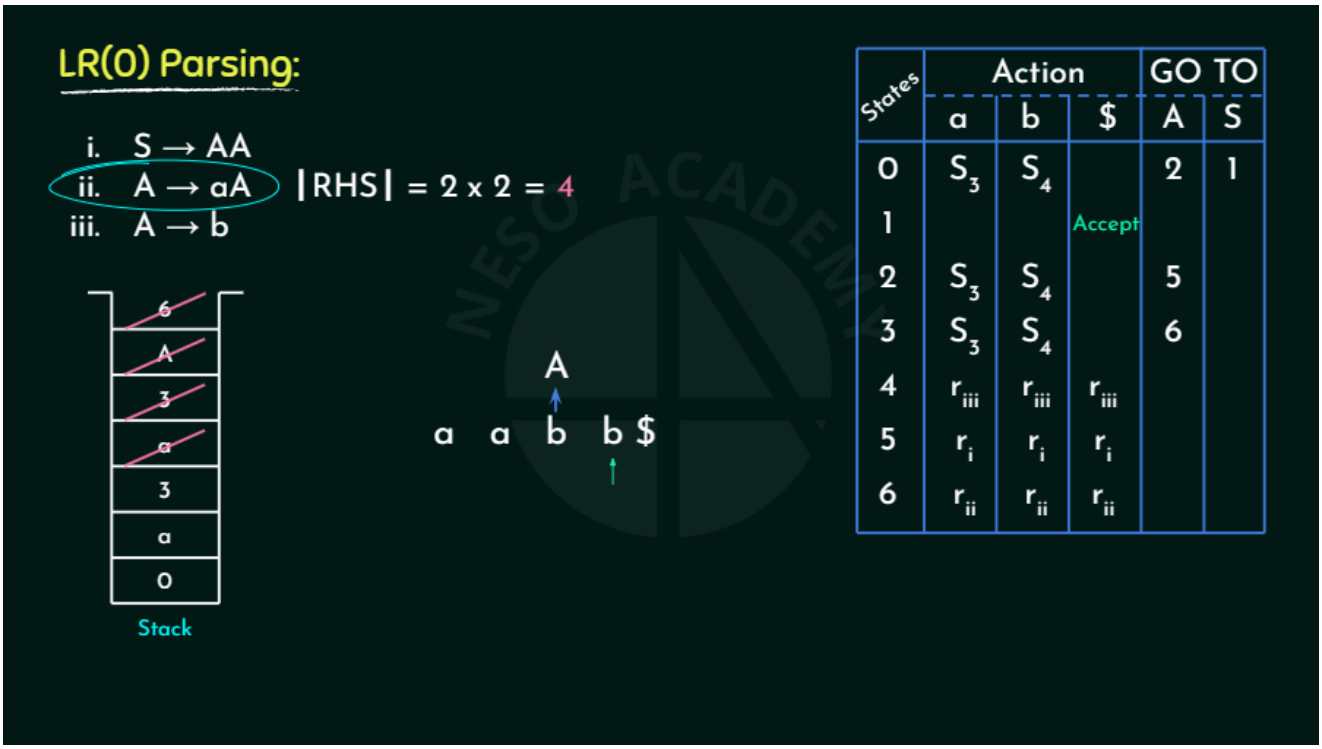
StackS → AAA → aA A → bLR(0) Parsing:1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SAa a b b $a03a34bSn : 1.Push the i/p 2.Push the State 'n'



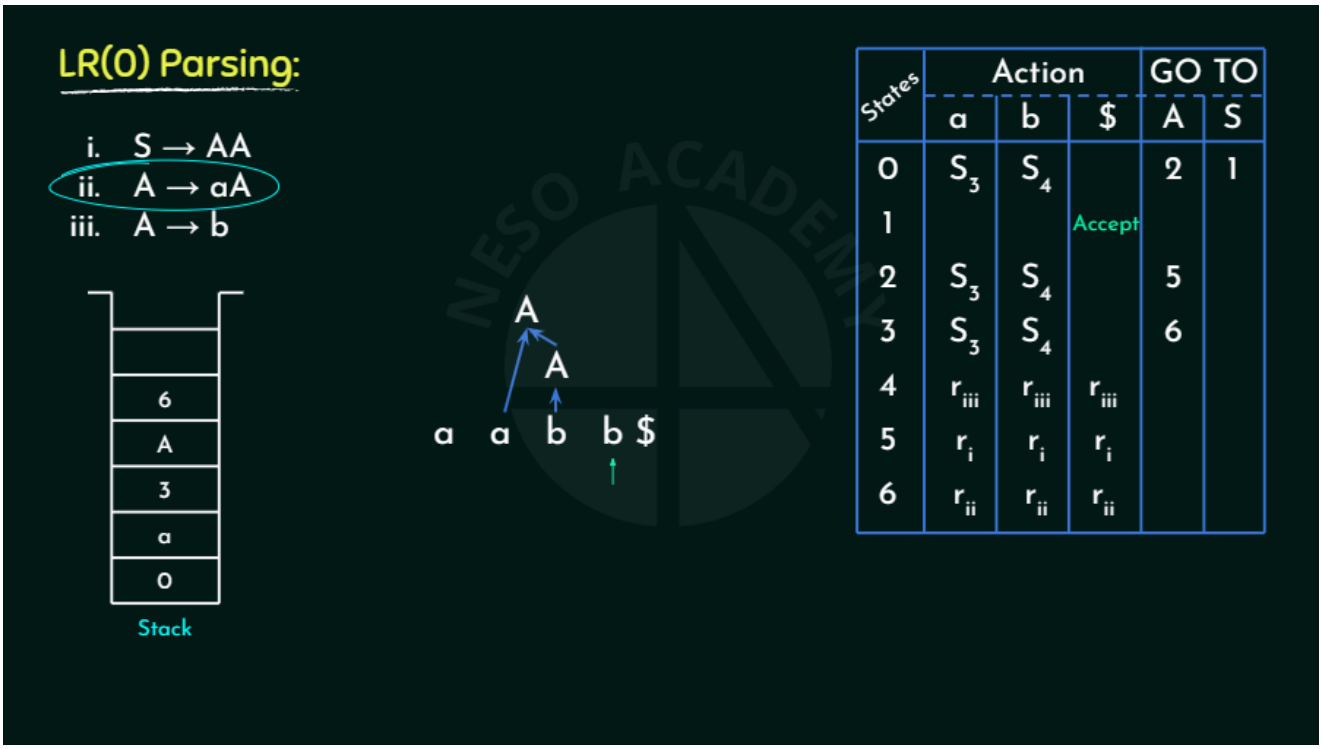
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $a03a34bIRHSIRHS= 1x 2= 21ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



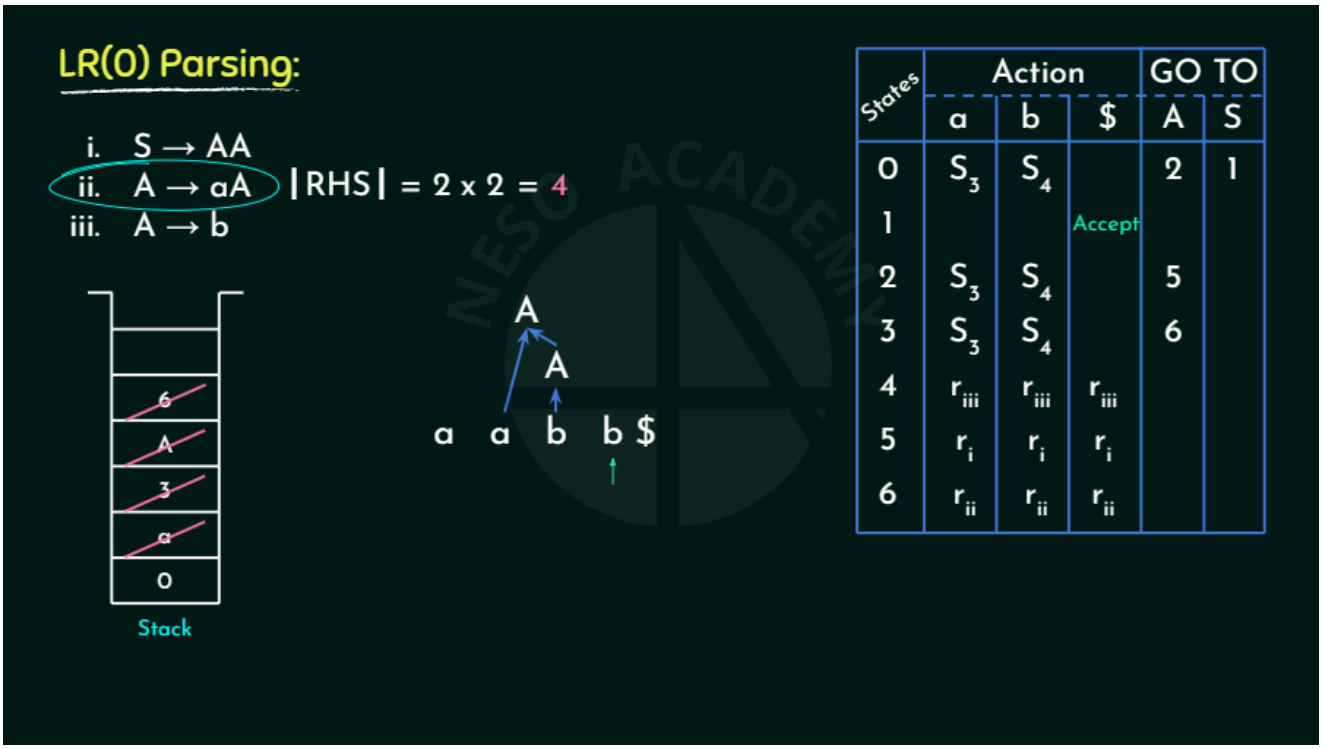
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $a03a3AA61ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



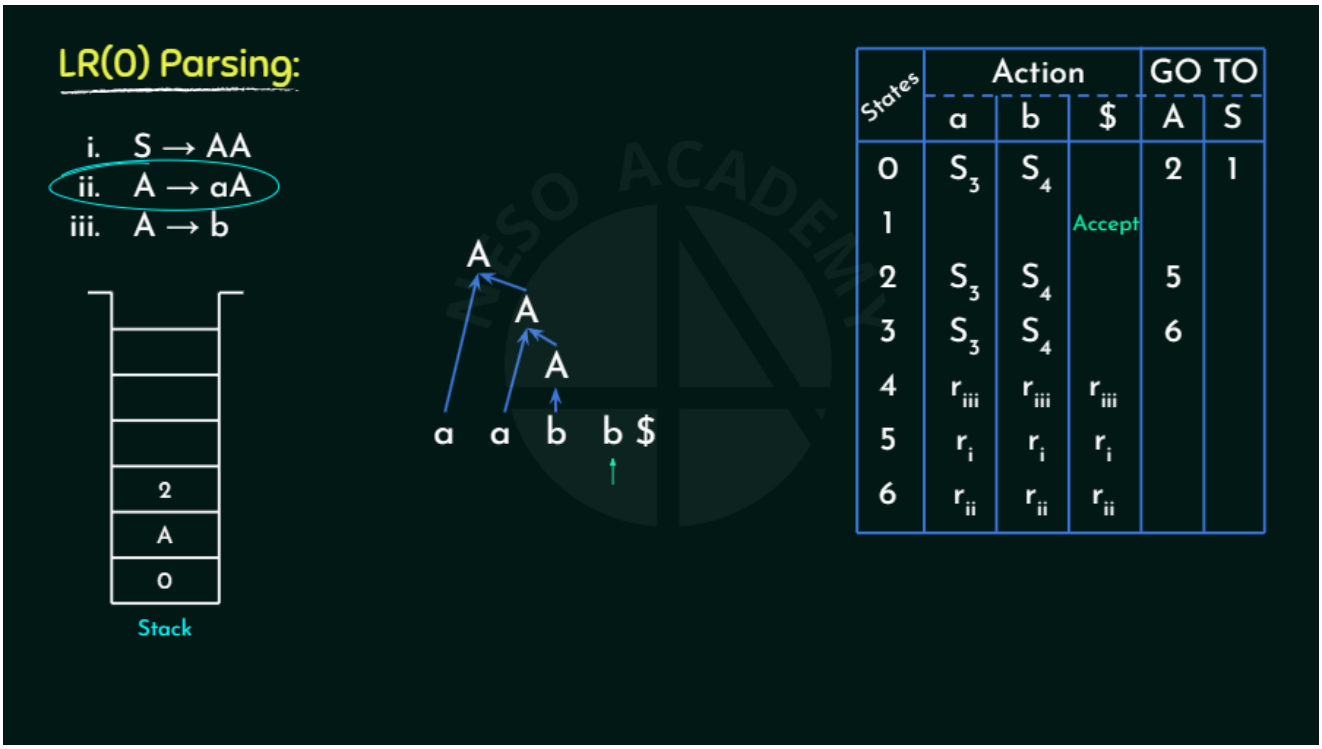
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $a03a3AA6IRHSIRHS= 2x 2= 41ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



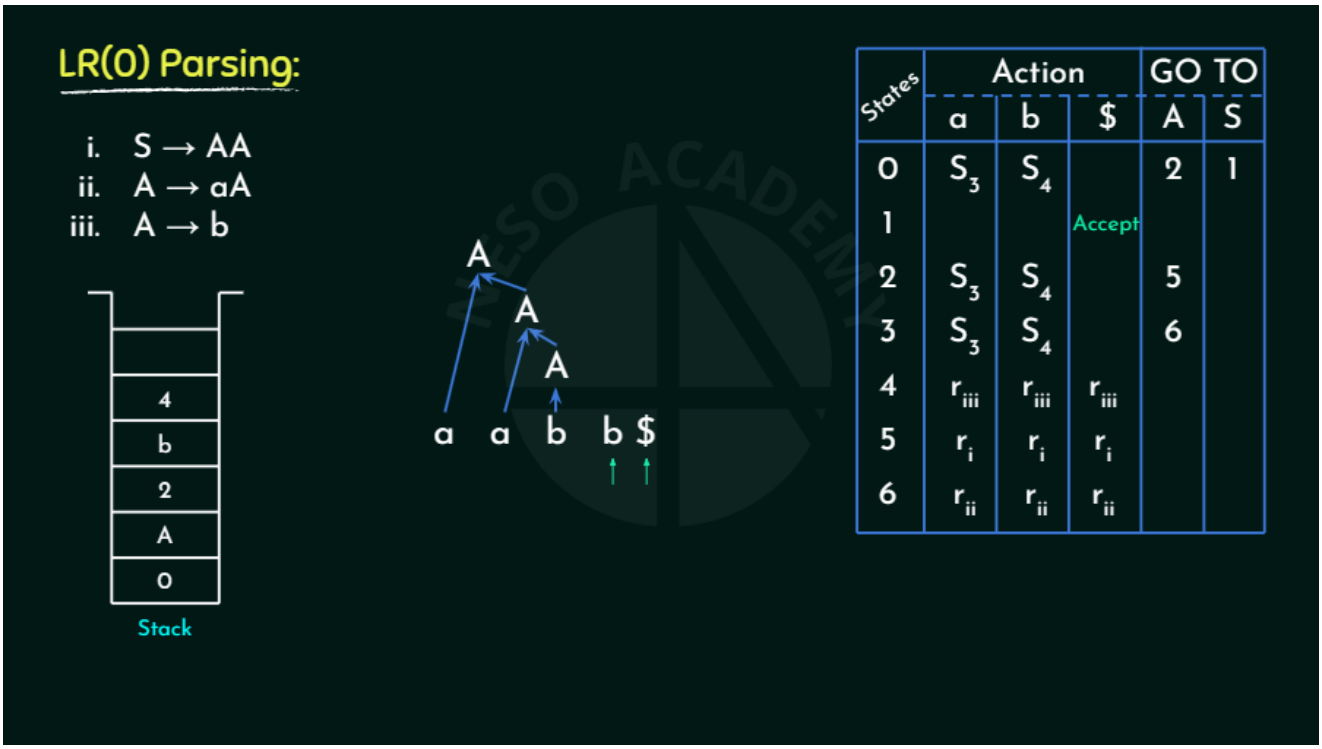
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $a0A3A6A 1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



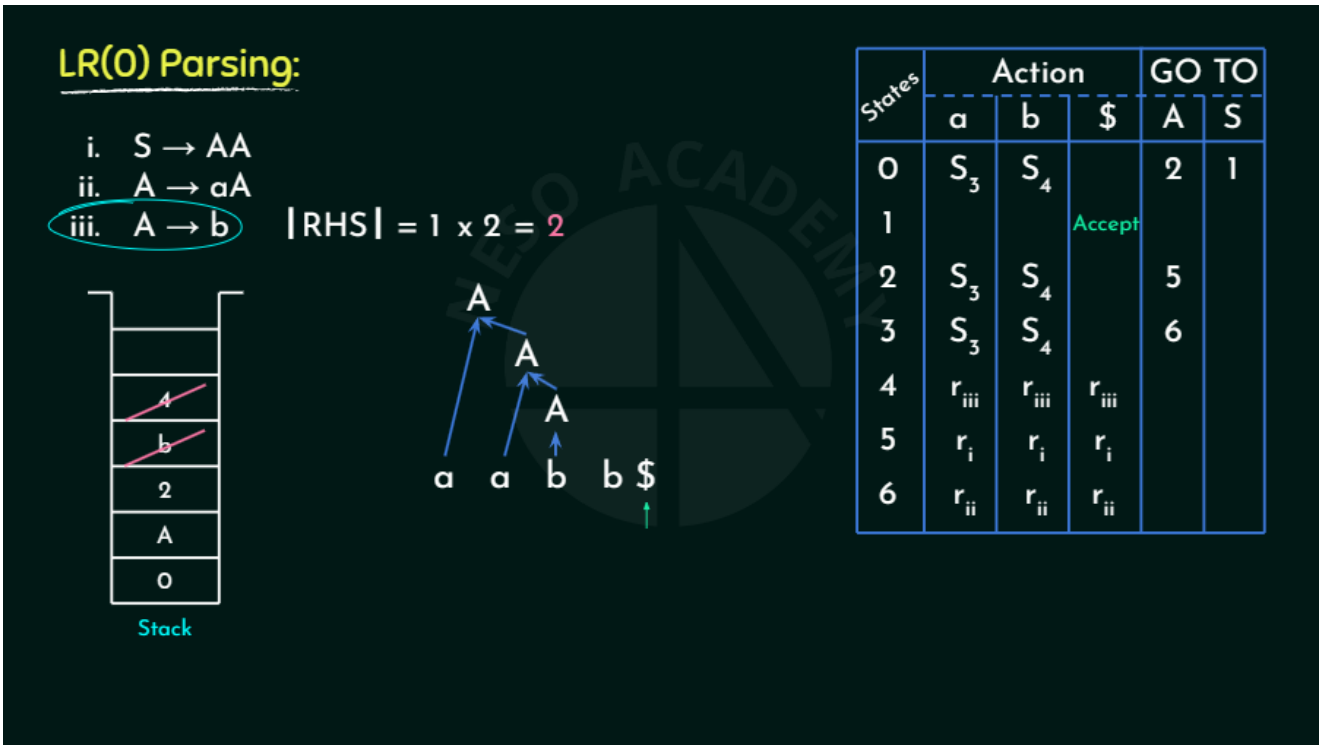
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $a0A3AIRHSIRHS= 2x 2= 46A 1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



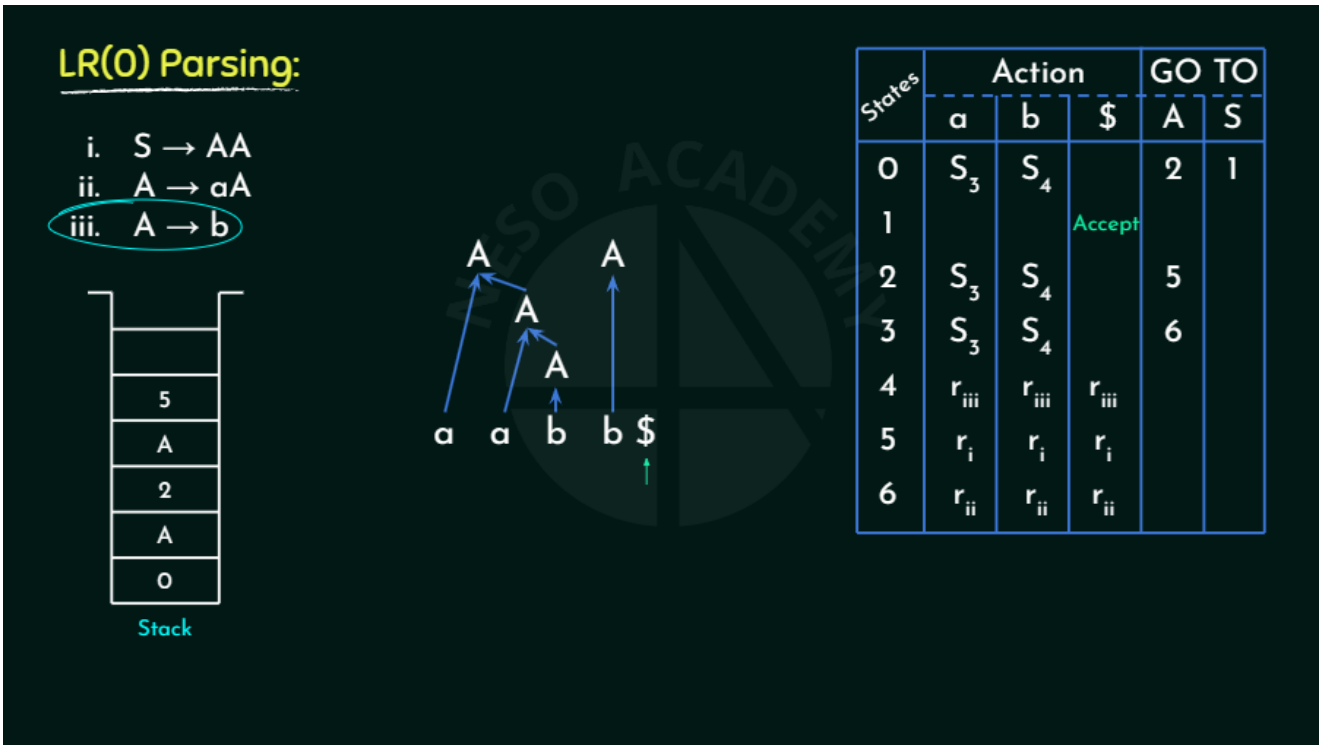
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $A0AA A 21ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



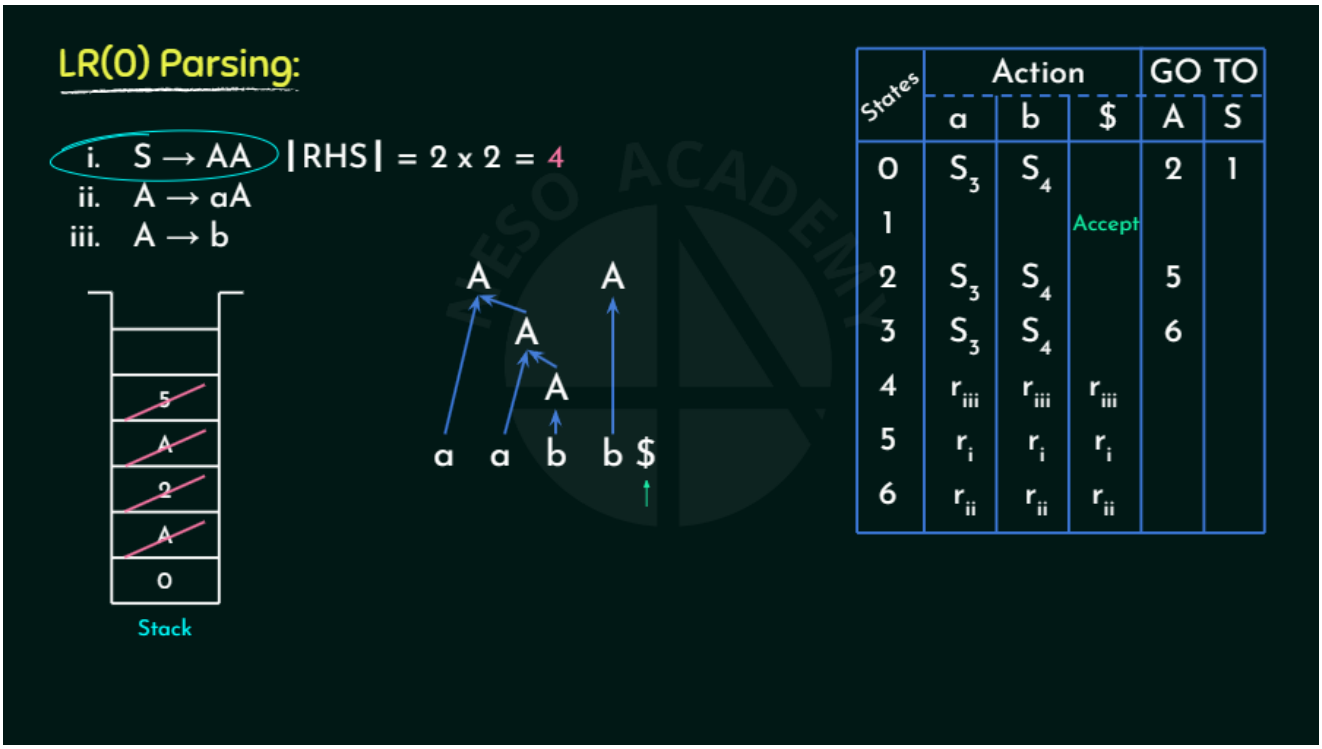
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $A0AA A 2b41ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



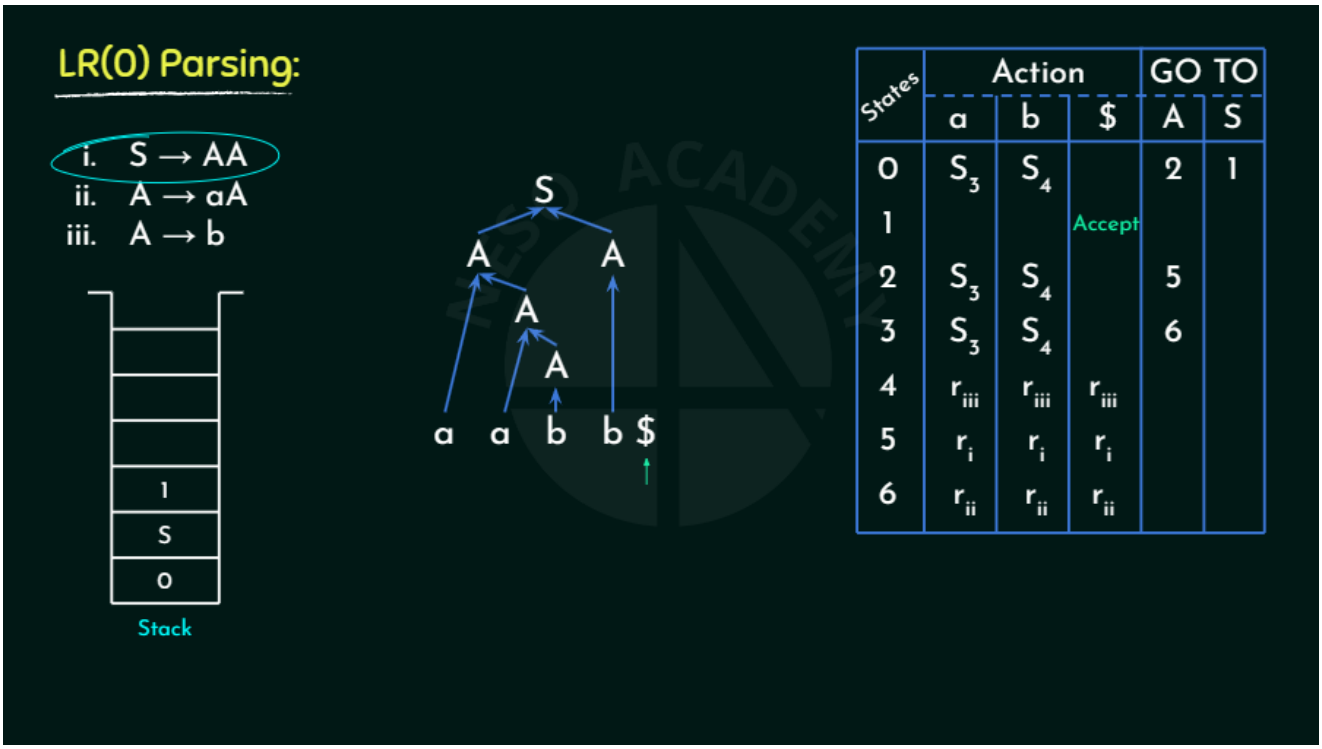
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $A0AA A 2b4IRHSIRHS= 1x 2= 21ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA



Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $A0AA A 2AA51ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA

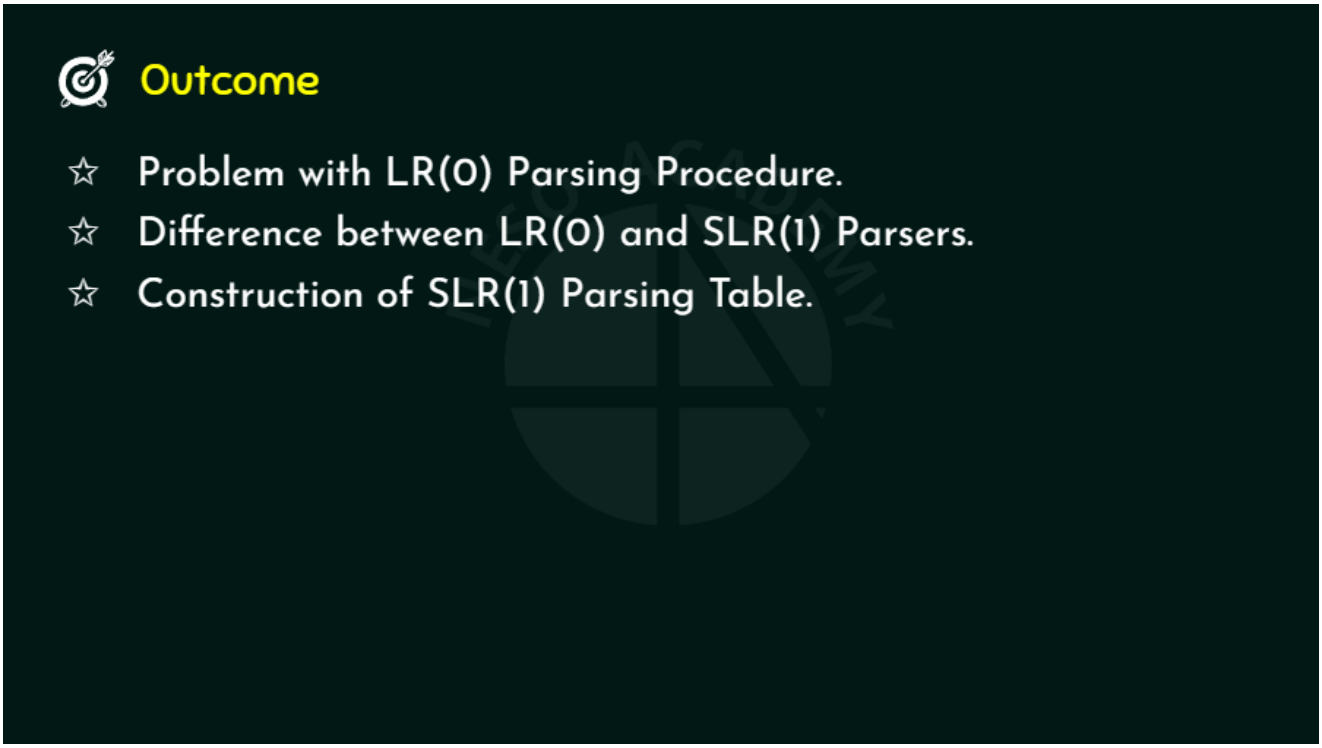


Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $A0AA A 2AA5IRHSIRHS= 2x 2= 41ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA

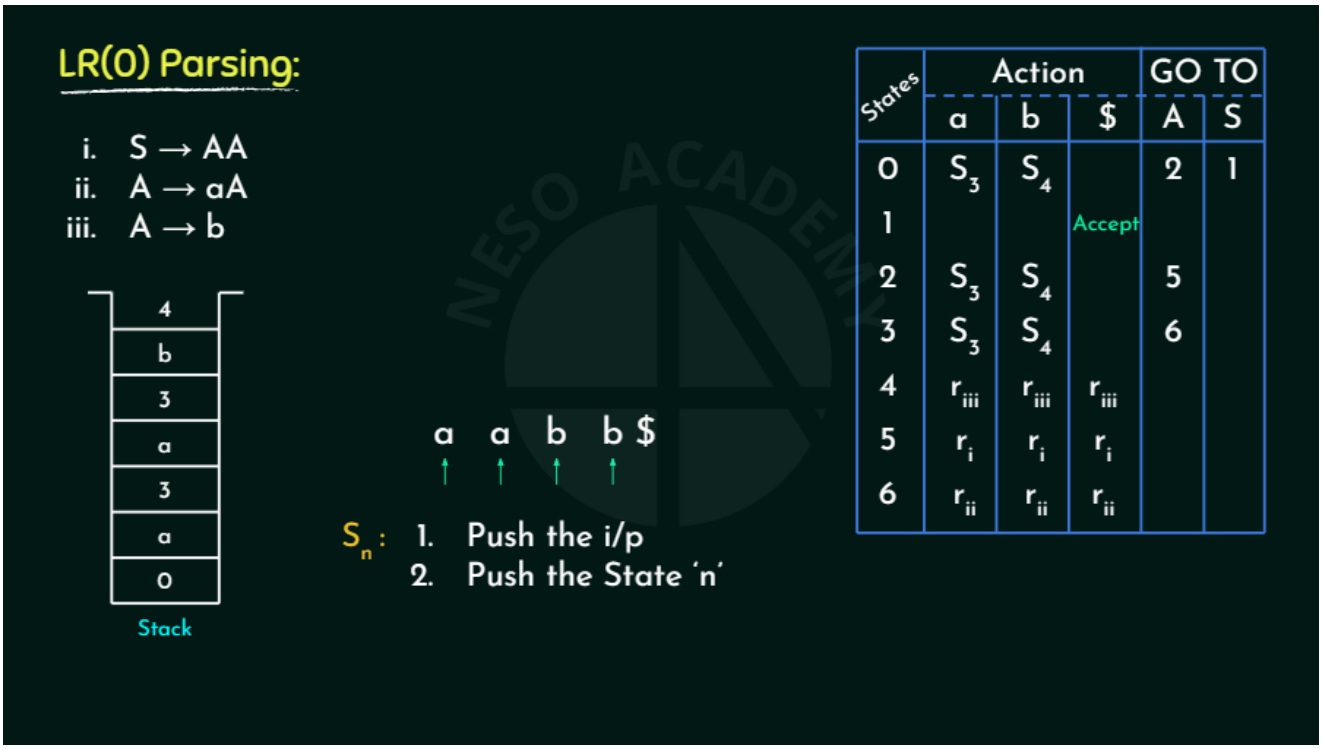


Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $S0AA A A1S1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriirii56ab$SA

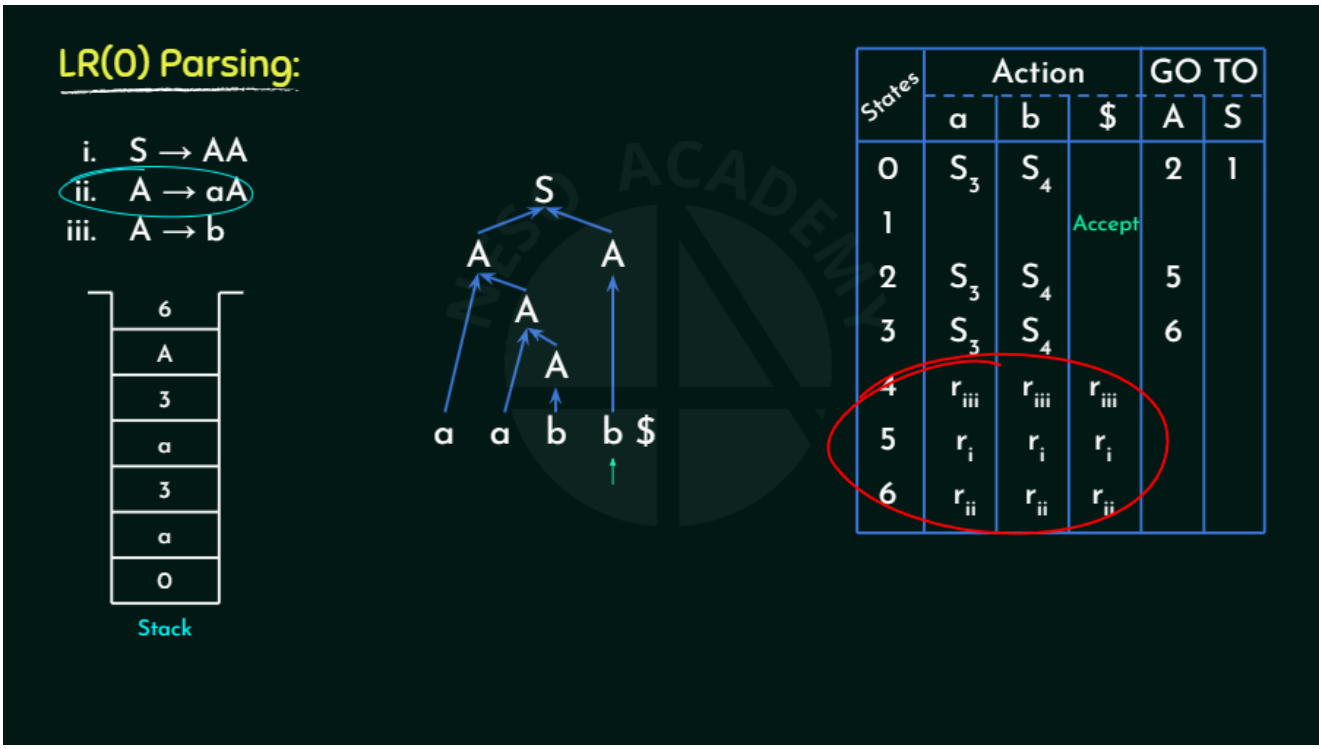
Compiler Design SLR(1) Parsers



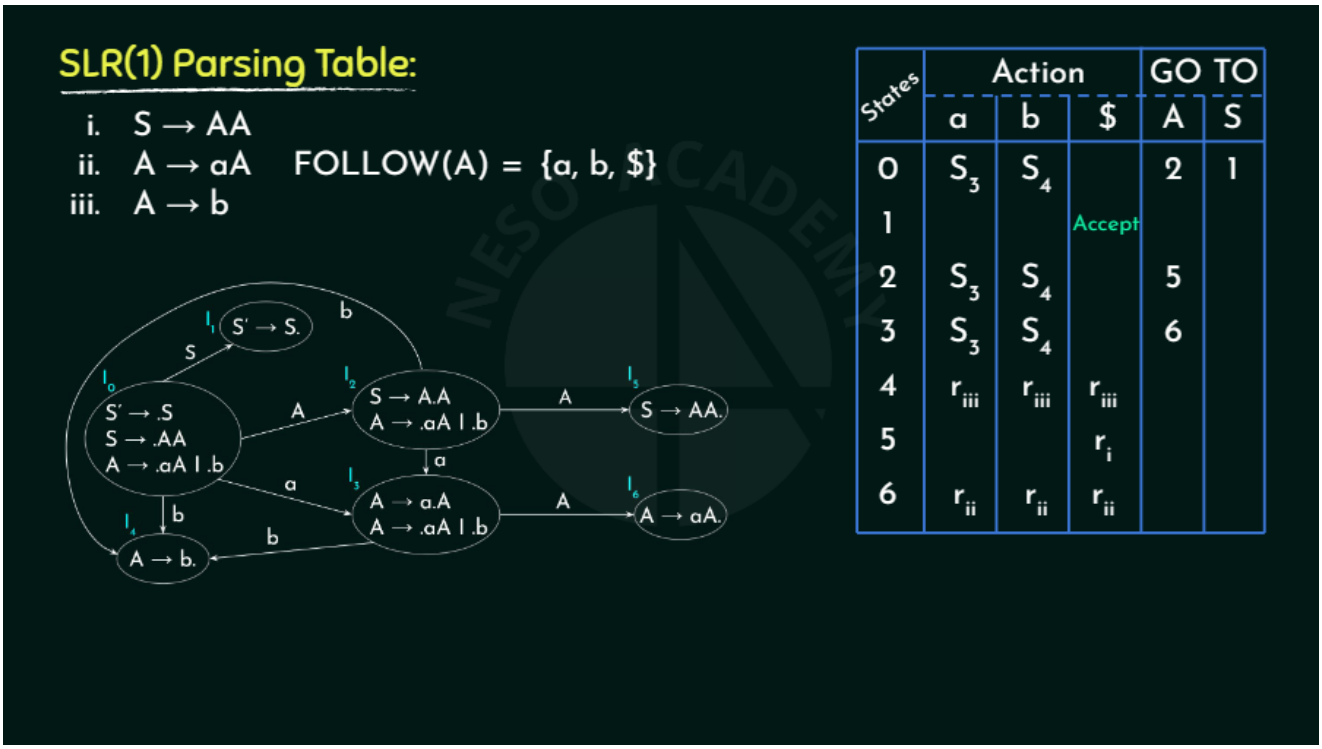
Outcome☆Problem with LR(0) Parsing Procedure.☆Difference between LR(0) and SLR(1) Parsers.☆Construction of SLR(1) Parsing Table.



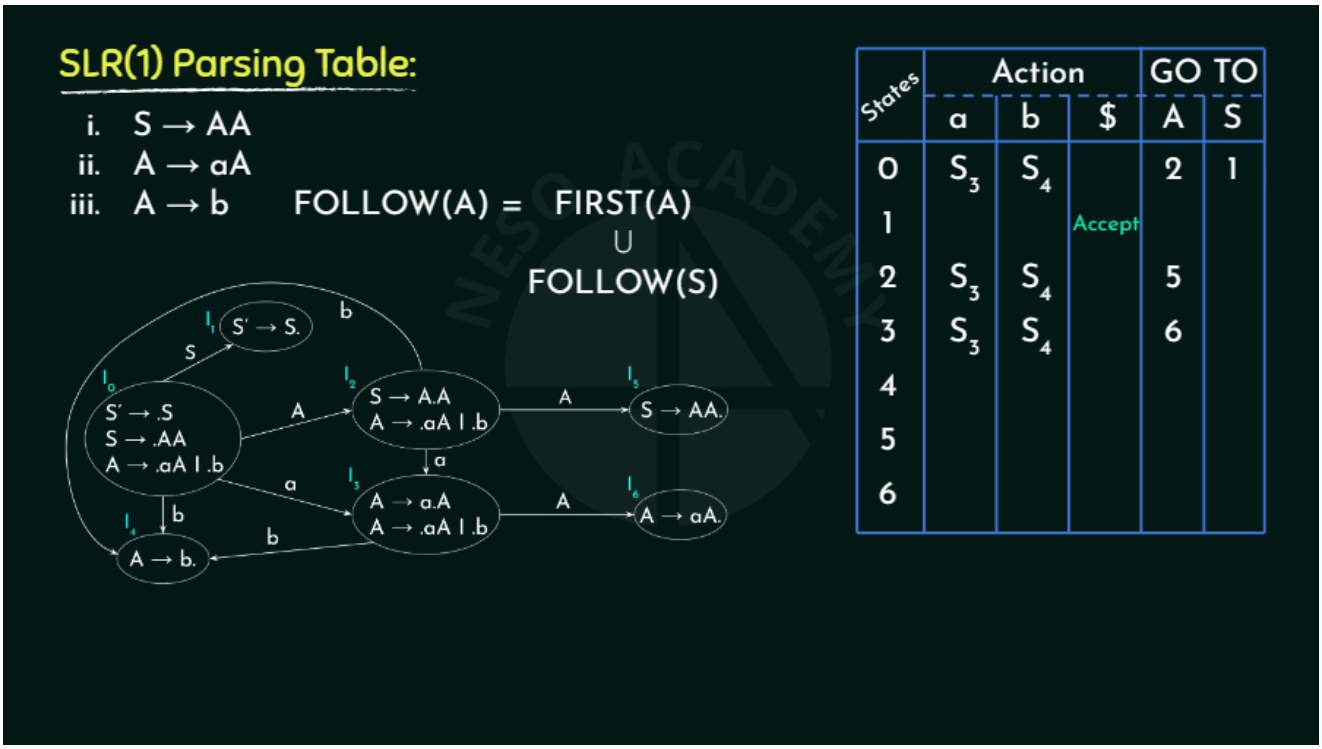
StackLR(0) Parsing:1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriiri56ab$SAa a b b $a03a34bSn : 1.Push the i/p 2.Push the State 'n' i.S → AAii.A → aA iii.A → b



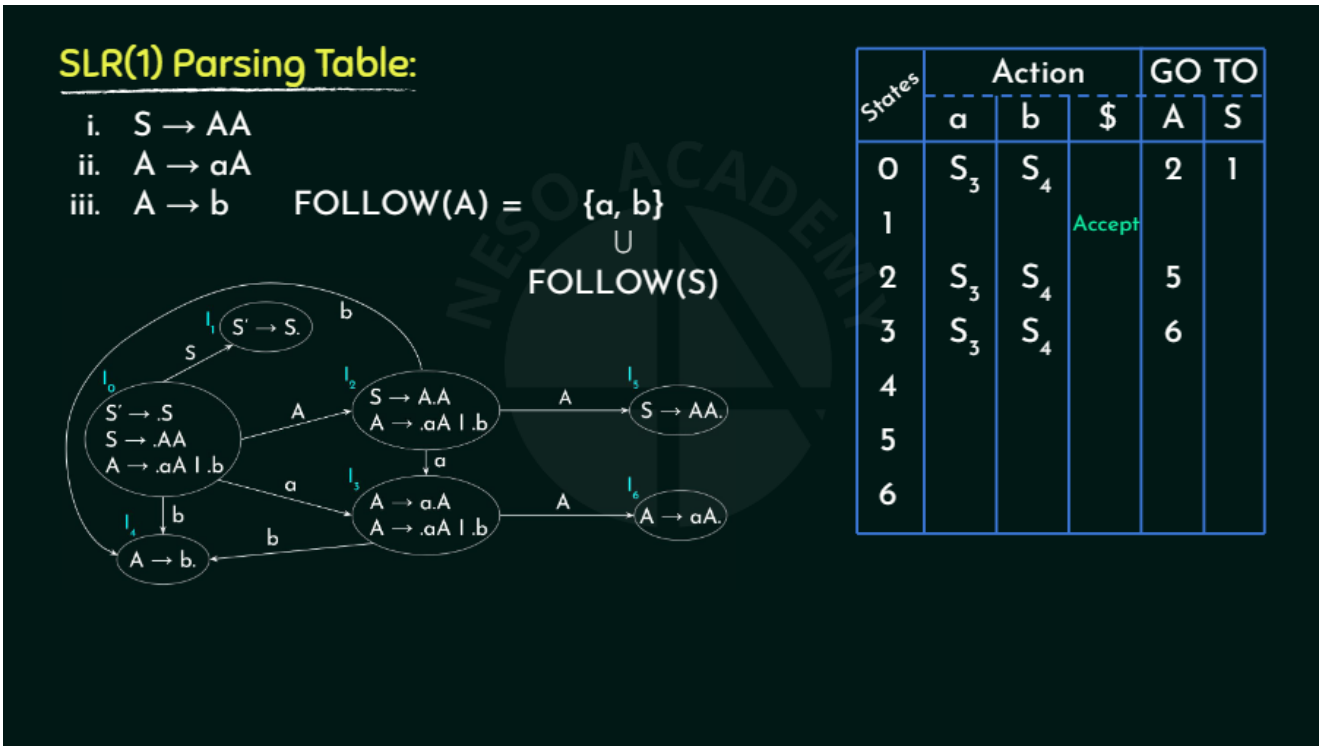
Stacki.S → AAii.A → aA iii.A → bLR(0) Parsing:a a b b $a03a3AA61ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriiri56ab$SAA A AS



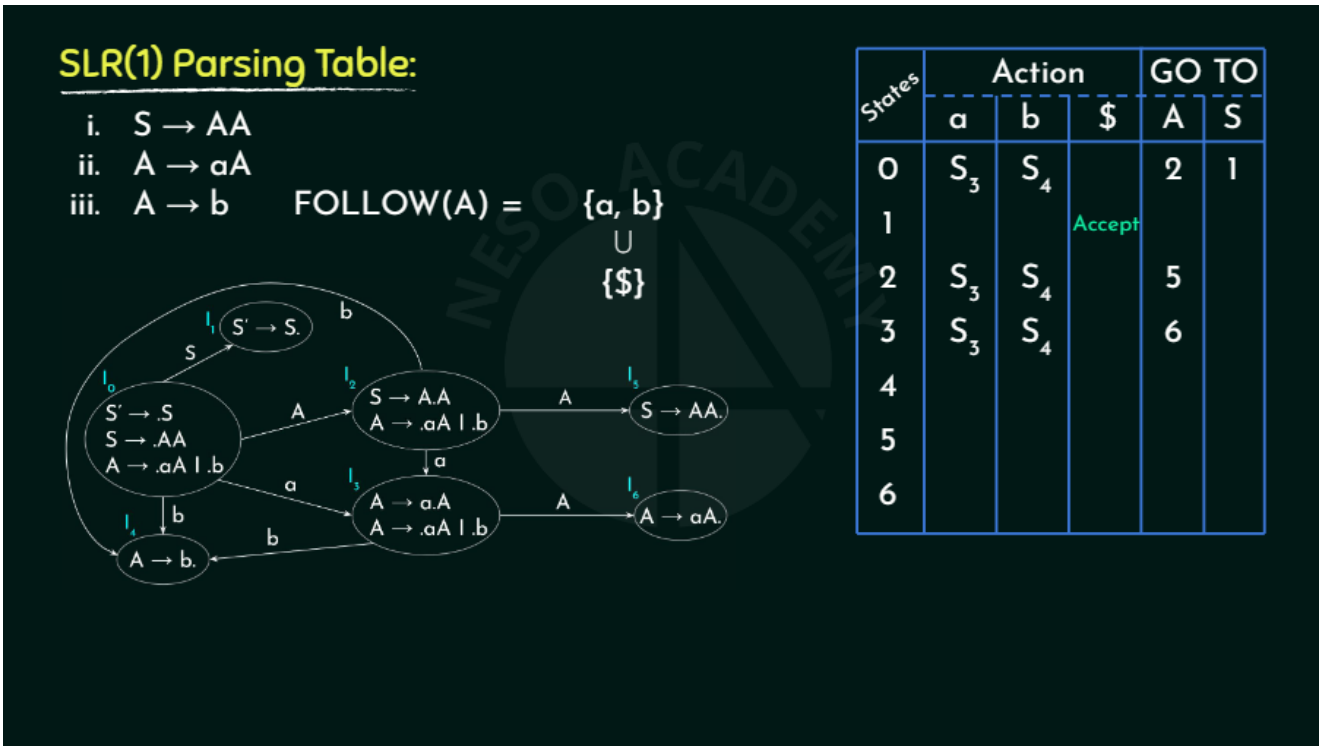
i.S → AAii.A → aA iii.A → bSLR(1) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiriirii56ab$SAriiiriiiriiiriFOLLOW(A) = {a, b, $}



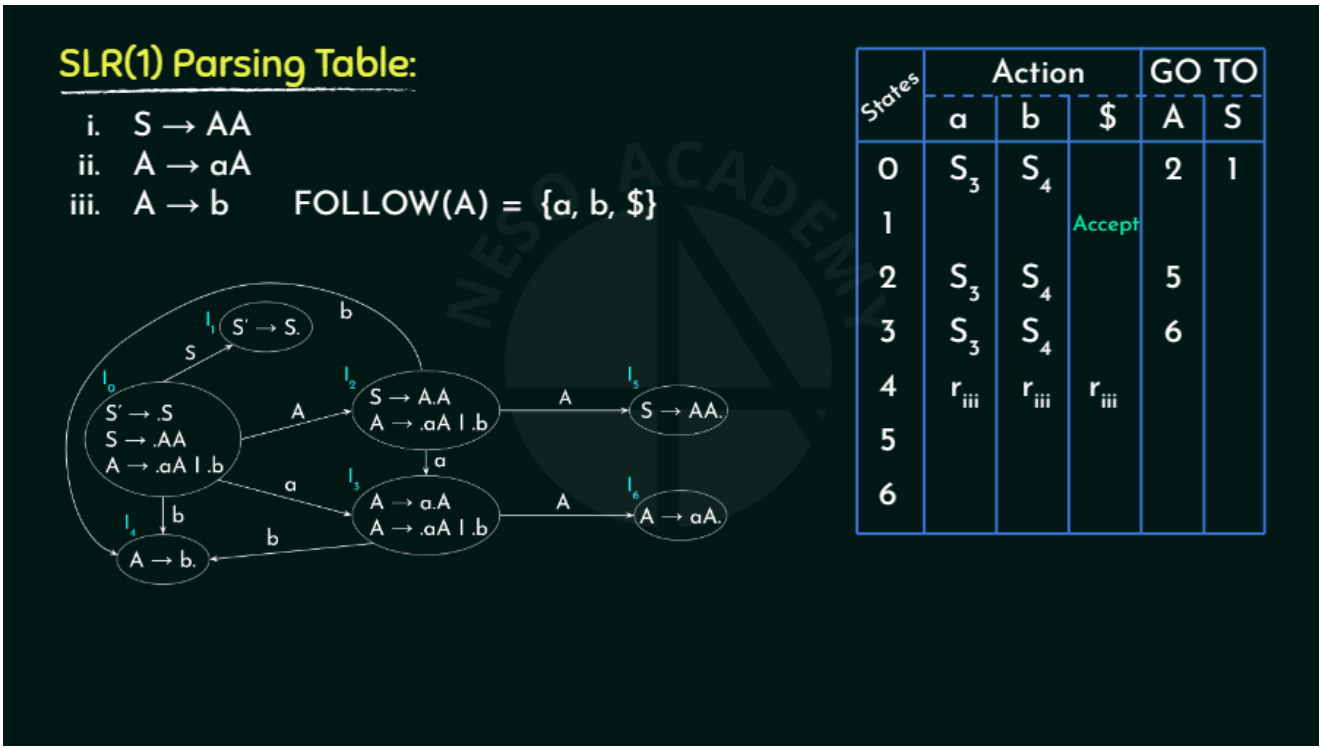
i.S → AAii.A → aA iii.A → bSLR(1) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S456ab$SAFOLLOW(A) = FIRST(A) ∪ FOLLOW(S)



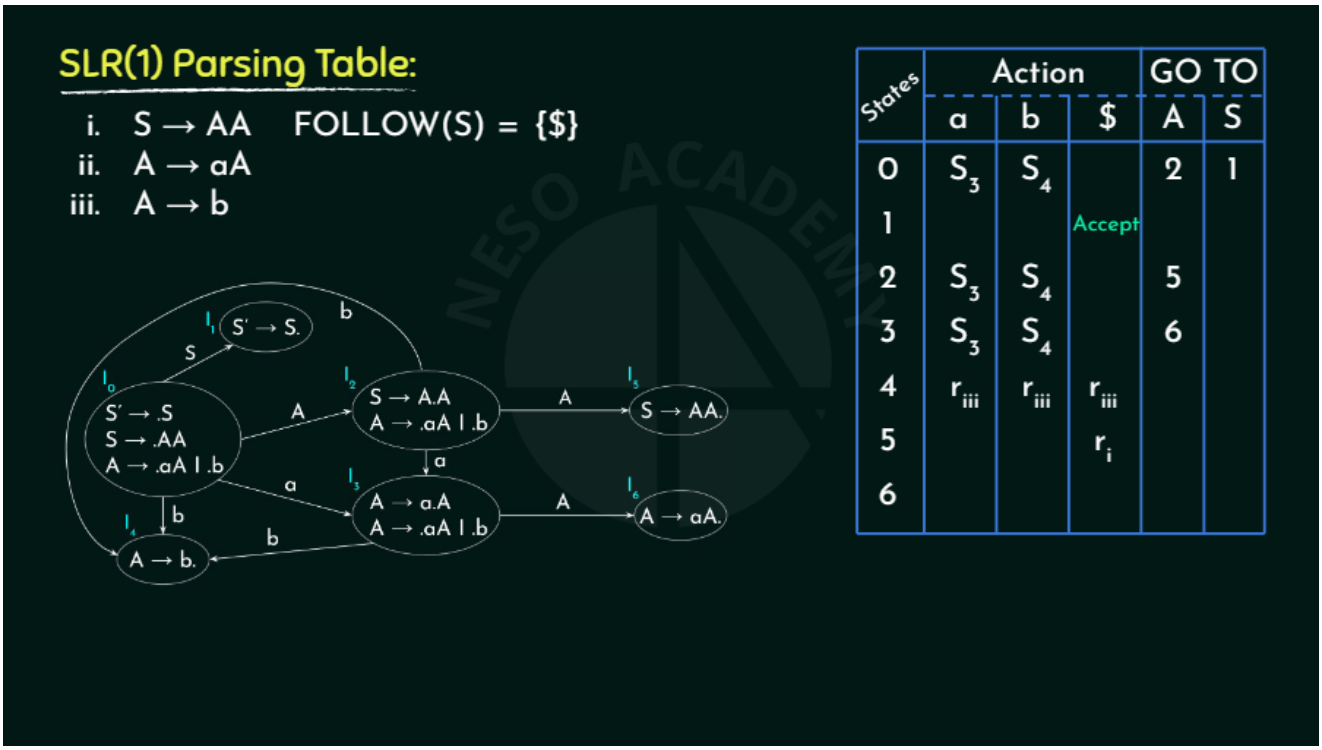
i.S → AAii.A → aA iii.A → bSLR(1) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S456ab$SAFOLLOW(A) = {a, b} ∪ FOLLOW(S)



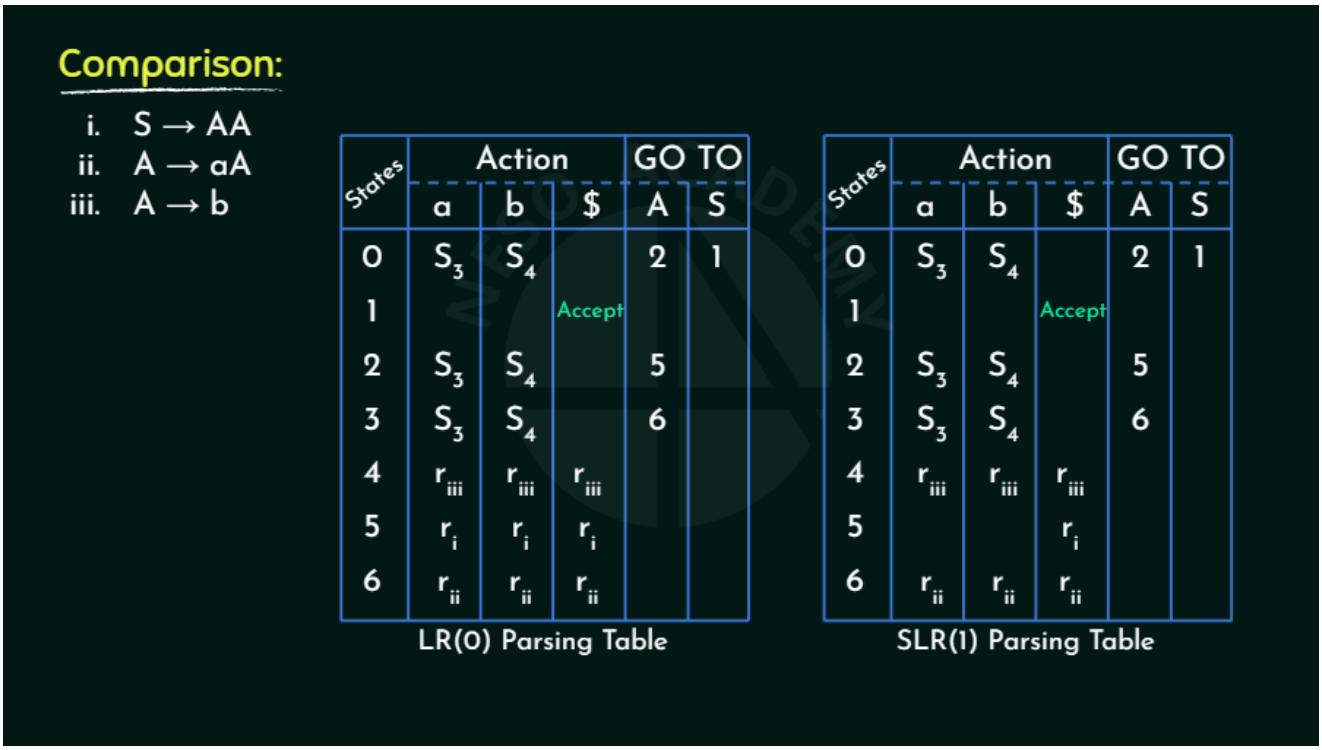
i.S → AAii.A → aA iii.A → bSLR(1) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S456ab$SAFOLLOW(A) = {a, b} ∪ {$}



i.S → AAii.A → aA iii.A → bSLR(1) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S456ab$SAriiiriiiriiiFOLLOW(A) = {a, b, $}

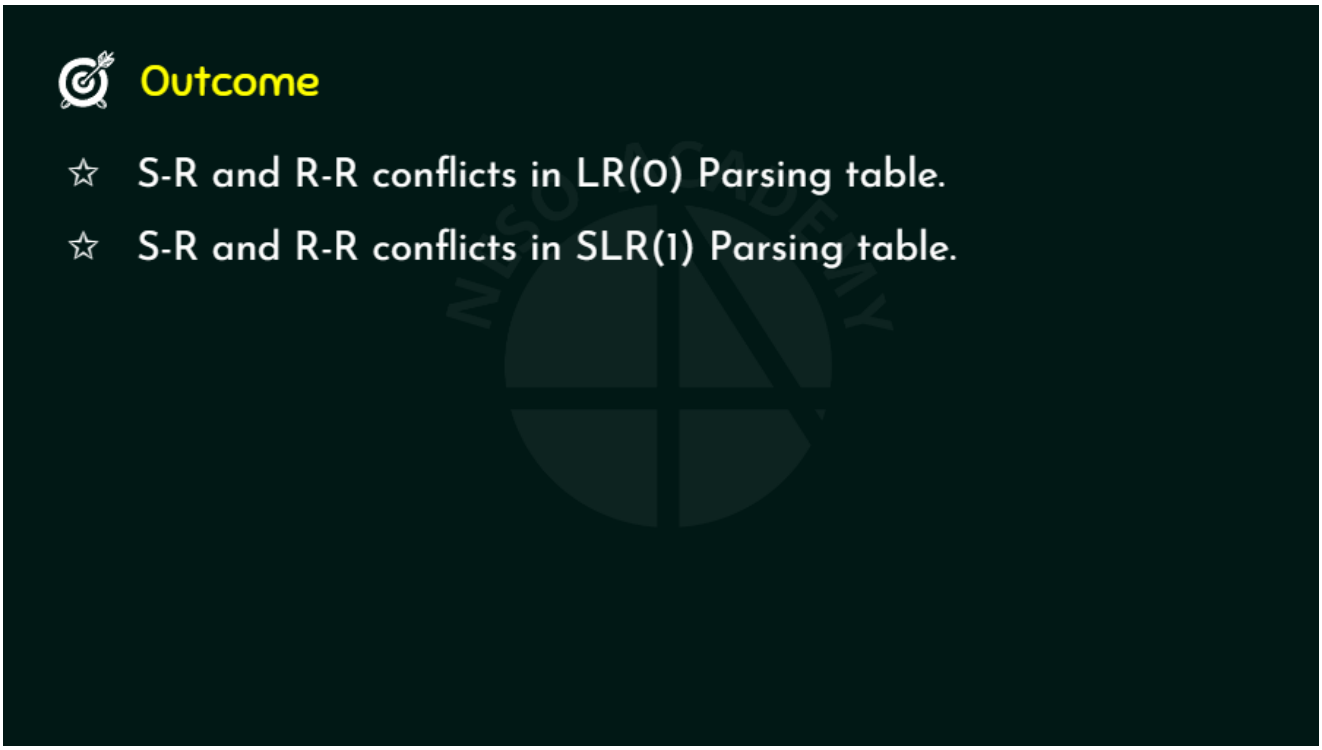


i.S → AAii.A → aA iii.A → bSLR(1) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S456ab$SAriiiriiiriiiriFOLLOW(S) = {$}

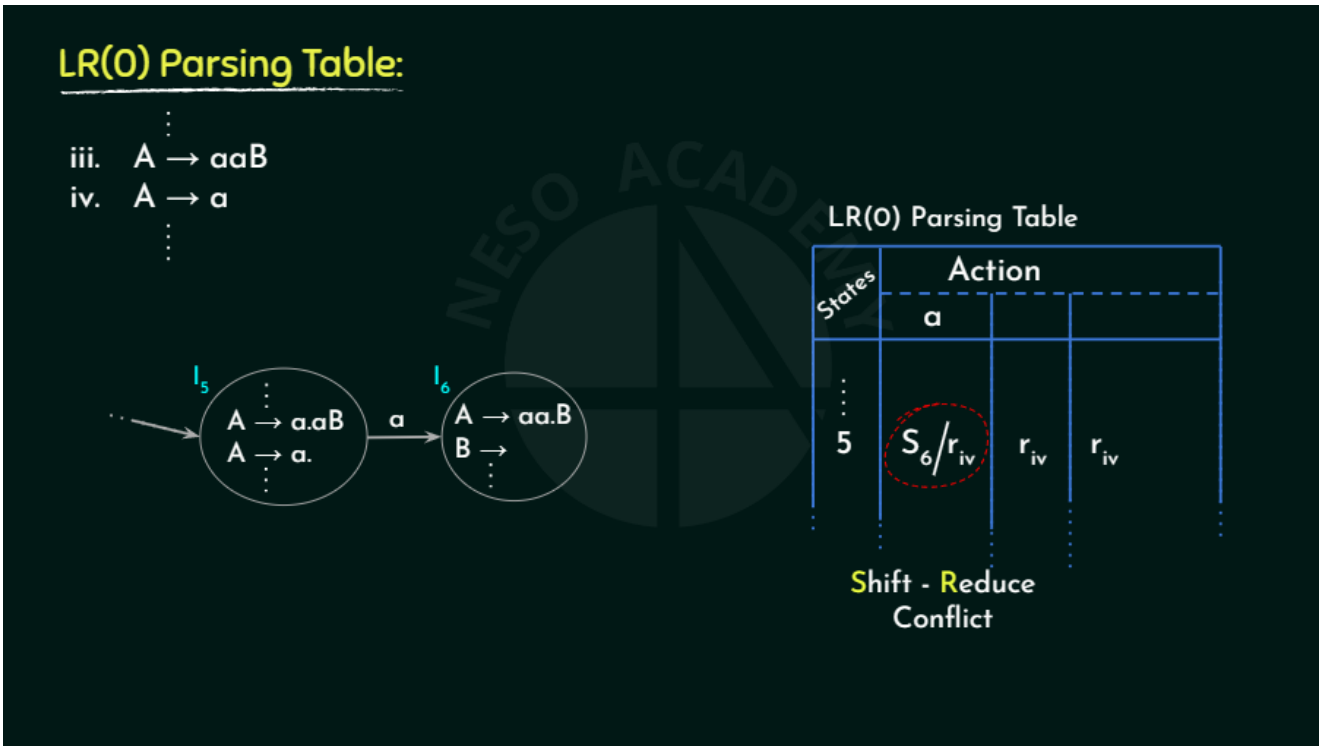


Comparison:i.S → AAii.A → aA iii.A → b1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriii56ab$SALR(0) Parsing Table1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiriirii56ab$SAriiiriiiriiiriSLR(1) Parsing Table

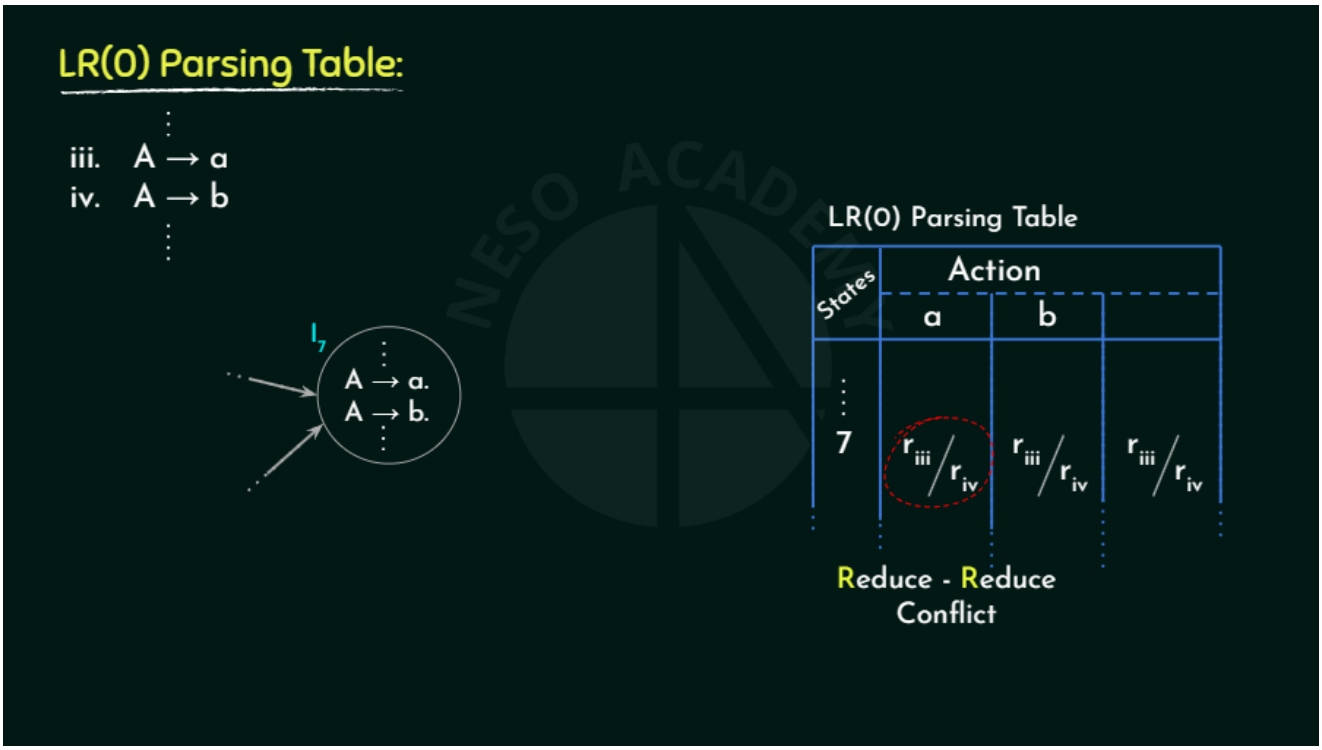
Compiler Design Conflicts in LR(0) and SLR(1) Parsers



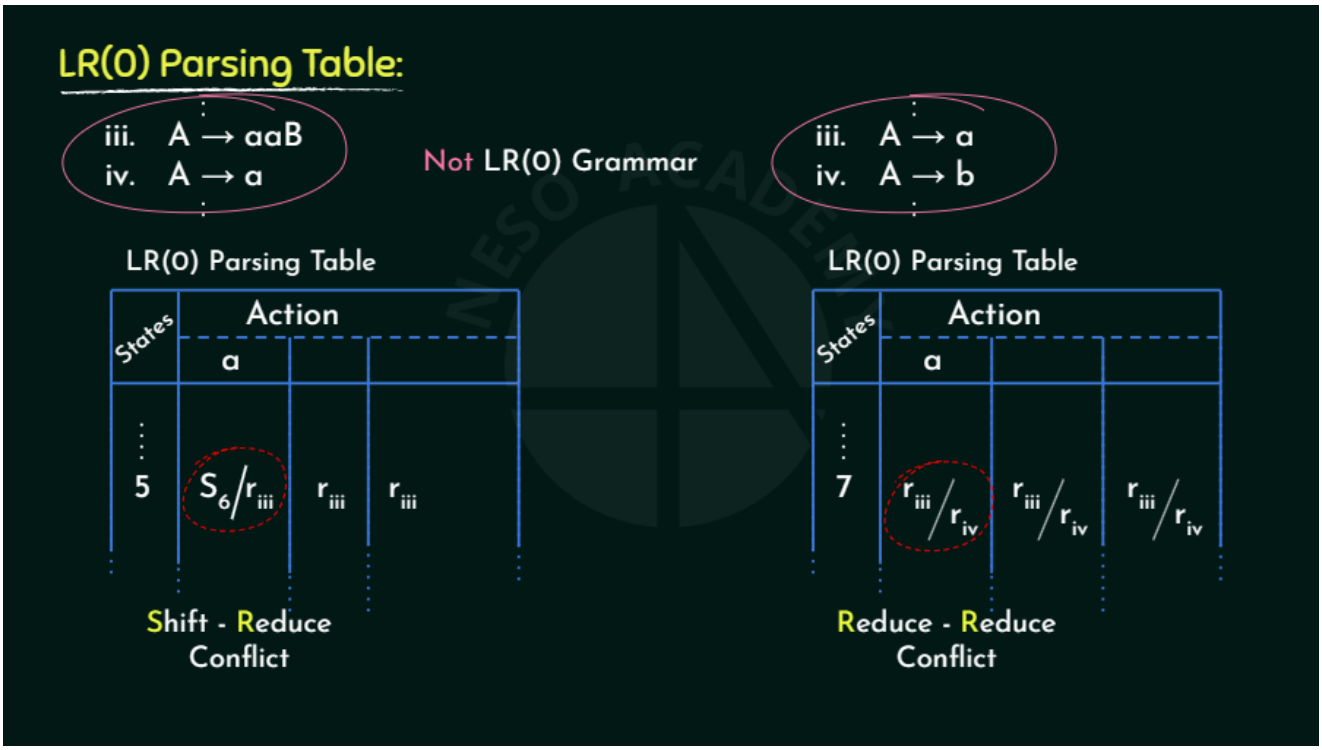
Outcome☆S-R and R-R conflicts in LR(0) Parsing table.☆S-R and R-R conflicts in SLR(1) Parsing table.



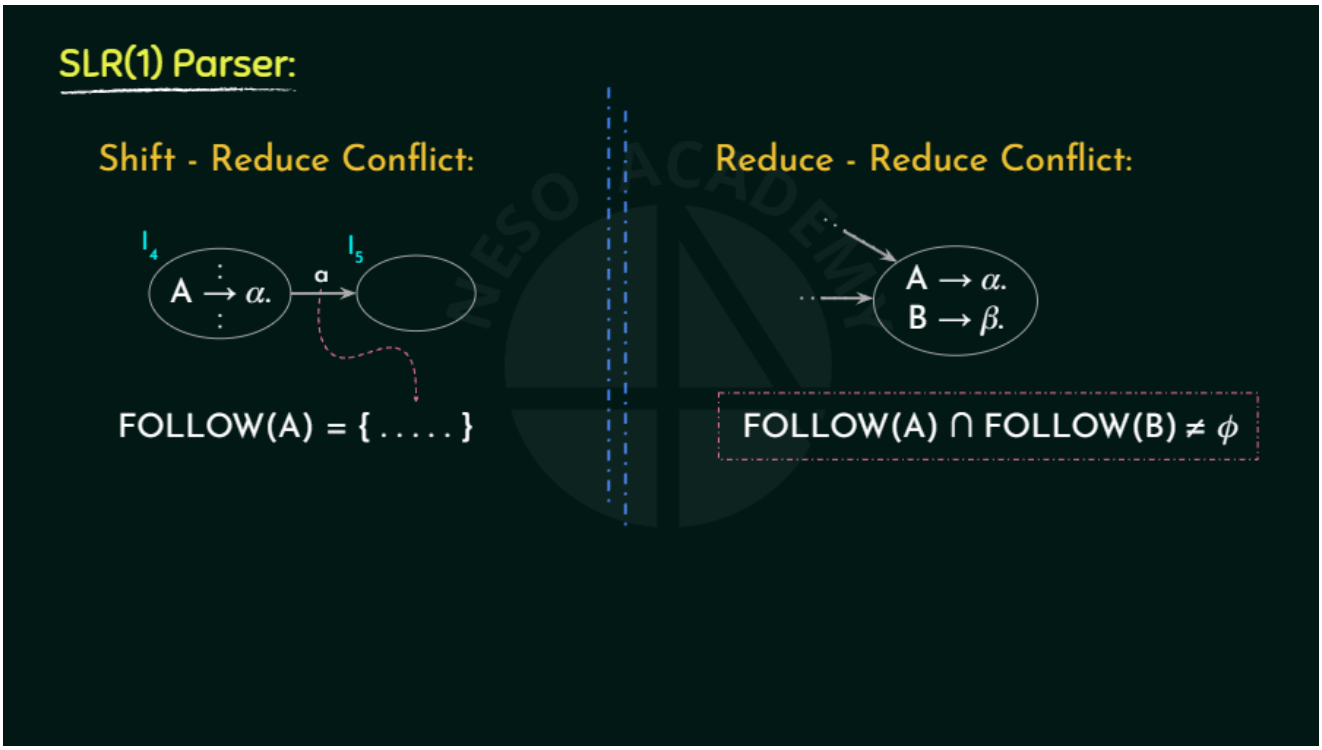
iii.A → aaB iv.A → aLR(0) Parsing Table:aA → aa.B B → I6LR(0) Parsing TableActionStatesS6arivriv5A → a.aB A → a.I5rivShift - ReduceConflict



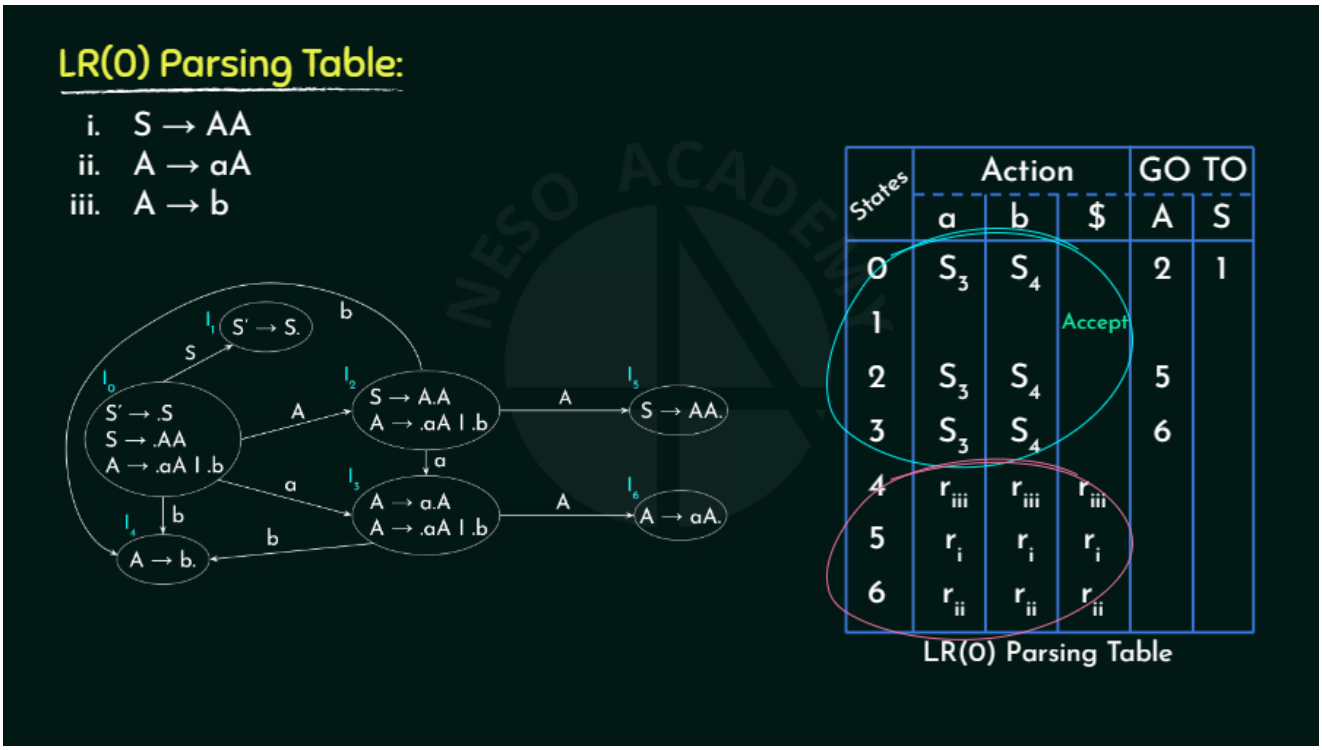
LR(0) Parsing Table:iii.A → a iv.A → bLR(0) Parsing TableActionStatesa7A → a. A → b.I7riiirivriiirivriiirivReduce - ReduceConflictb



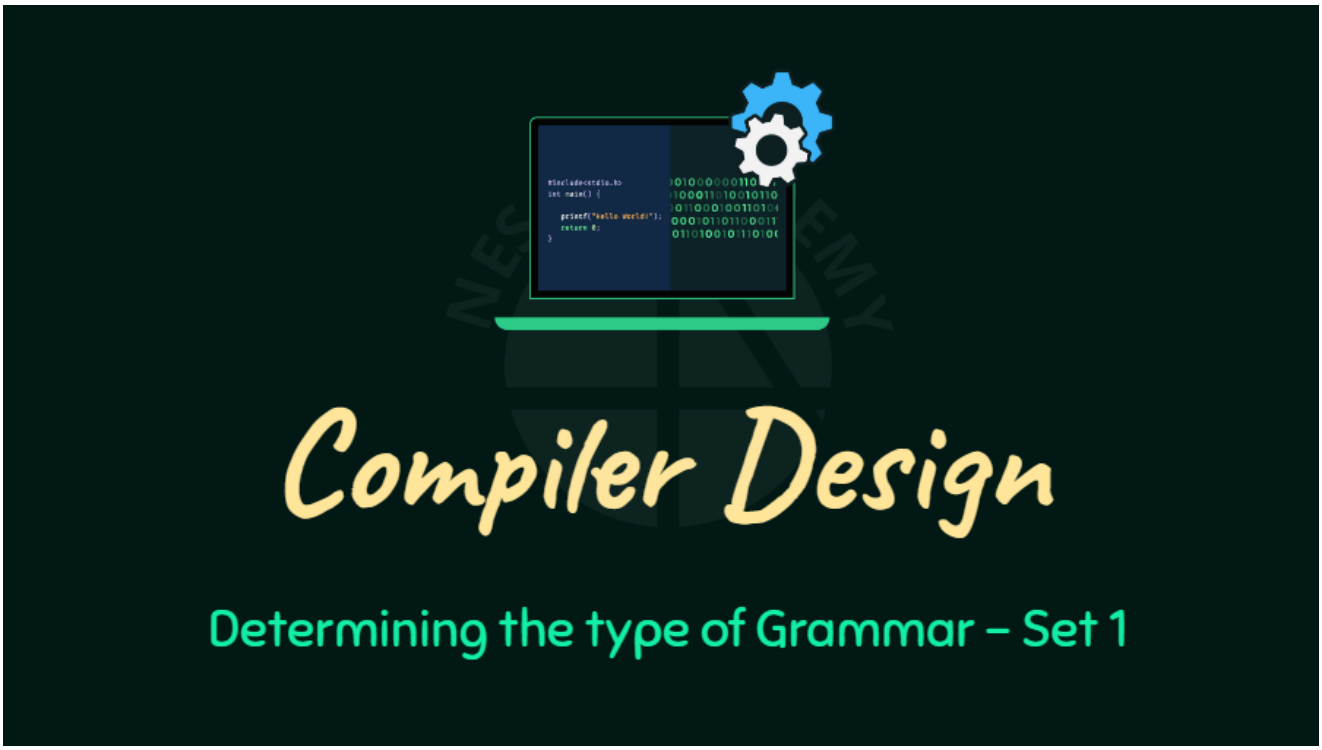
LR(0) Parsing Table:iii.A → a iv.A → bLR(0) Parsing TableActionStatesa7riiirivriiirivriiirivReduce - ReduceConflictLR(0) Parsing TableActionStatesS6ariiiriii5riiiShift - ReduceConflictiii.A → aaB iv.A → aNot LR(0) Grammar



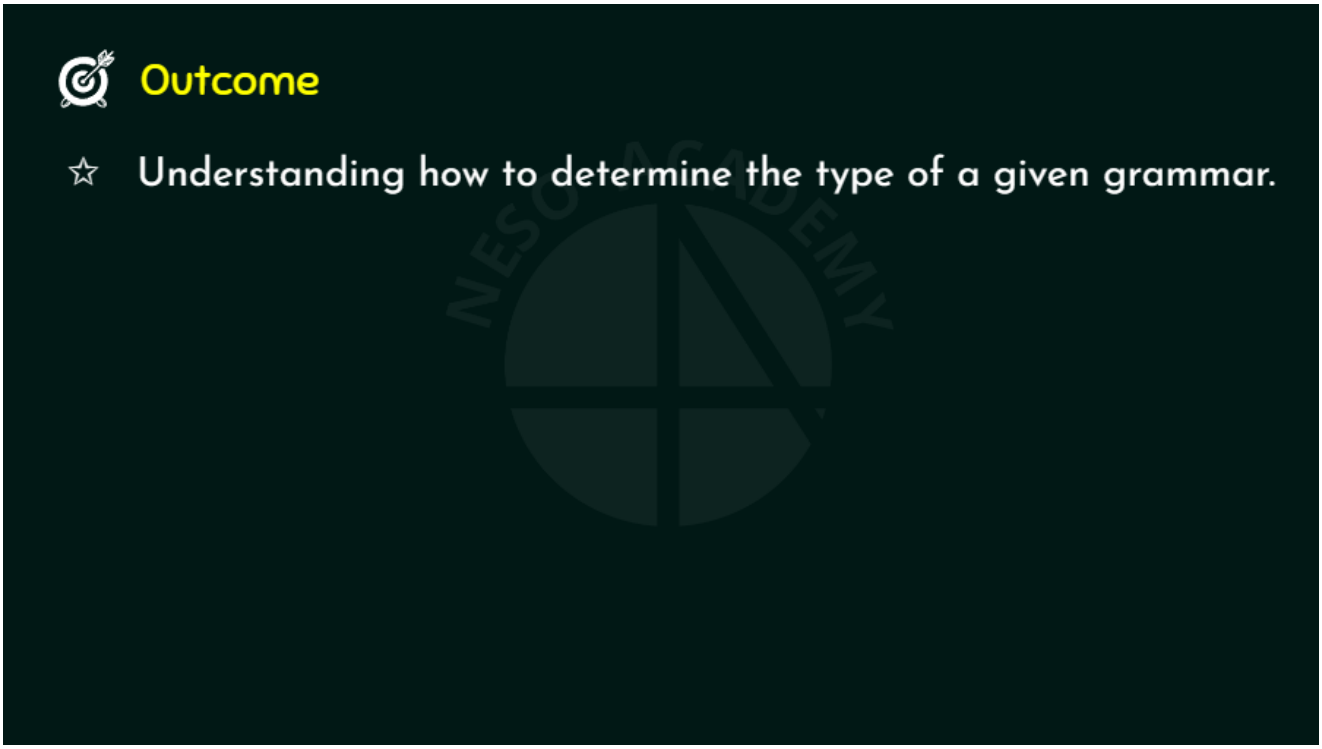
≠ ϕSLR(1) Parser:Shift - Reduce Conflict:A → α.I4I5aReduce - Reduce Conflict:A → α.B → β.FOLLOW(A) = { . . . . . }FOLLOW(A) ∩ FOLLOW(B)



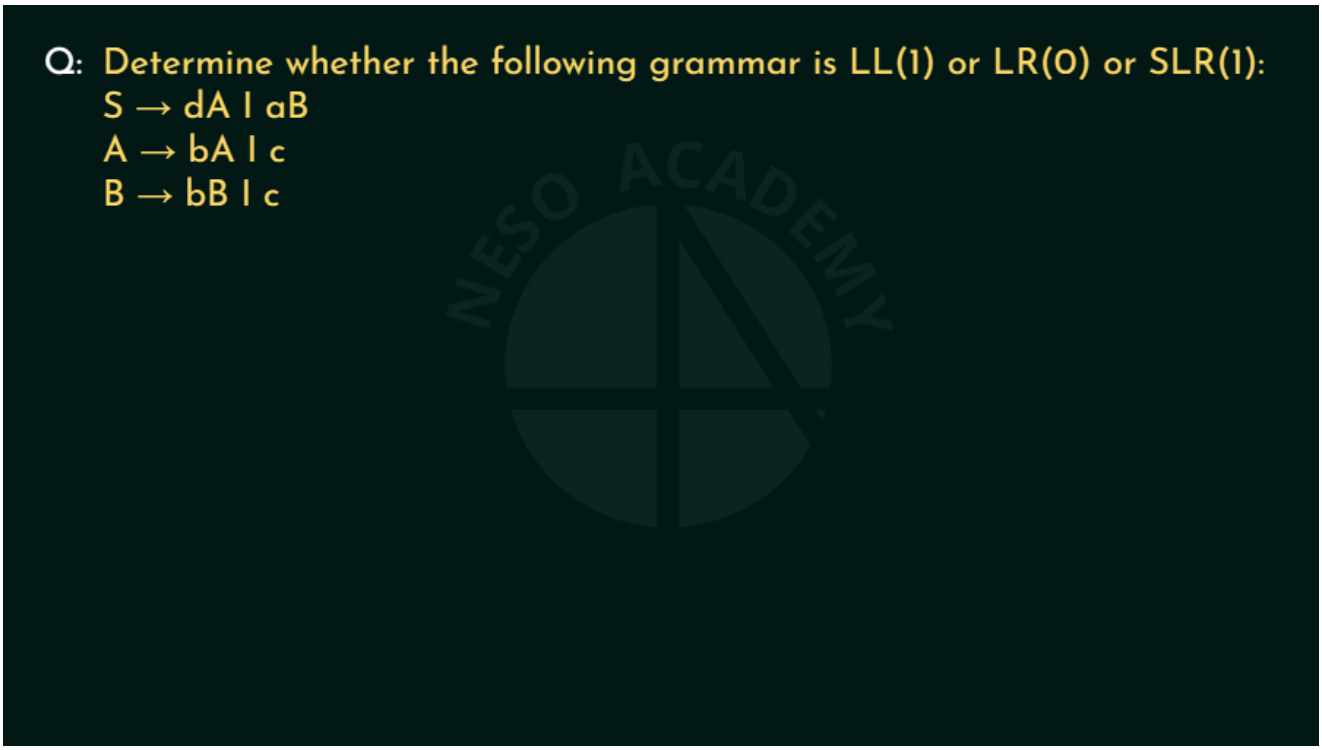
i.S → AAii.A → aA iii.A → bLR(0) Parsing Table:1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriiri56ab$SALR(0) Parsing Table



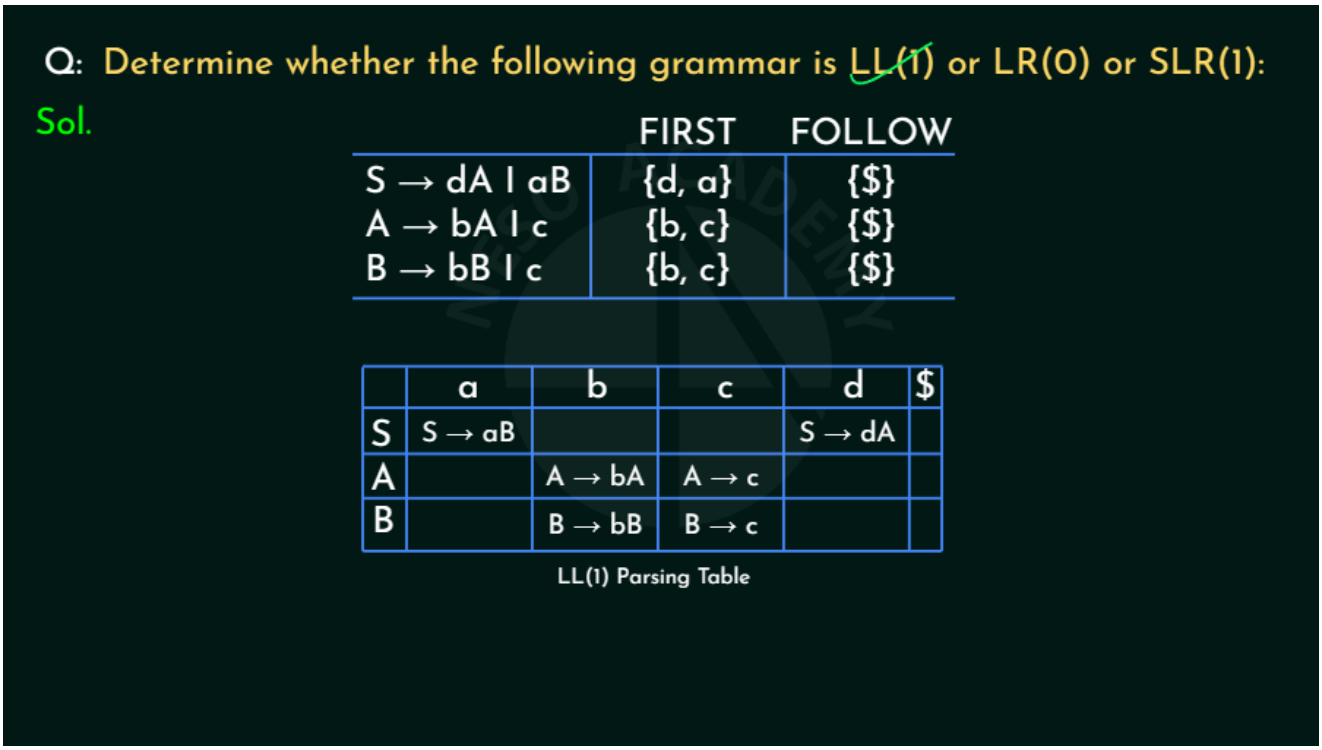
Compiler Design Determining the type of Grammar - Set 1



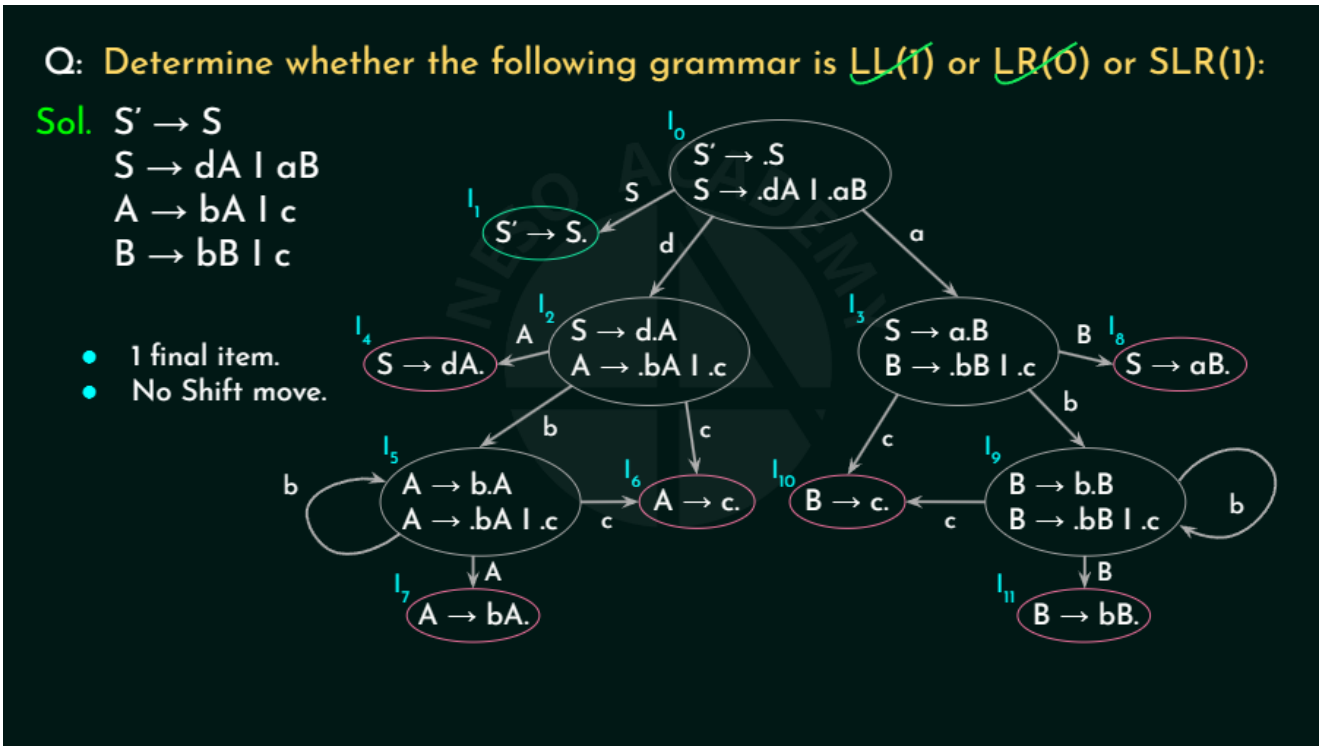
Outcome☆Understanding how to determine the type of a given grammar.



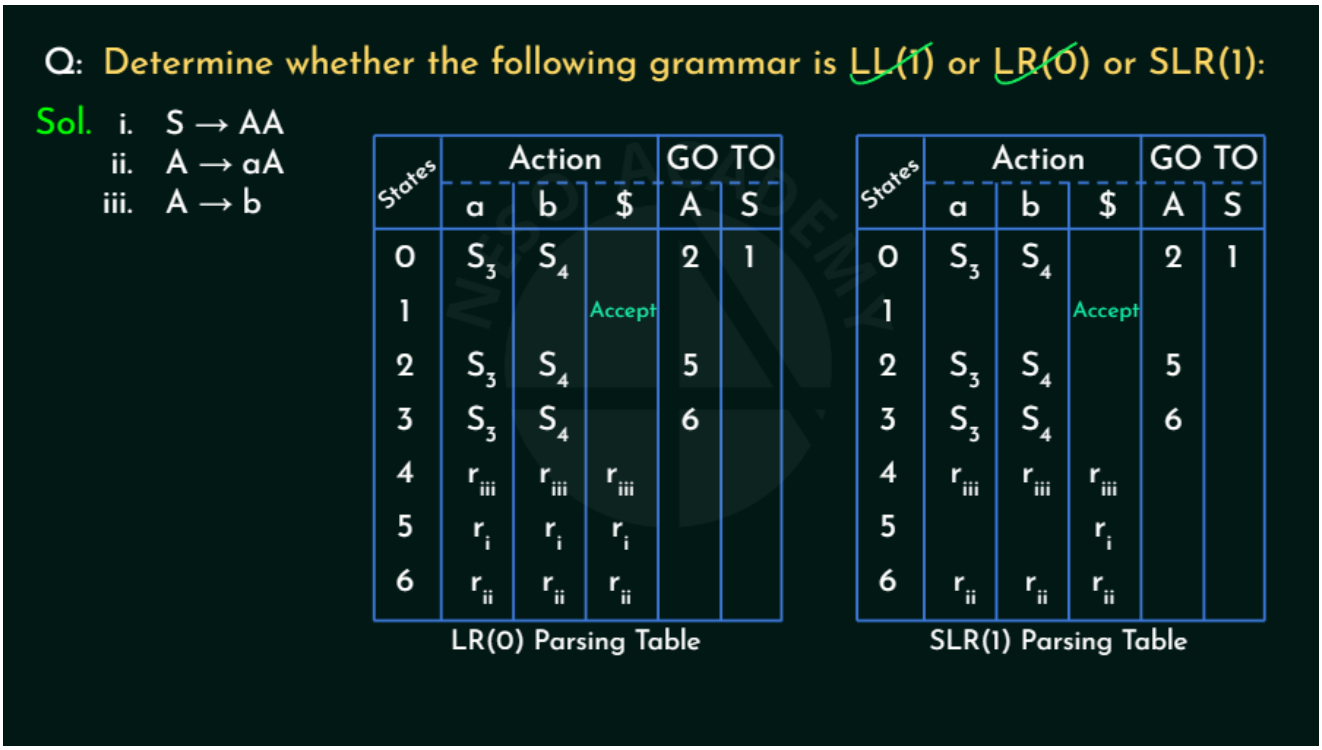
Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → dA I aB A → bA I cB → bB I c



LL(1) Parsing TableadSAB$cbQ:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → dA I aB A → bA I cB → bB I cFOLLOW{$}FIRST{d, a}{b, c}{$}{$}{b, c}S → aBB → cA → cA → bAB → bBS → dASol.



I9I5I3I2I0Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → SS → dA I aB A → bA I cB → bB I cSol.S' → .SS → .dA I .aBS → d.AA → .bA I .cS → a.BB → .bB I .cA → b.AA → .bA I .cB → b.BB → .bB I .cS' → S.I1S → dA.I4A → c.I6A → bA.I7S → aB.I8B → c.I10B → bB.I11SdaAbccbABbccBb●1 final item.●No Shift move.



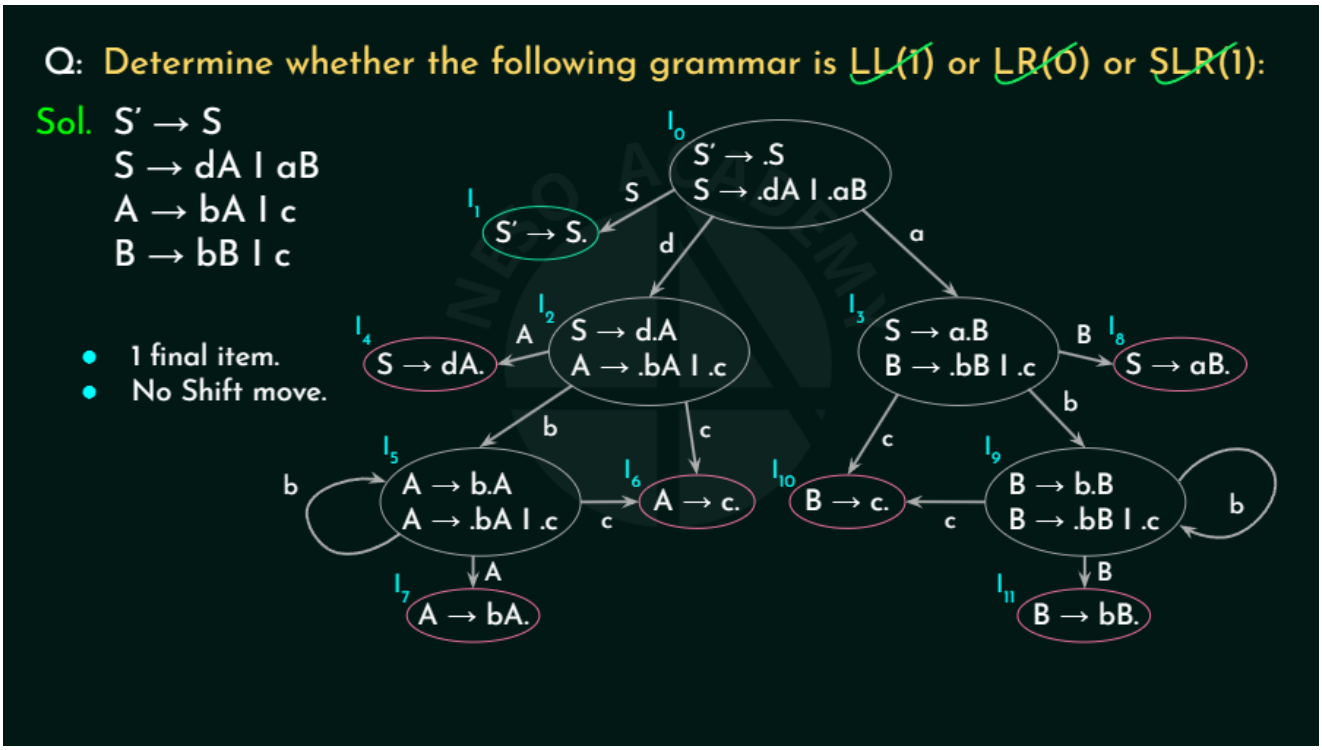
States	Action			GO TO	
	a	b	$	A	S
0	S_3	S_4		2	1
1			Accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_{iii}	r_{iii}	r_{iii}		
5	r_i	r_i	r_i		
6	r_{ii}	r_{ii}	r_{ii}		

LR(0) Parsing Table

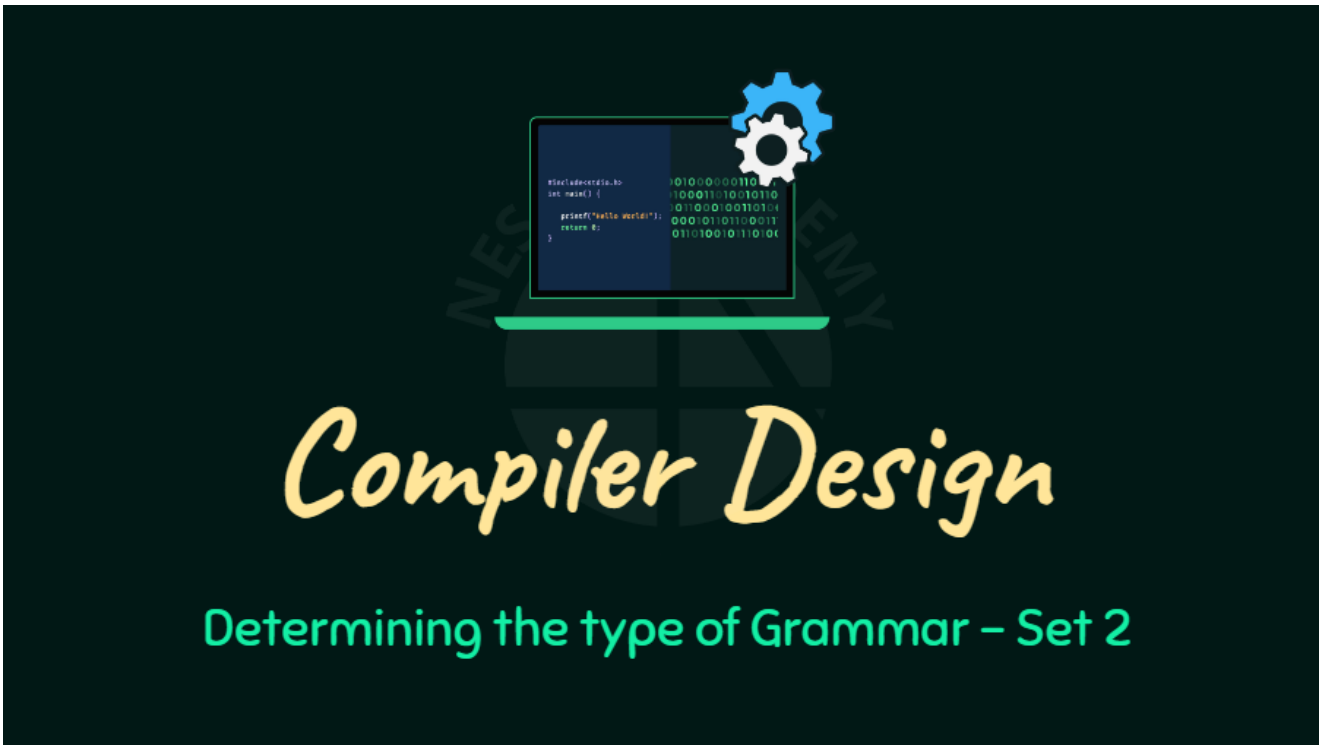
States	Action			GO TO	
	a	b	$	A	S
0	S_3	S_4		2	1
1			Accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_{iii}	r_{iii}	r_{iii}		
5			r_i		
6	r_{ii}	r_{ii}	r_{ii}		

SLR(1) Parsing Table

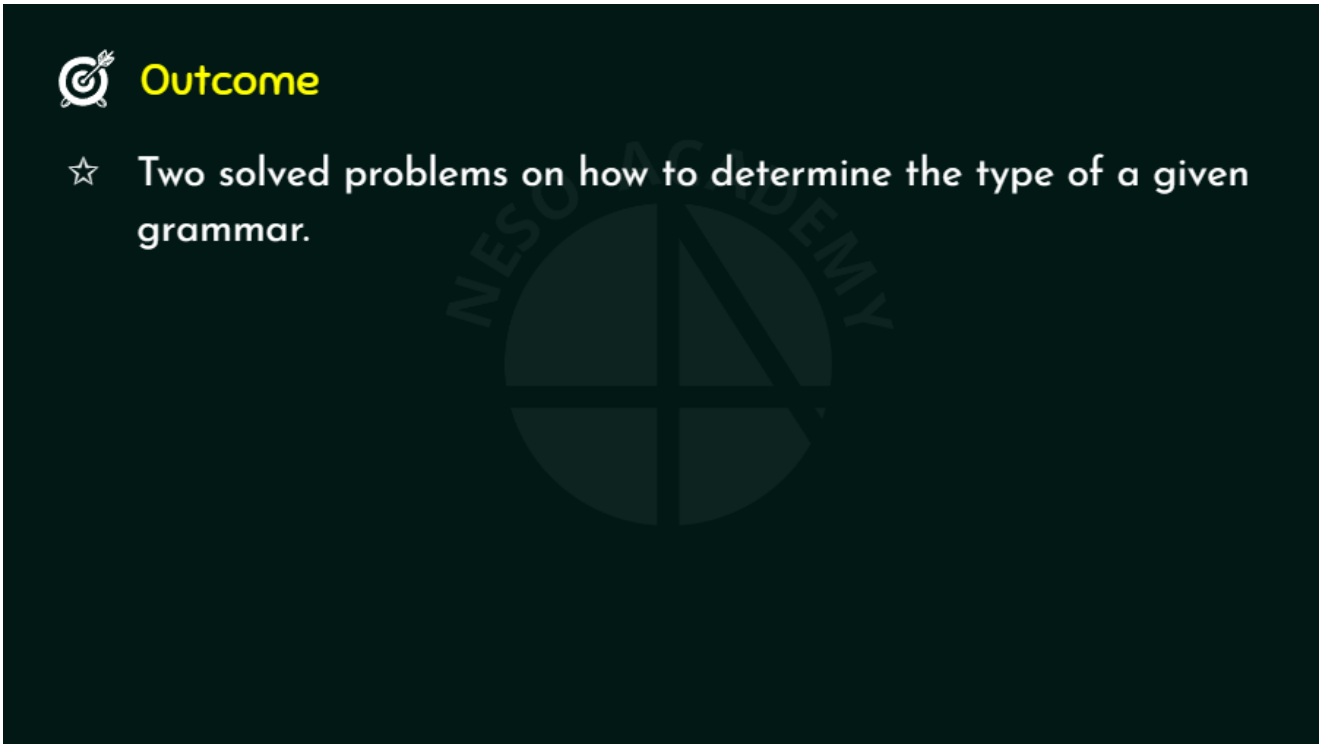
Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):Sol.i.S → AAii.A → aA iii.A → b1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriii56ab$SALR(0) Parsing Table1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiriirii56ab$SAriiiriiiriiiriSLR(1) Parsing Table



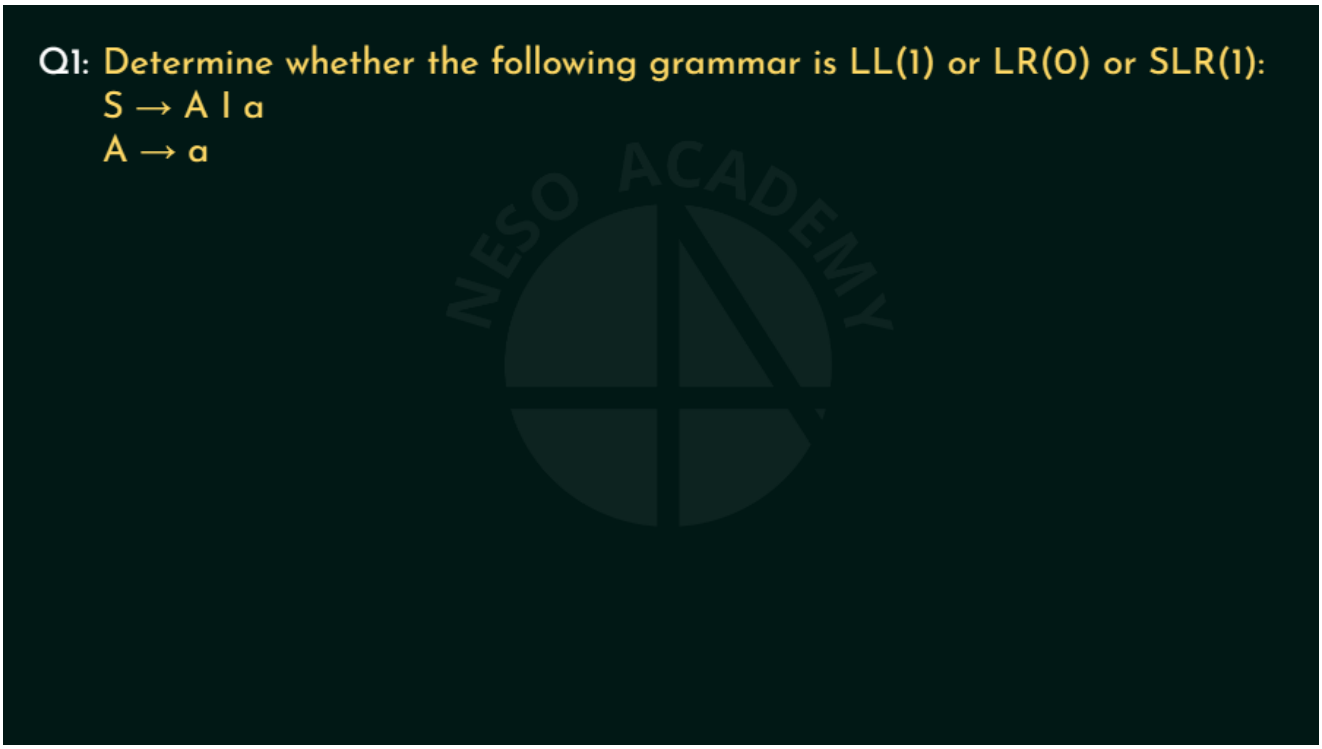
I9I5I3I2I0Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → SS → dA | aB A → bA | cB → bB | cSol.S' → .SS → .dA | .aBS → d.AA → .bA | .cS → a.BB → .bB | .cA → b.AA → .bA | .cB → b.BB → .bB | .cS' → S.I1S → dA.I4A → c.I6A → bA.I7S → aB.I8B → c.I10B → bB.I11SdaAbccbABbccBb●1 final item.●No Shift move.



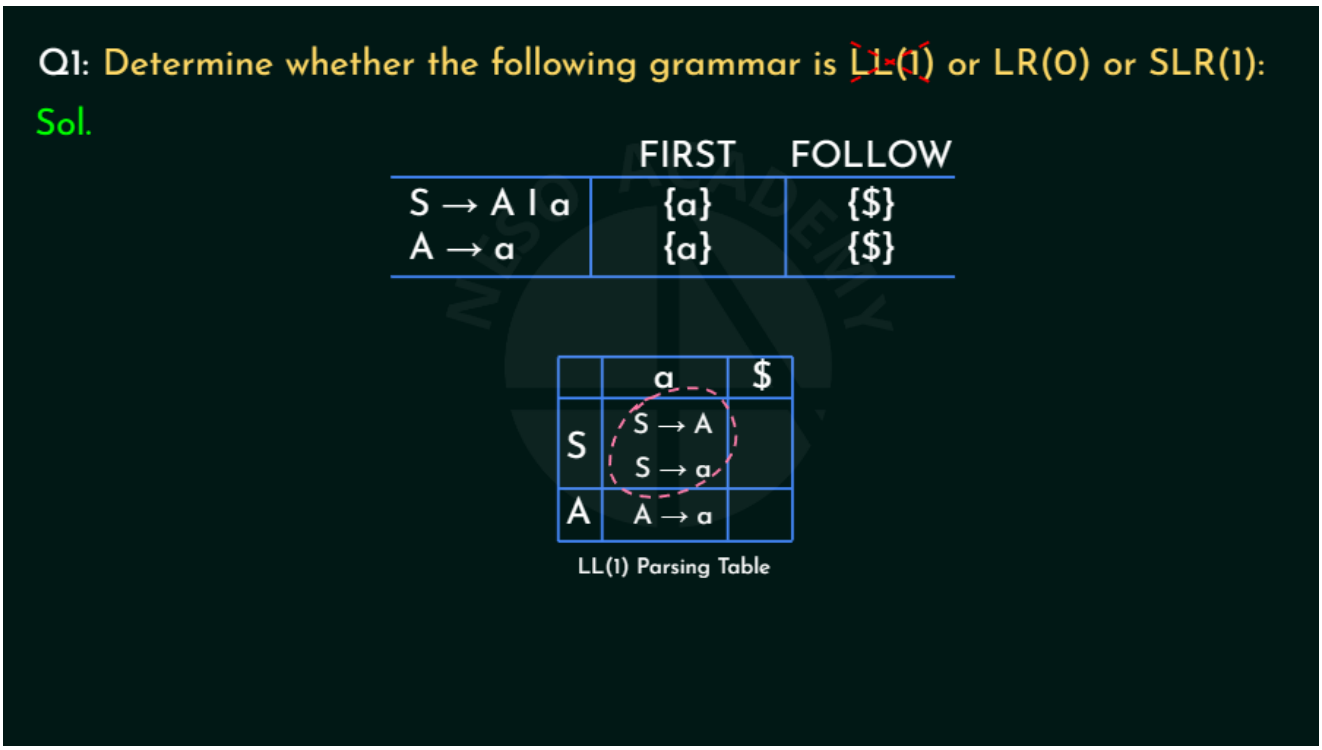
Compiler Design Determining the type of Grammar - Set 2



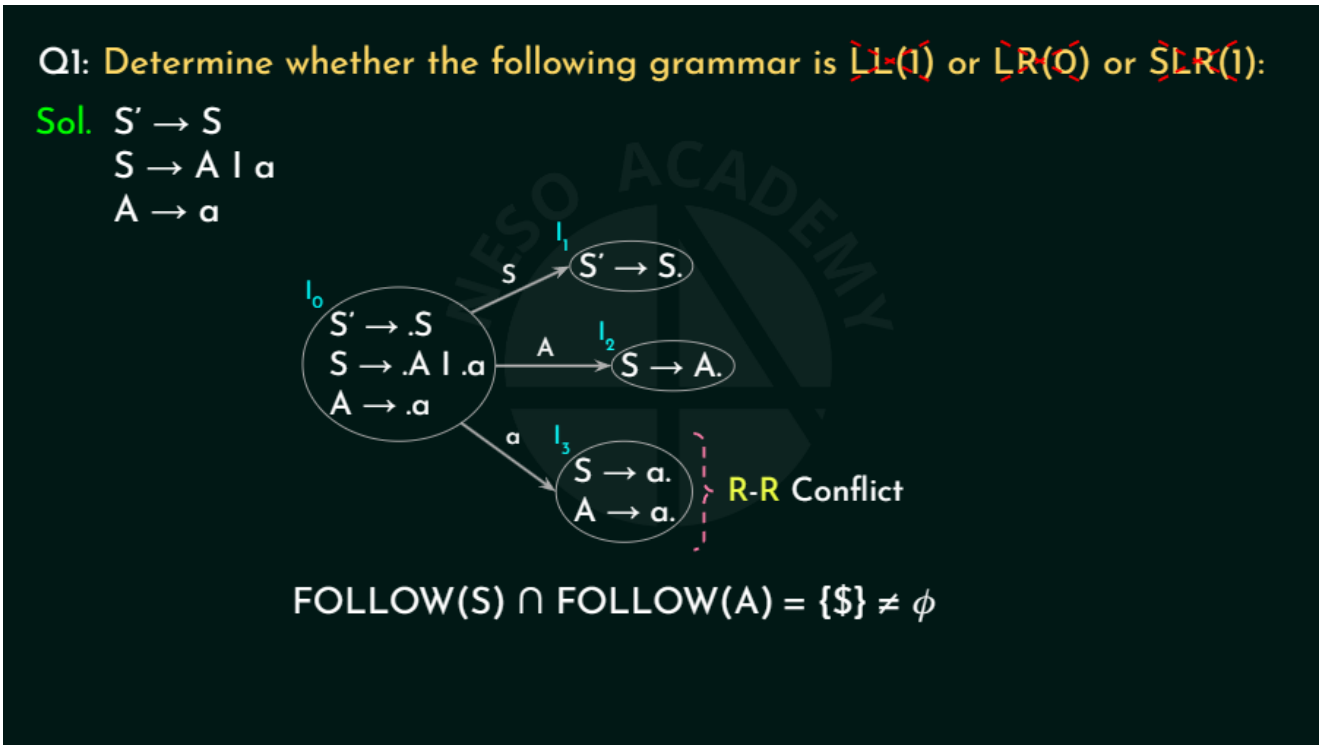
Outcome☆Two solved problems on how to determine the type of a given grammar.



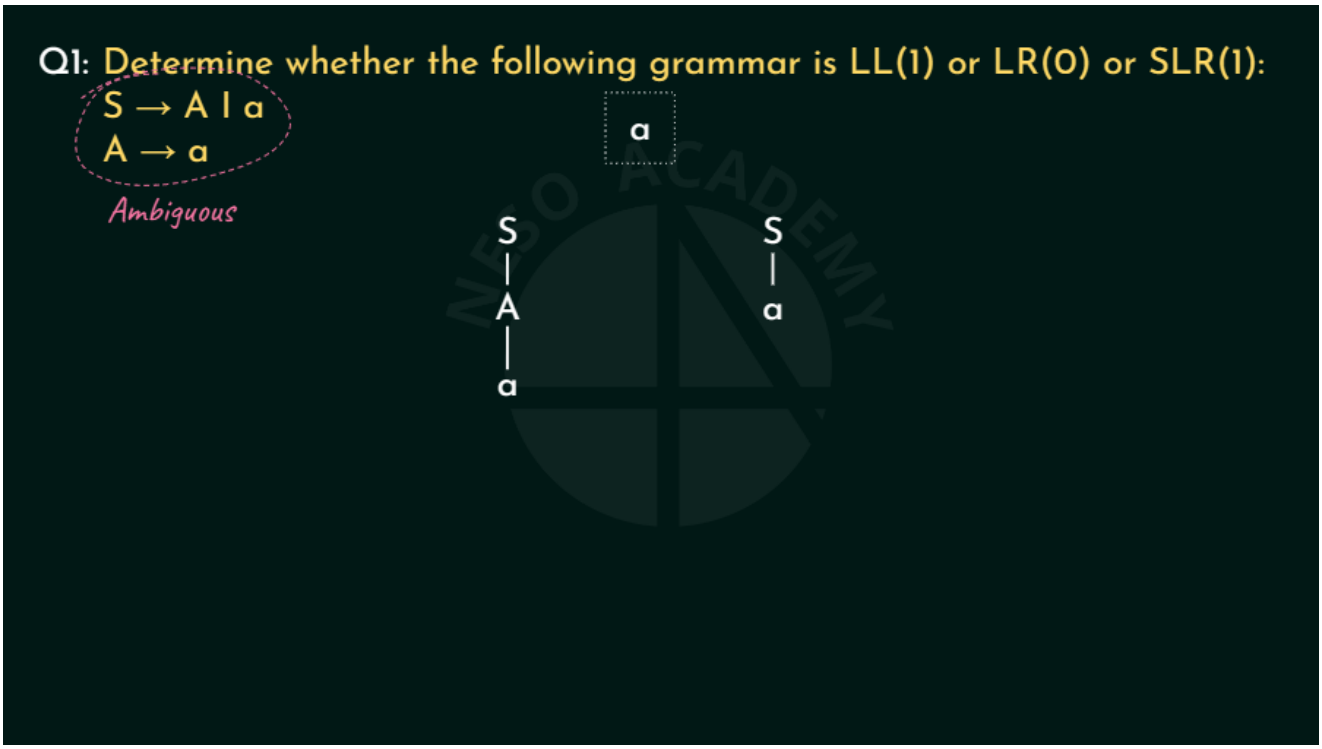
Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → A I a A → a



Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):LL(1) Parsing TableaSA$S → A I a A → aFOLLOW{$}FIRST{a}{$}{a}S → AS → aA → aSol.



I3I0Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):Sol.S' → SS → A I a A → aS' → .SS → .A I .a A → .aS → a. A → a.S' → S.I1S → A.I2SAaR-R ConflictFOLLOW(S) ∩ FOLLOW(A)= {$}≠ ϕ



Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → A I a A → aaS A a S a Ambiguous

Q2: Determine whether the following grammar is LL(1) or LR(0) or SLR(1):

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → (L) I a L → L,S I S

Q2: Determine whether the following grammar is LL(1) or LR(0) or SLR(1):

Sol.

	FIRST	FOLLOW
$S \rightarrow (L) \mid a$	{(, a}	{$, ,,)}
$L \rightarrow L,S \mid S$	{(, a}	{,,)}

	(	)	,	a	$
S	S → (L)			S → a	
L	L → S			L → S	

LL(1) Parsing Table

{$Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → (L) I a L → L,S I SFOLLOWFIRST{(, a}{,,)}{(, a}Sol., ,,)}LL(1) Parsing Table(aSL$,)S → (L)L → SL → SS → a

Q2: Determine whether the following grammar is LL(1) or LR(0) or SLR(1):

S → (L) I a

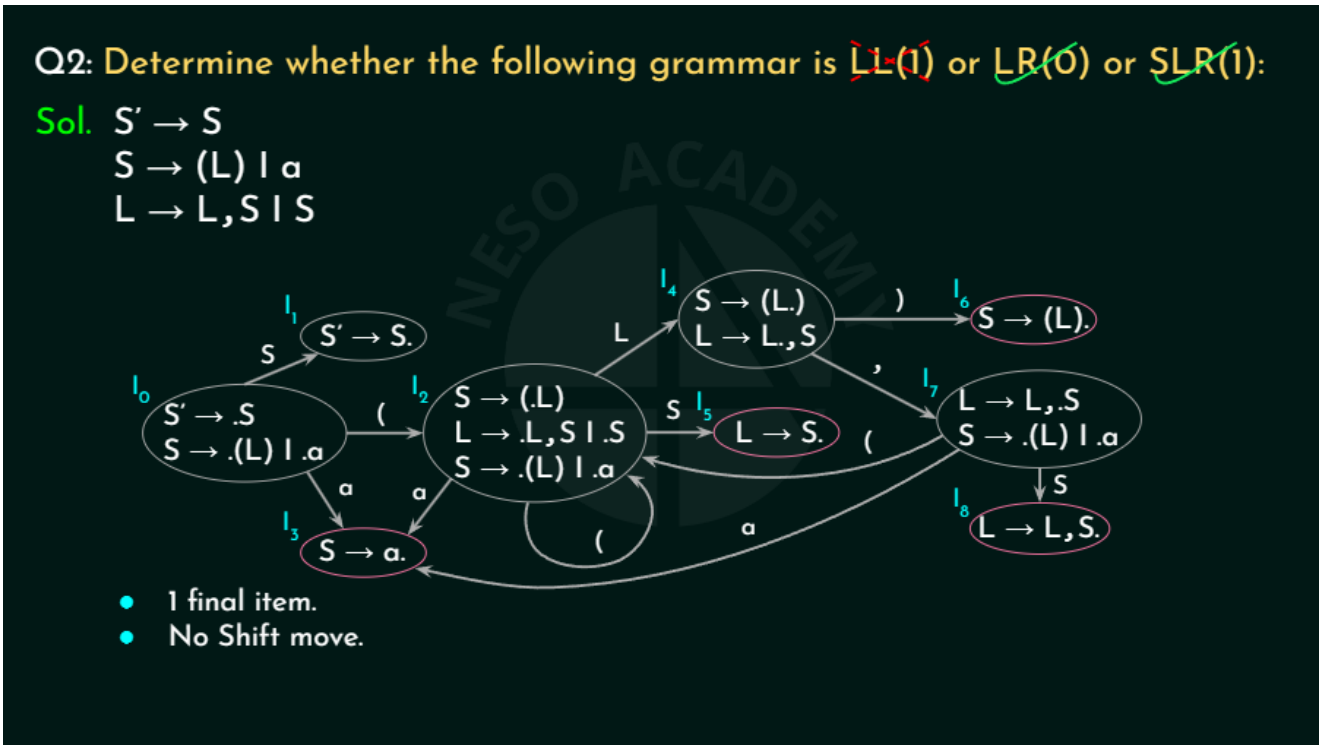
L → L,S I S

Top down Parser:

In order to construct Top down parsers the Context Free Grammars should not have,

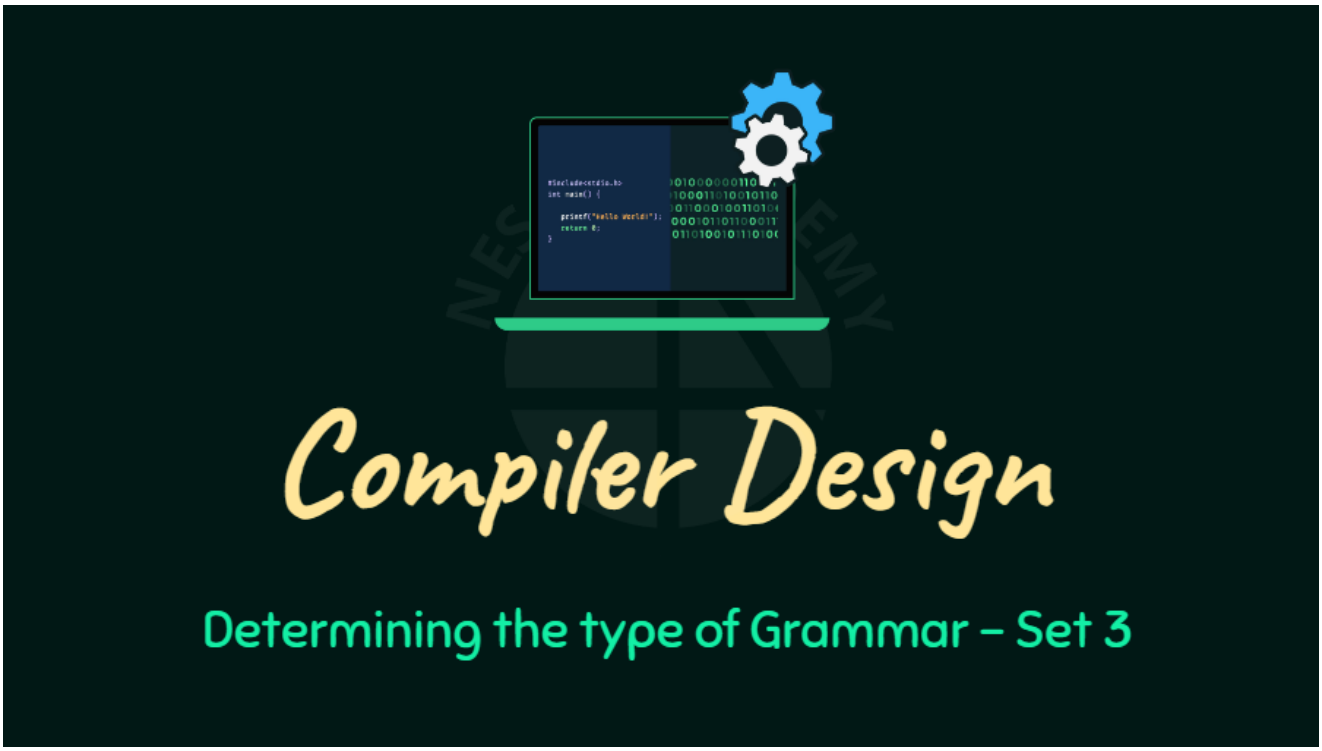
1. Left Recursion,
2. Non-determinism.

Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → (L) I a L → L,S I STop down Parser:In order to construct Top down parsers the Context Free Grammars should not have,1.Left Recursion,2.Non-determinism.

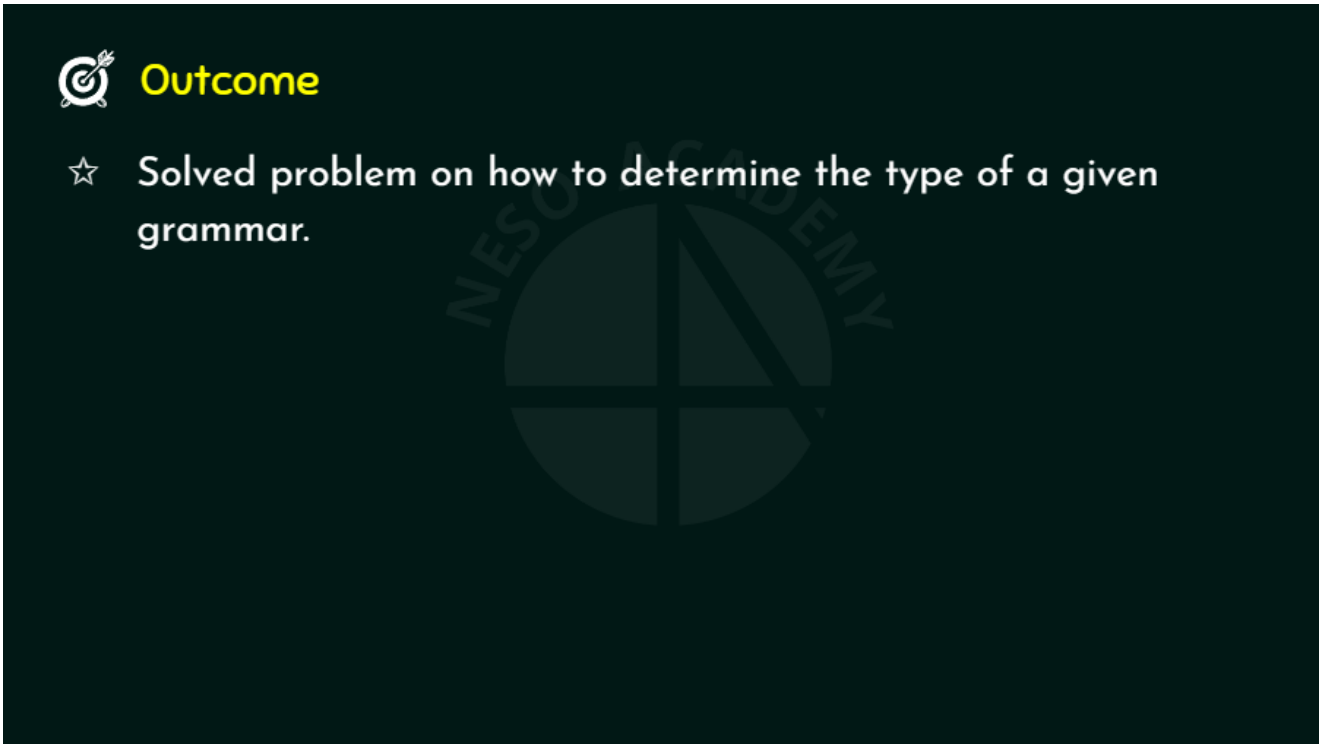


I7I2I0Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):Sol.S' → SS → (L) I a L → L,S I SS' → .SS → .(L) I .a S → (.L)L → .L,S I .SS → .(L) I .aS → a.S → (L.)L → L.,SL → L,.SS → .(L) I .a S' → S.I1S(I3aI4L → S.I5LSS → (L).I6a(),SL → L,S.I8(a●1 final

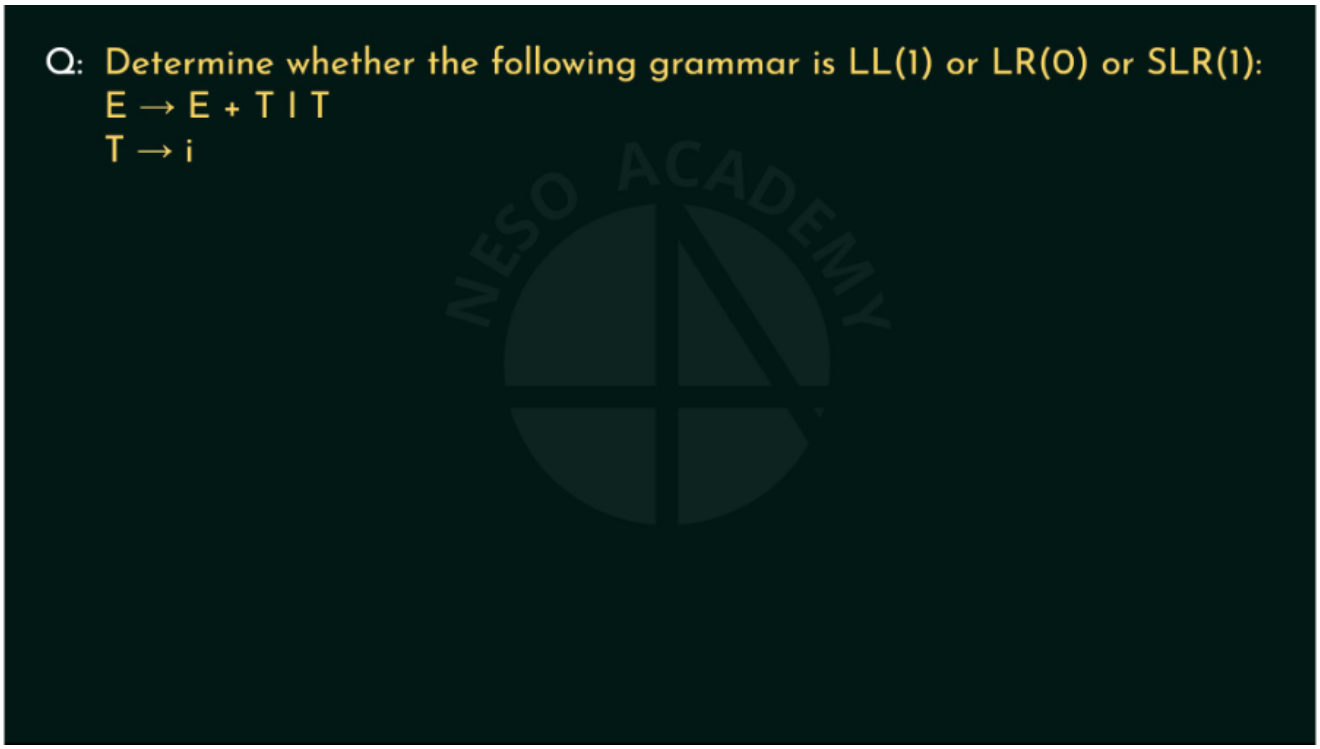
item.●No Shift move.

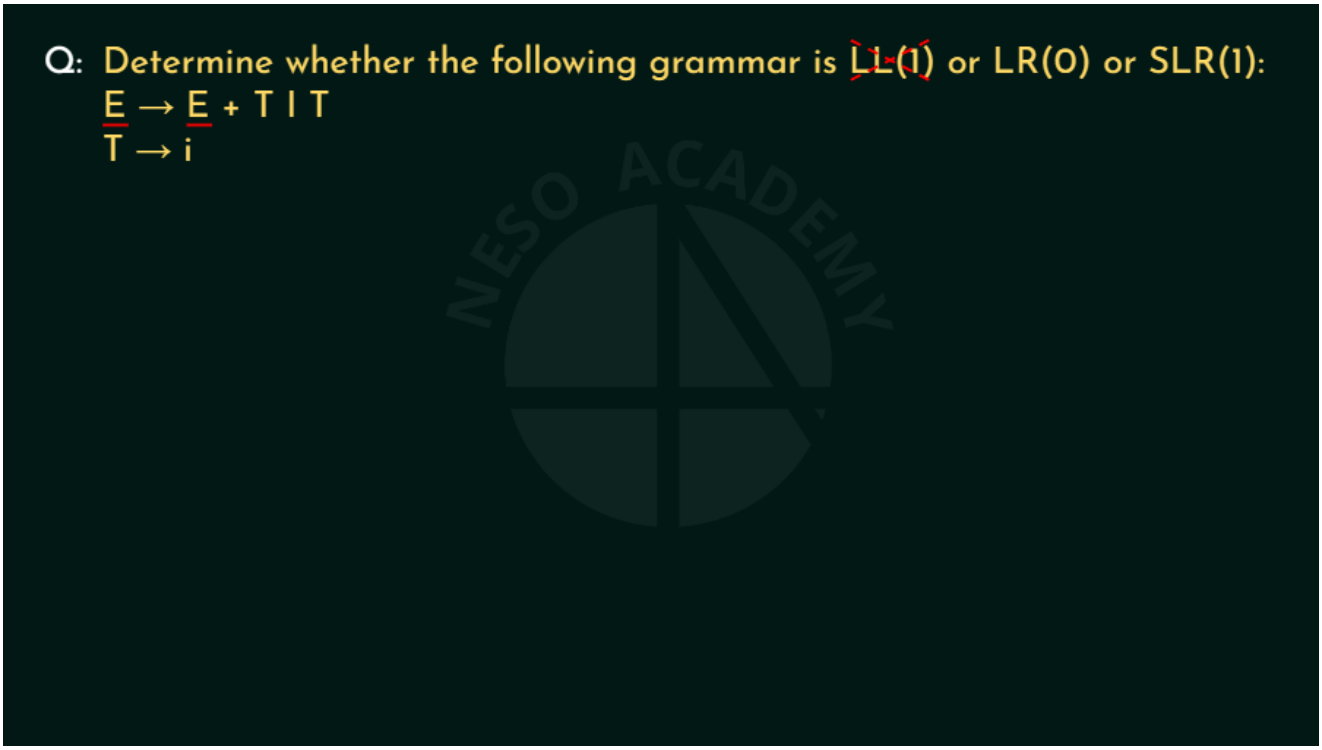


Compiler Design Determining the type of Grammar - Set 3

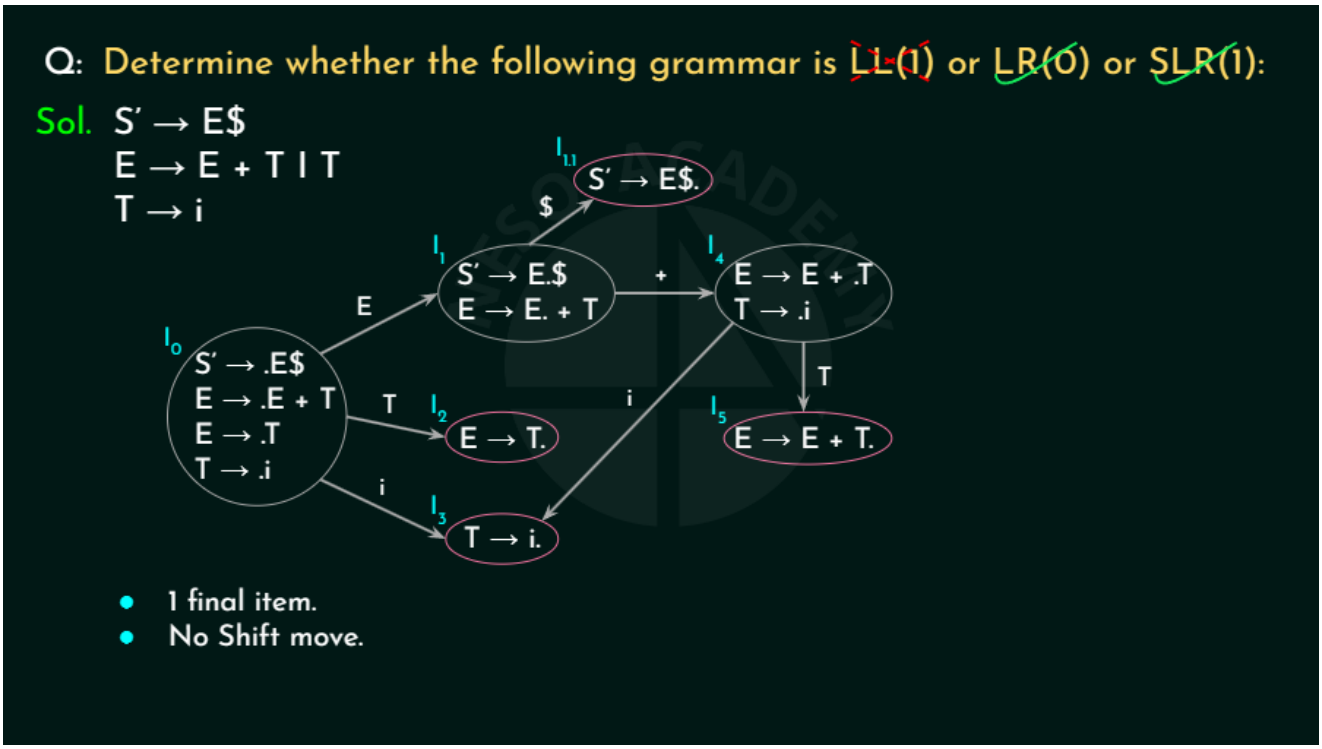


Outcome☆Solved problem on how to determine the type of a given grammar.

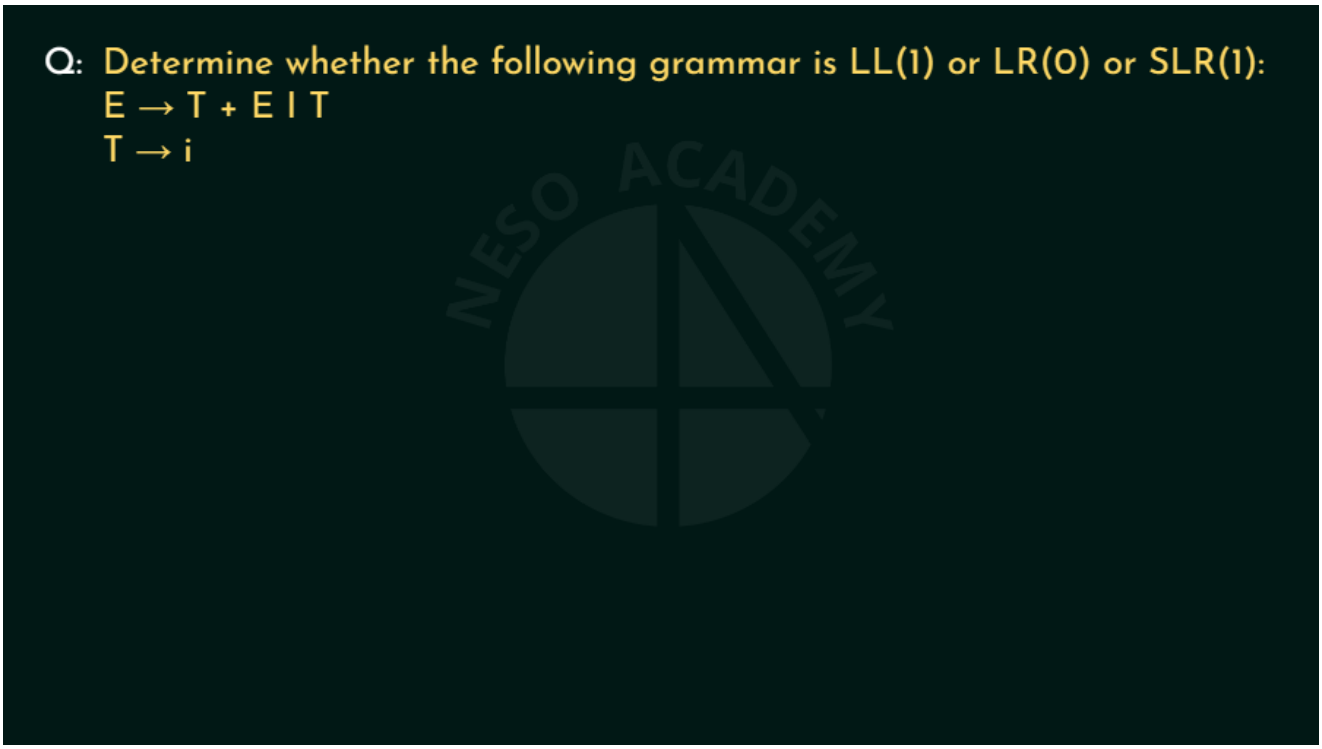




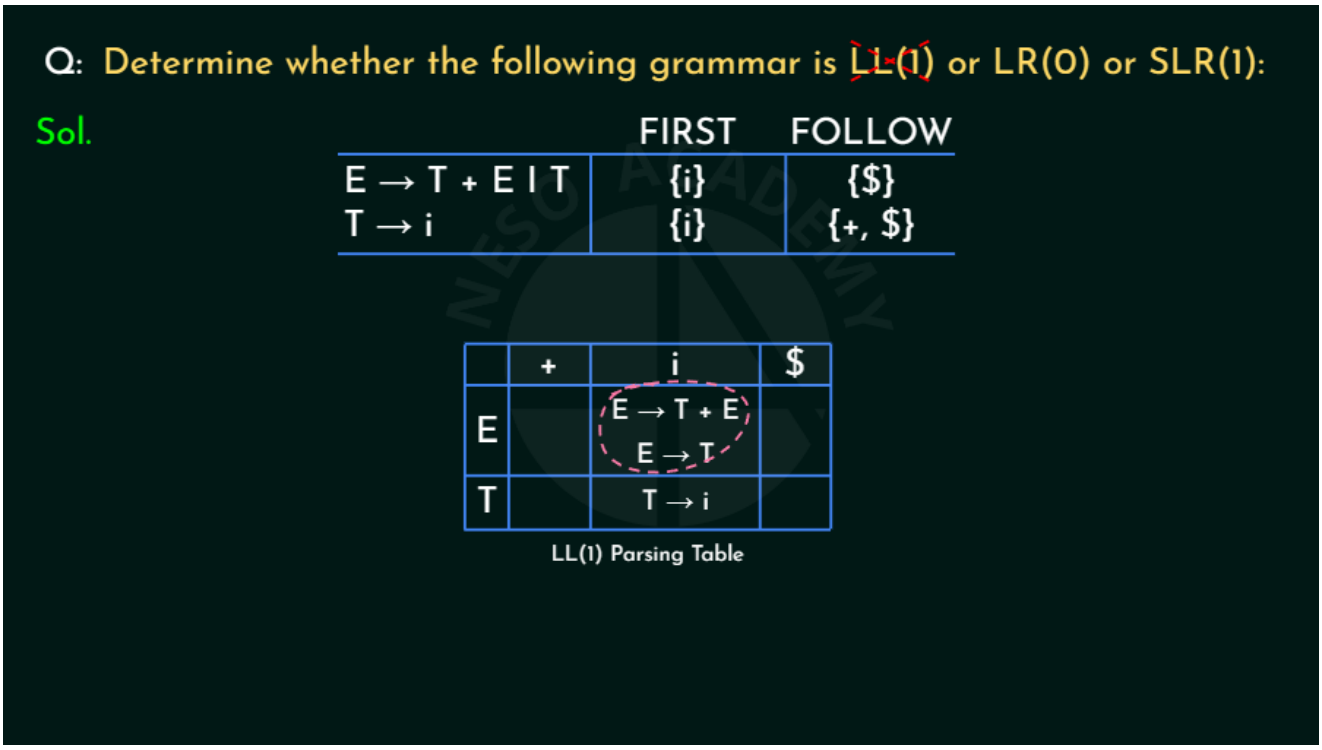
Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):E → E + T I TT → i



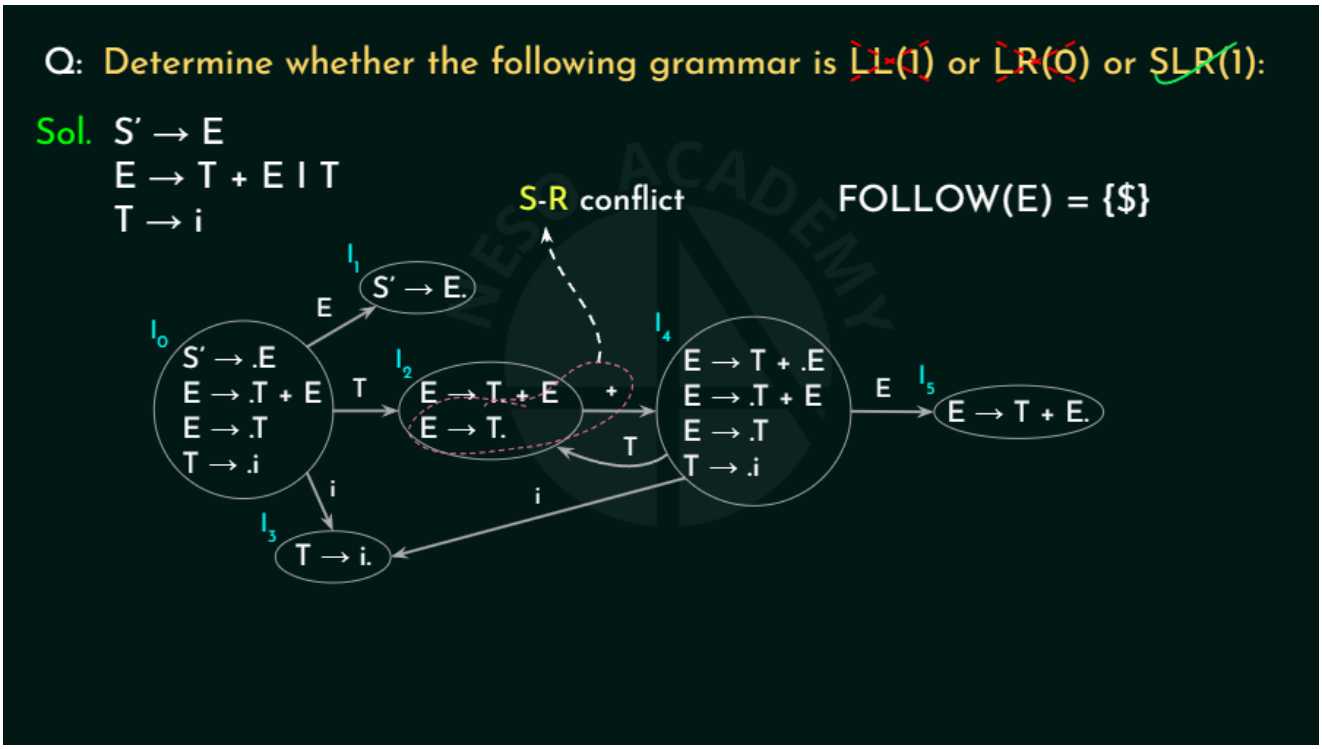
I4I1Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):Sol.S' → E$E → E + T I T T → iS' → .E$E → .E + T E → .T T → .iI0S' → E.$E → E. + TE → E + .TT → .iE → T.I2T → i.I3EE → E + T.I5TS' → E$.I1.1$Ti+i●1 final item.●No Shift move.



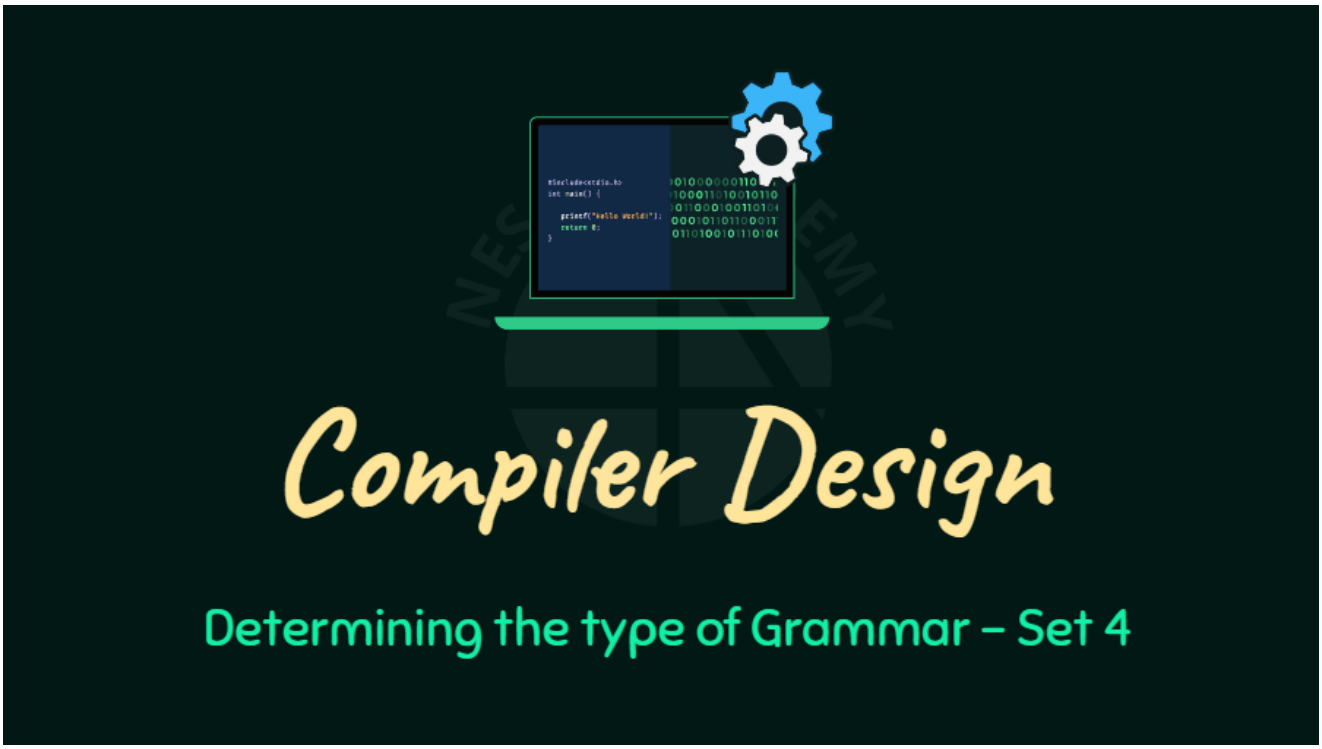
Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):E → T + E I T T → i



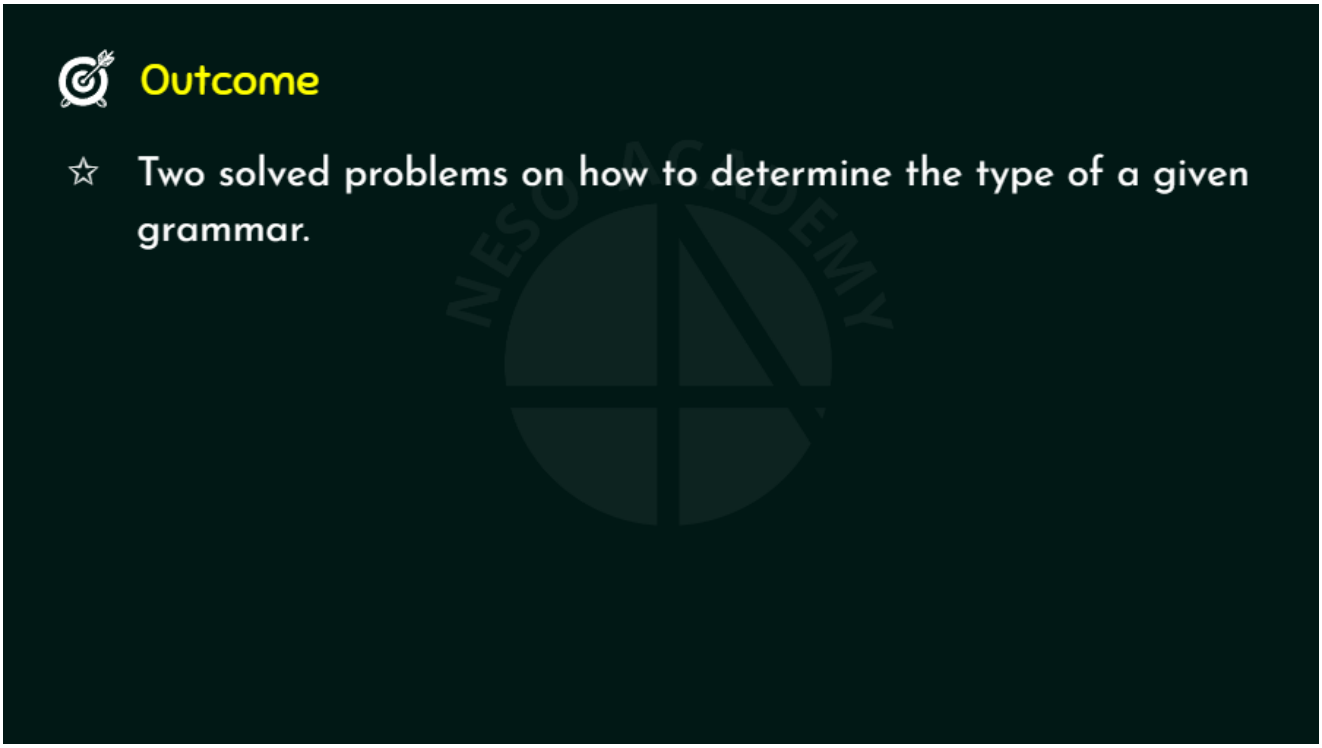
Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):E → T + E I T T → iFOLLOW{$}FIRST{i}{+, $}{i}Sol.E → T + EE → TT → iLL(1) Parsing Table+ET$i



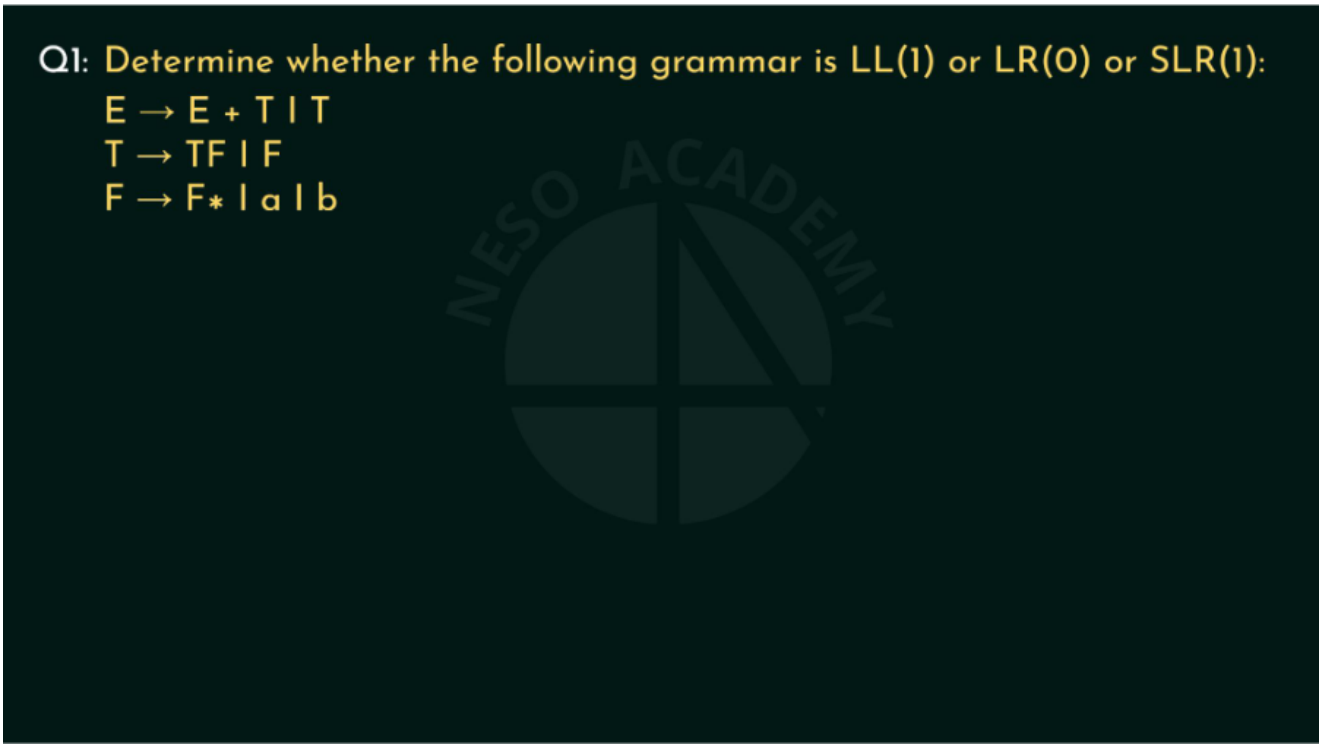
Q:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → EE → T + E I T T → iSol.S' → .EE → .T + E E → .T T → .iE → T. + E E → T. E → T + .E E → .T + E E → .T T → .i I0S' → E.I1I2T → i.I3iTEI4+TiE → T + E. I5ES-R conflictFOLLOW(E)= {$}

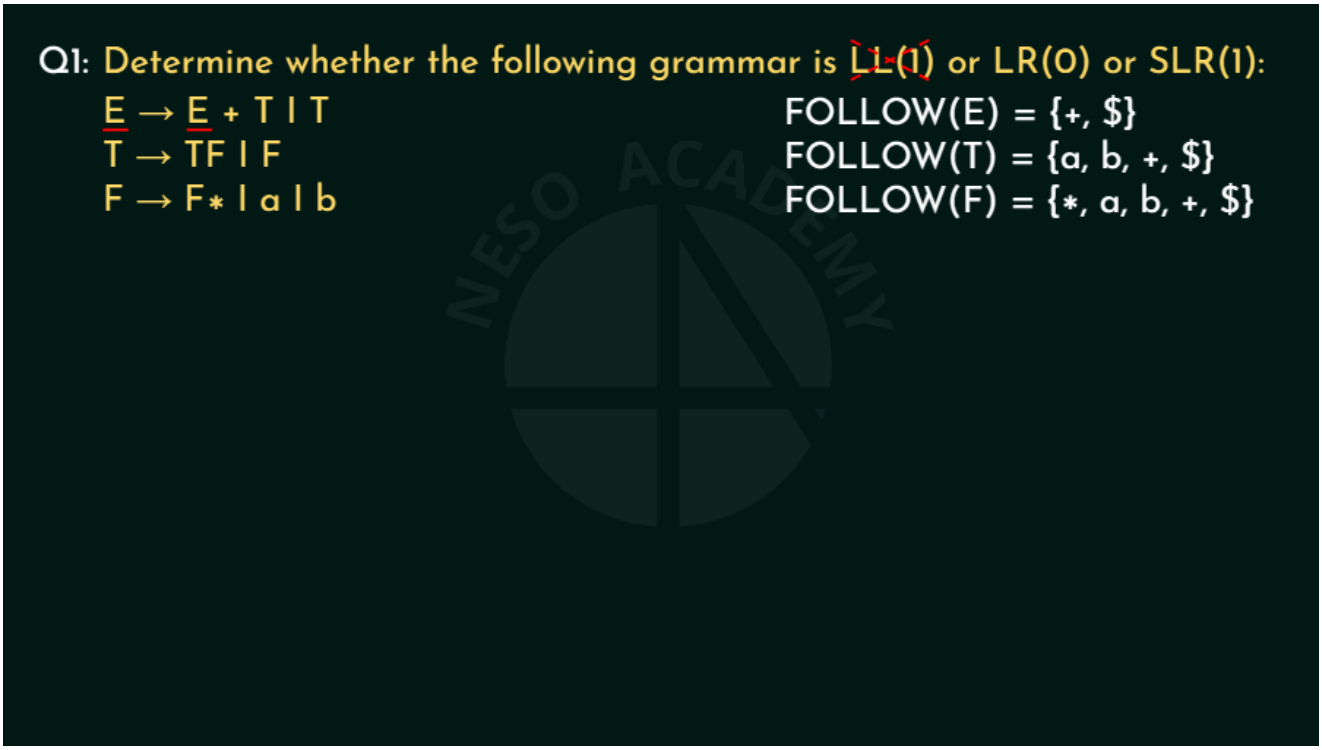


Compiler Design Determining the type of Grammar - Set 4

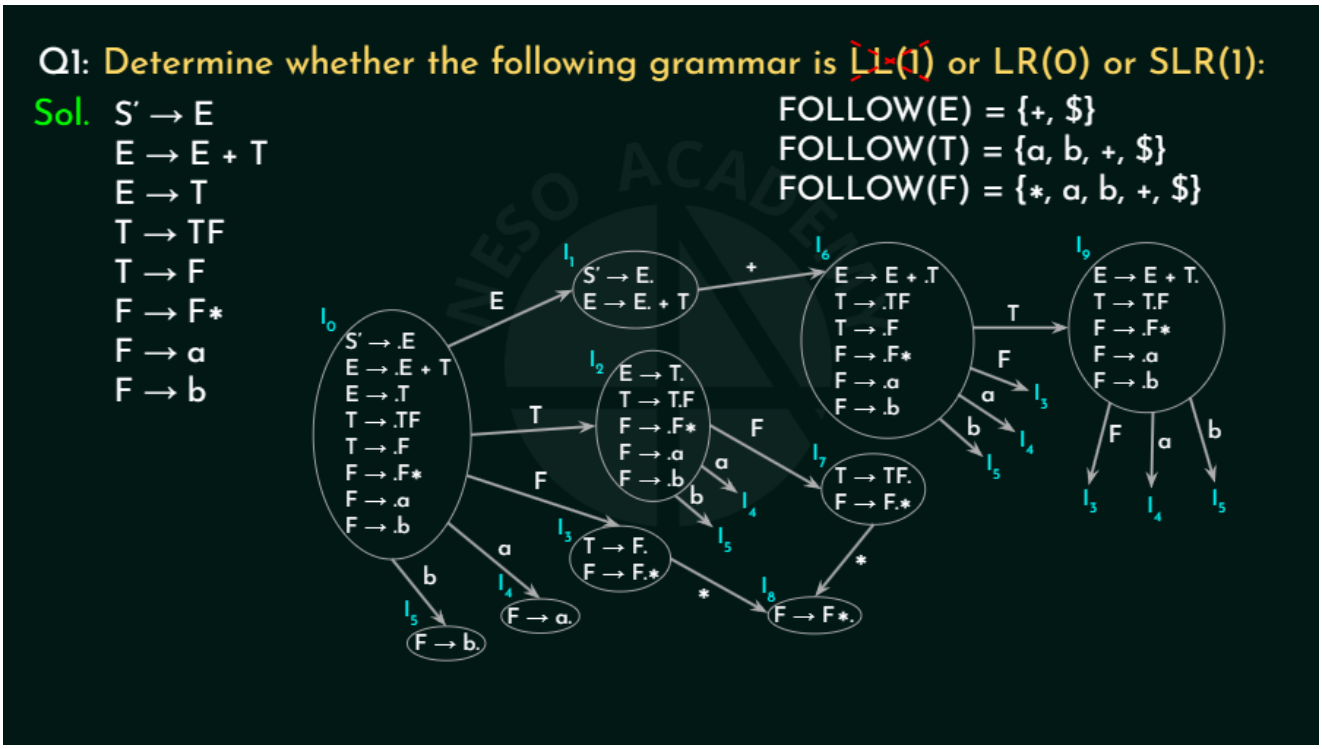


Outcome☆Two solved problems on how to determine the type of a given grammar.

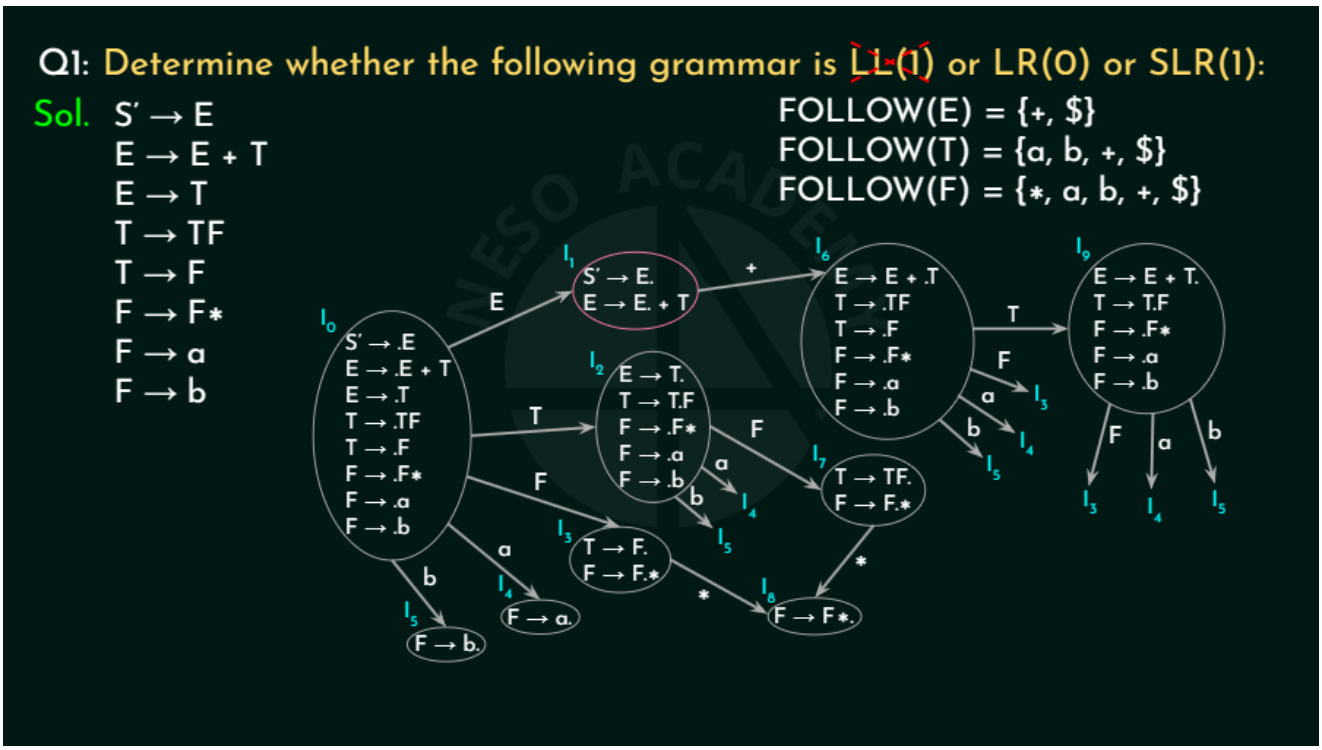




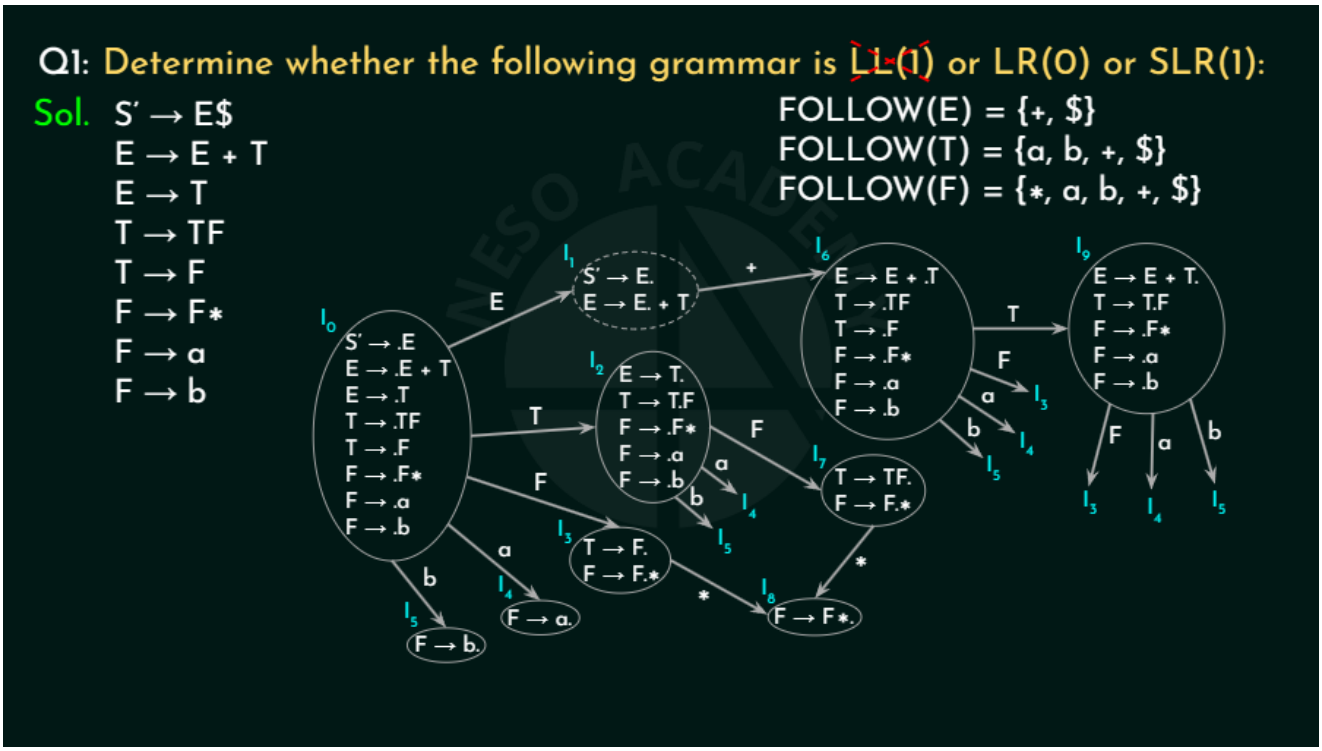
Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):E → E + T I TT → TF I FF → F+ I a I bFOLLOW(E) = {+, $}FOLLOW(T)= {a, b, +, $}FOLLOW(F), a, b, +, $}= {+**



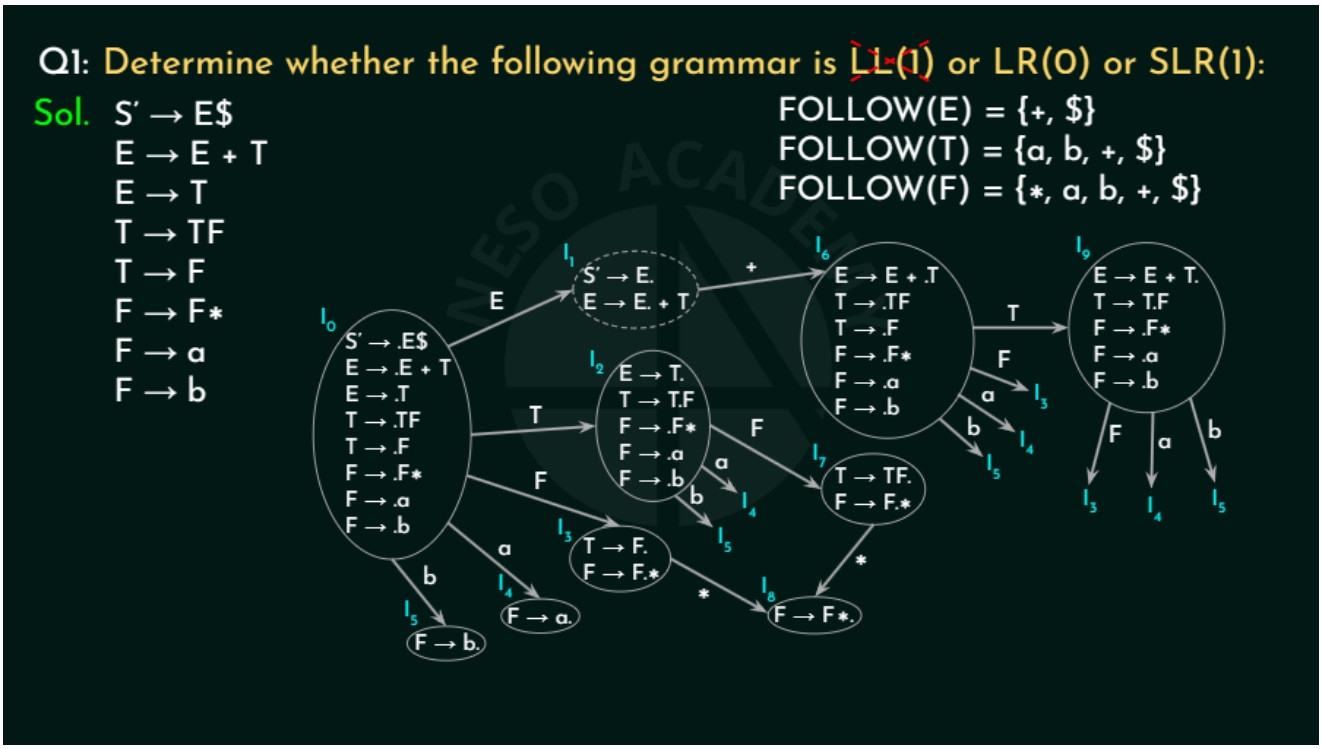
I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → EE → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S' → .EE → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S' → E.E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7I2FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baF



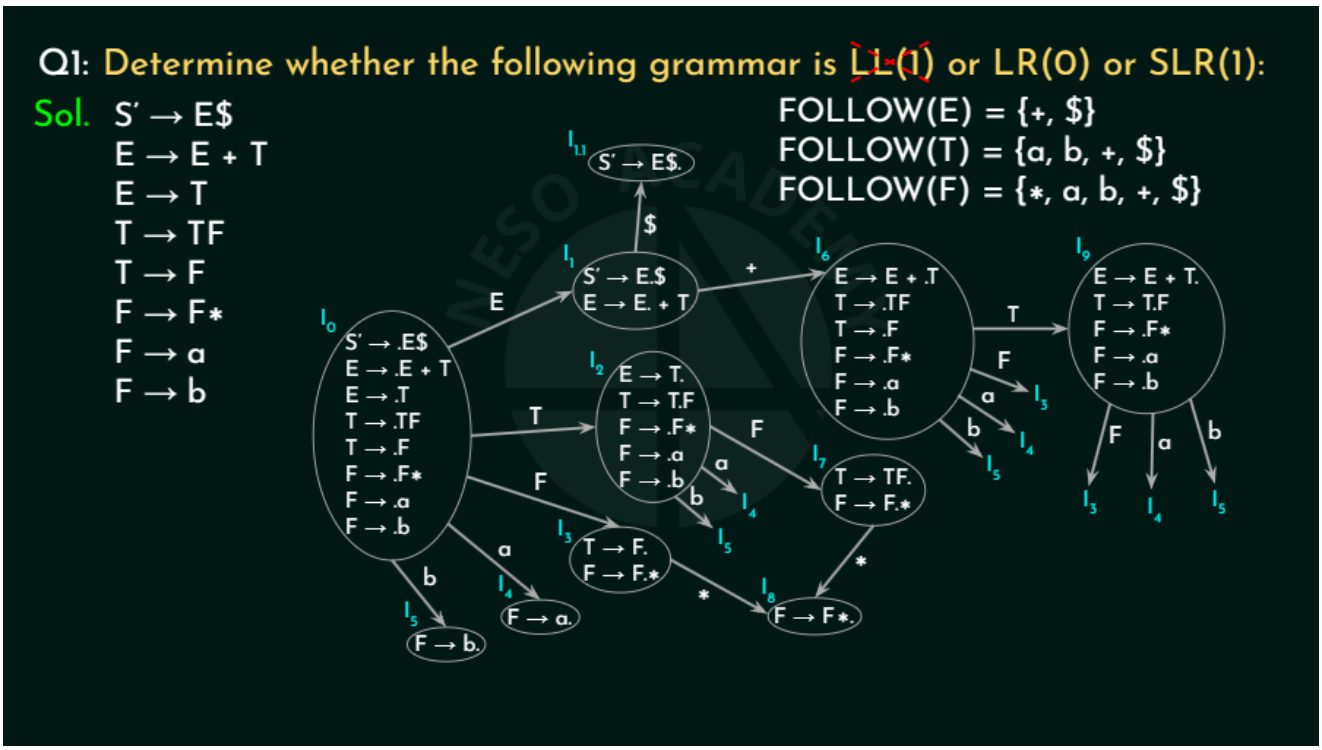
I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → EE → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .EE → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7I2FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baF



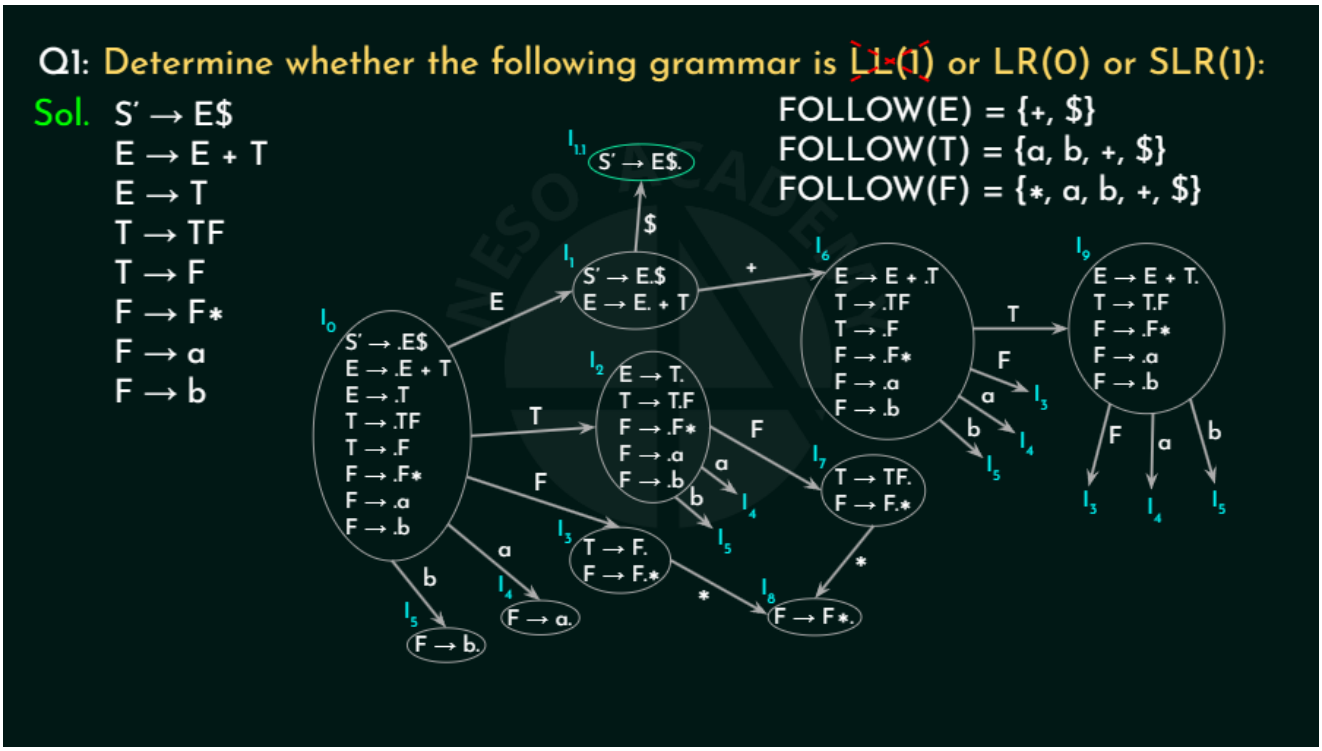
I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .EE → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7I2FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baF



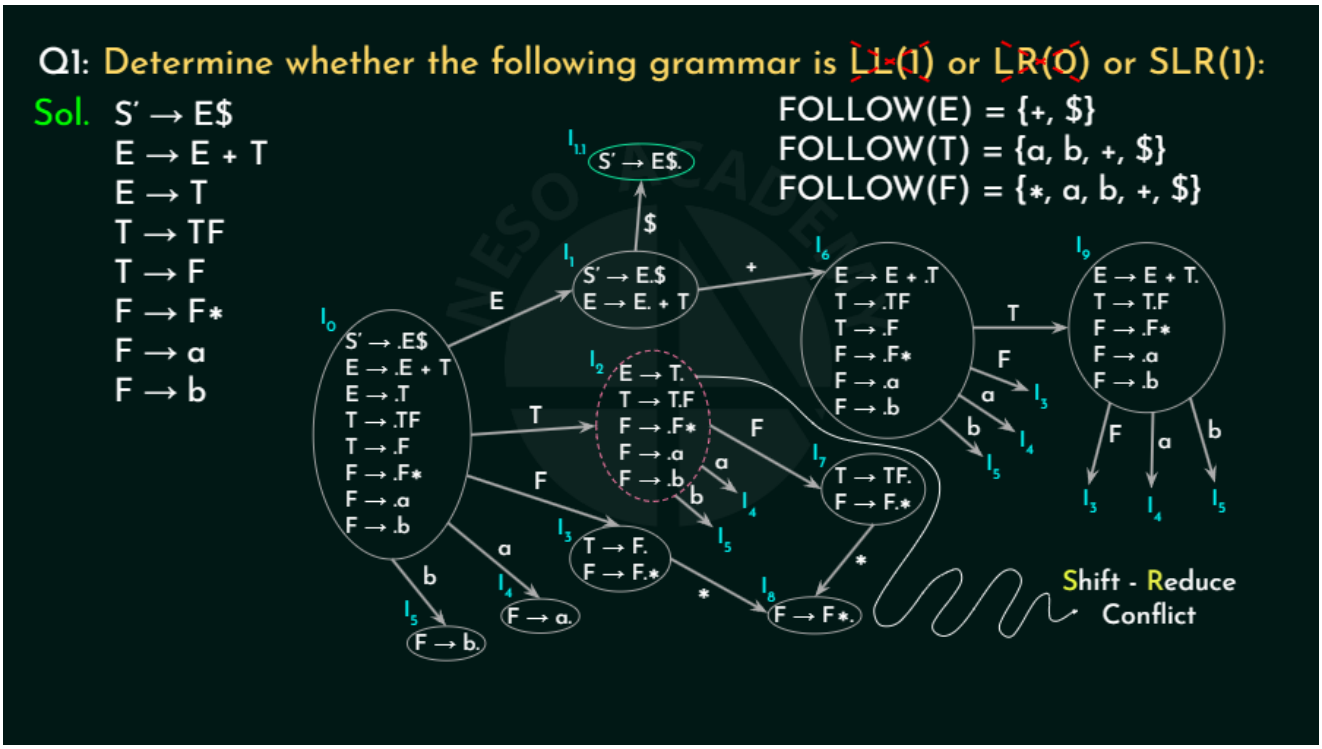
I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S' → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S' → E.E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7I2FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baF



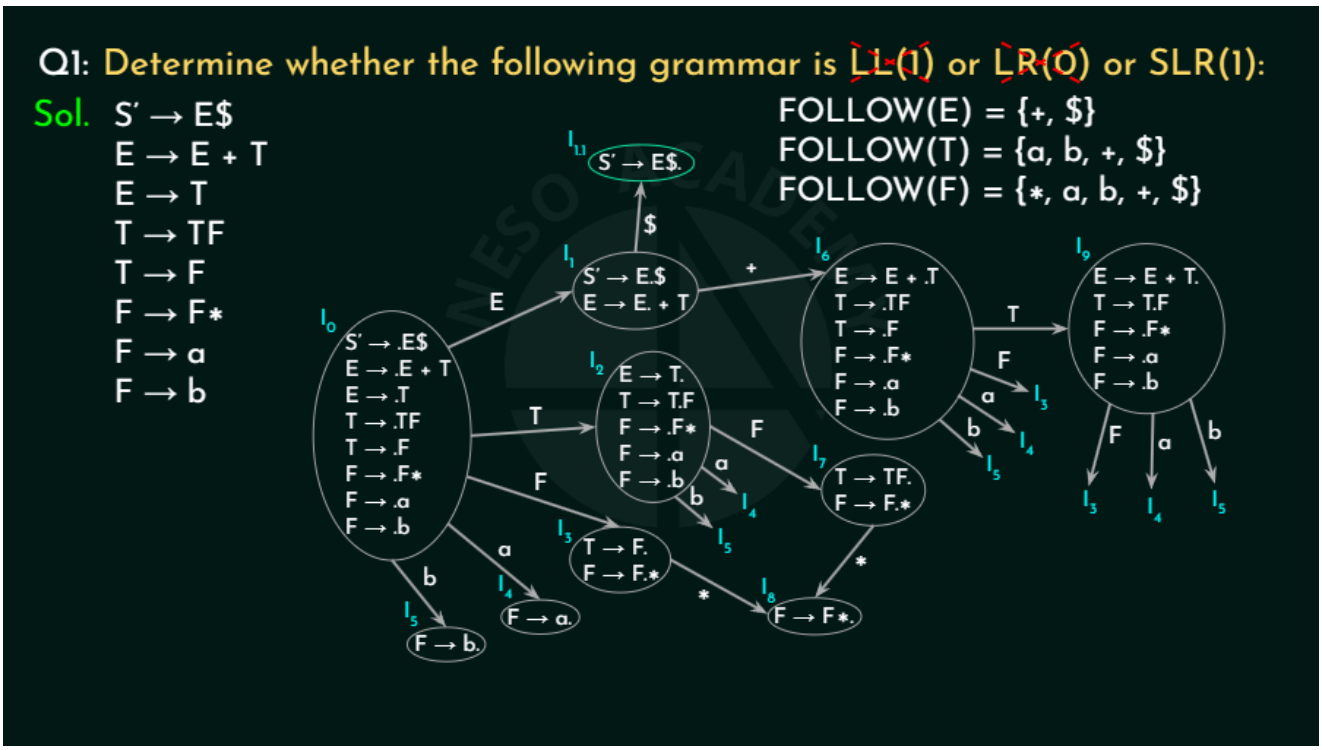
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



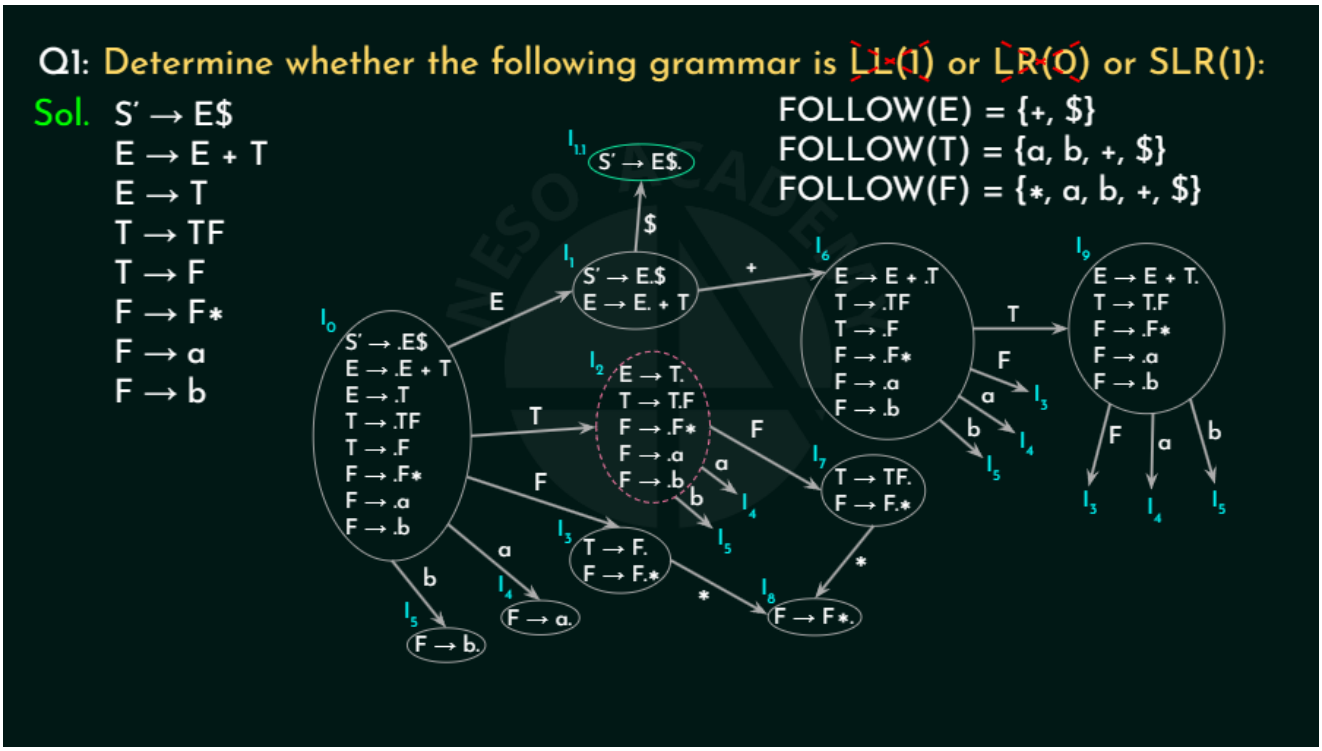
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



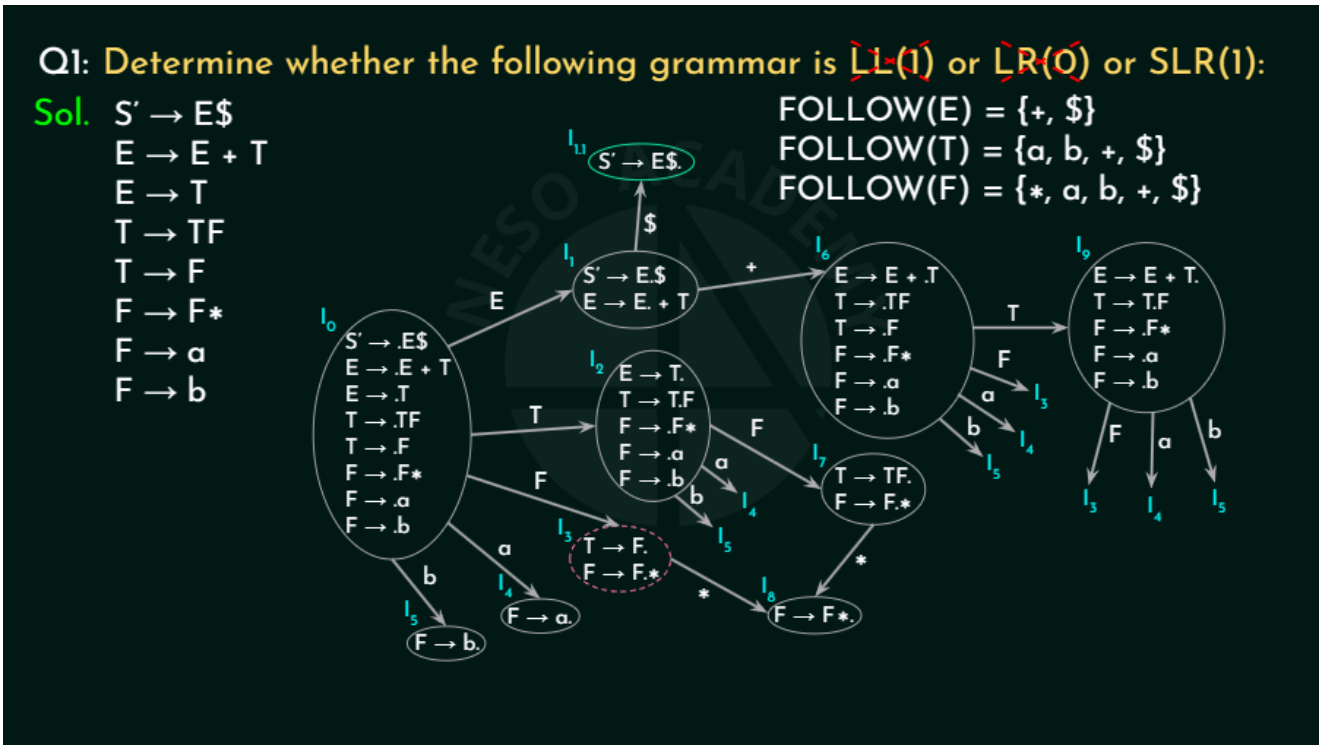
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S' → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S' → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS' → E$.I1.1$Shift - Reduce Conflict



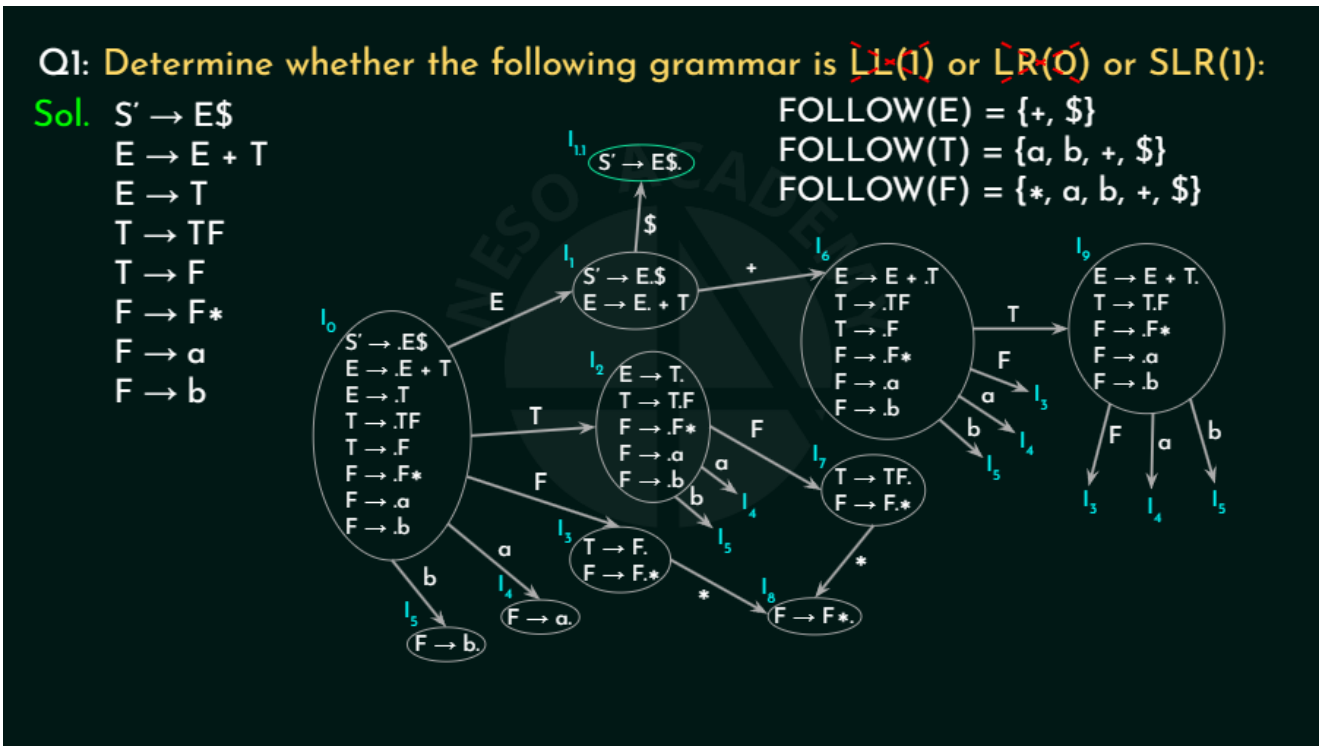
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



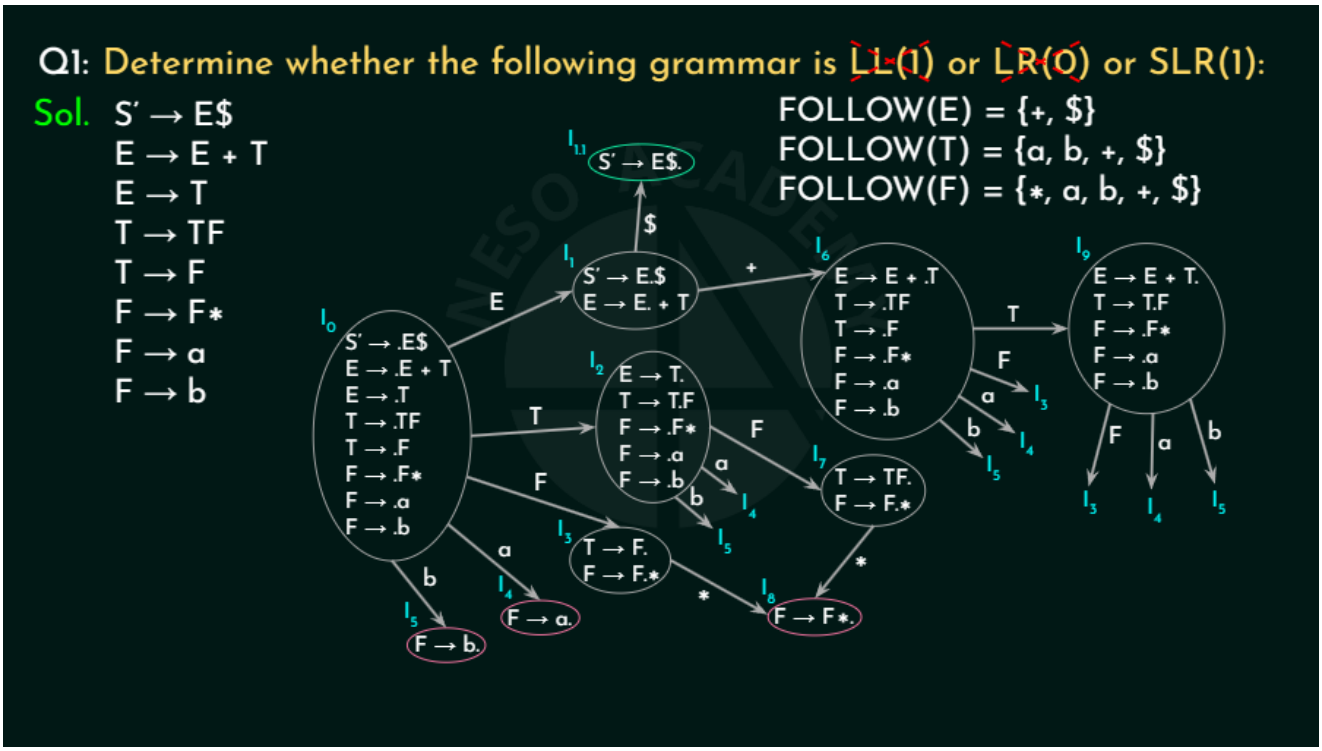
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



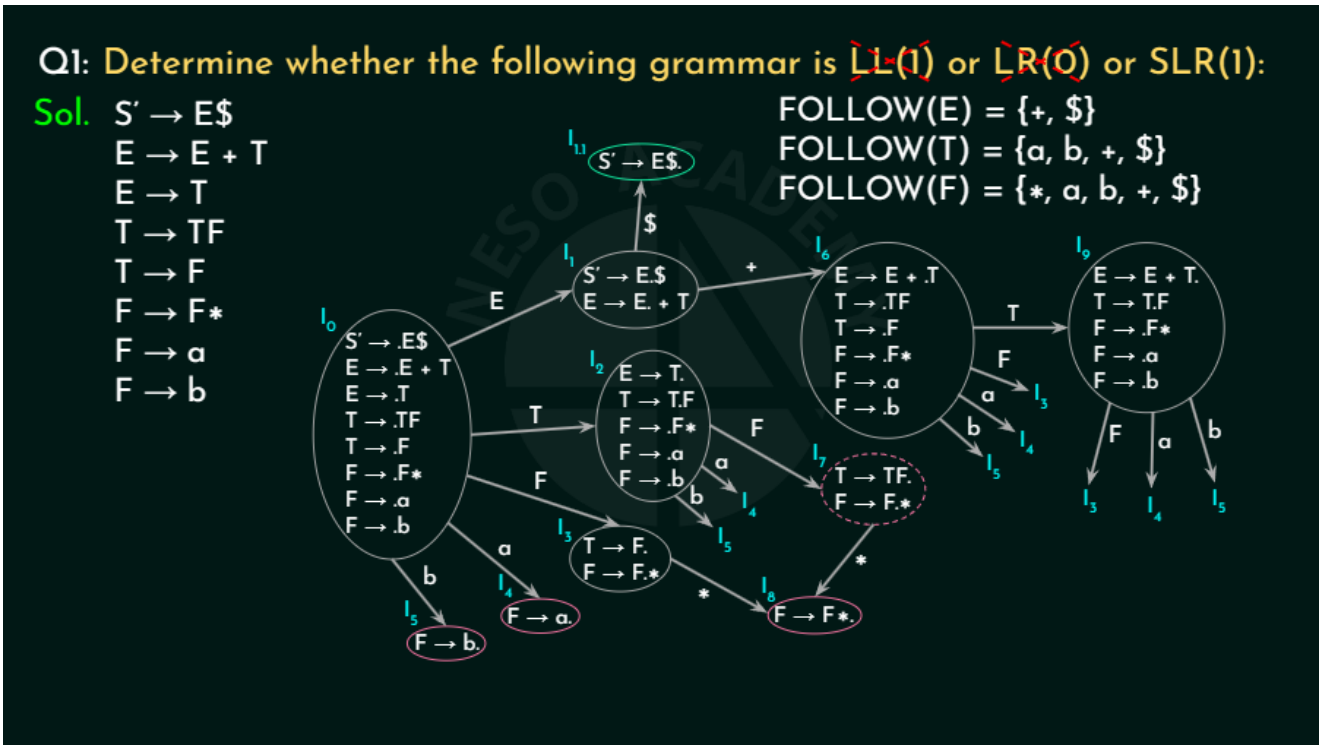
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S' → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S' → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS' → E$.I1.1$



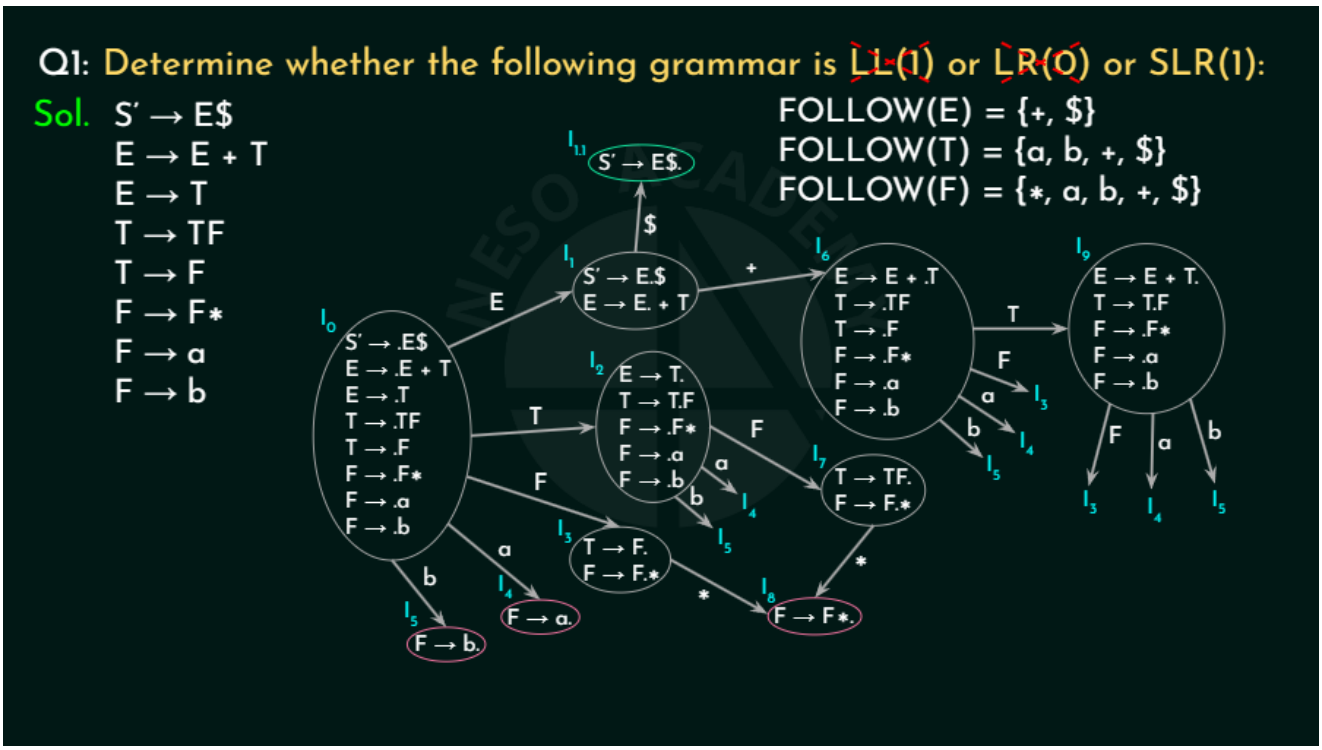
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



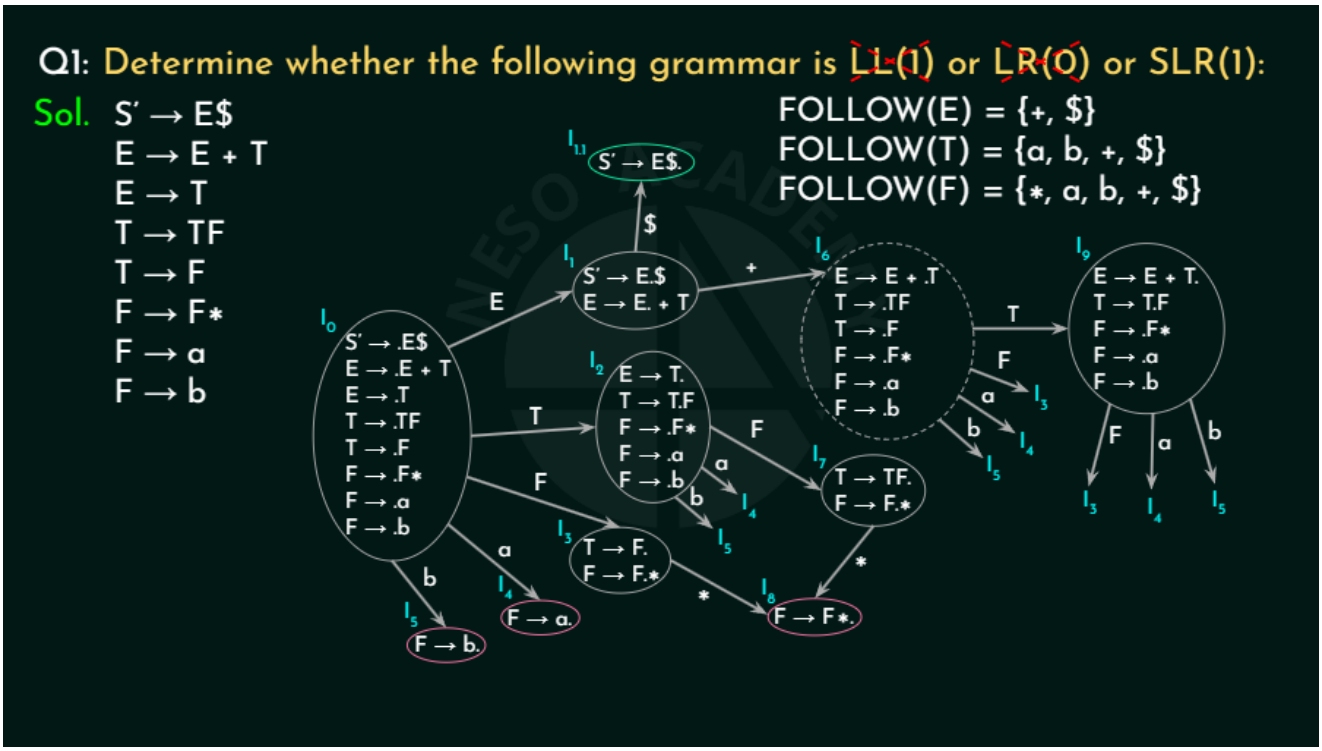
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



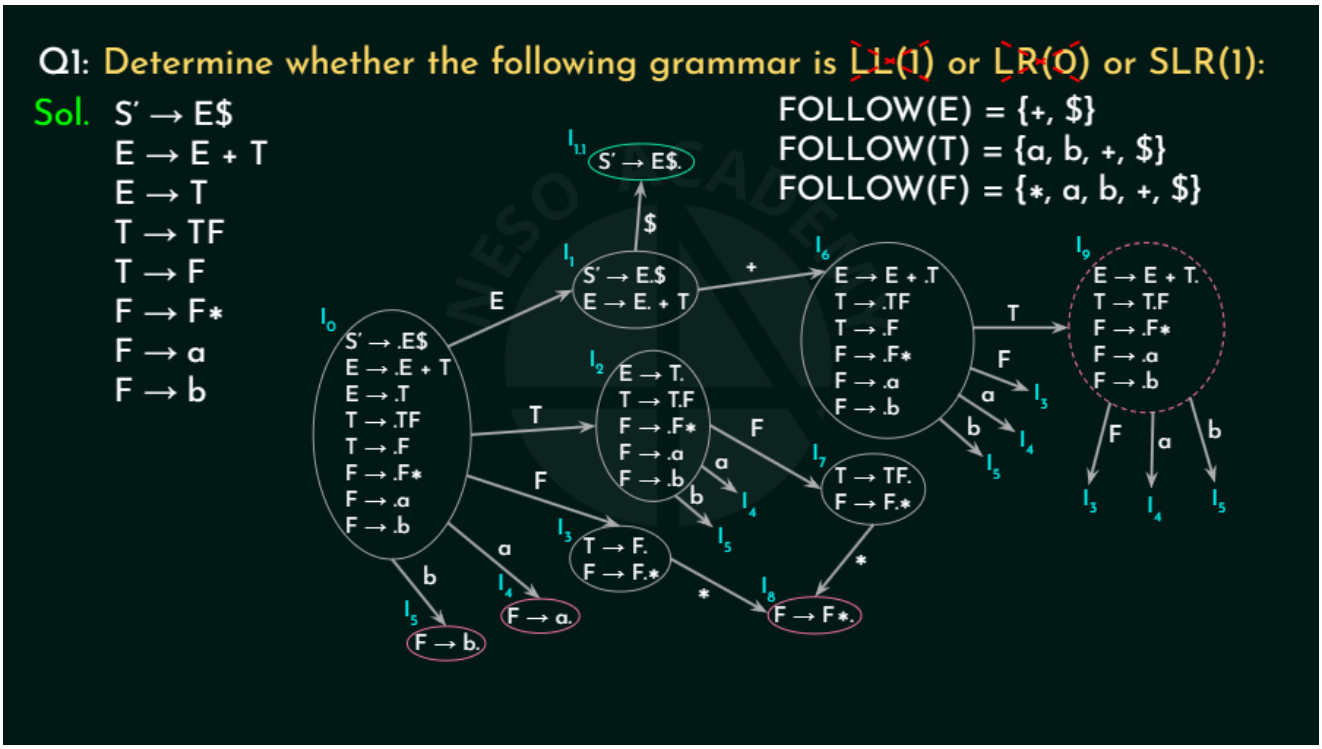
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S' → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S' → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS' → E$.I1.1$



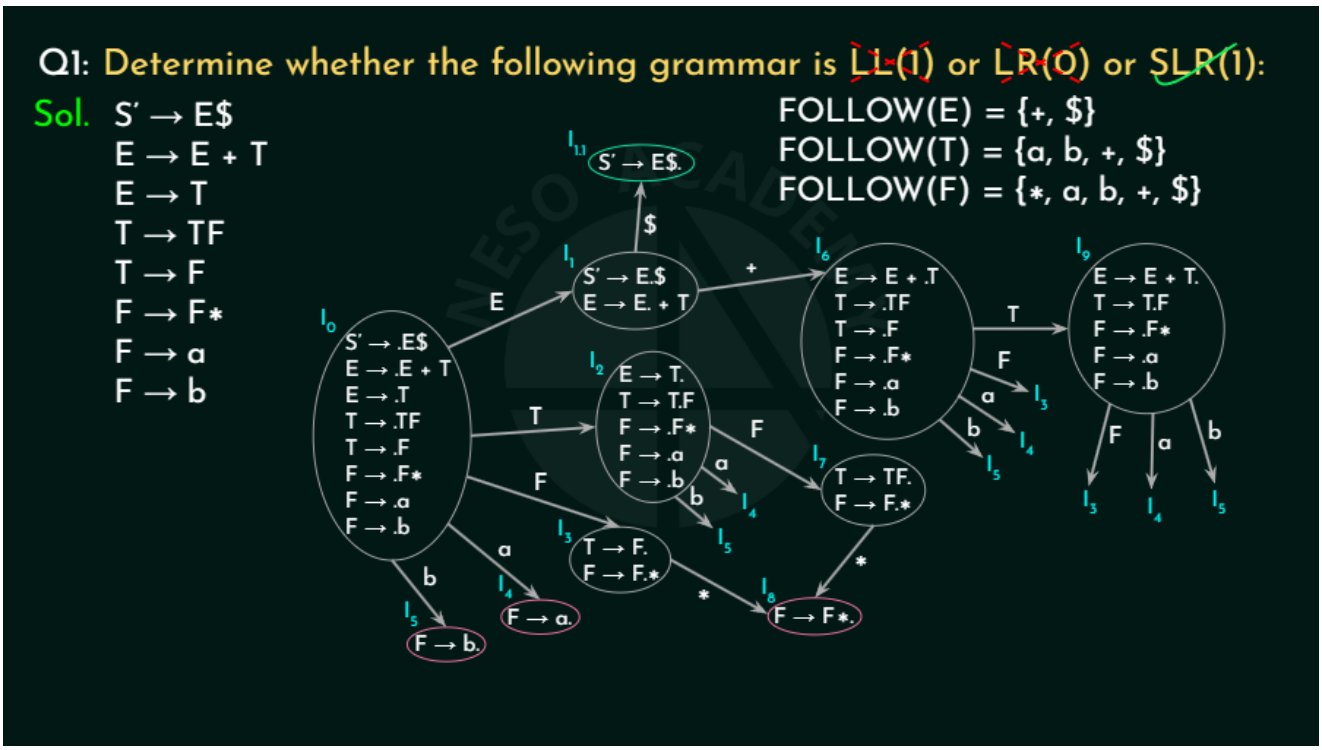
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



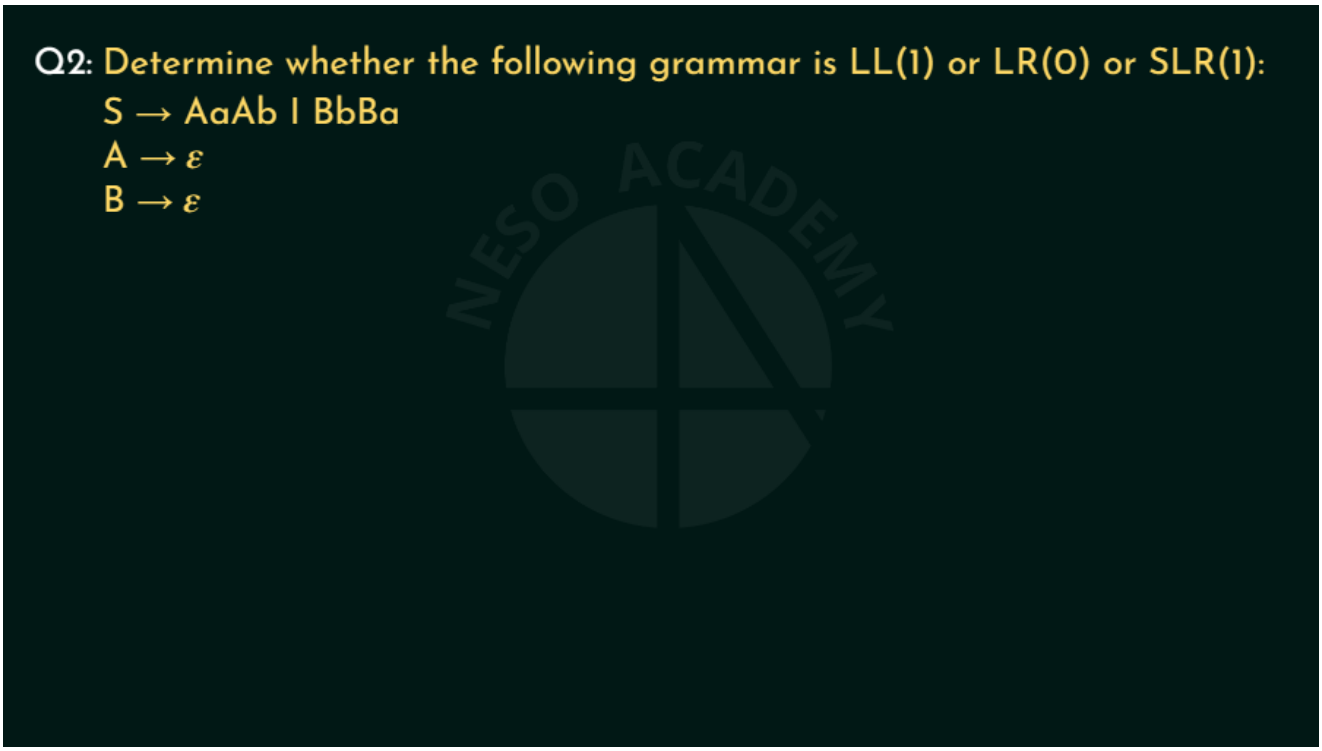
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



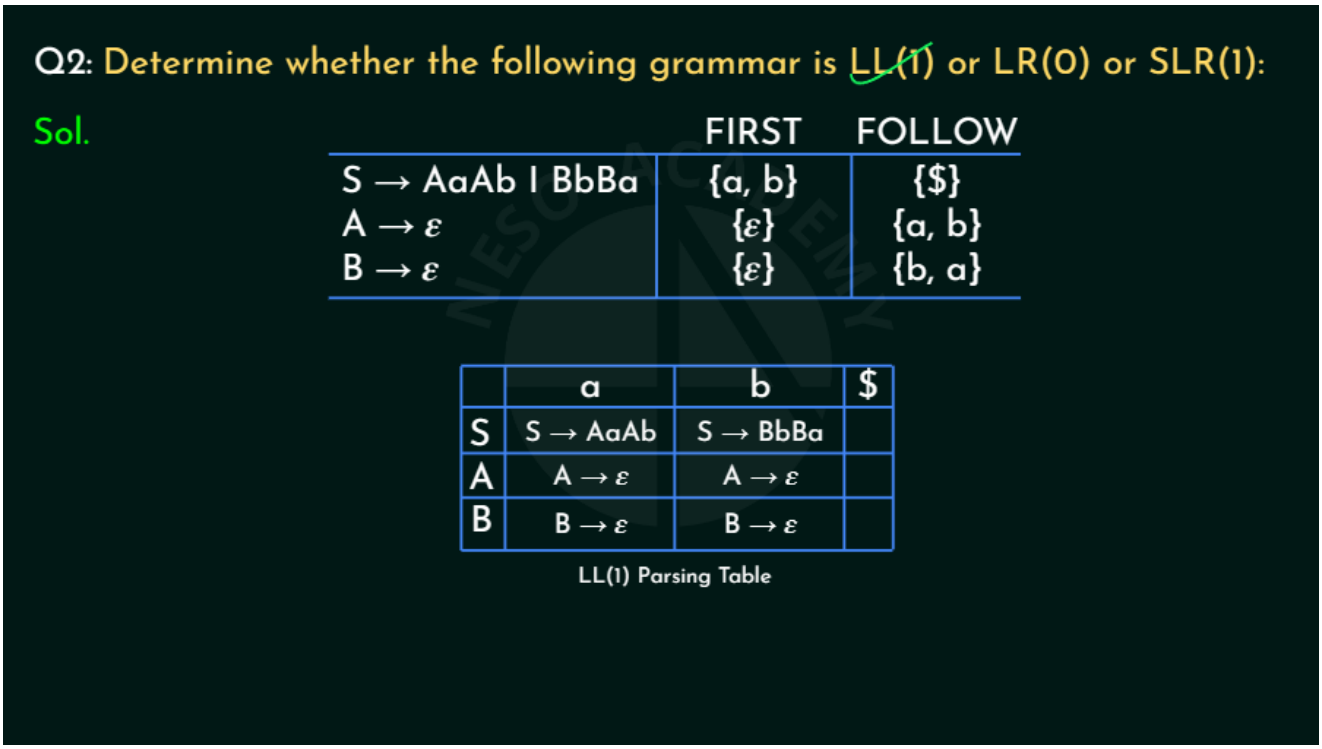
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S' → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S' → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS' → E$.I1.1$



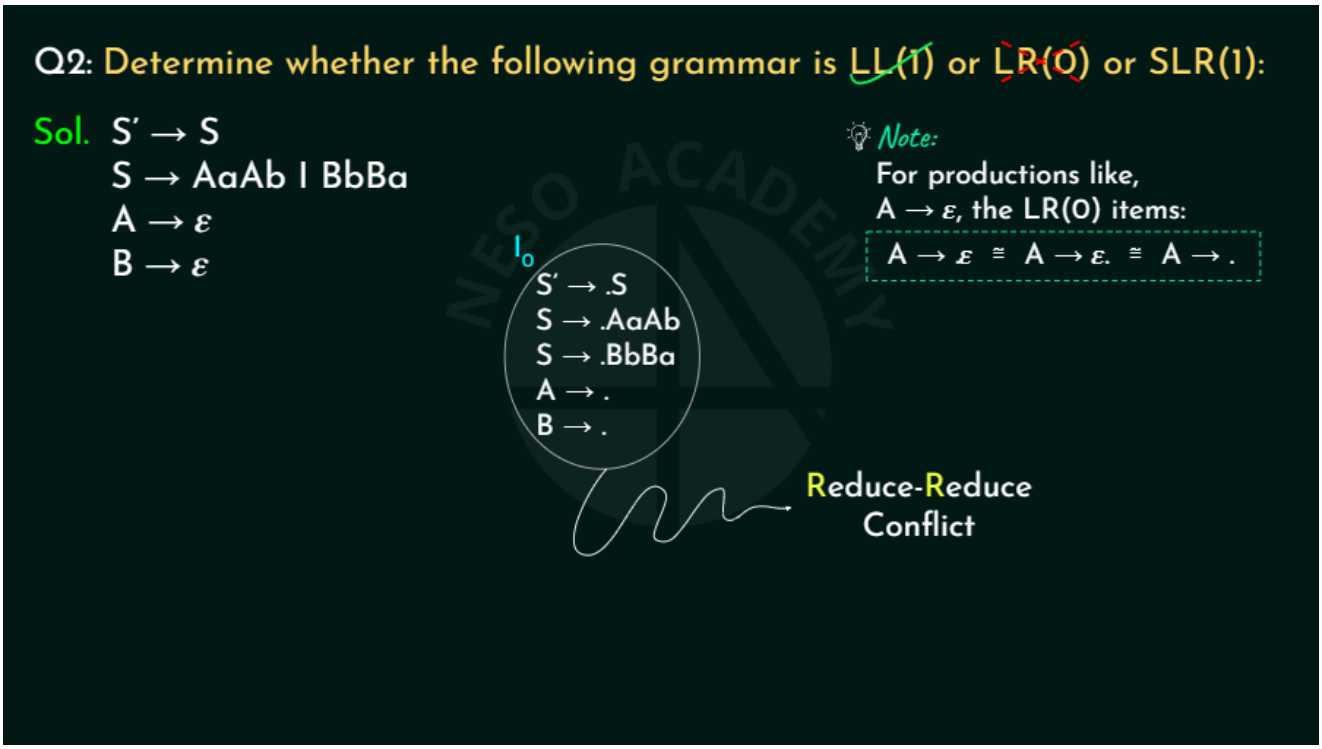
I2I6I0I9Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S’ → E$E → E + T E → TT → TF T → FF → F+F → aF → bSol.*FOLLOW(E) = {+, $}FOLLOW(T) = {a, b, +, $}FOLLOW(F) = {+, a, b, +, $}*S’ → .E$E → .E + T E → .TT → .TF T → .FF → .FF → .aF → .b*E → T.T → T.FF → .FF → .aF → .b*E → E + .TF → .FF → .aF → .b*T → .TFT → .FE → E + T. T → T.F F → .FF → .aF → .b*S’ → E.$E → E. + TI1T → F.F → F.+*I3F → a.I4F → b.I5baFTE+T → TF.F → F.+*I7FaI4bI5F → F+.*I8*TFI3I4I5ab*I3I5I4baFS’ → E$.I1.1$



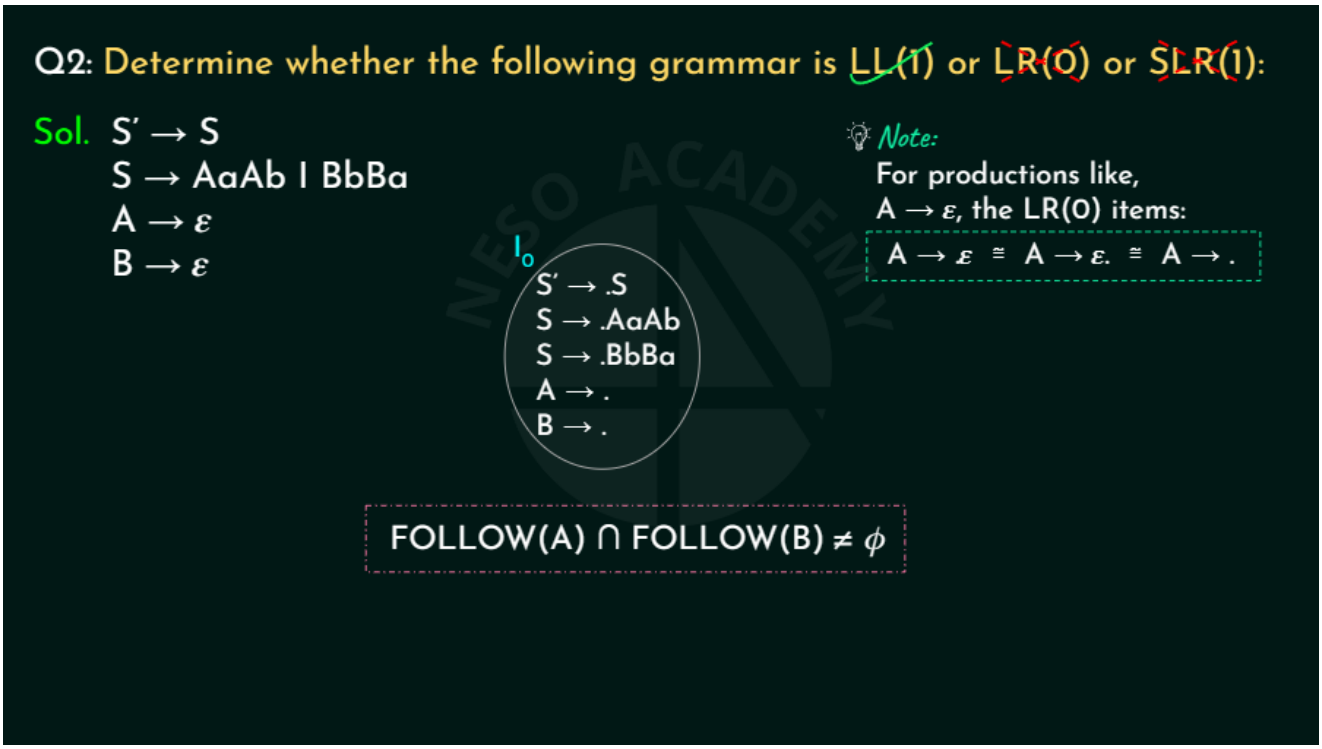
Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → AaAb I BbBaA → εB → ε



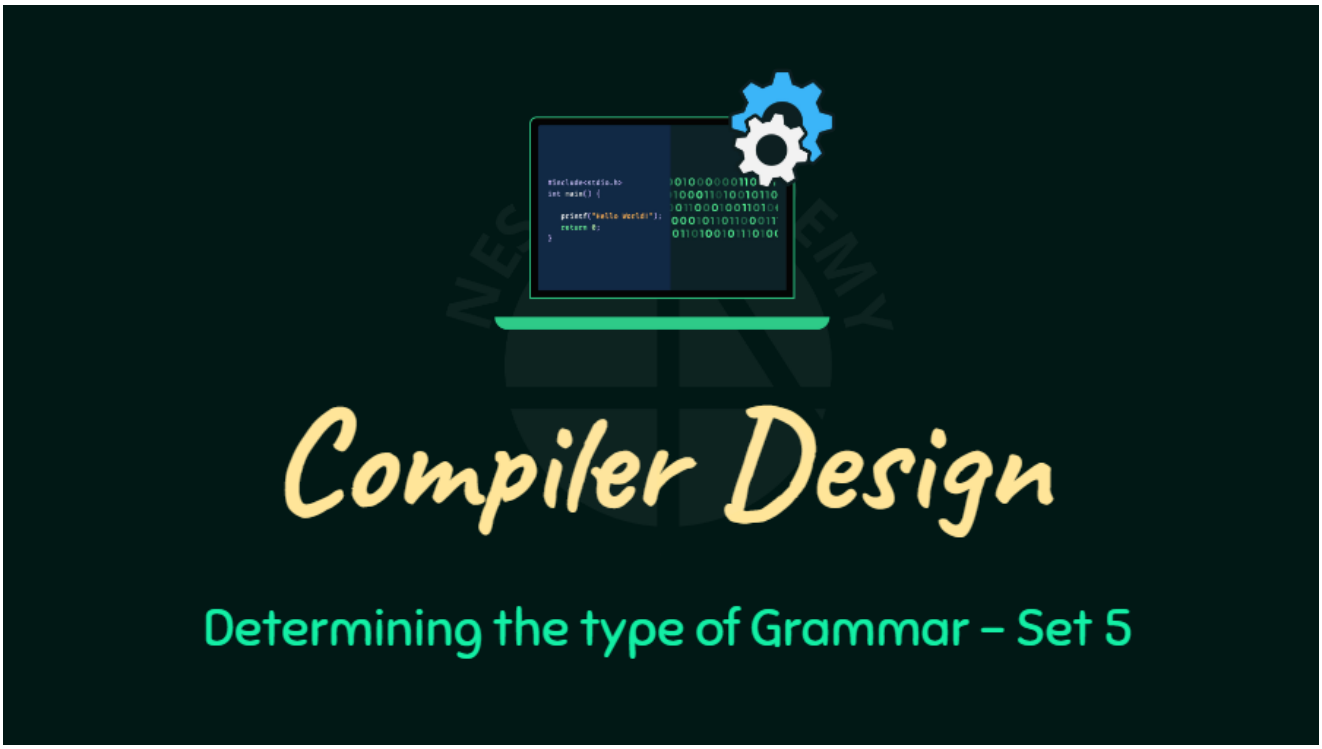
Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):{$}S → AaAb I BbBaA → εB → εFOLLOWFIRST{a, b}{ε}{a, b}{b, a}{ε}Sol.LL(1) Parsing TableaSAB$bS → AaAbA → εS → BbBaA → εB → εB → ε



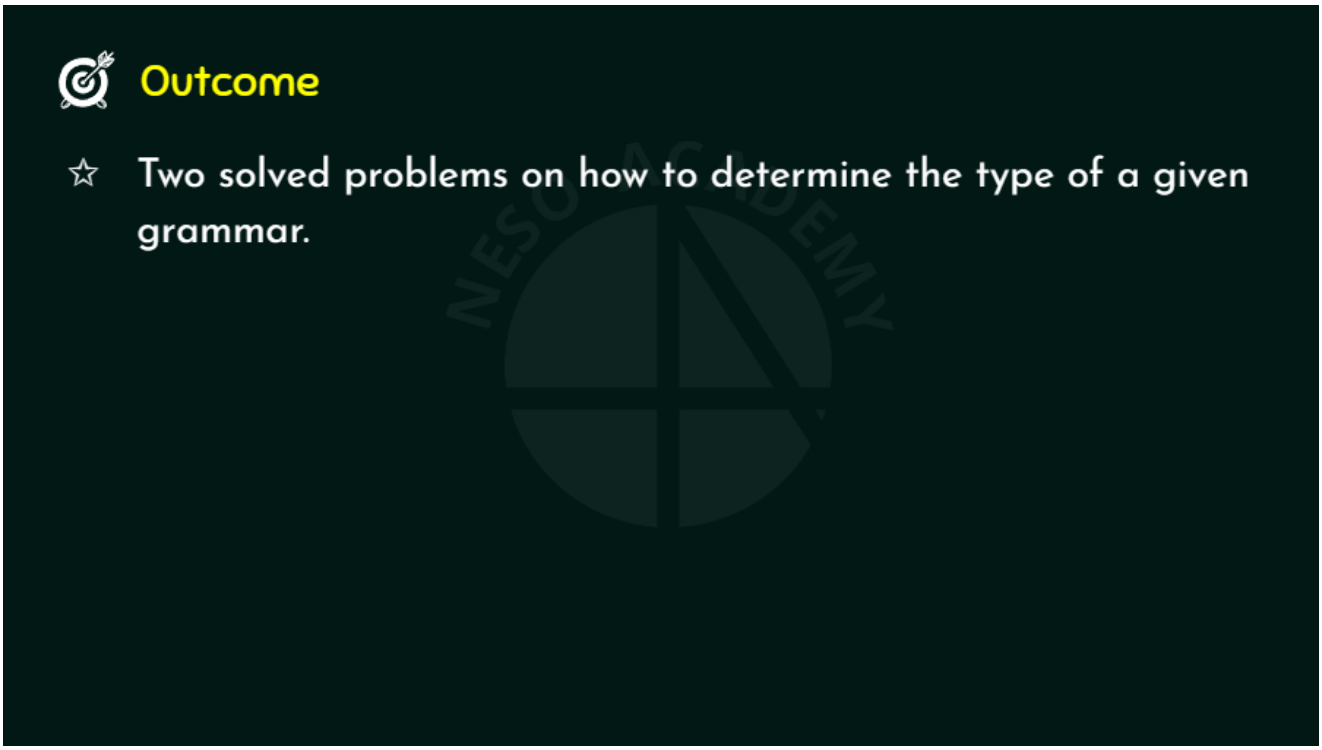
I0Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → SS → AaAb I BbBaA → εB → εSol.S' → .SS → .AaAb S → .BbBaA → .B → .Note:For productions like,A → ε, the LR(0) items:A → .ε ≅ A → ε. ≅ A → .Reduce-Reduce Conflict



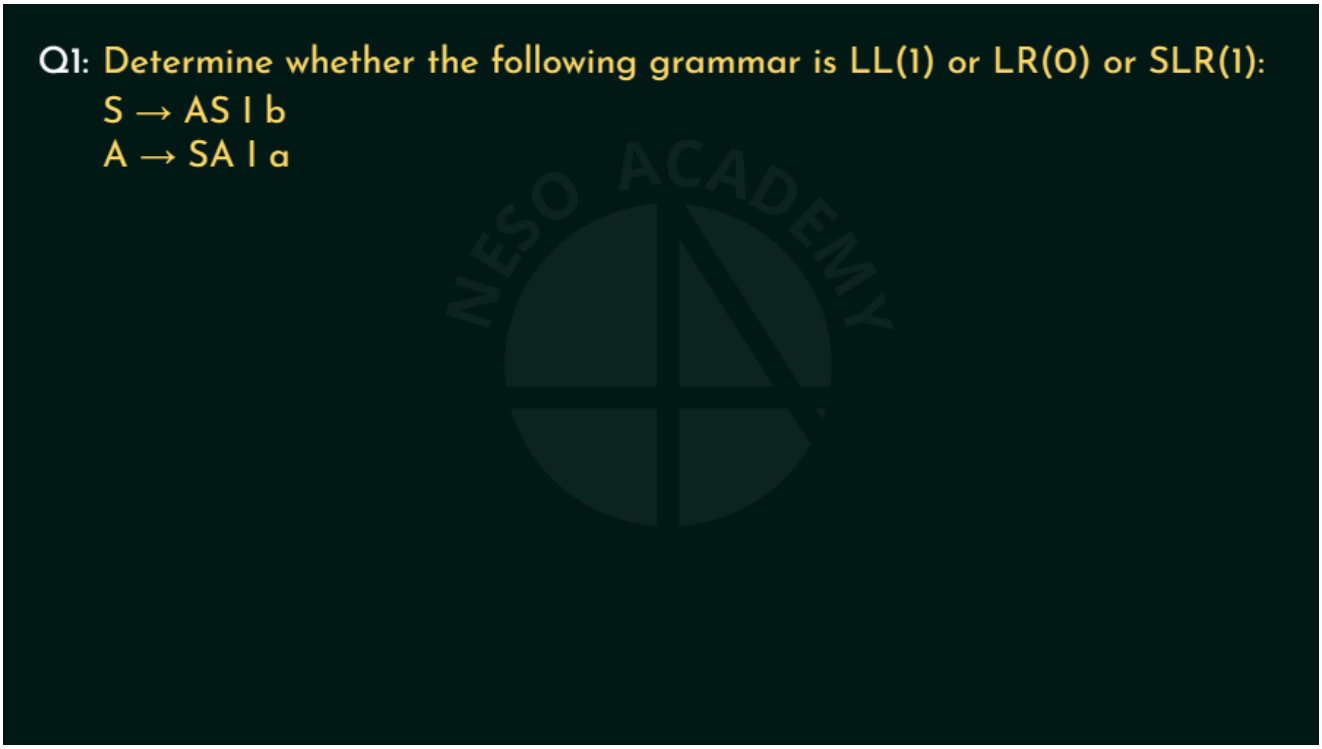
≠ ϕI0Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → SS → AaAb I BbBaA → εB → εSol.S' → .SS → .AaAb S → .BbBaA → .B → .Note:For productions like,A → ε, the LR(0) items:A → .ε ≌ A → ε. ≌ A → .FOLLOW(A) ∩ FOLLOW(B)



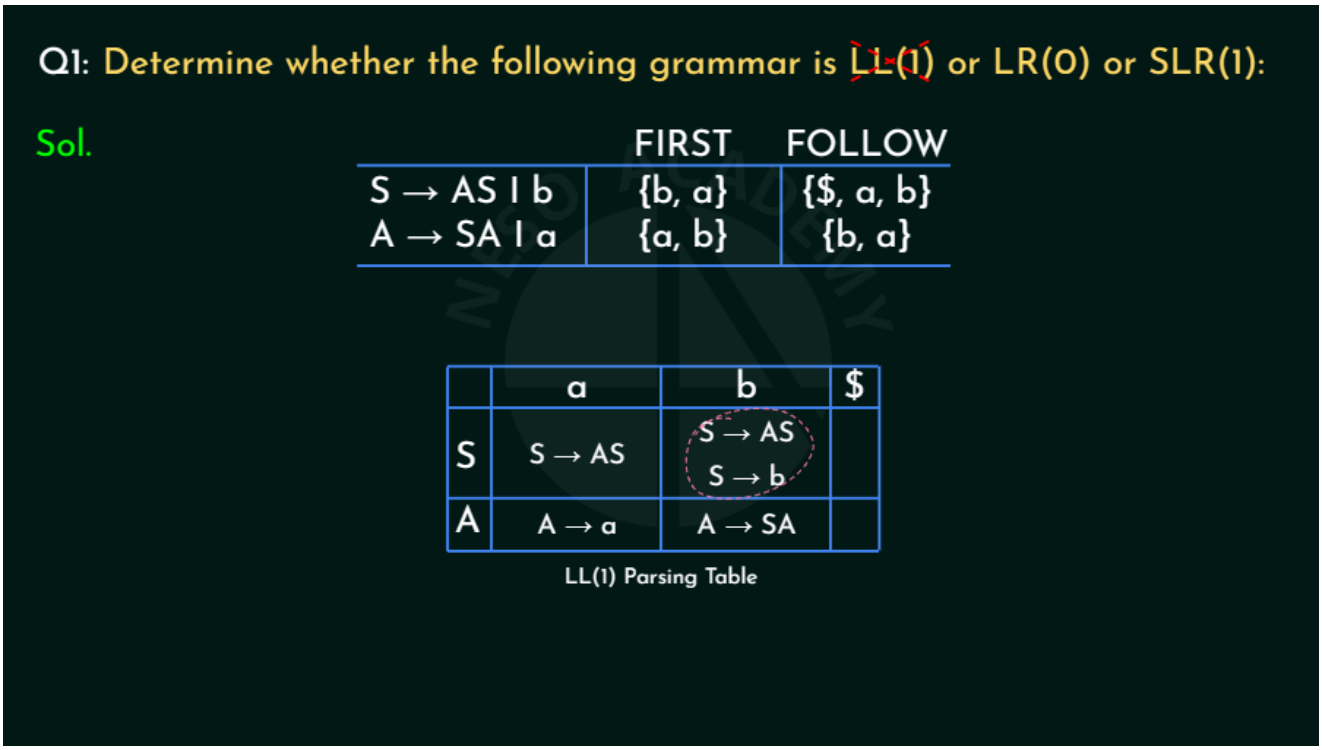
Compiler Design Determining the type of Grammar - Set 5



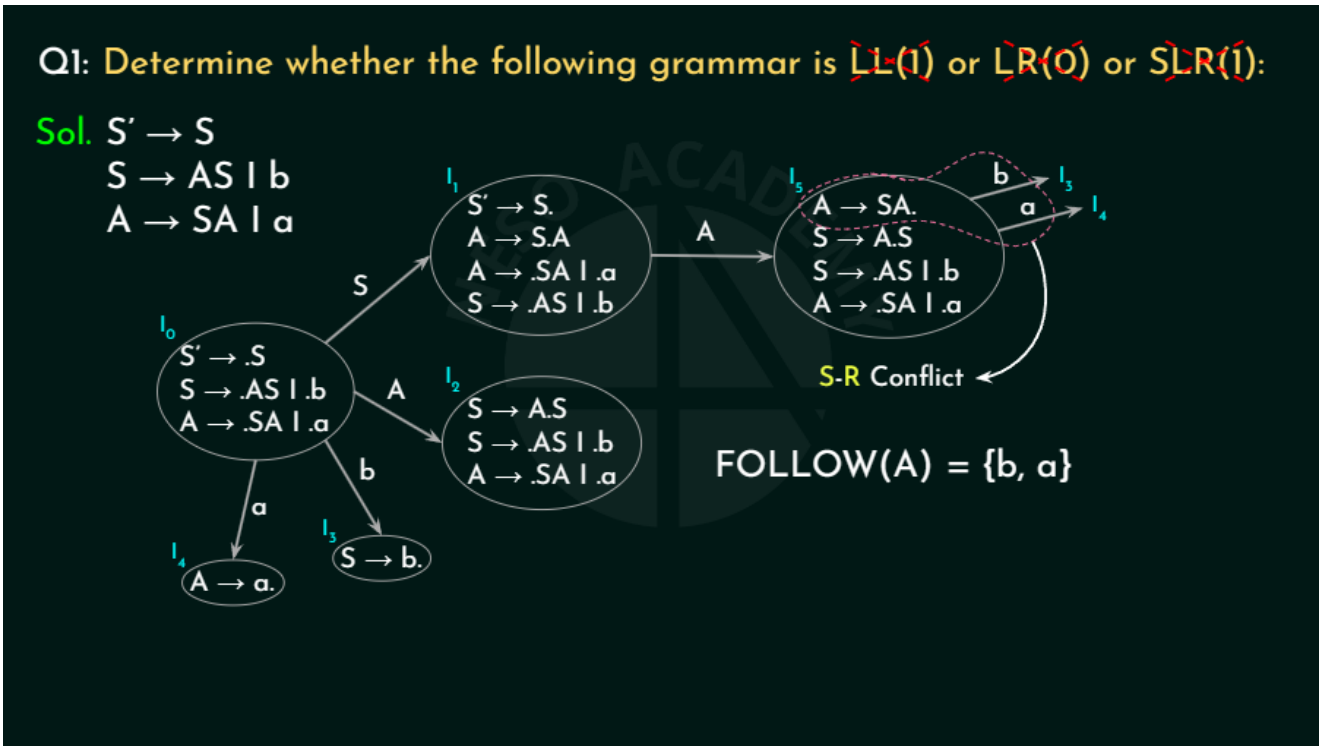
Outcome☆Two solved problems on how to determine the type of a given grammar.



Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → AS I bA → SA I a

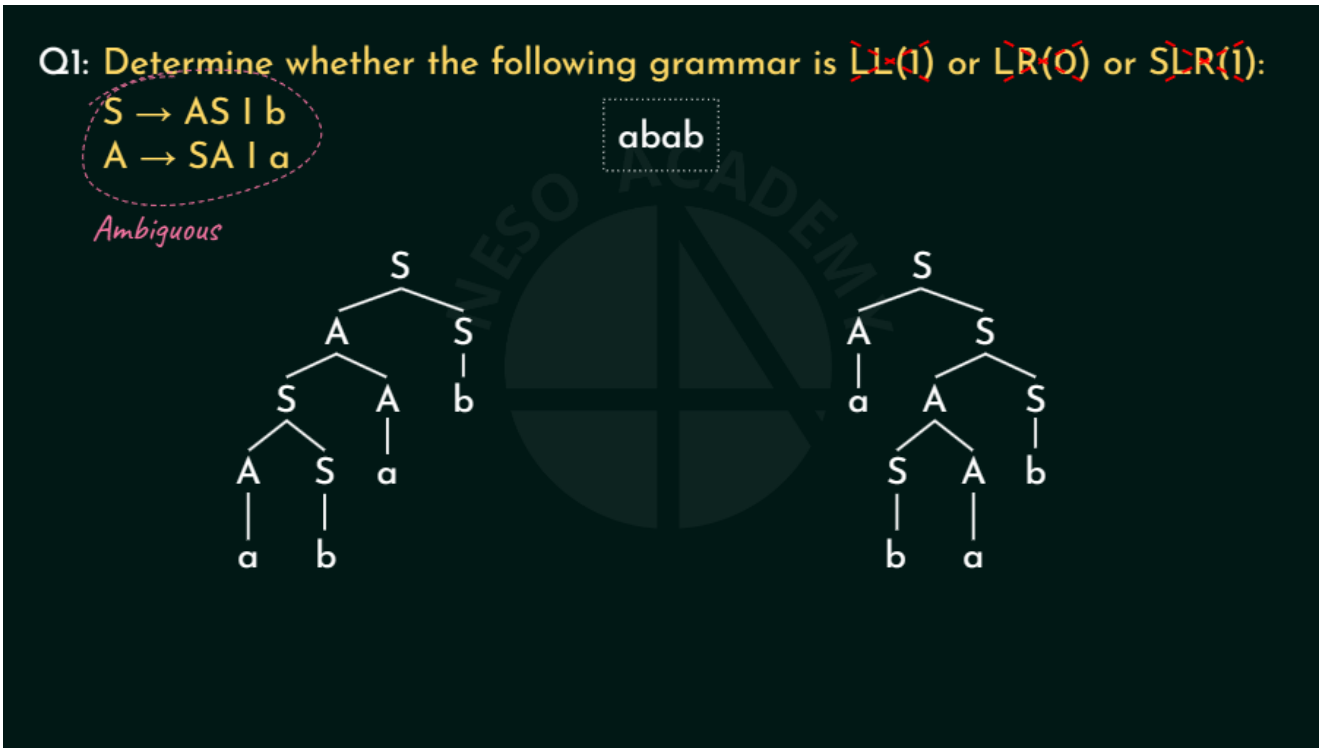


Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → ASS → ASS → bA → SAA → a, a, b}, a}, b}S → AS I bA → SA I aFOLLOWFIRST{b, a}{b{a{$Sol.LL(1) Parsing TableaA$bS

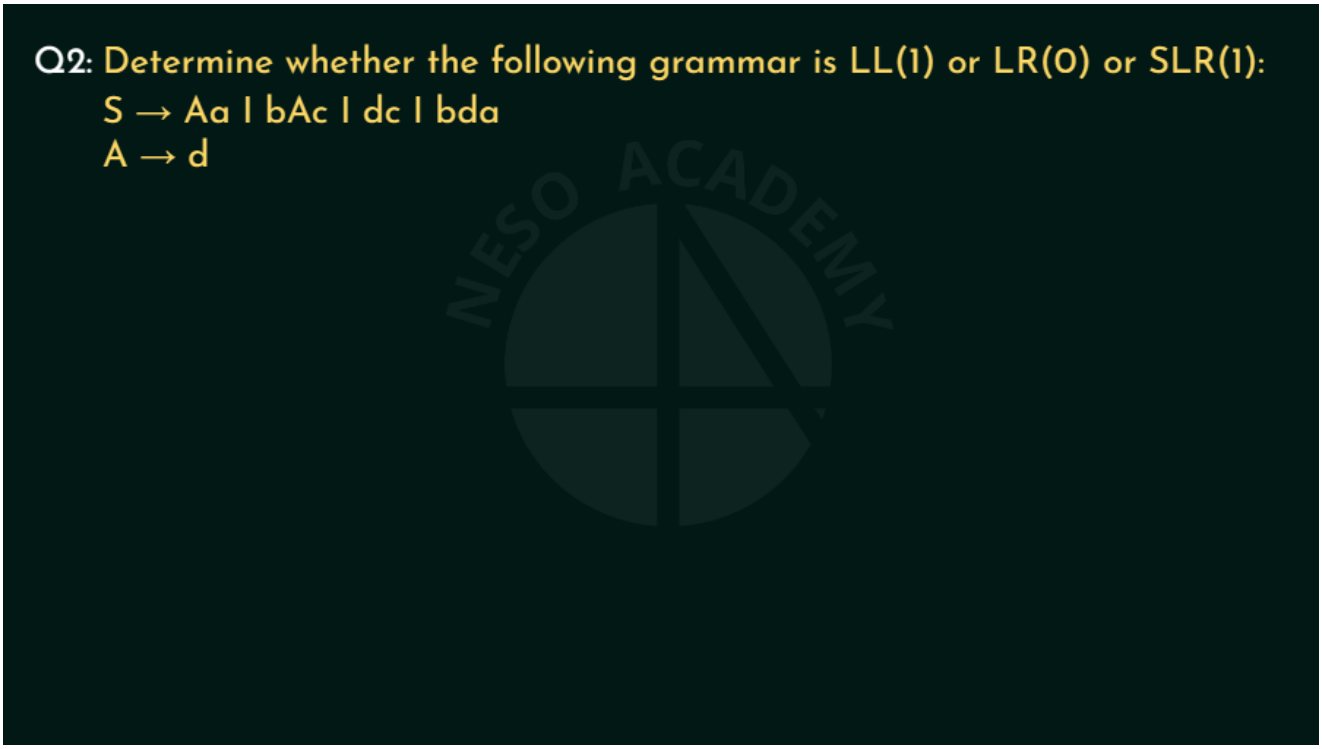


I5I1Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → SS → AS I bA → SA I aSol.S' → .SS → .AS I .bA → .SA I .aI0S' → S.A → S.AA → .SA I .aS → .AS I .bSS → A.SS → .AS I .bA → .SA I .a I2AS → b.I3A → a.I4baA → SA.S → A.SS → .AS I .bA

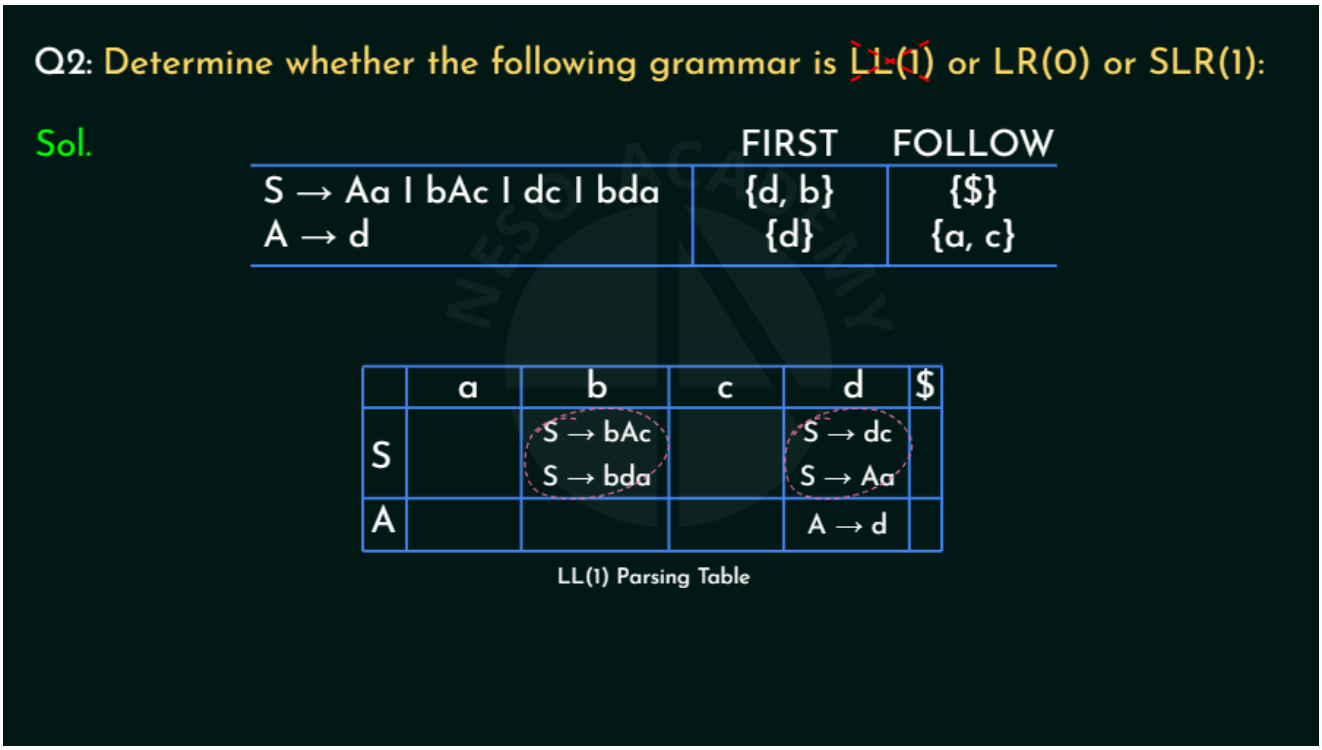
→ .SA I .aAI3bI4aS-R ConflictFOLLOW(A)= {b, a}



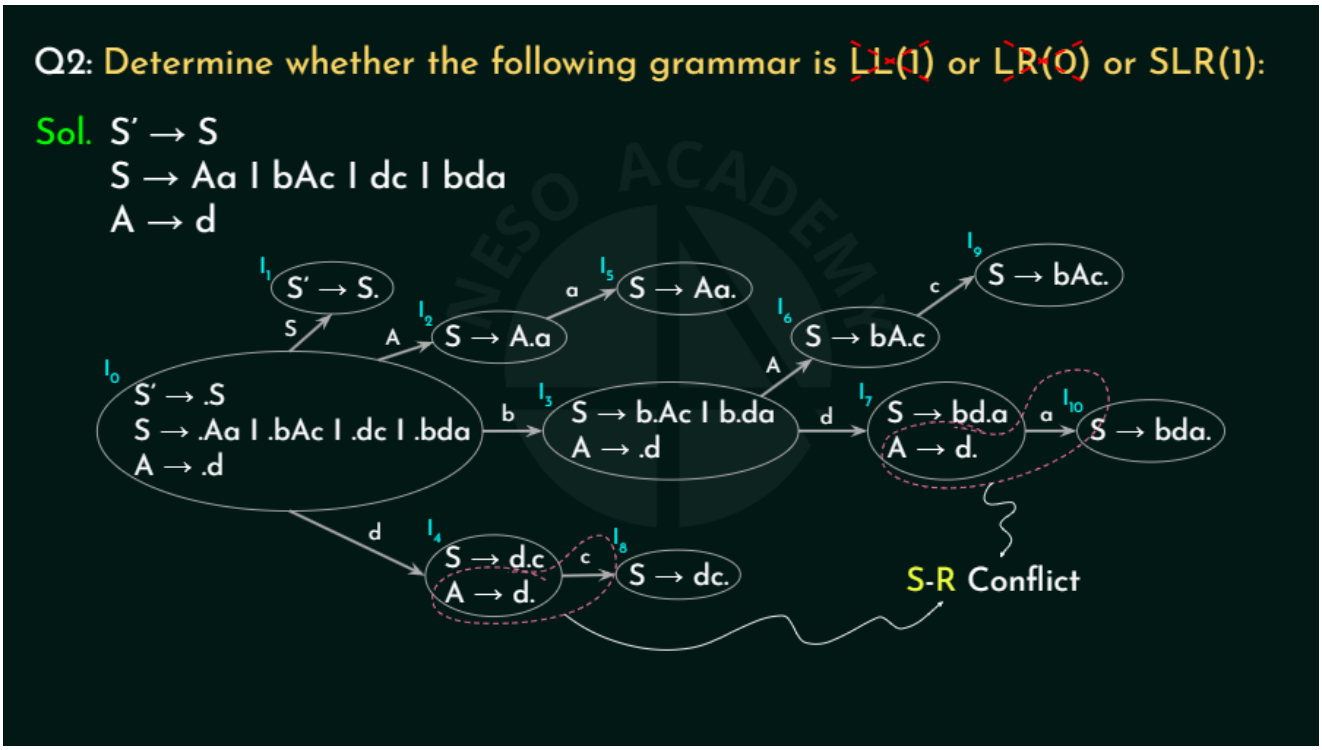
Q1:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → AS I bA → SA I aababSSAASSAababSASASSAabbaAmbiguous



Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S → Aa I bAc I dc I bdaA → d

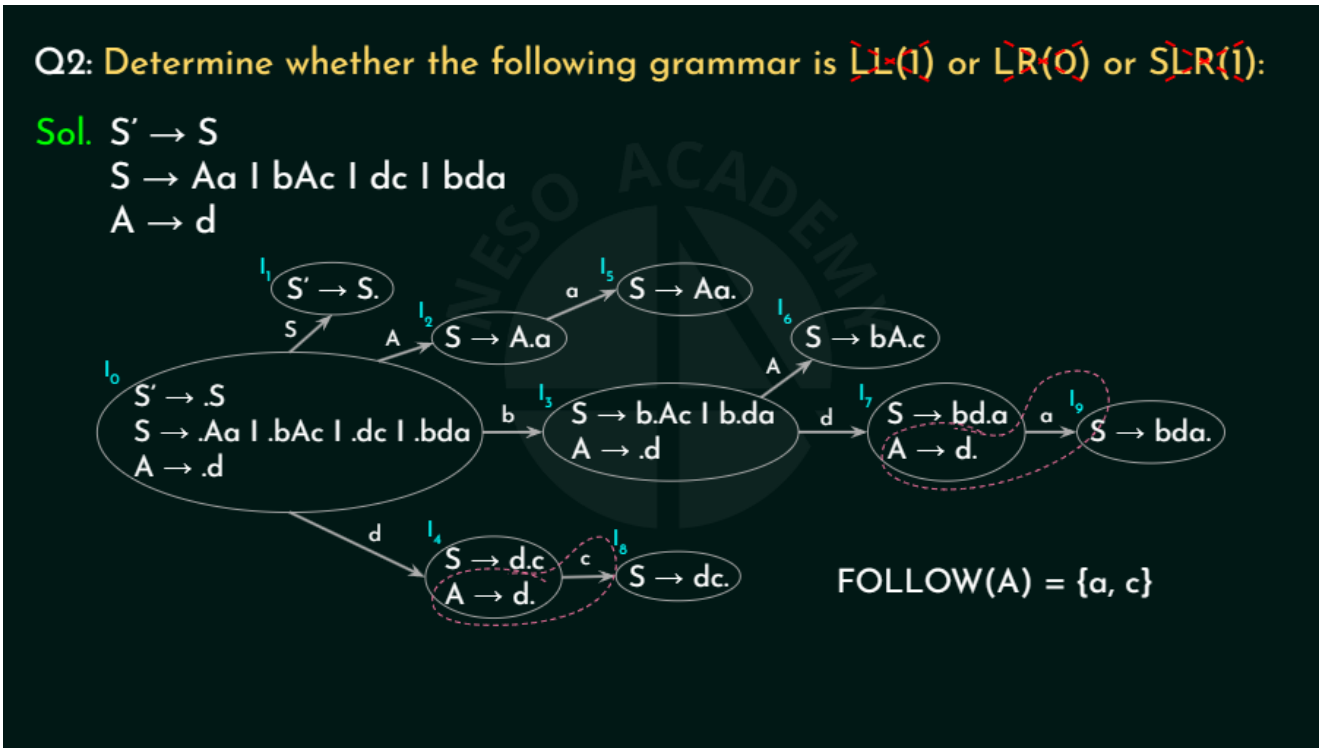


Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):, b}{d}S → Aa I bAc I dc I bdaA → dFOLLOWFIRST{a, c}{d{$}Sol.LL(1) Parsing TableadSA$cbA → dS → bAcS → dcS → AaS → bda



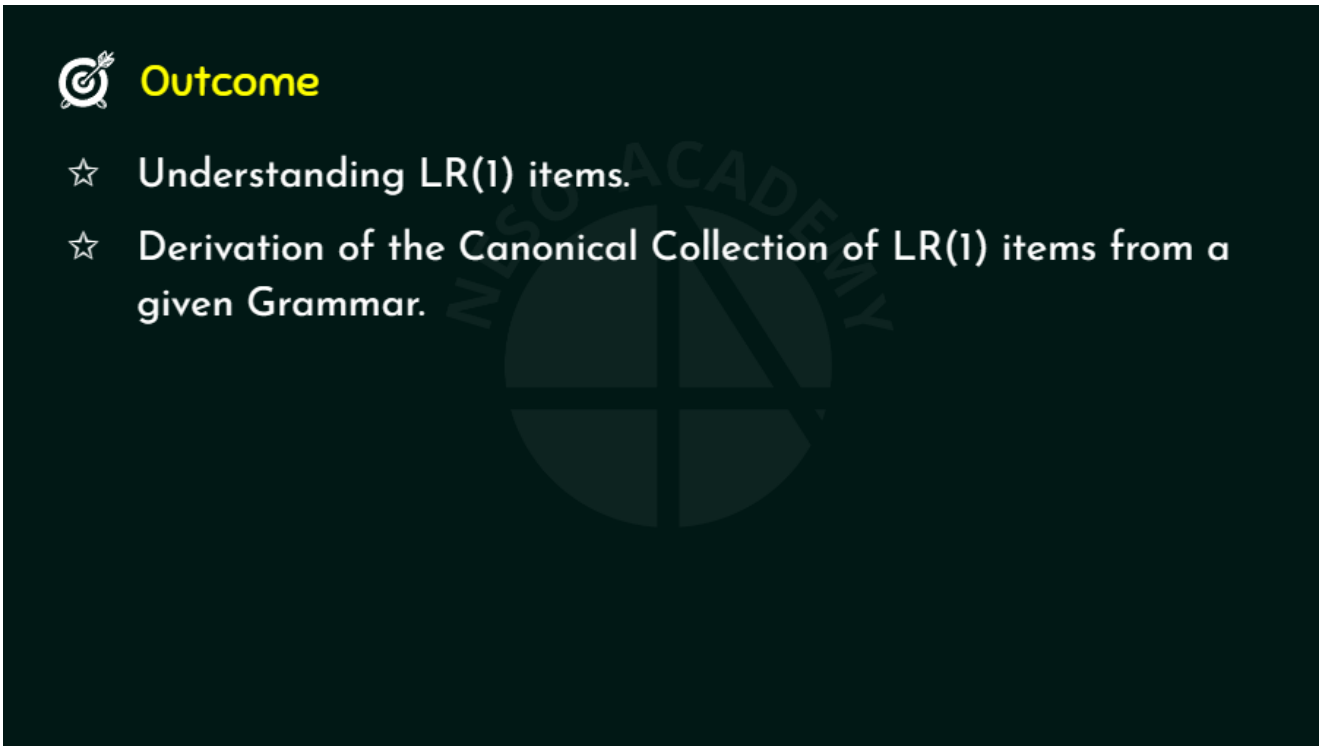
I4S → b.Ac I b.daA → .dS' → .S S → .Aa I .bAc I .dc I .bdaA → .dI0I7I3Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → S S → Aa I bAc I dc I bdaA → dSol.S → d.cA → d.S → bd.aA → d.S' → S. I1SS → A.aI2AbdS → Aa.I5aS → dc.I8cS →

bA.cI6AdS → bda.I10aS-R ConflictS → bAc.I9c

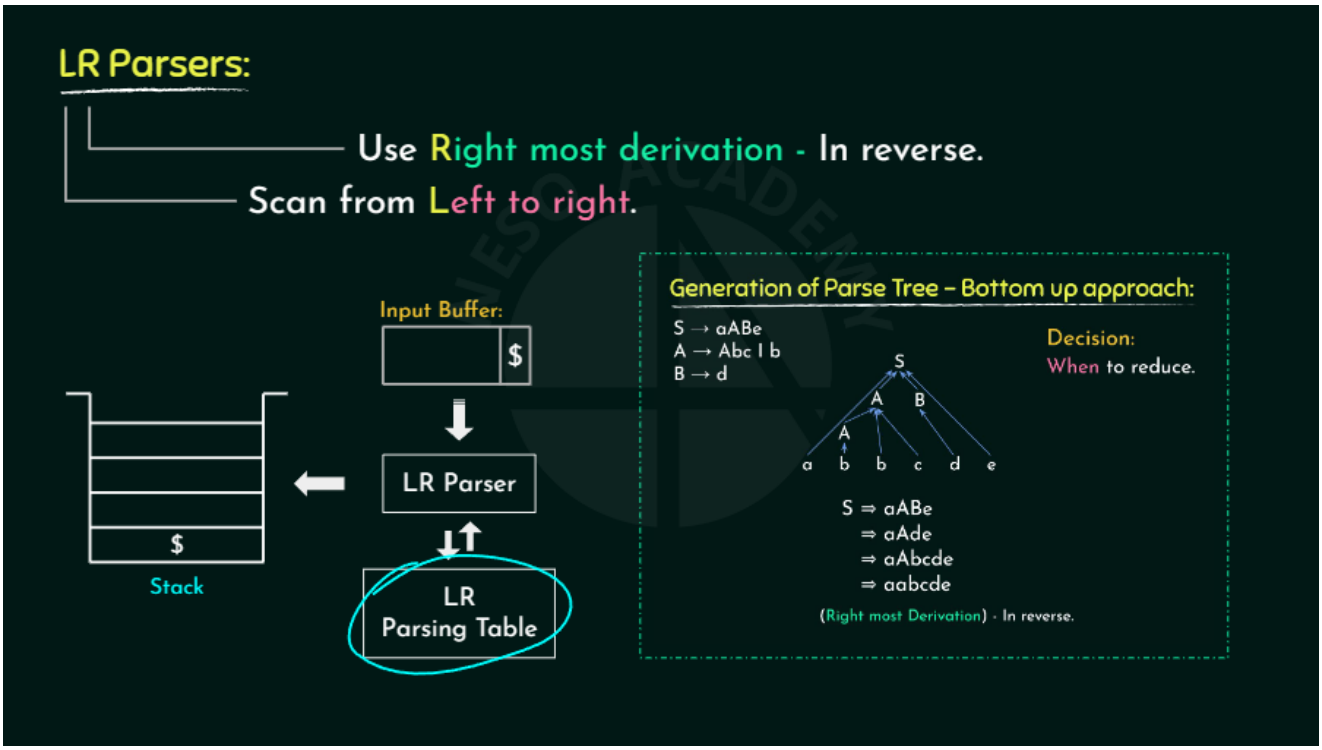


= {a, c}S → b.Ac I b.daA → .dS' → .S S → .Aa I .bAc I .dc I .bdaA → .dI0I7I3Q2:Determine whether the following grammar is LL(1) or LR(0) or SLR(1):S' → S S → Aa I bAc I dc I bdaA → dSol.S → A.aI2S → d.cA → d.I4S' → S. I1S → Aa.I5S → bA.cI6S → bd.aA → d.S → dc.I8S → bda.I9SAbdacAdaFOLLOW(A)

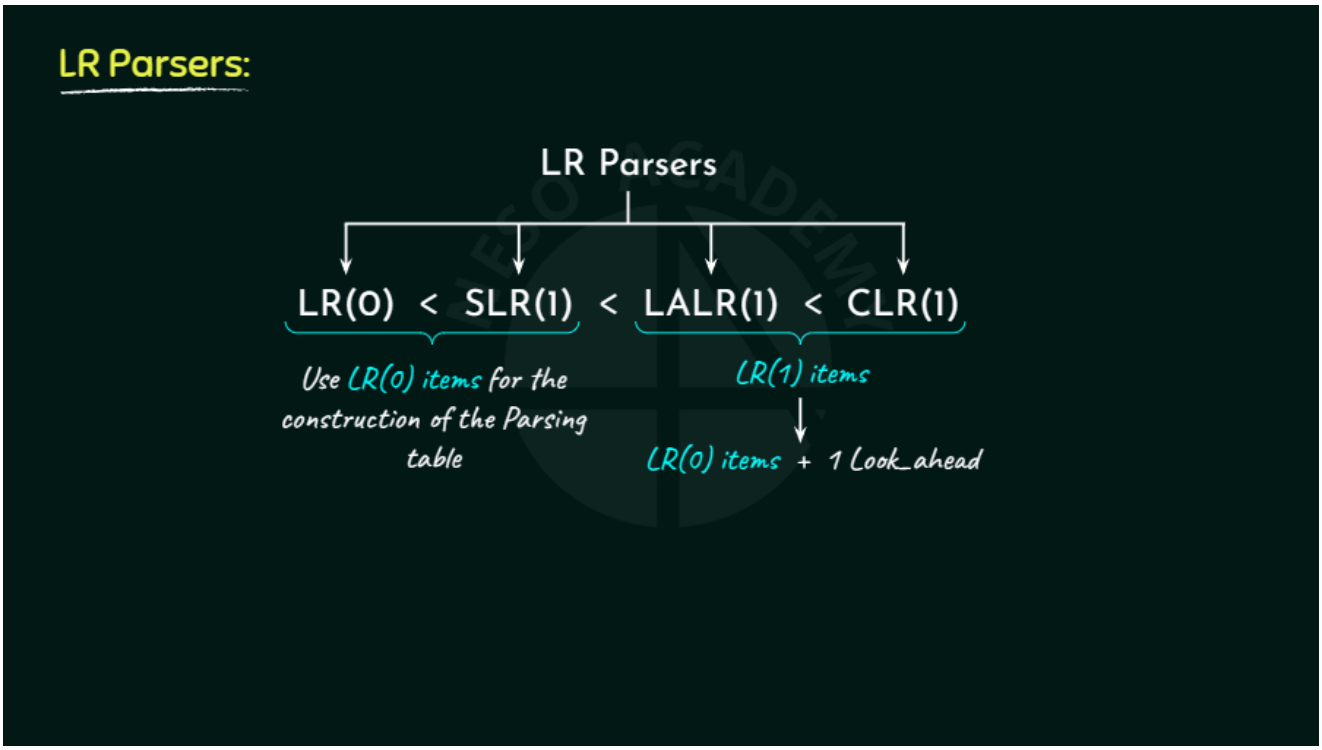
Compiler Design Canonical Collection of LR(1) items



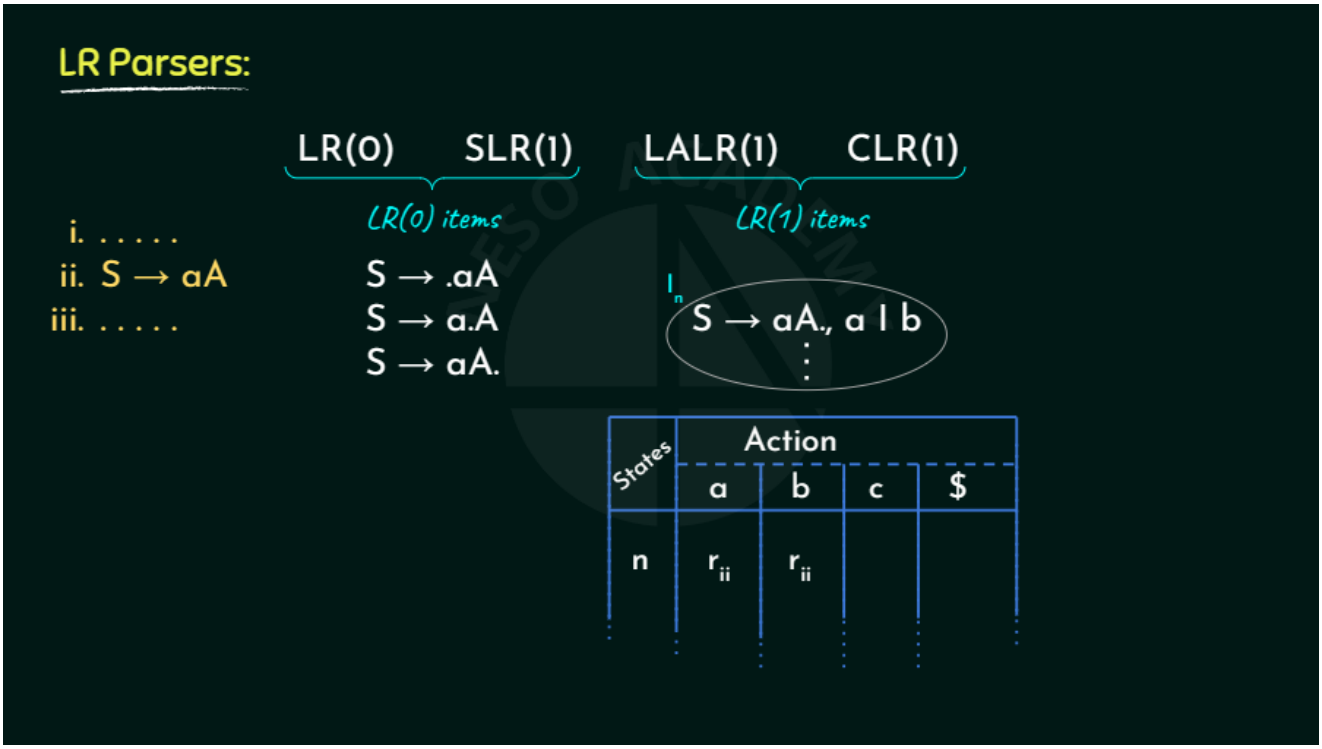
Outcome☆Understanding LR(1) items.☆Derivation of the Canonical Collection of LR(1) items from a given Grammar.



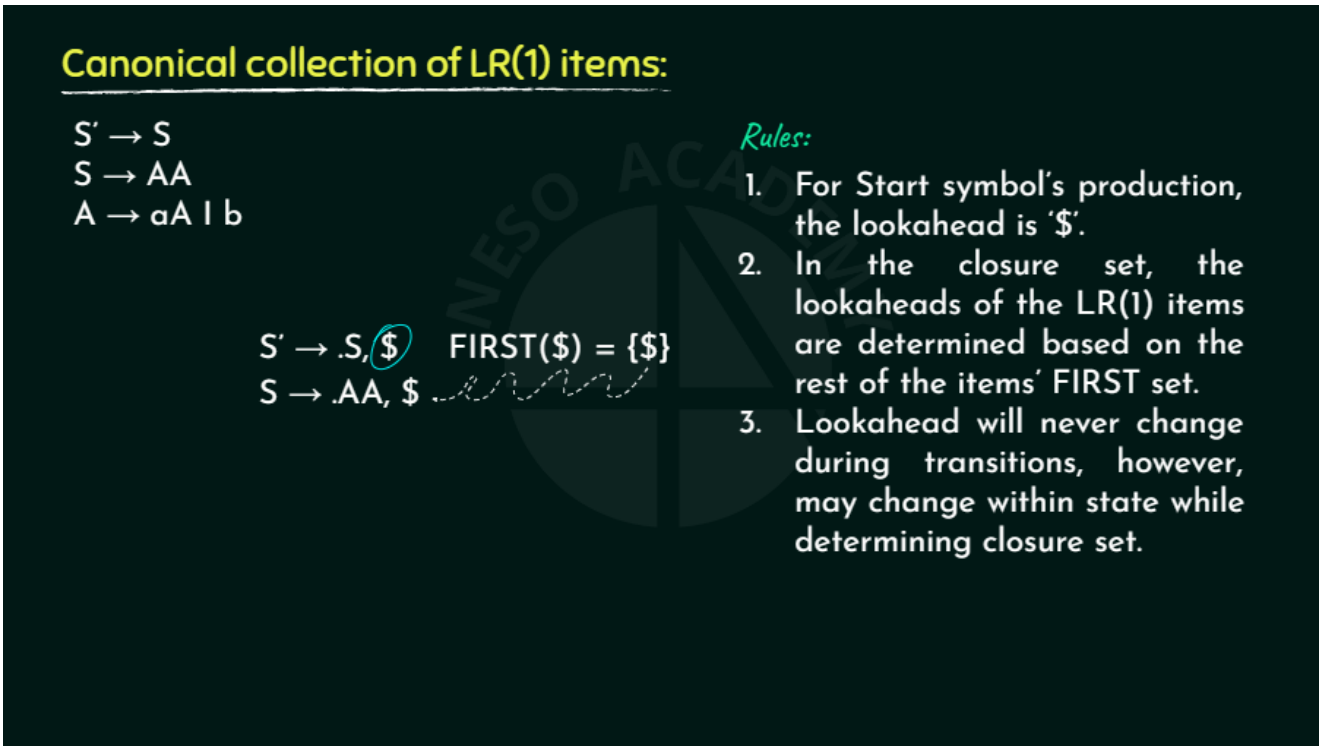
LR Parsers:$$LR ParserLR Parsing TableInput Buffer:StackScan from Left to right.Use Right most derivation - In reverse.



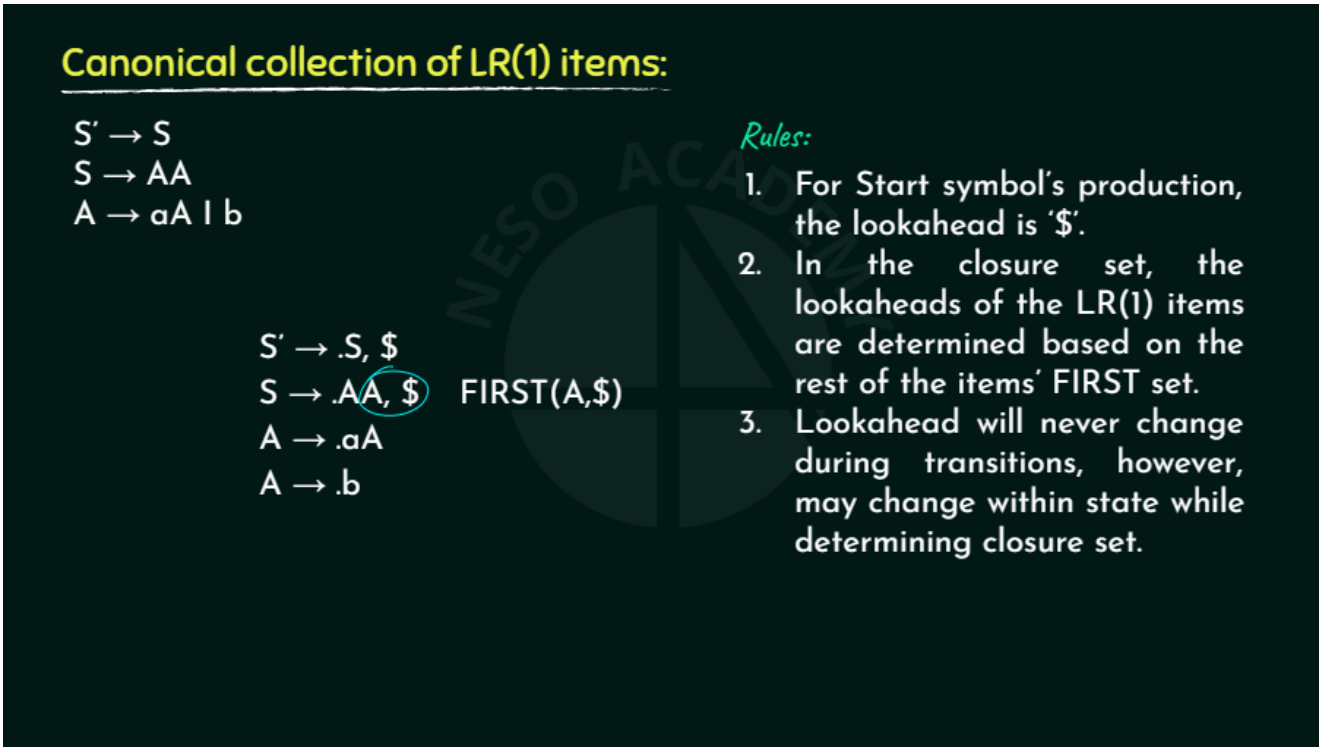
LR Parsers:LR ParsersLR(0)CLR(1)LALR(1)SLR(1)<<<Use LR(0) items for the construction of the Parsing tableLR(1) items LR(0) items + 1 Look_ahead



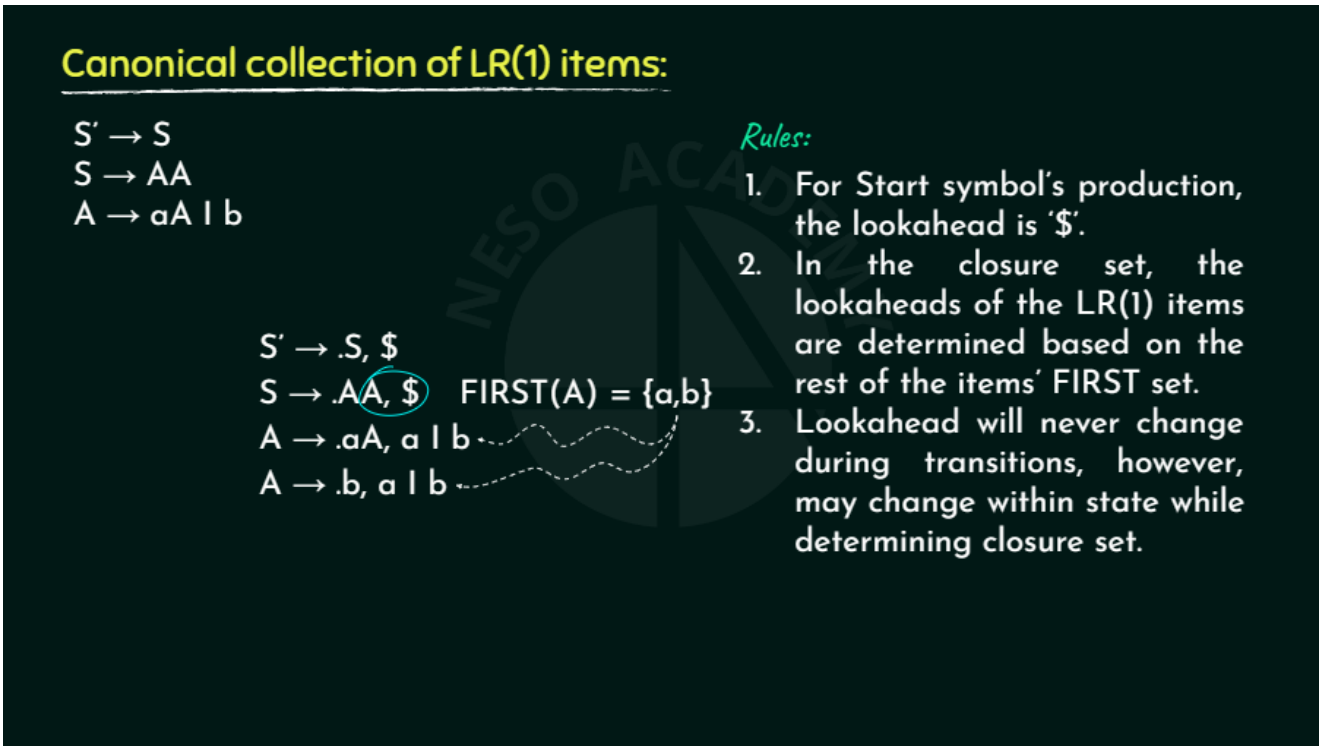
i.. . . . . ii.S → aAiii.. . . . . LR Parsers:LR(0)CLR(1)LALR(1)SLR(1)LR(1) items LR(0) items S → .aAS → a.AS → aA.I bS → aA., a InriiriiStatescbnActiona$



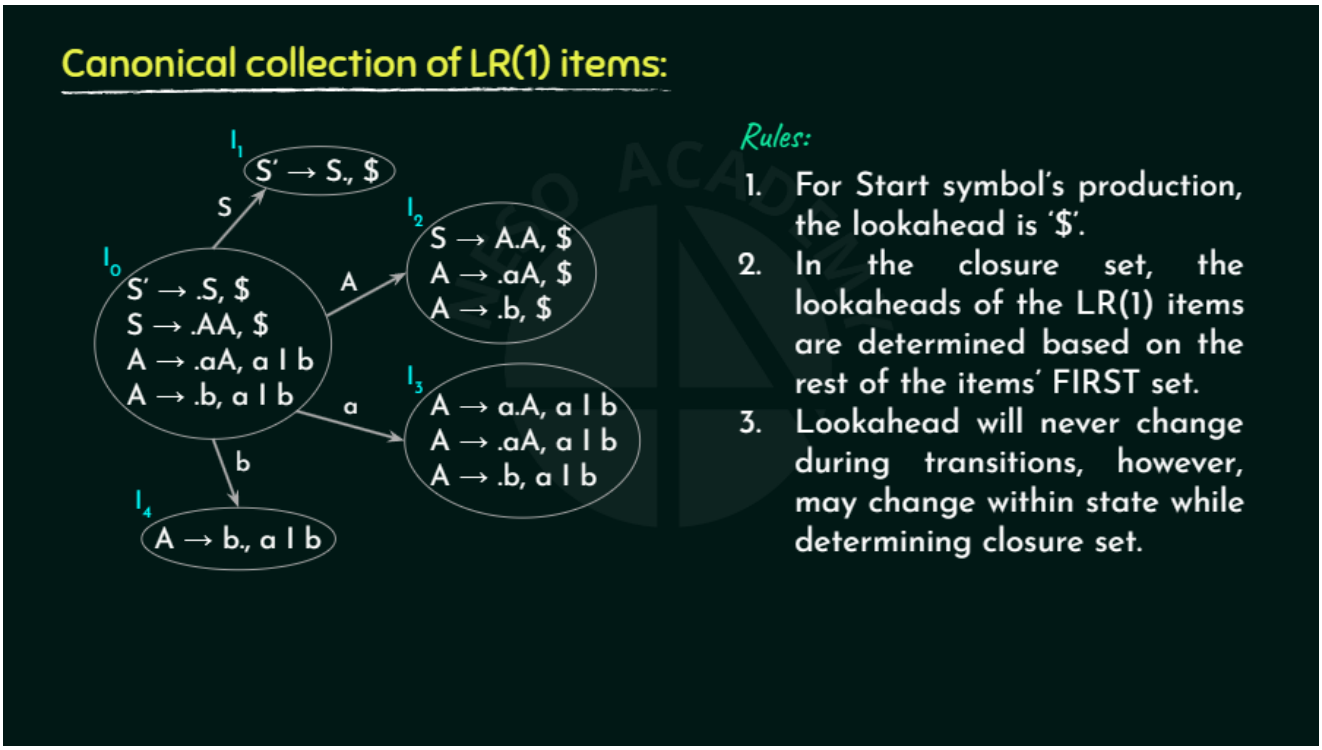
= {$}Canonical collection of LR(1) items:S' → SS → AAA → aA I bRules:1.For Start symbol's production, the lookahead is '$'.2.In the closure set, the lookaheads of the LR(1) items are determined based on the rest of the items' FIRST set.3.Lookahead will never change during transitions, however, may change within state while determining closure set.S' → .S, $S → .AAFIRST($), $



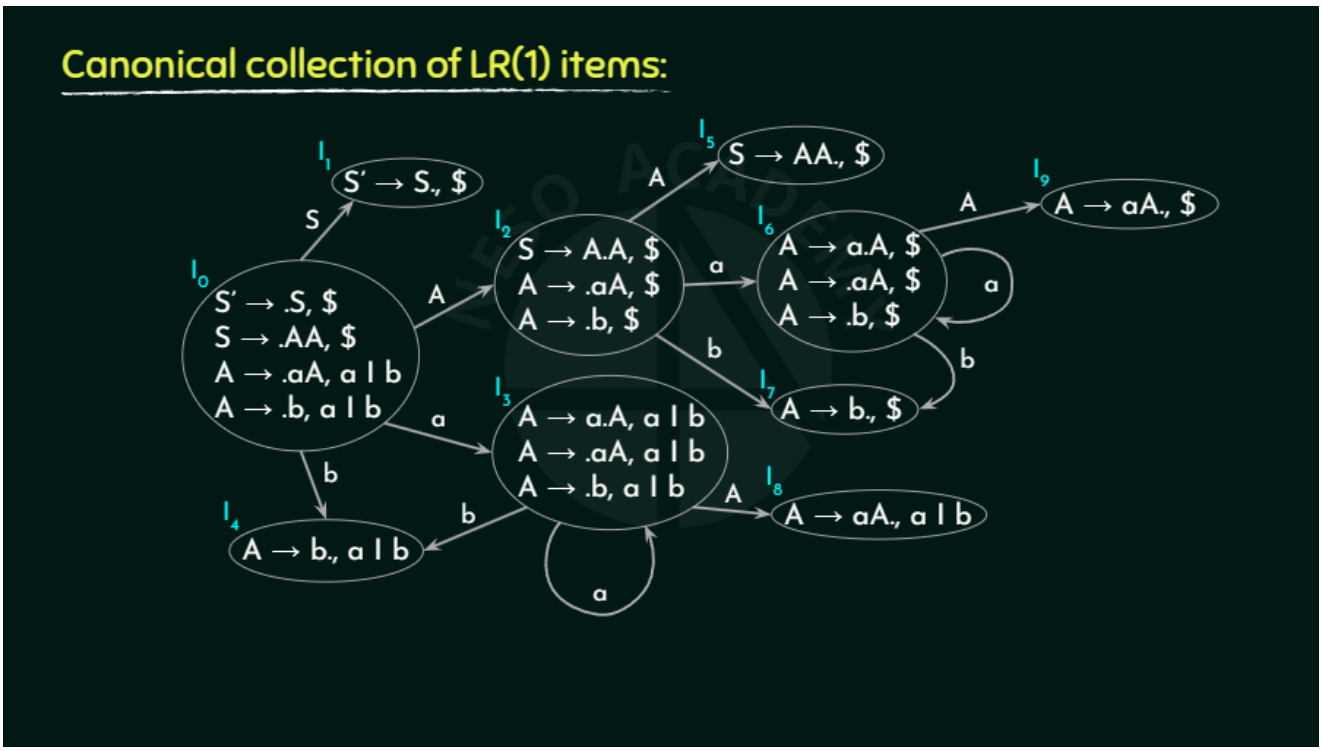
Canonical collection of LR(1) items:S’ → SS → AAA → aA I bRules:1.For Start symbol’s production, the lookahead is ‘$’.2.In the closure set, the lookaheads of the LR(1) items are determined based on the rest of the items’ FIRST set.3.Lookahead will never change during transitions, however, may change within state while determining closure set.S’ → .S, $S → .AA, $A → .aAA → .bFIRST(A,$)



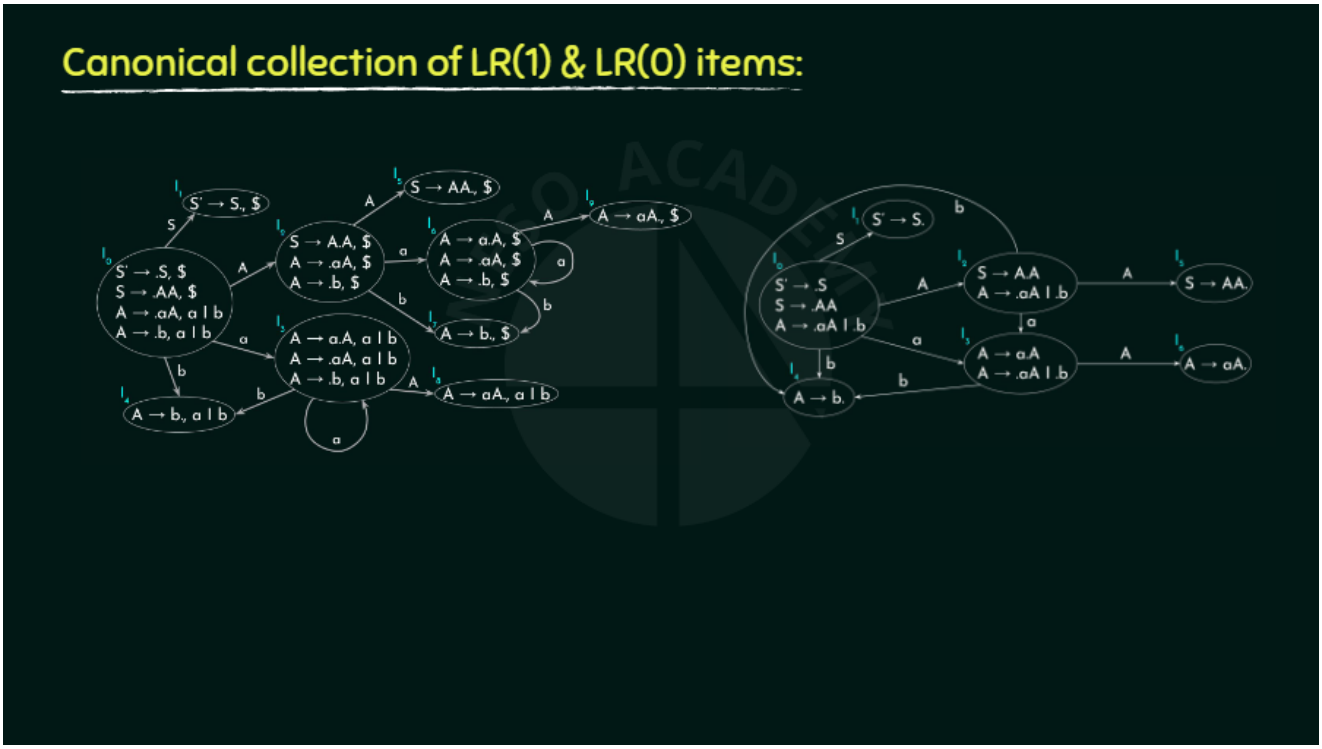
Canonical collection of LR(1) items:S’ → SS → AAA → aA I bRules:1.For Start symbol’s production, the lookahead is ‘$’.2.In the closure set, the lookaheads of the LR(1) items are determined based on the rest of the items’ FIRST set.3.Lookahead will never change during transitions, however, may change within state while determining closure set.S’ → .S, $S → .AA, $A → .aAA → .bFIRST(A)= {a,b}, a I b, a I b



Canonical collection of LR(1) items:Rules:1.For Start symbol's production, the lookahead is '$'.2.In the closure set, the lookaheads of the LR(1) items are determined based on the rest of the items' FIRST set.3.Lookahead will never change during transitions, however, may change within state while determining closure set.I2I3A → .aAA → .bS' → .S, $S → .AA, $A → .aA, a I bA → .b, a I bS → A.A, $A → a.A, a I b, $, $, a I b, a I bA → .aAA → .bI0S' → S., $I1A → b., a I bI4SAab

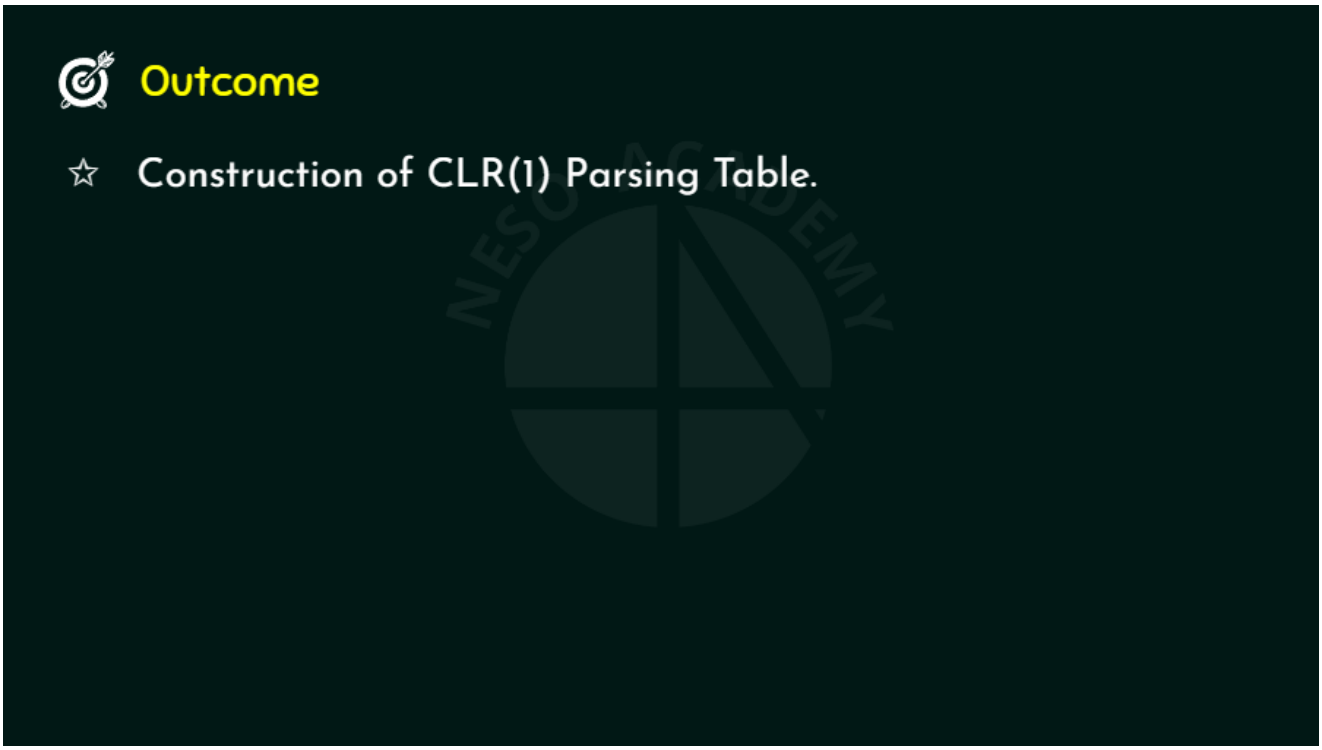


I6I2A → .aAA → .bCanonical collection of LR(1) items:S’ → .S, $S → .AA, $A → .aA, a I bA → .b, a I bS → A.A, $A → a.A, a I bA → b., a I b, $, $, a I b, a I bA → .aAA → .bI0S’ → S., $I1I4SAabS → AA., $I5AA → a.A, $A → .aAA → .b, $, $aA → b., $I7bA → aA., a I bI8I3AabA → aA., $I9Aab

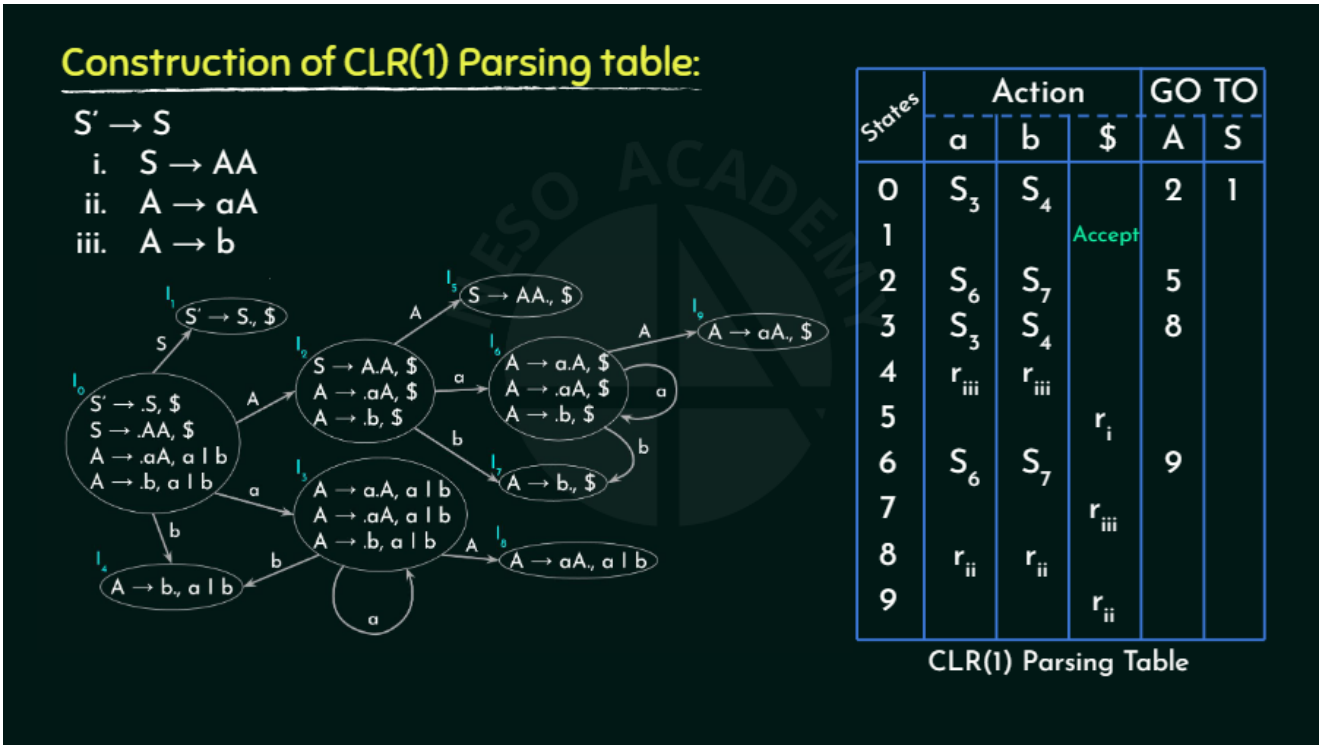


Canonical collection of LR(1) & LR(0) items:

Compiler Design CLR(1) Parser

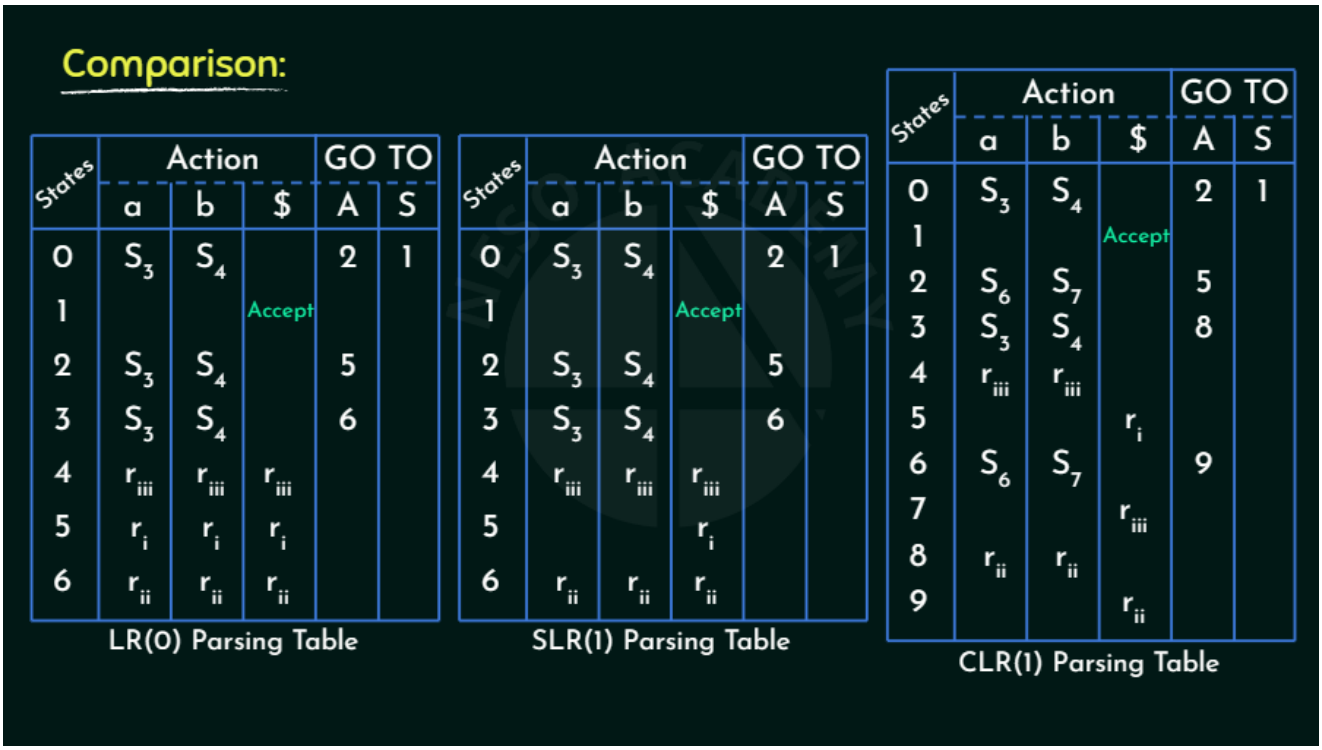


Outcome☆Construction of CLR(1) Parsing Table.



States	Action			GO TO	
	a	b	$	A	S
0	S_3	S_4		2	1
1			Accept		
2	S_6	S_7		5	
3	S_3	S_4		8	
4	r_{iii}	r_{iii}			
5			r_i		
6	S_6	S_7		9	
7			r_{iii}		
8	r_{ii}	r_{ii}			
9			r_{ii}		

Construction of CLR(1) Parsing table:S’ → Si.S → AAii.A → aAiii.A → bAccept1S3S42riiS6S75S3S48riiiriiiriririiS6S791092345867riiiActionGO TOStatesab$SACLR(1) Parsing Table

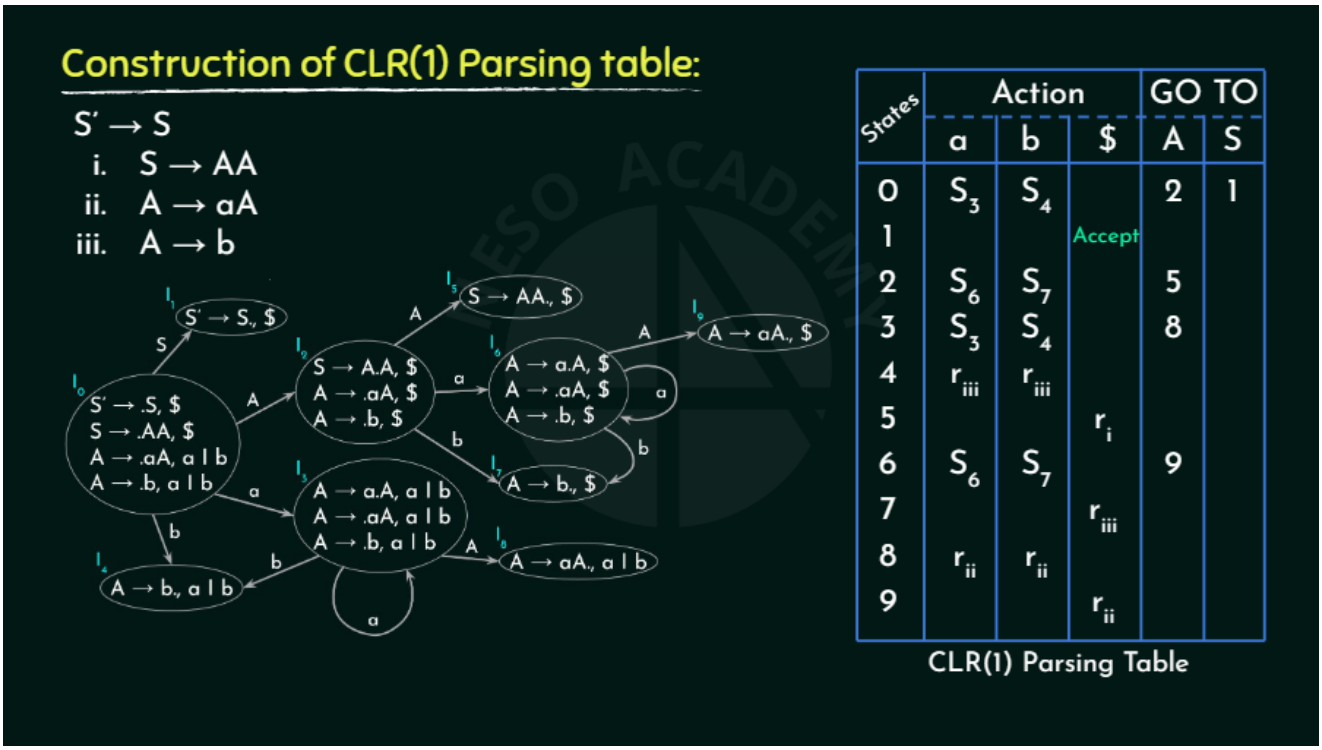


Comparison:1Accept01S3S429rii2S6S753S3S484riiiriii5ri8riirii6S6S797riiiActionGO TOStatesab$SACLR(1) Parsing Table1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriii56ab$SALR(0) Parsing Table1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiriirii56ab$SAriiiriiiriiiriSLR(1) Parsing Table

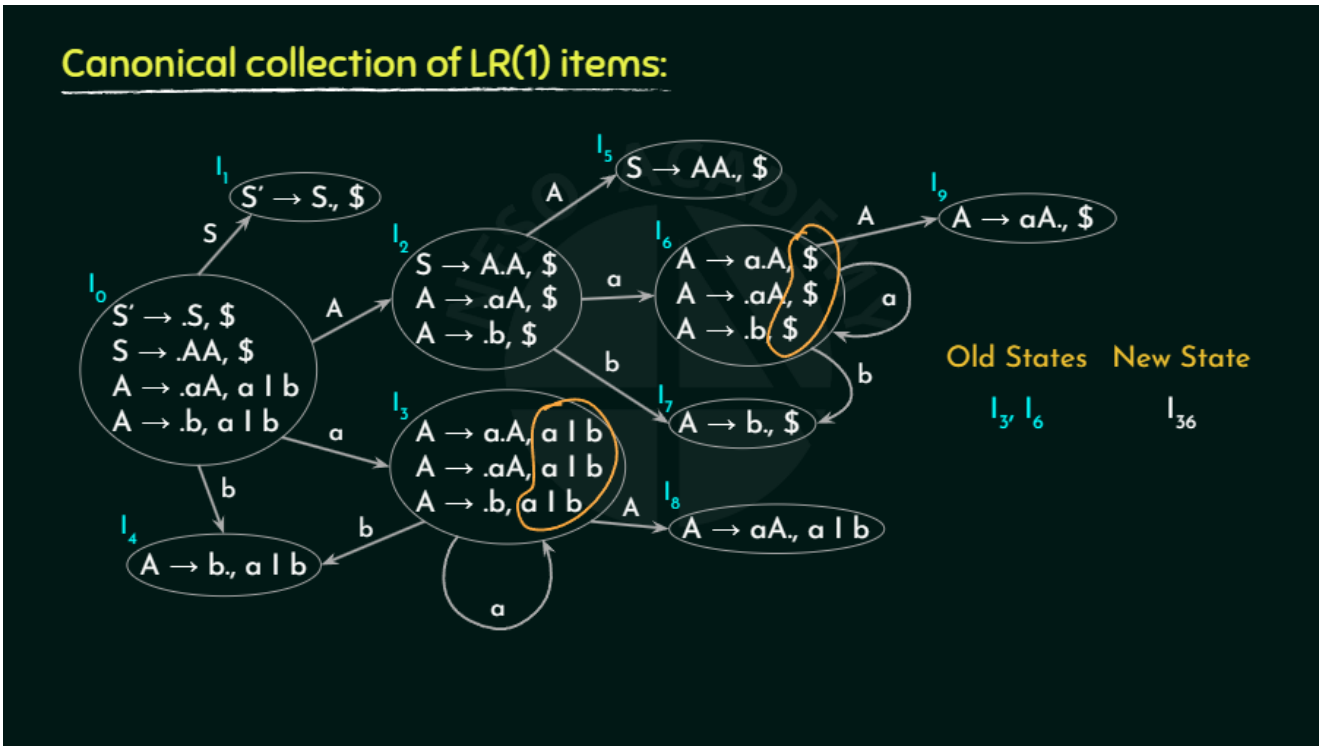
Compiler Design LALR(1) Parser

Outcome☆Conversion from CLR(1) to LALR(1) Parsing Table.

Comparison:1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiiriiiriiiriririririiriii56ab$SALR(0) Parsing Table1ActionGO TOStates0123456S3AcceptS42S3S4S3S4riiriirii56ab$SAriiiriiiriiiriSLR(1) Parsing Table

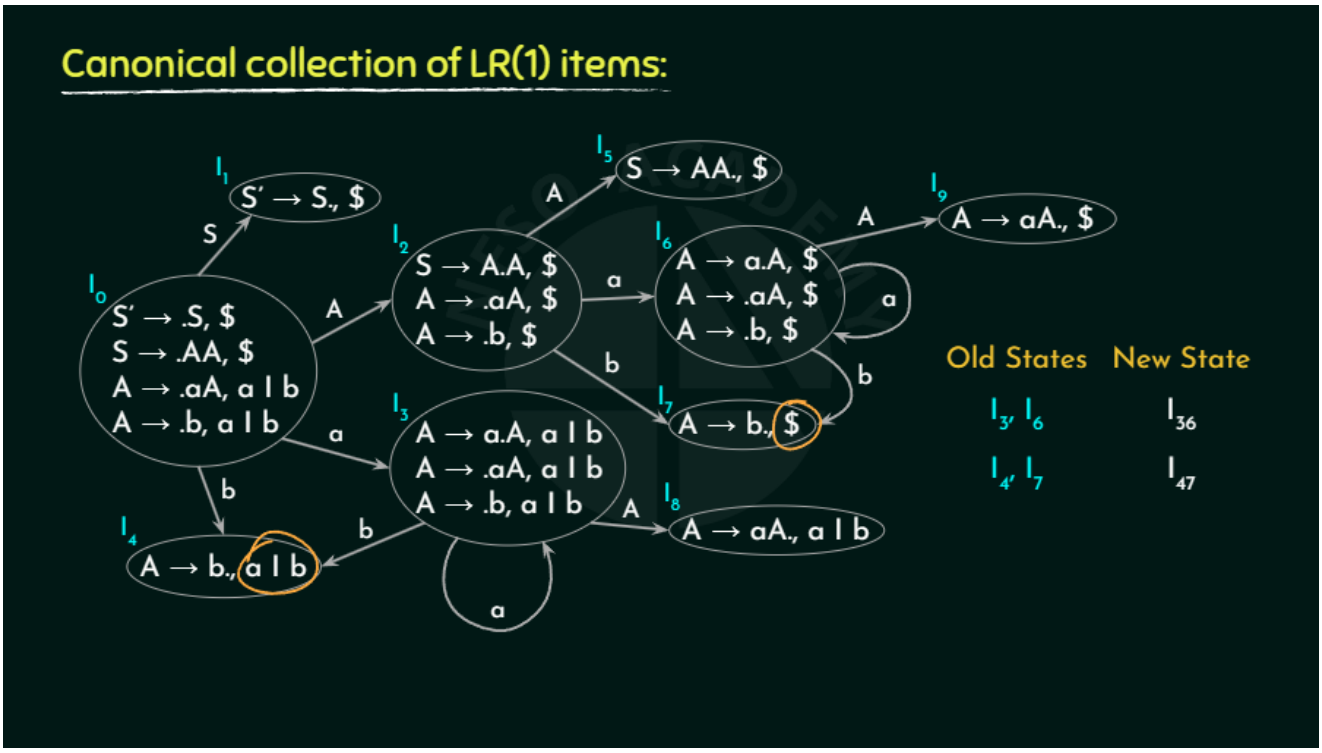


Construction of CLR(1) Parsing table:S’ → Si.S → AAii.A → aAiii.A → bAccept1S3S42riiS6S75S3S48riiiriiiriririiriiS6S791092345867riiiActionGO TOStatesab$SACLR(1) Parsing Table

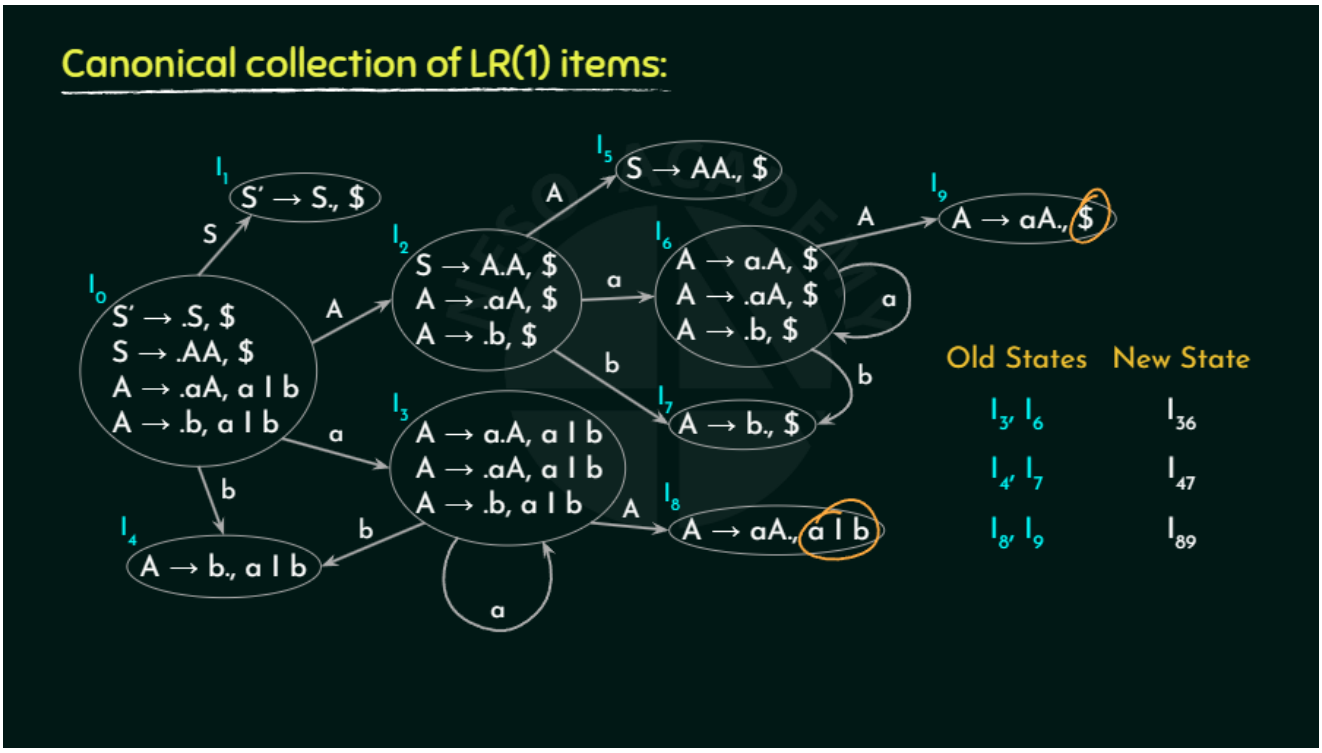


Canonical collection of LR(1) items:I6I2A → .aAA → .bS’ → .S, $S → .AA, $A → .aA, a | bA → .b, a | bS → A.A, $A → a.A, a | bA → b., a | b, $, $, a | b, a | bA → .aAA → .bI0S’ → S., $I1I4SAabS → AA., $I5AA → a.A, $A → .aAA → .b, $, $aA → b., $I7bA → aA., a |

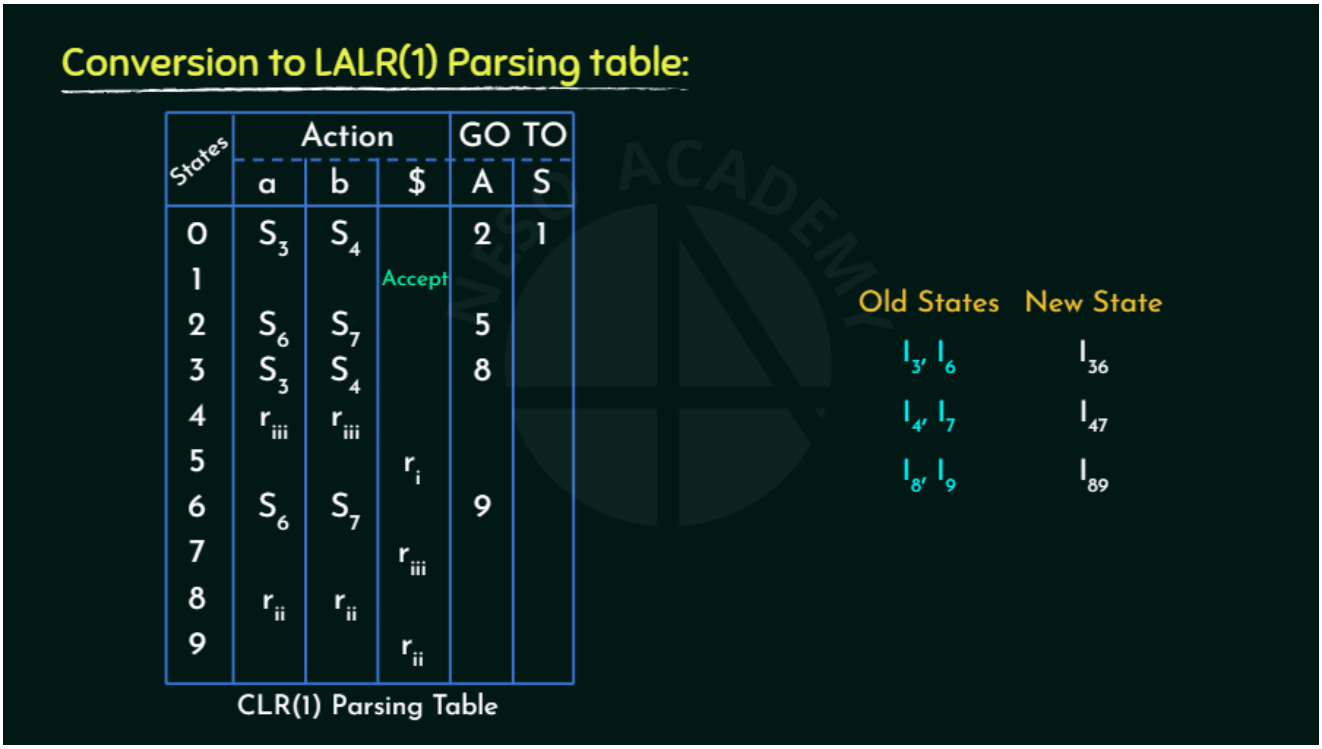
bI8I3AabA → aA., $I9AabOld StatesNew StateI3, I6I36



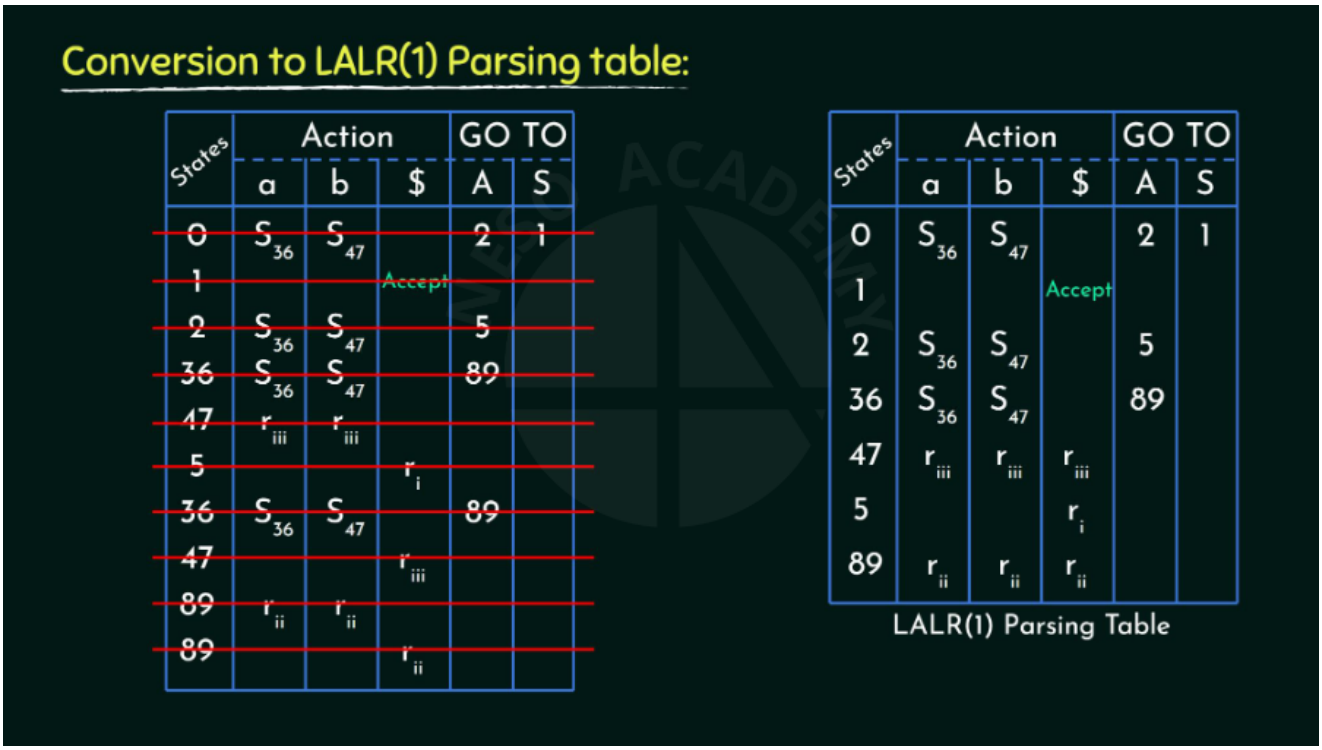
Canonical collection of LR(1) items:I6I2A → .aAA → .bS' → .S, $S → .AA, $A → .aA, a | bA → .b, a | bS → A.A, $A → a.A, a | bA → b., a | b, $, $, a | b, a | bA → .aAA → .bI0S' → S., $I1I4SAabS → AA., $I5AA → a.A, $A → .aAA → .b, $, $aA → b., $I7bA → aA., a | bI8I3AabA → aA., $I9AabOld StatesNew StateI3, I6I36I4, I7I47



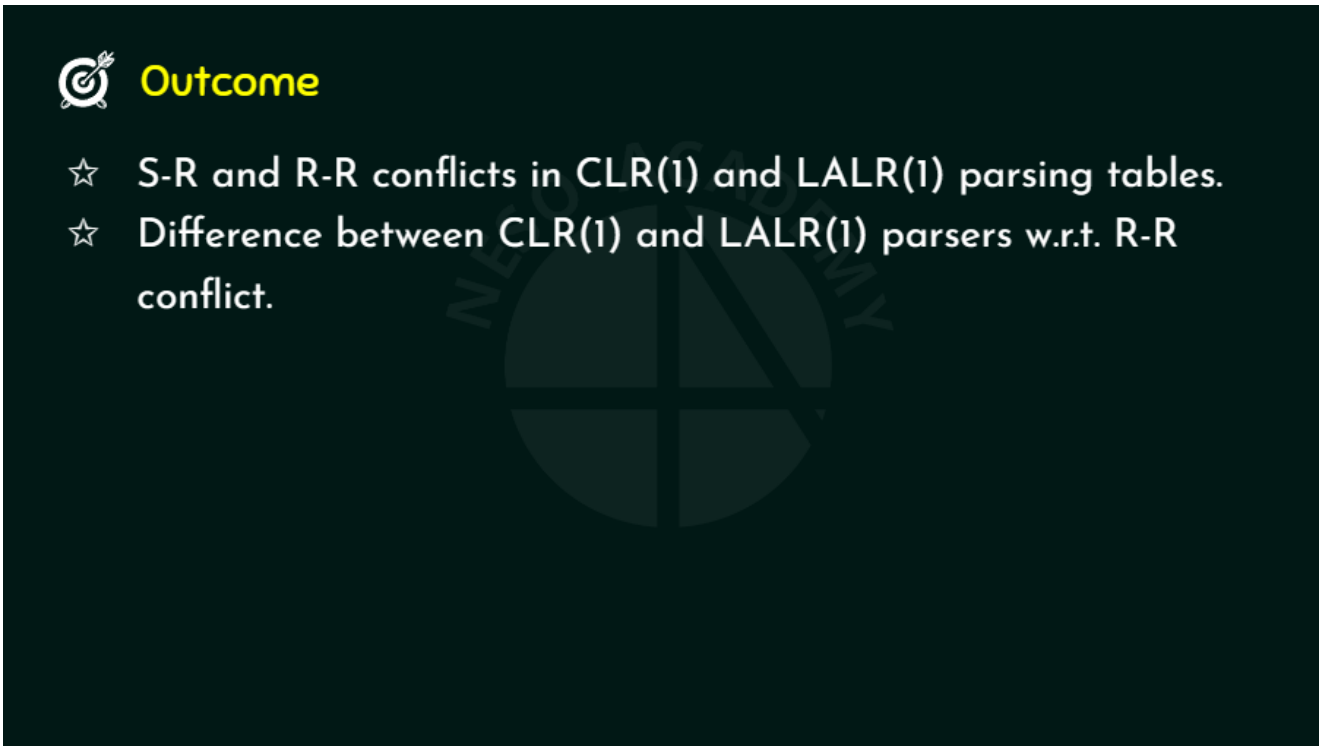
Canonical collection of LR(1) items:I6I2A → .aAA → .bS’ → .S, $S → .AA, $A → .aA, a I bA → .b, a I bS → A.A, $A → a.A, a I bA → b., a I b, $, $, a I b, a I bA → .aAA → .bI0S’ → S., $I1I4SAabS → AA., $I5AA → a.A, $A → .aAA → .b, $, $aA → b., $I7bA → aA., a I bI8I3AabA → aA., $I9AabOld StatesNew StateI3, I6I36I4, I7I8, I9I47I89



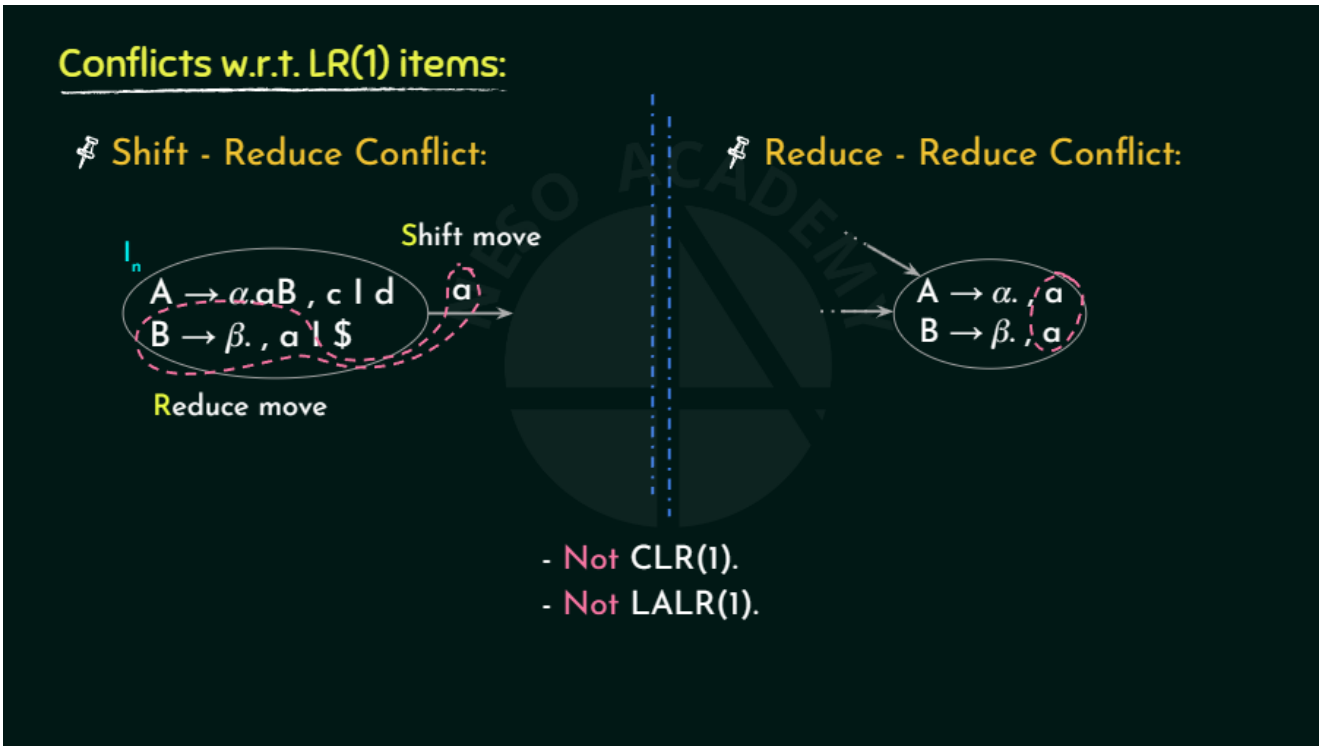
Conversion to LALR(1) Parsing table:Accept1S3S42riiS6S75S3S48riiiriiiriririiriiS6S791092345867riiiActionGO TOStatesab$SACLR(1) Parsing TableOld StatesNew StateI3, I6I36I4, I7I8, I9I47I89



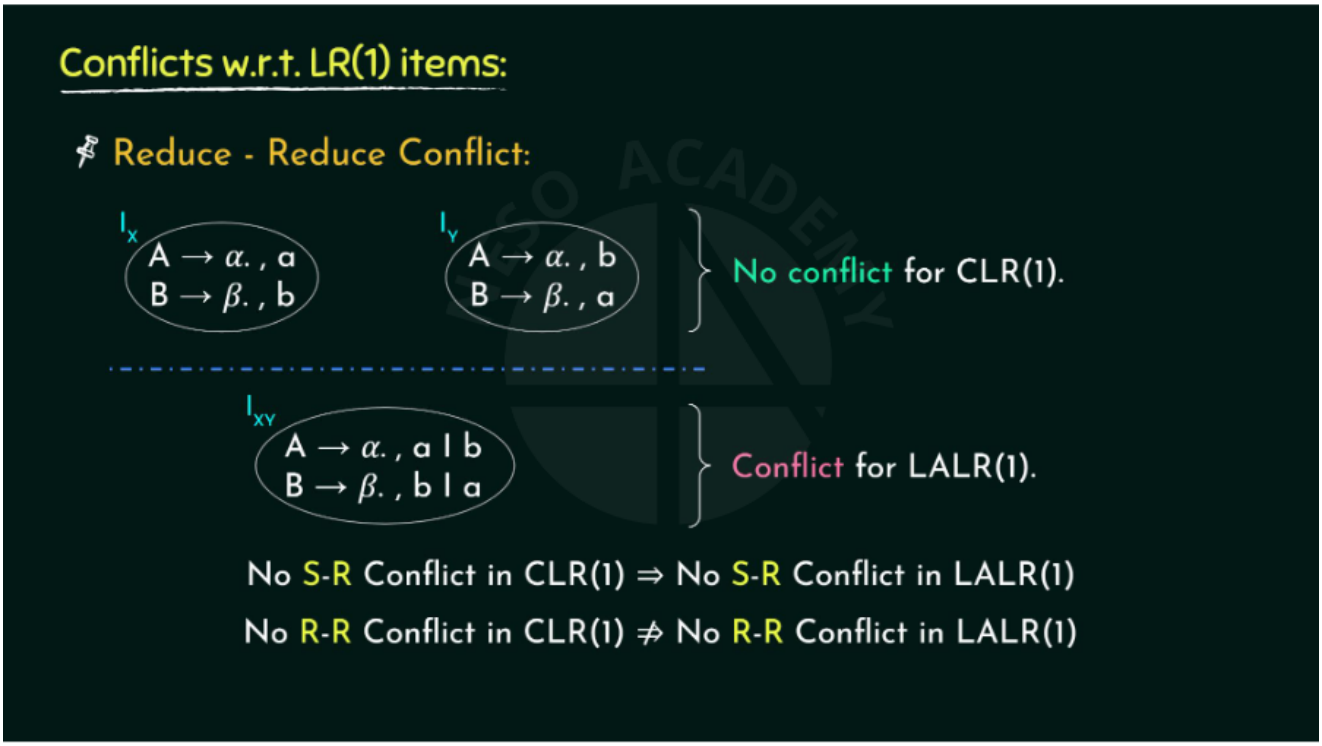
Compiler Design Conflicts in CLR(1) and LALR(1) Parsers

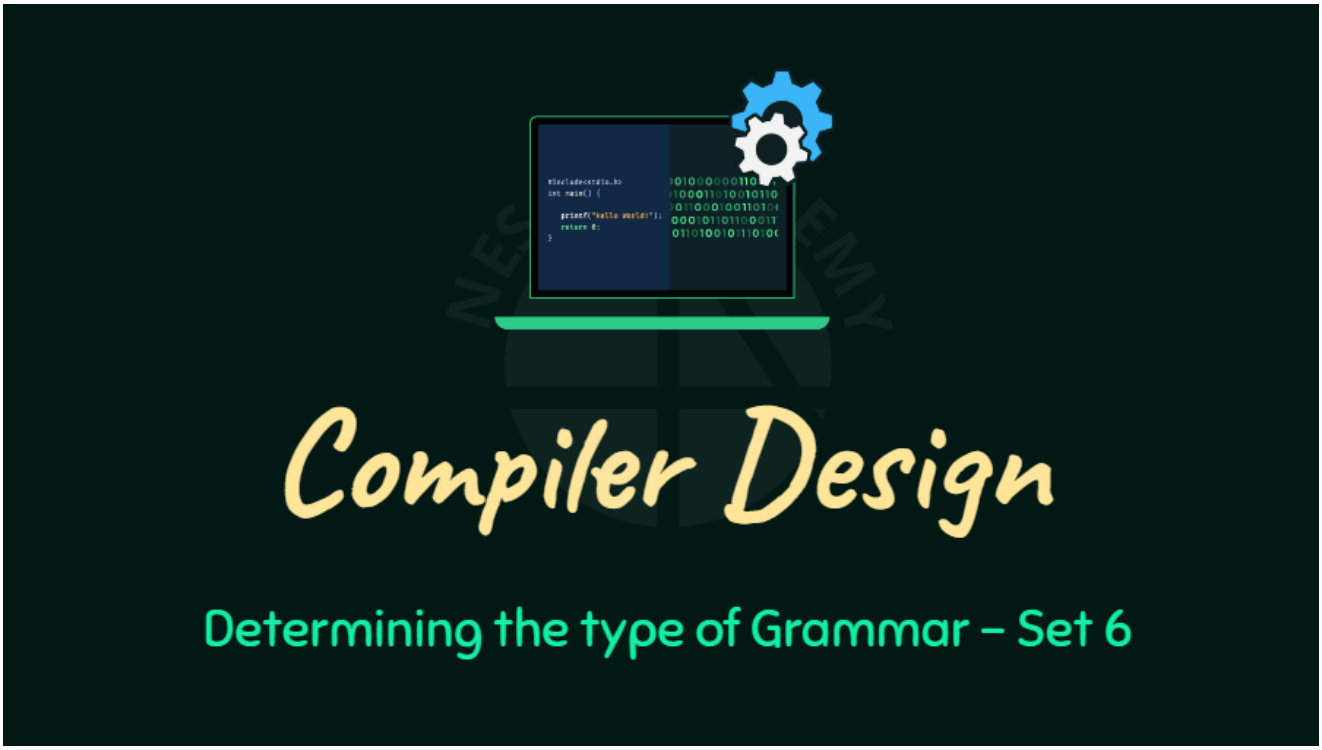


Outcome☆S-R and R-R conflicts in CLR(1) and LALR(1) parsing tables.☆Difference between CLR(1) and LALR(1) parsers w.r.t. R-R conflict.

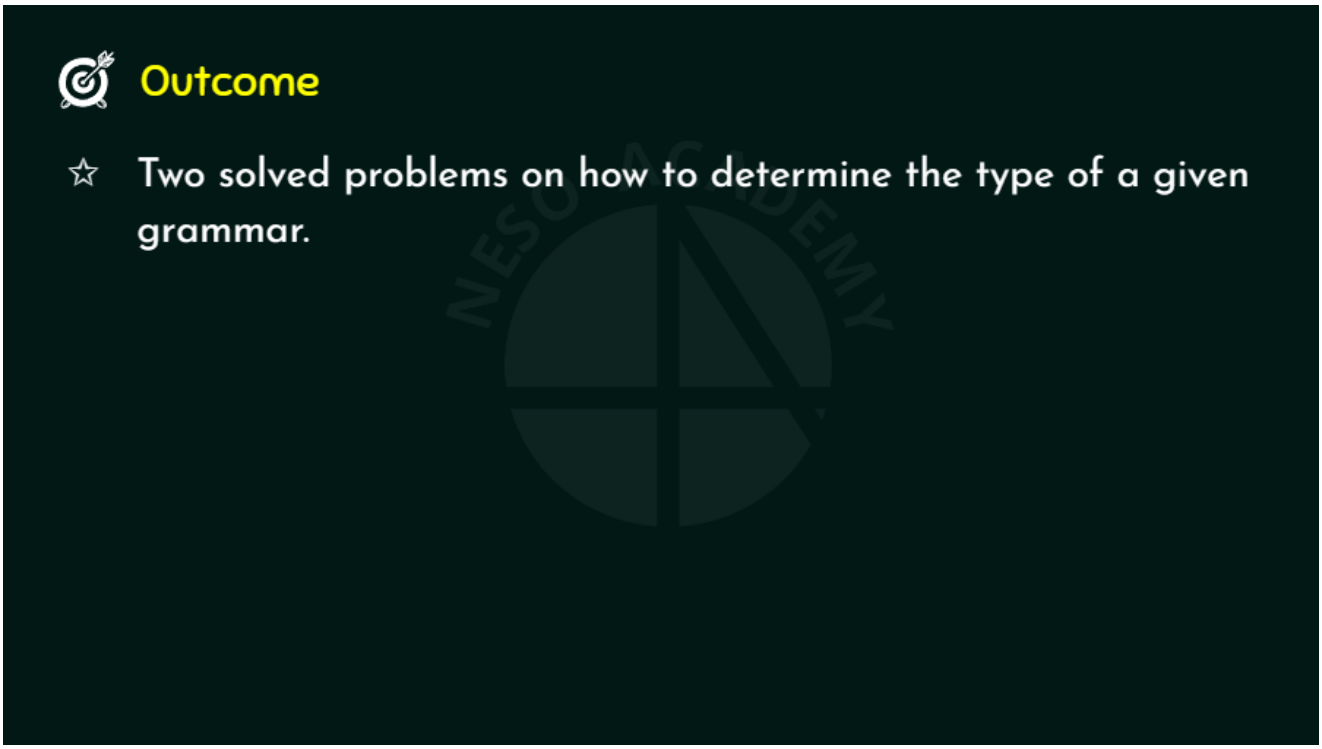


Conflicts w.r.t. LR(1) items:Shift - Reduce Conflict:$A \rightarrow \alpha.$, $aB \rightarrow \beta.$, $aA \rightarrow \alpha.aB$, c I $dB \rightarrow \beta.$, a I $Ina- Not CLR(1).- Not LALR(1).Reduce moveReduce - Reduce Conflict:Shift move

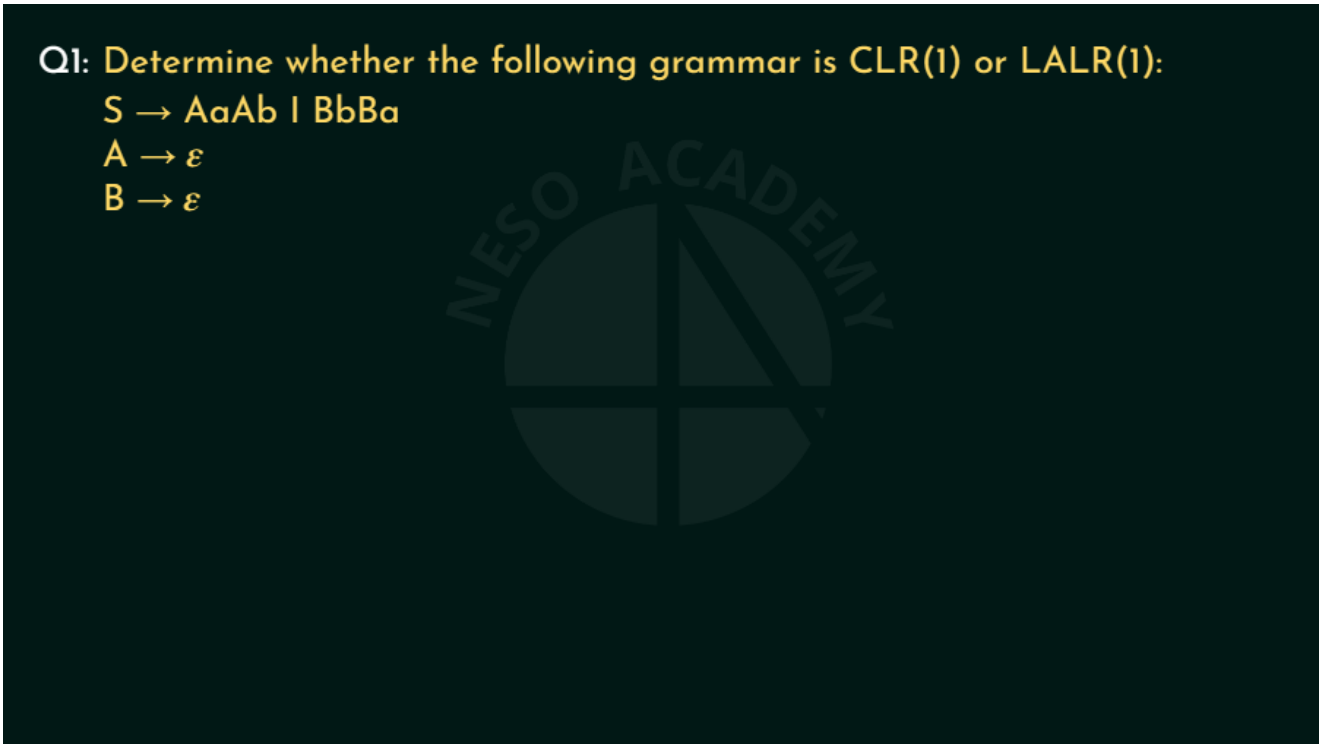




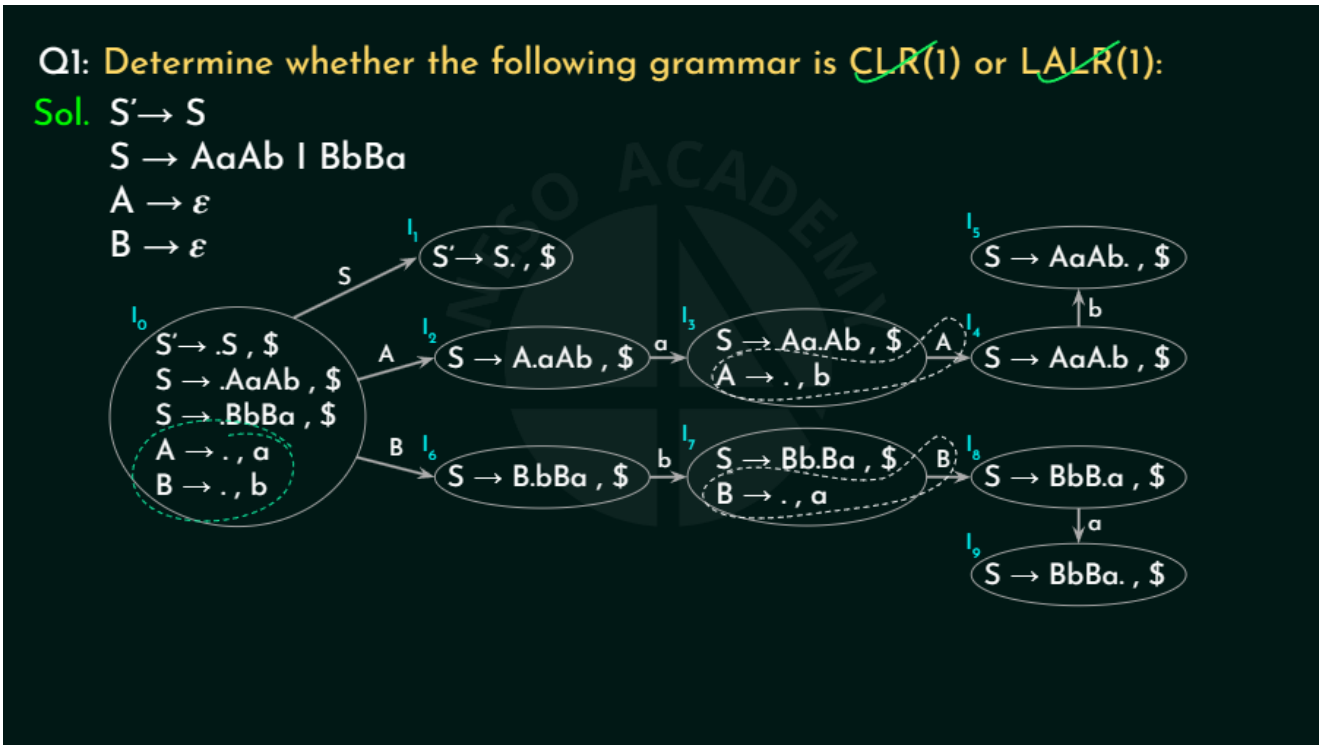
Compiler Design Determining the type of Grammar - Set 6



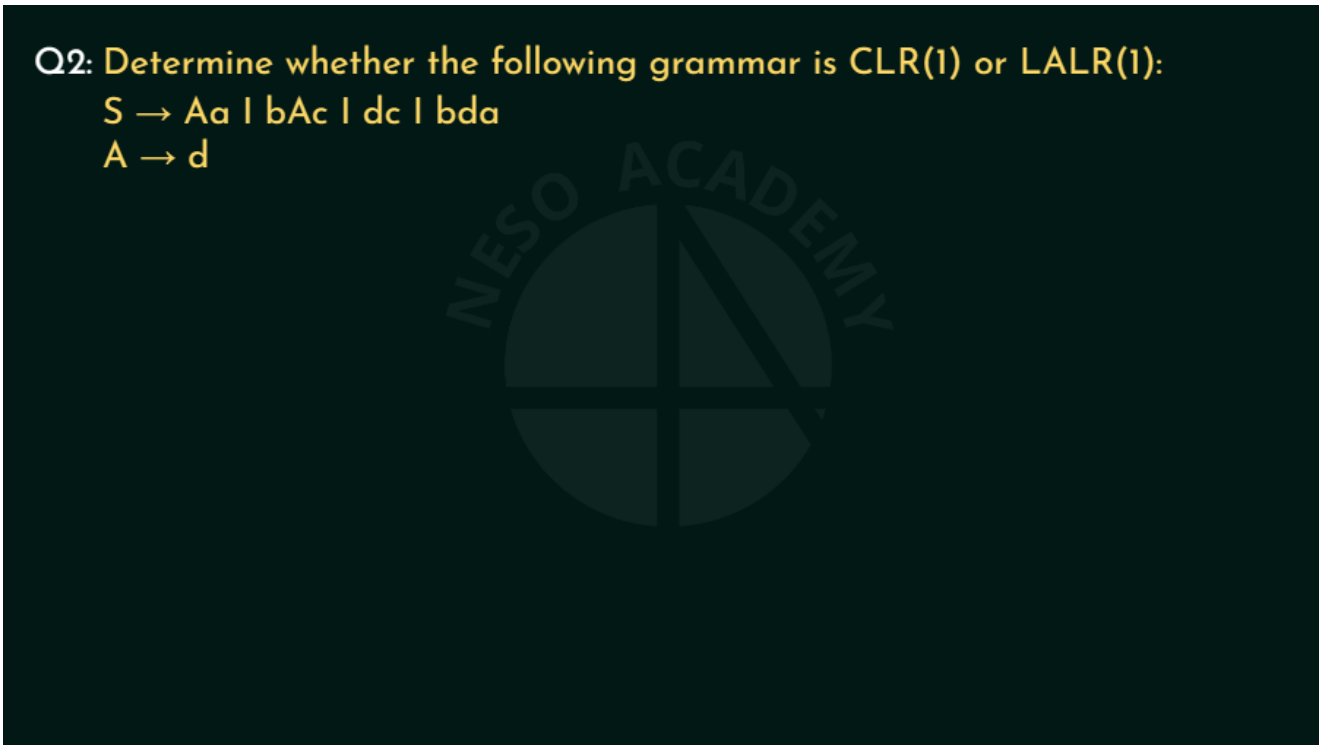
Outcome☆Two solved problems on how to determine the type of a given grammar.



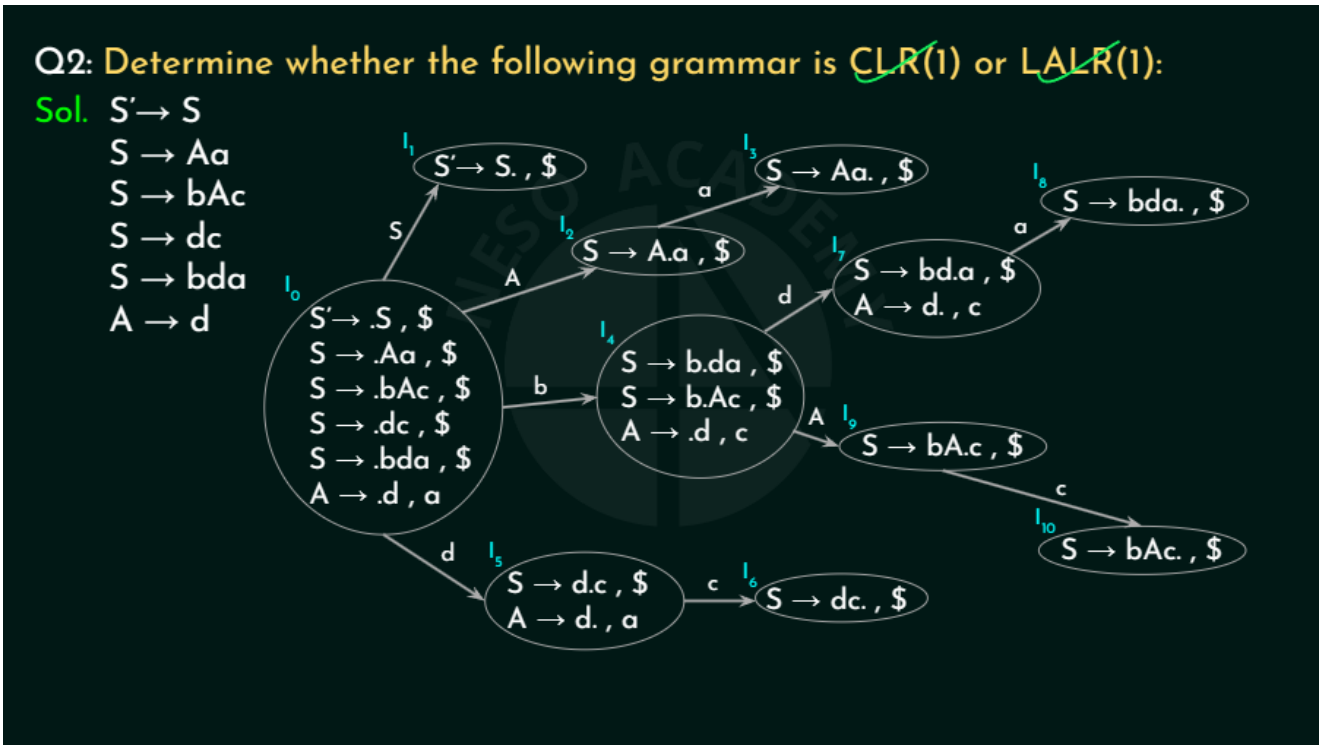
Q1:Determine whether the following grammar is CLR(1) or LALR(1):S → AaAb I BbBaA → εB → ε



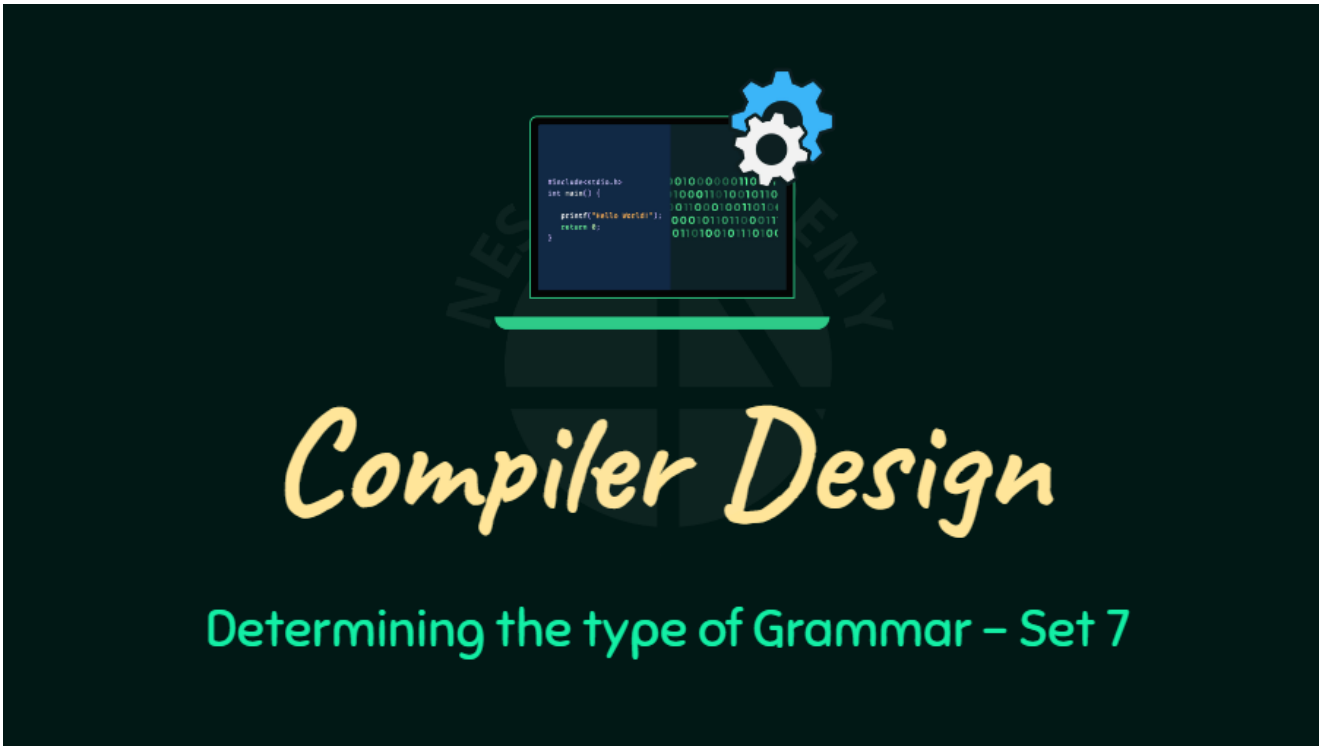
Q1:Determine whether the following grammar is CLR(1) or LALR(1):Sol.S'→ SS → AaAb I BbBaA → εB → εI0S'→ .S , $S → .AaAbS → .BbBaA → .B → ., a, b, $, $SI3S → Aa.Ab , $S → B.bBa , $S → Bb.Ba , $S → BbB.a , $S → BbBa. , $A → .B → ., b, aS → AaA.b , $I4S → AaAb. , $I5I7S → A.aAb , $I2I8I9I6aAbBbaABS'→ S. , $I1



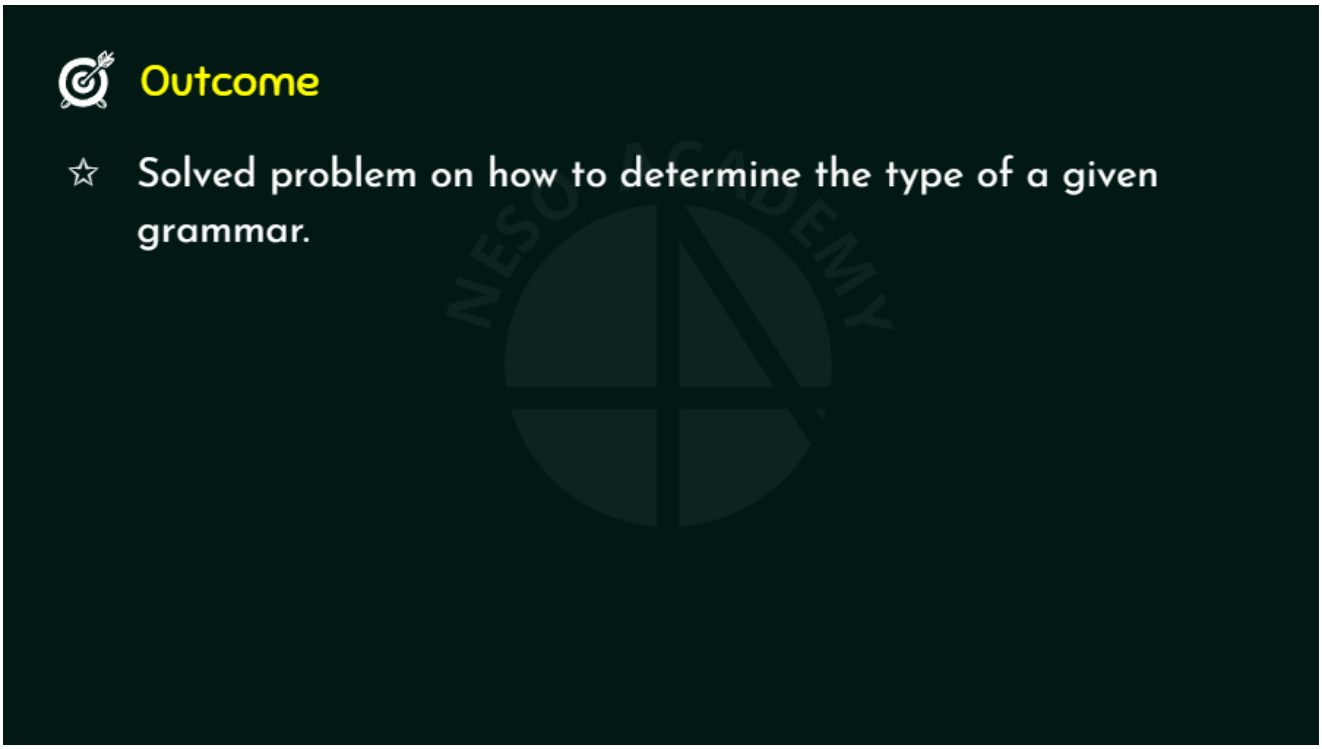
Q2:Determine whether the following grammar is CLR(1) or LALR(1):S → Aa I bAc I dc I bdaA → d



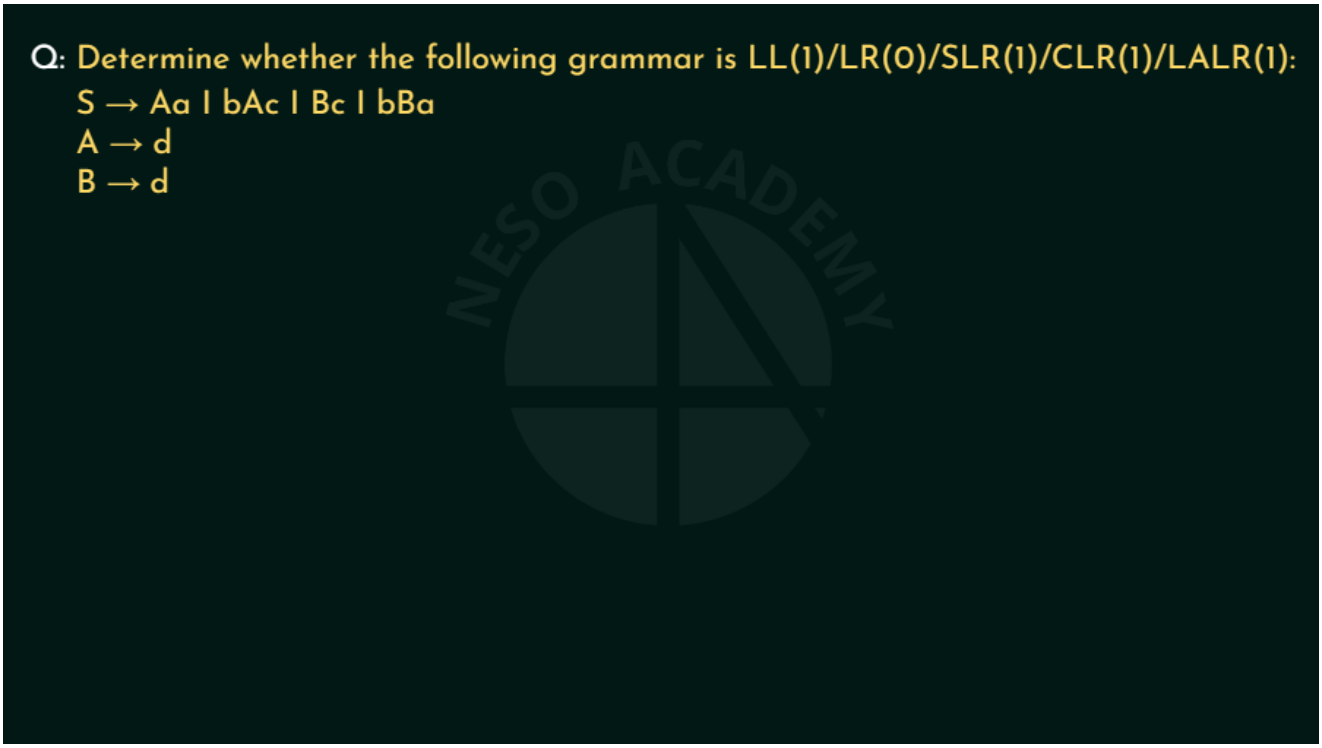
I4S → b.da , $S → b.Ac , $Q2:Determine whether the following grammar is CLR(1) or LALR(1):Sol.S'→ SS → Aa S → bAc S → dcS → bdaA → dI0S'→ .S , $S → .Aa , $ S → .bAc , $S → .dc , $S → .bda , $A → .d , aS'→ S. , $I1S → A.a , $ I2S → Aa. , $ I3A → .d, cS → d.c , $A → d. , aI5S → dc. , $ I6S → bd.a , $A → d. , cI7S → bda. , $I8S → bA.c , $I9S → bAc. , $I10ASAabdcdca



Compiler Design Determining the type of Grammar - Set 7



Outcome☆Solved problem on how to determine the type of a given grammar.



Q:Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):S → Aa I bAc I Bc I bBaA → dB → d

Q: Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):

Sol.

	FIRST	FOLLOW
S → Aa I bAc I Bc I bBa	{d, b}	{$}
A → d	{d}	{a, c}
B → d	{d}	{c, a}

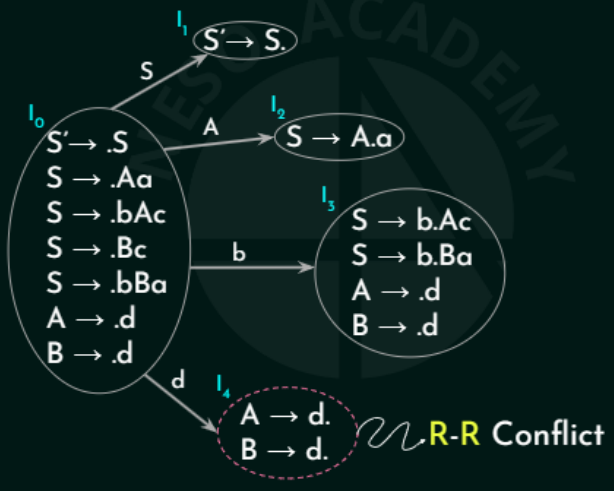
	a	b	c	d	$
S		S → bAc S → bBa		S → Aa S → Bc	
A				A → d	
B				B → d	

LL(1) Parsing Table

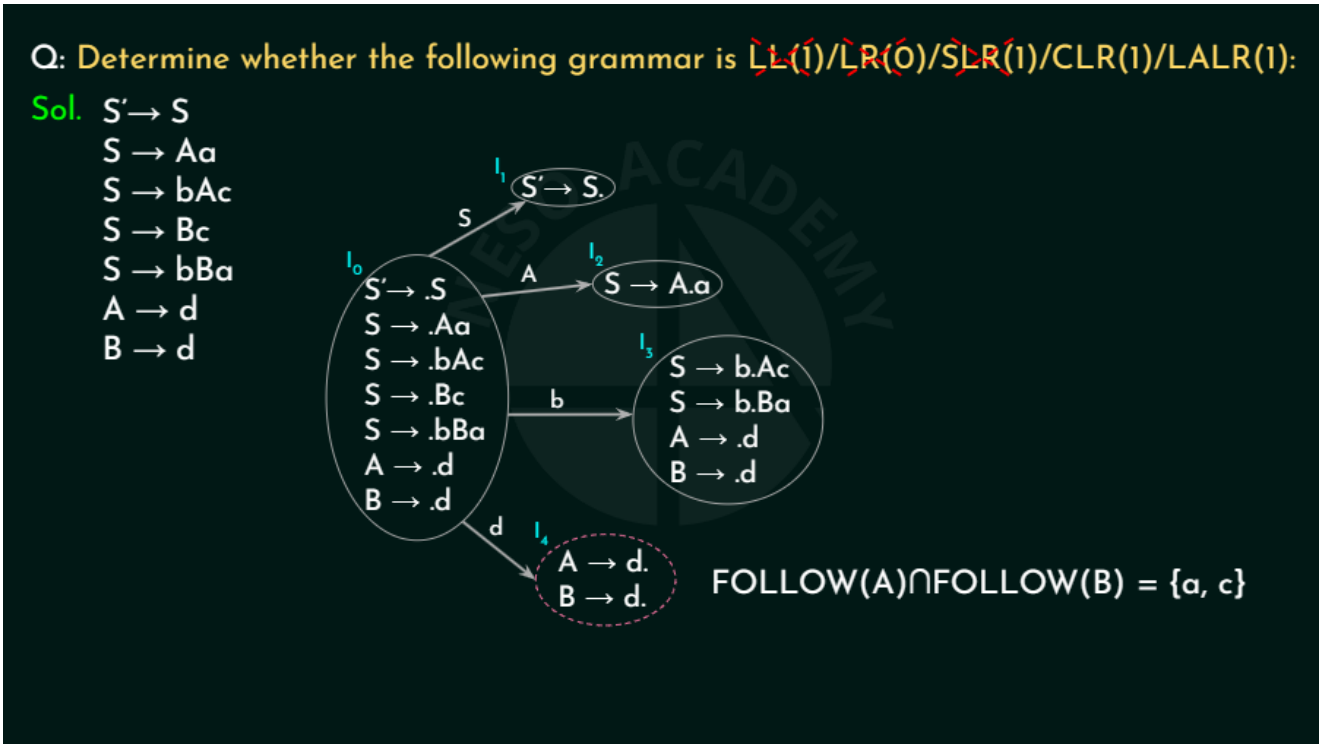
, a}, c}, b}Q:Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):S → Aa I bAc I Bc I bBaA → dB → dFOLLOW{$}FIRST{d}{d}S → bAcB → dA → dS → AaSol.LL(1) Parsing TableadSAB$cbS → bBaS → Bc{d{a{c

Q: Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):

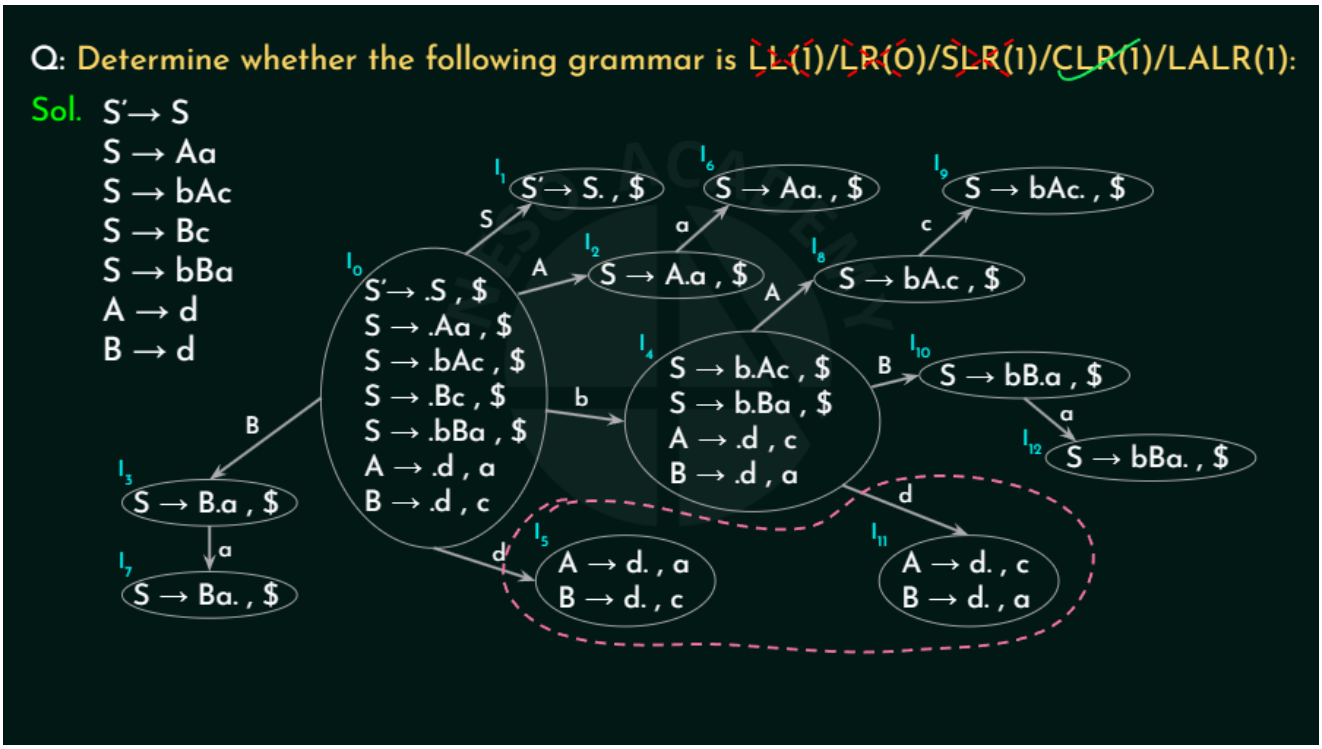
Sol. S'→ S
S → Aa
S → bAc
S → Bc
S → bBa
A → d
B → d



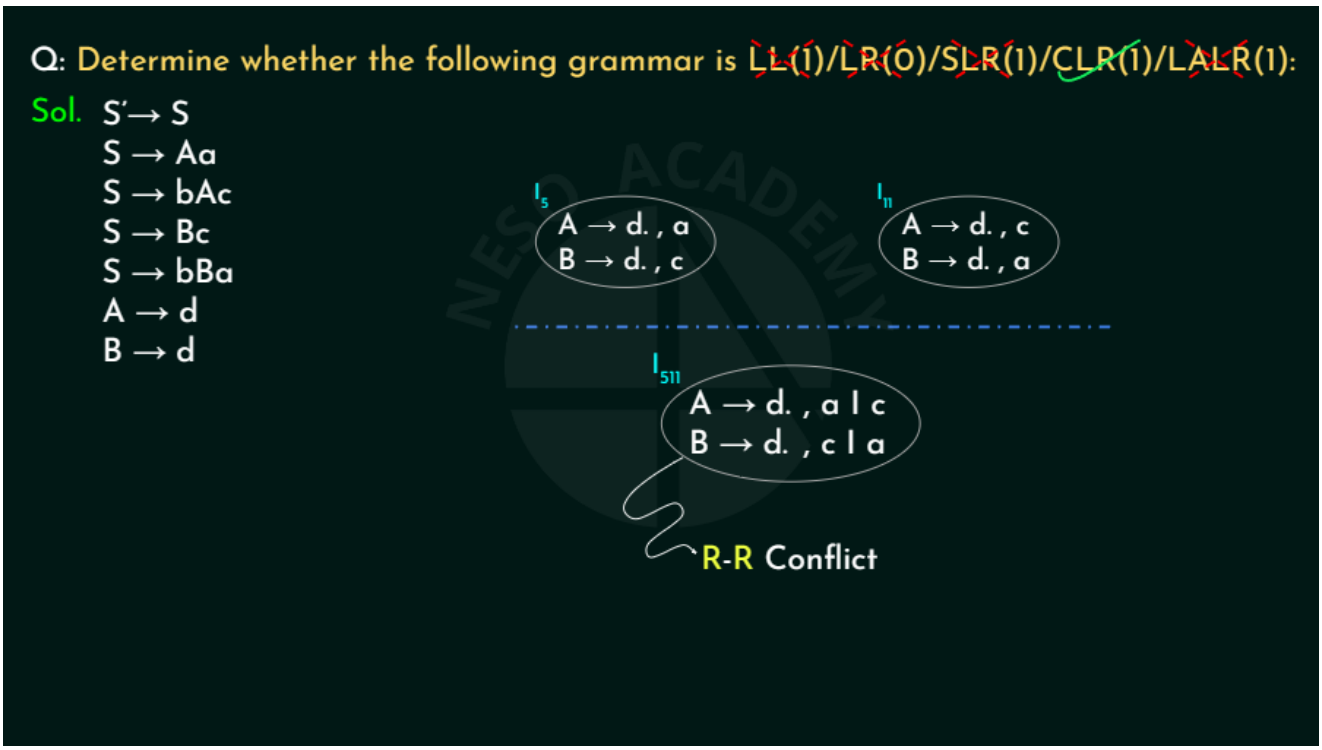
I0S'→ .SQ:Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):Sol.S'→ SS → Aa S → bAc S → Bc S → bBaA → dB → dS → .Aa S → .bAc S → .Bc S → .bBaA → .dB → .dS'→ S.I1S → A.a I2I3S → b.Ac S → b.BaA → .dB → .dA → d.B → d.I4SAbdR-R Conflict



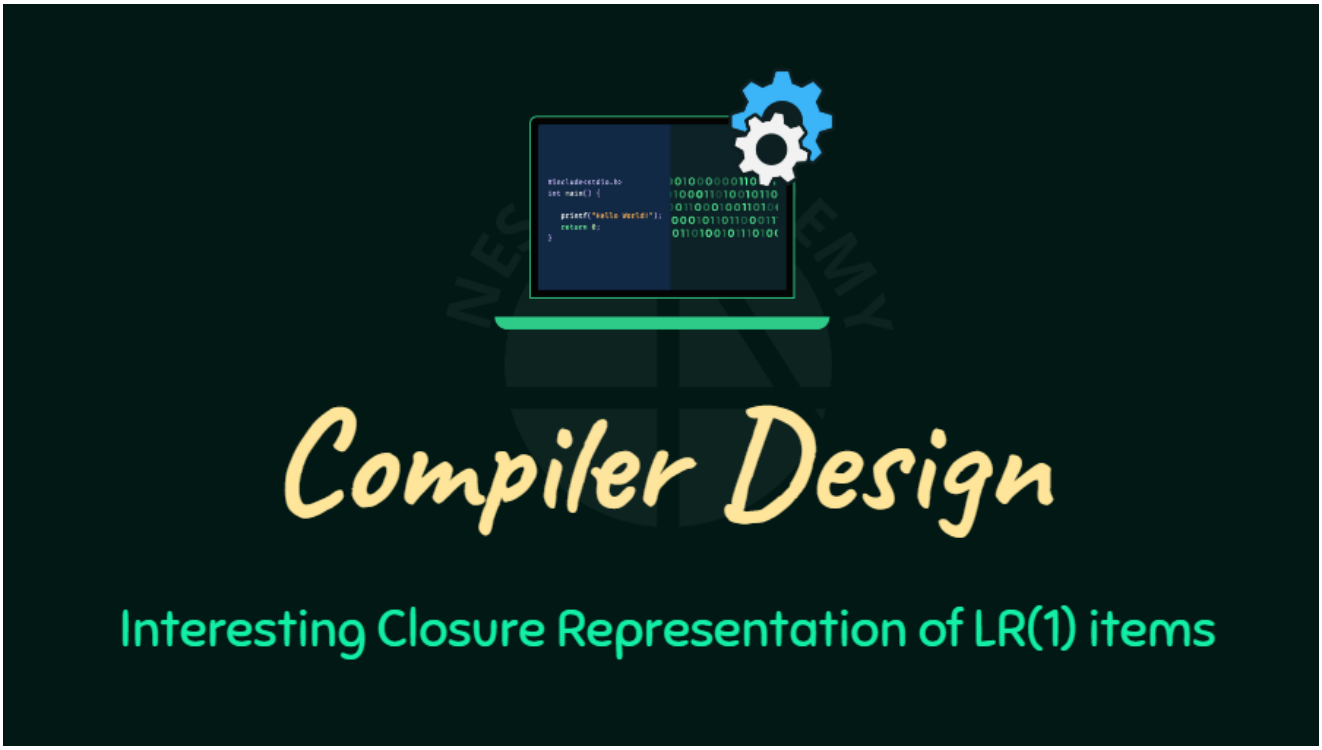
I0S'→ .SQ:Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):Sol.S'→ SS → Aa S → bAc S → Bc S → bBaA → dB → dS → .Aa S → .bAc S → .Bc S → .bBaA → .dB → .dS'→ S.I1S → A.a I2I3S → b.Ac S → b.BaA → .dB → .dA → d.B → d.I4SAbdFOLLOW(A)∩FOLLOW(B)= {a, c}



S → bBa. , $I12Q:Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):Sol.I0S'→ .S , $S'→ SS → Aa S → bAc S → Bc S → bBaA → dB → dS → .Aa , $ S → .bAc , $S → .Bc , $ S → .bBa , $A → .dB → .dS'→ S. , $I1S → A.a , $ I2I4S → b.Ac , $ S → b.Ba , $A → .dB → .dA → d. , aB → d. , cI5S, c, aS → bA.c , $ I8S → Aa. , $ I6aS → bAc. , $ I9AacbBS → bB.a , $I10dA → d. , cB → d. , aI11d, a, cAS → B.a , $ I3S → Ba. , $ I7Ba

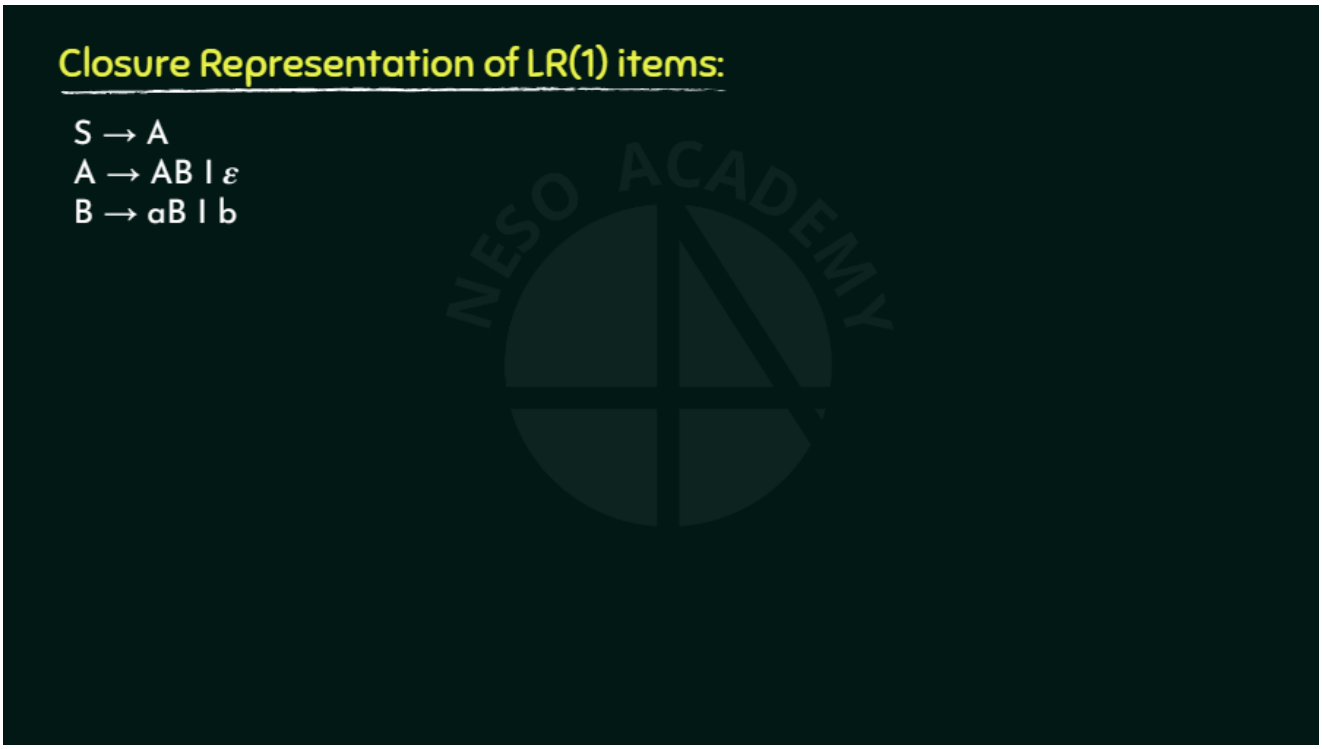


Q:Determine whether the following grammar is LL(1)/LR(0)/SLR(1)/CLR(1)/LALR(1):Sol.S'→ SS → Aa S → bAc S → Bc S → bBaA → dB → dA → d. , aB → d. , cI5A → d. , cB → d. , aI11A → d.B → d. I511, a I c, c I aR-R Conflict

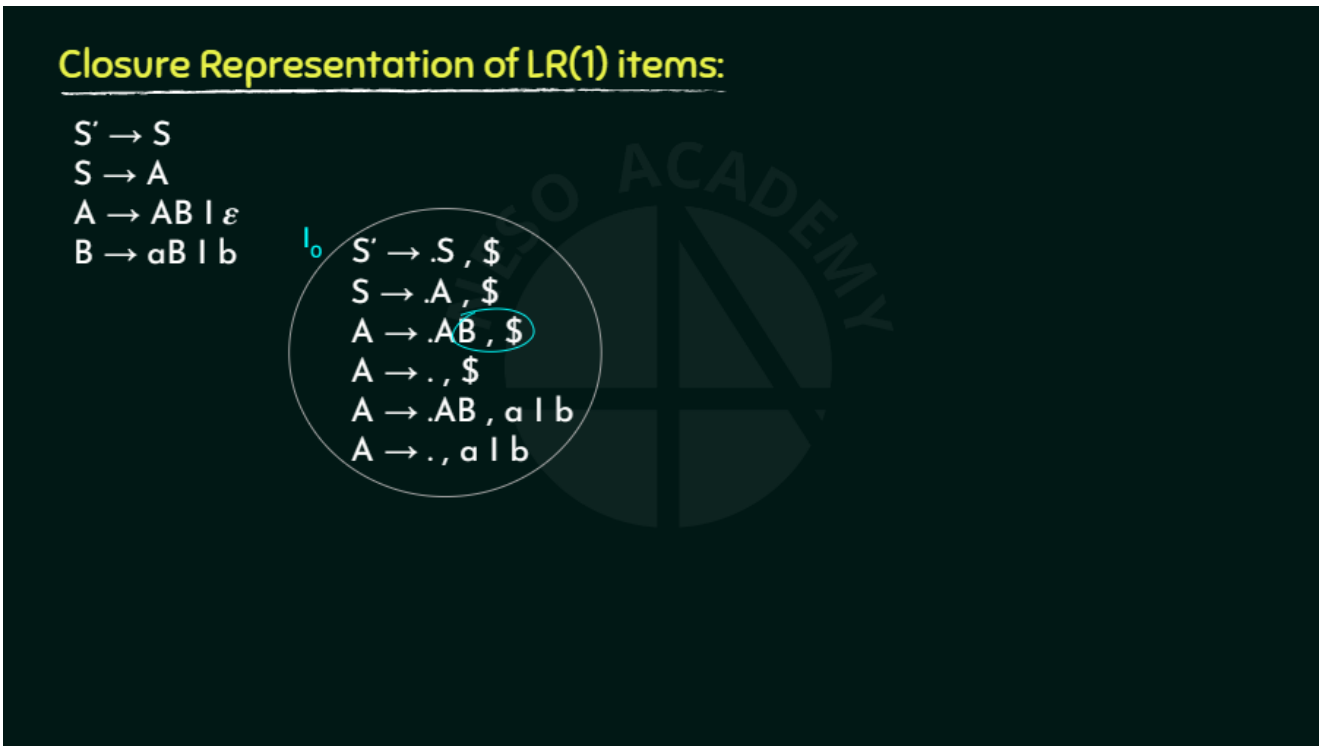


Compiler Design Interesting Closure Representation of LR(1) items

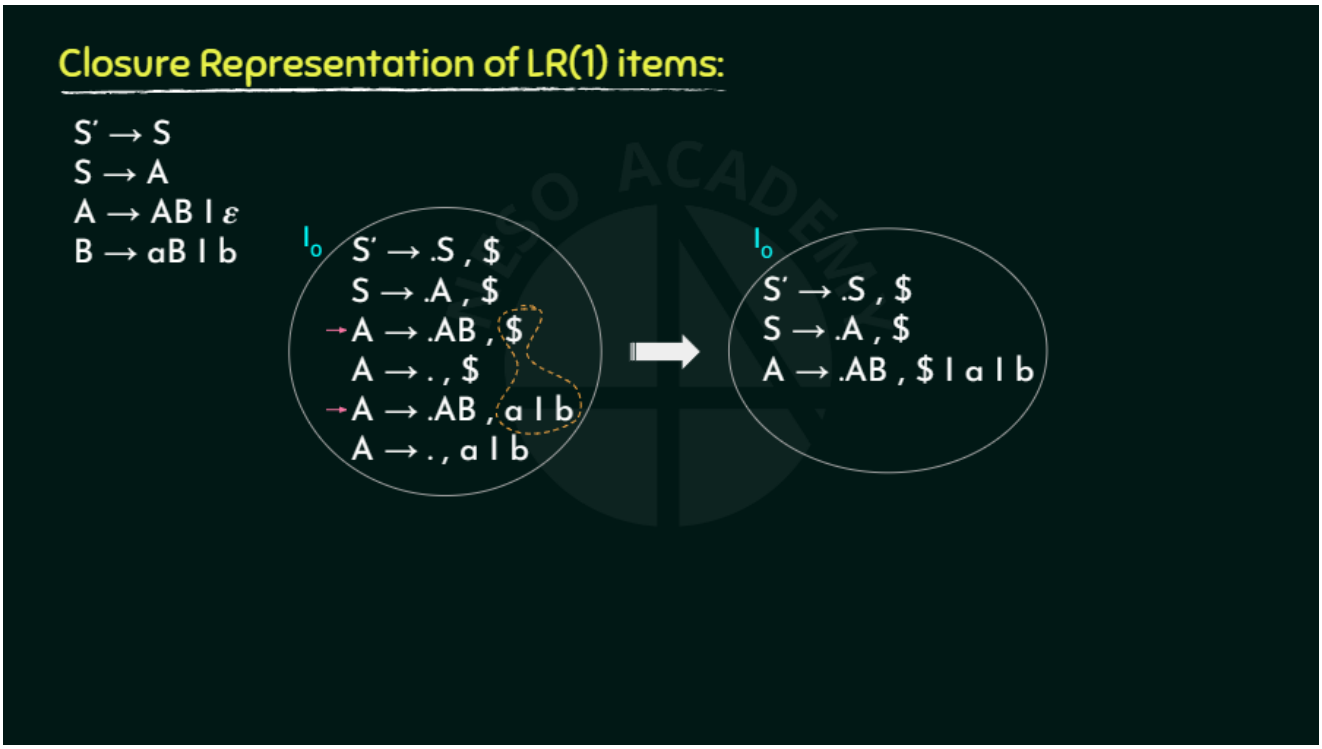
Outcome☆Canonical Collection of LR(1) items with merged Look-aheads.



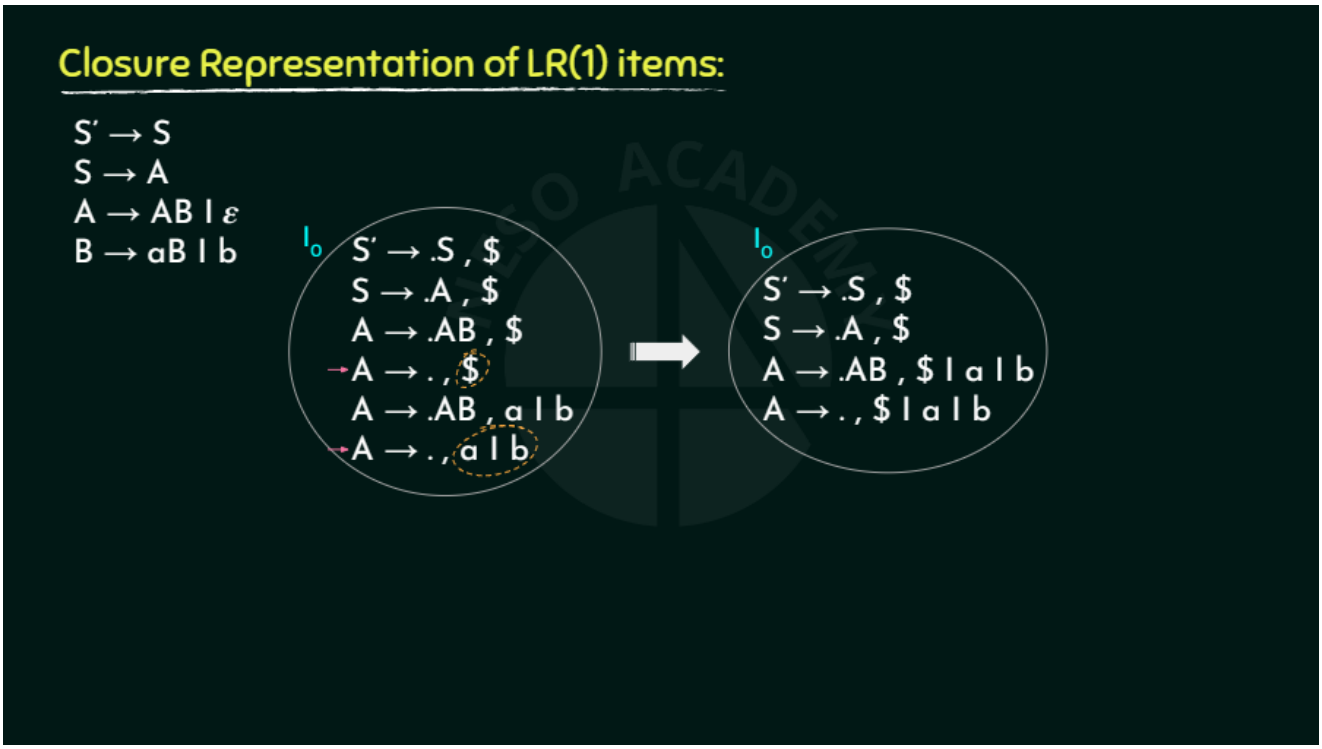
Closure Representation of LR(1) items:S → AA → AB I εB → aB I b



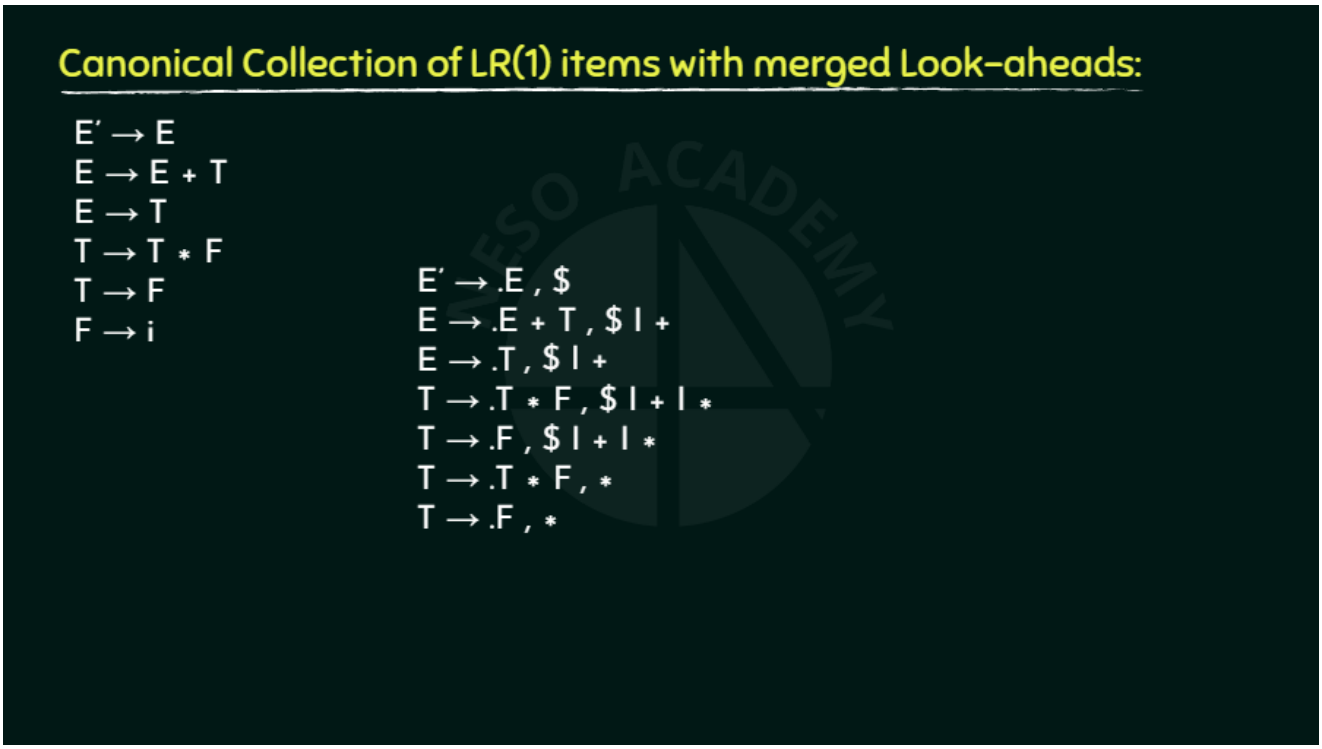
S' → SS → AA → AB I εB → aB I bI0S' → .S, $S → .AA → .ABA → .A → .ABA → ., $, $, $, a I b, a I bClosure Representation of LR(1) items:



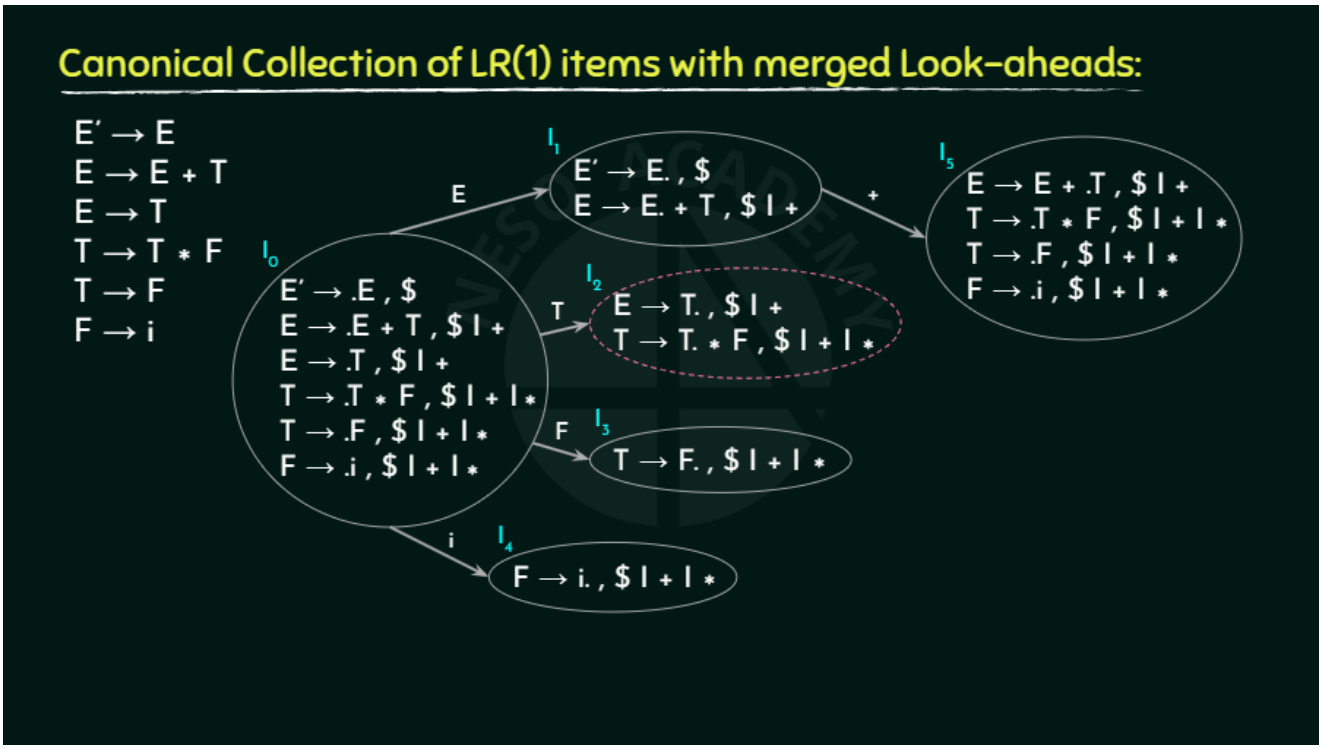
I0S' → .S , $S → .A , $S' → SS → AA → AB I εB → aB I bI0S' → .S, $S → .AA → .ABA → .A → .ABA → ., $, $, $, a I b, a I bA → .AB , $ I a I bClosure Representation of LR(1) items:



I0S' → SS → AA → AB I εB → aB I bI0S' → .S, $S → .AA → .ABA → .A → .ABA → ., $, $, $, a I b, a I bS' → .S , $S → .A , $A → .AB , $ I a I bA → . , $ I a I bClosure Representation of LR(1) items:

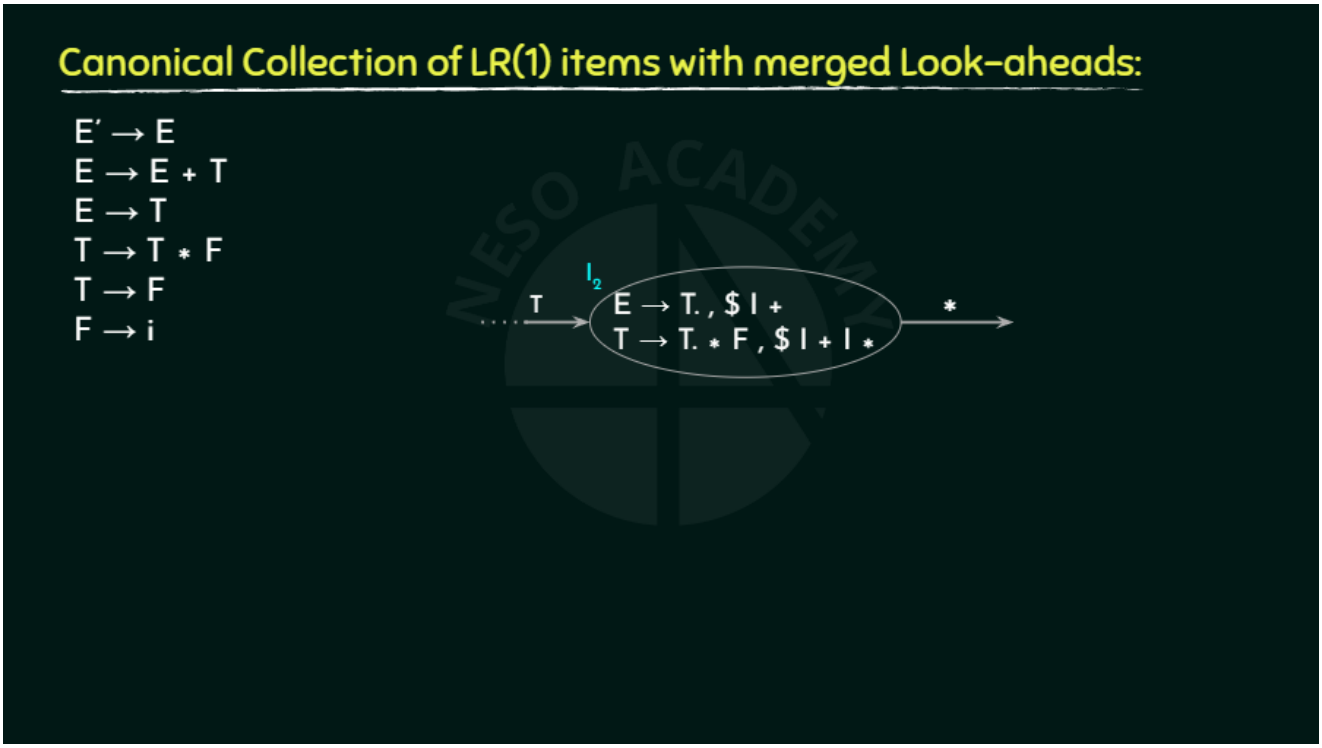


E’ → EE → E + TE → TT → T + FT → FF → iCanonical Collection of LR(1) items with merged Look-aheads:E’ → .E , $E → .E + TE → .T , $, $*I +I +T → .T + FT → .F*T → .T + FT → .F*, +*, +*, $I +, $I +I +*I +*

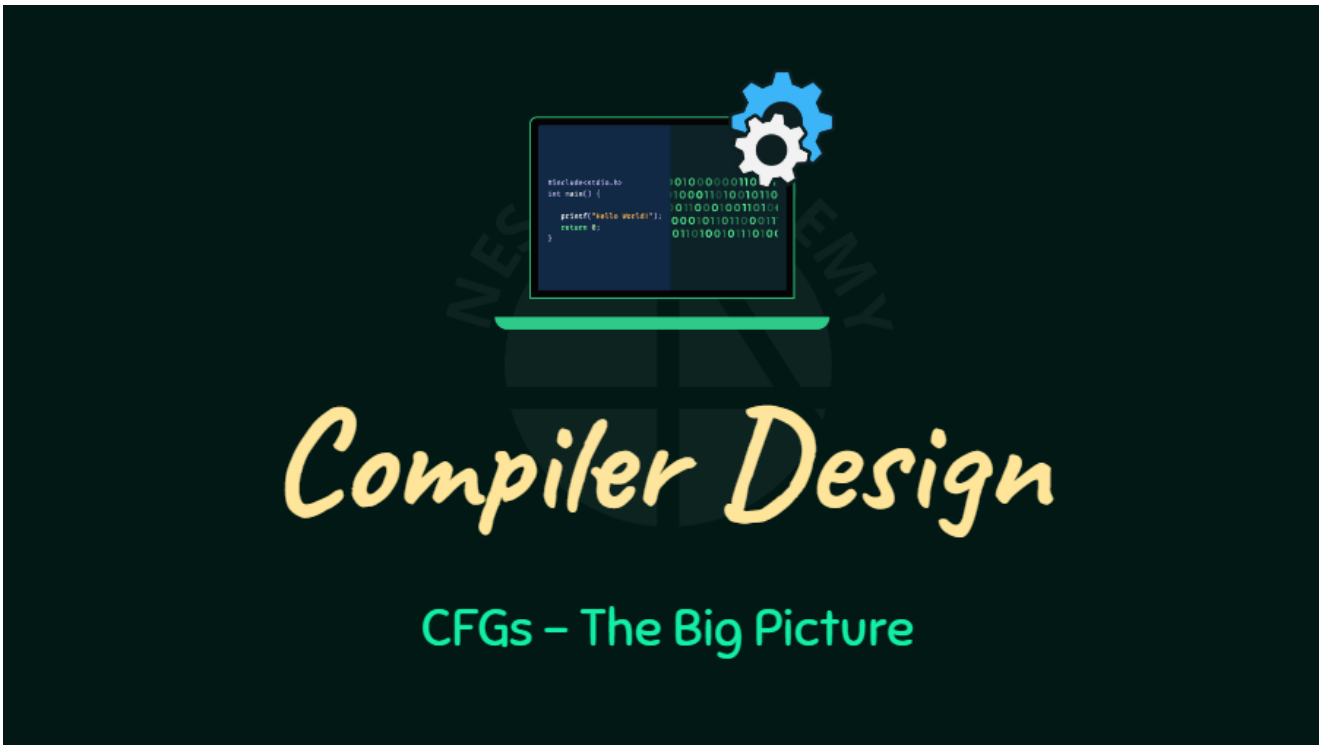


I5T → .T + F , $ I + T → .F , $ I +E’ → EE → E + TE → TT → T + FT → FF → iCanonical Collection of LR(1) items with merged Look-aheads:*I0E’ → .E , $E → .E + T , $ I + E → .T , $ I +T → .T + F , $ I + I +T → .F , $ I + I +F → .i , $ I + I + ***E’ → E. , $E → E. + T , $ I

+I1EE → T. , $ I +T → T. + F , $ I + I + *I2TI4F → i. , $ I + I + *I3T → F. , $ I + I +*FiI +*I +*, $ I + I + *F → .i E → E + .T , $ I + +***



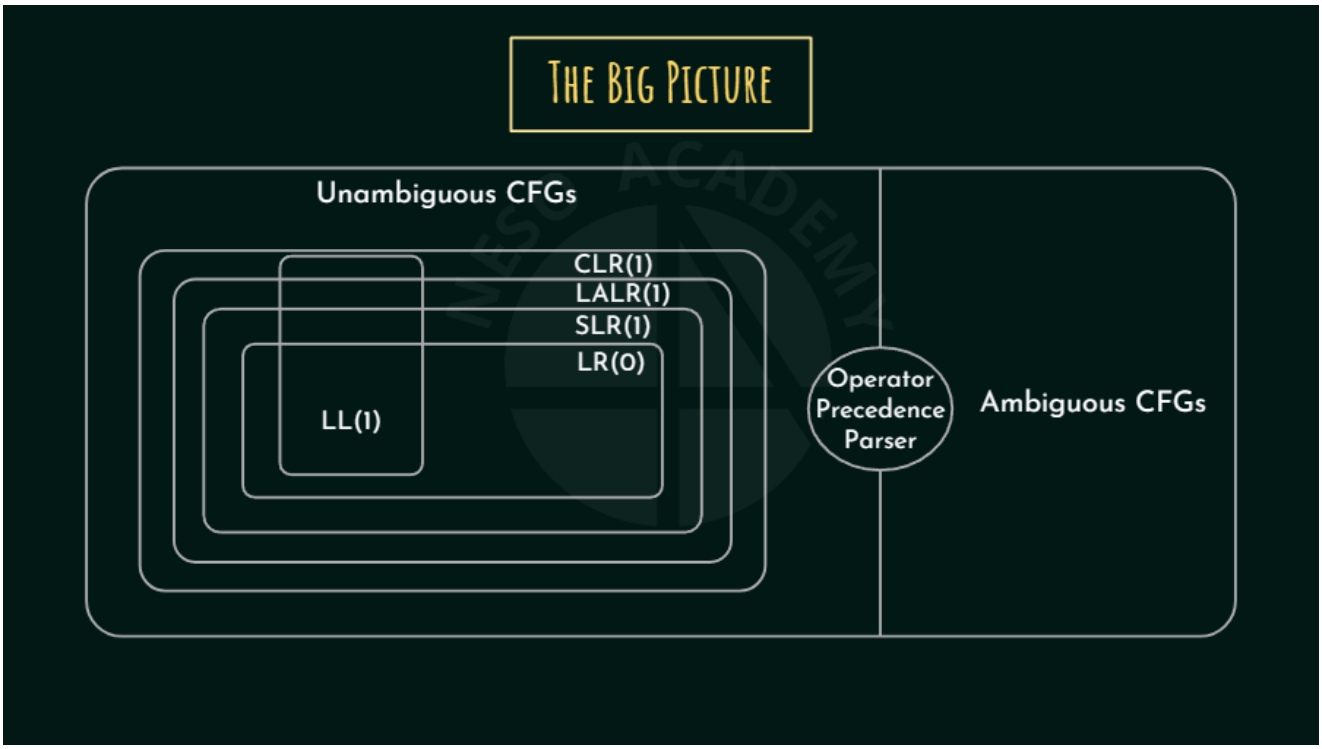
E' → EE → E + TE → TT → T + FT → FF → iCanonical Collection of LR(1) items with merged Look-aheads:*E → T. , $ I +T → T. + F , $ I + I + *I2*T*



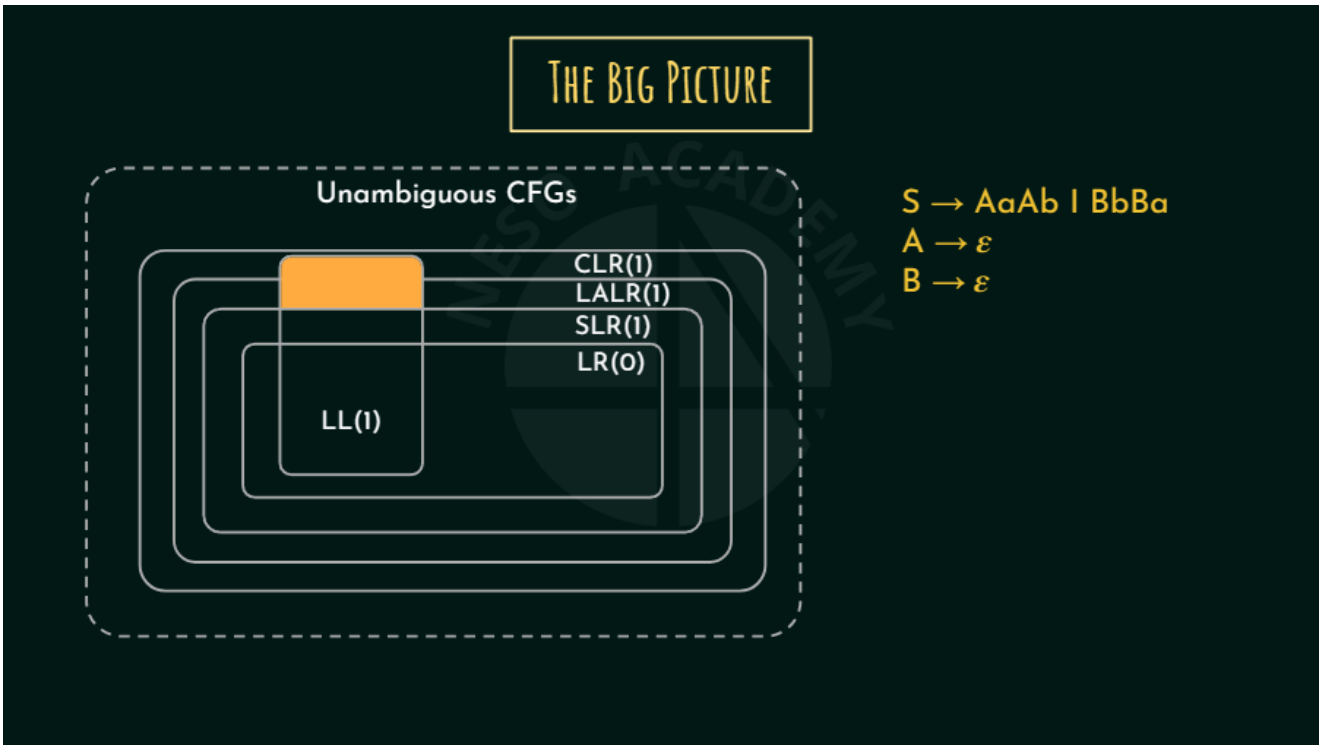
Compiler Design CFGs - The Big Picture

Outcome☆Relationship of all Context Free Grammars.

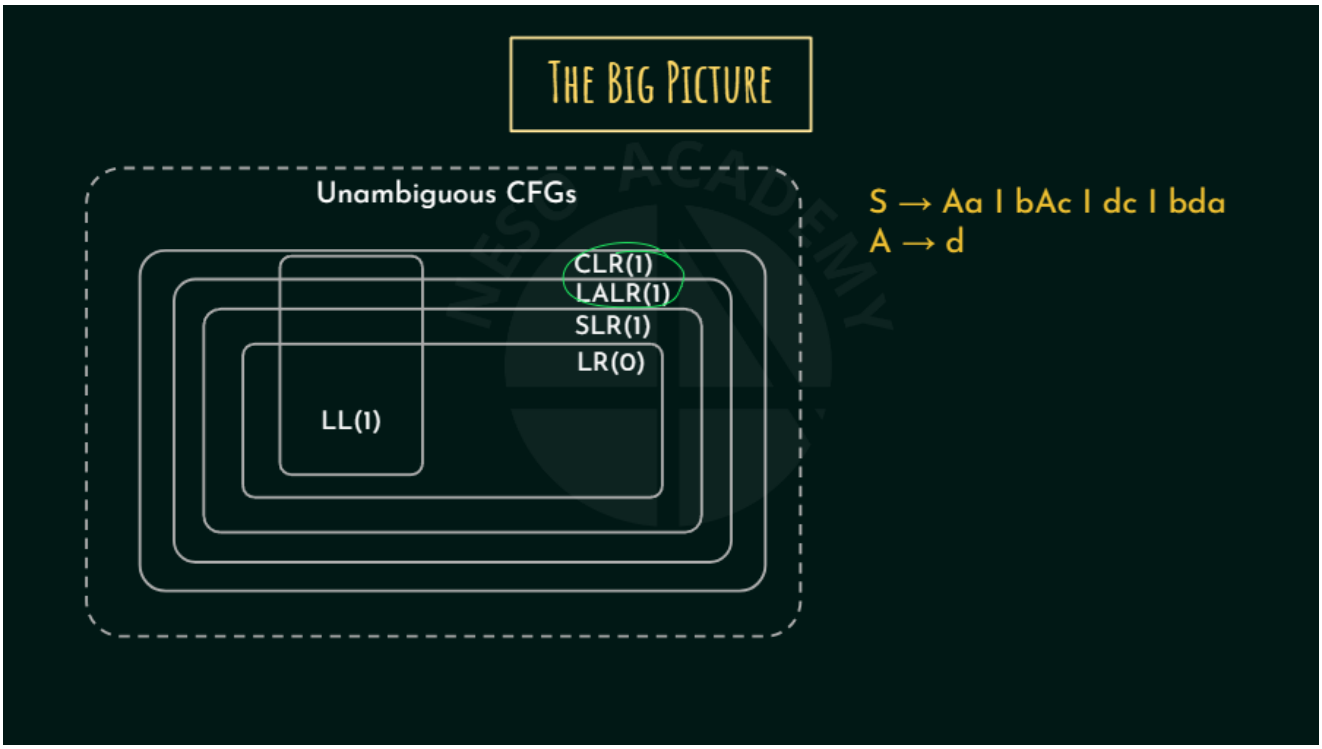
The Big PictureUnambiguous CFGsAmbiguous CFGsOperatorPrecedenceParser



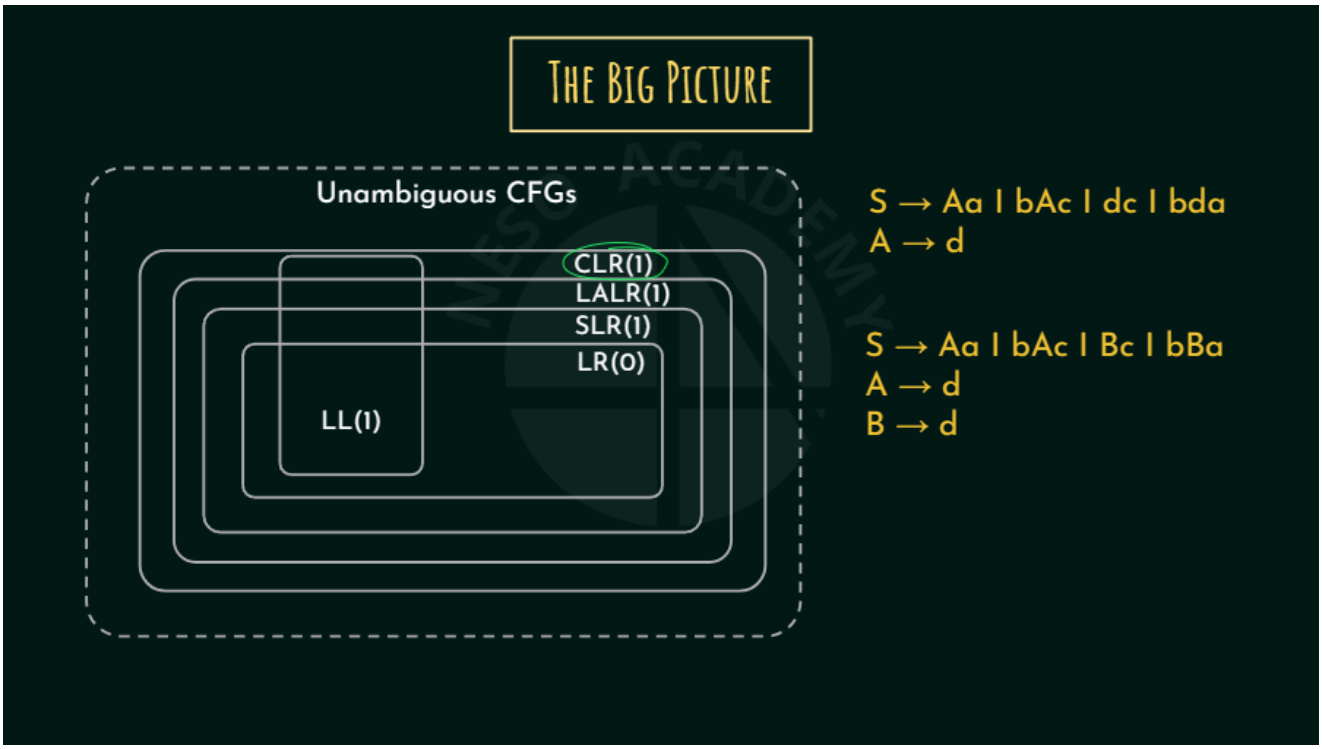
SLR(1)LALR(1)CLR(1)The Big PictureUnambiguous CFGsAmbiguous CFGsOperatorPrecedenceParserLR(0)LL(1)



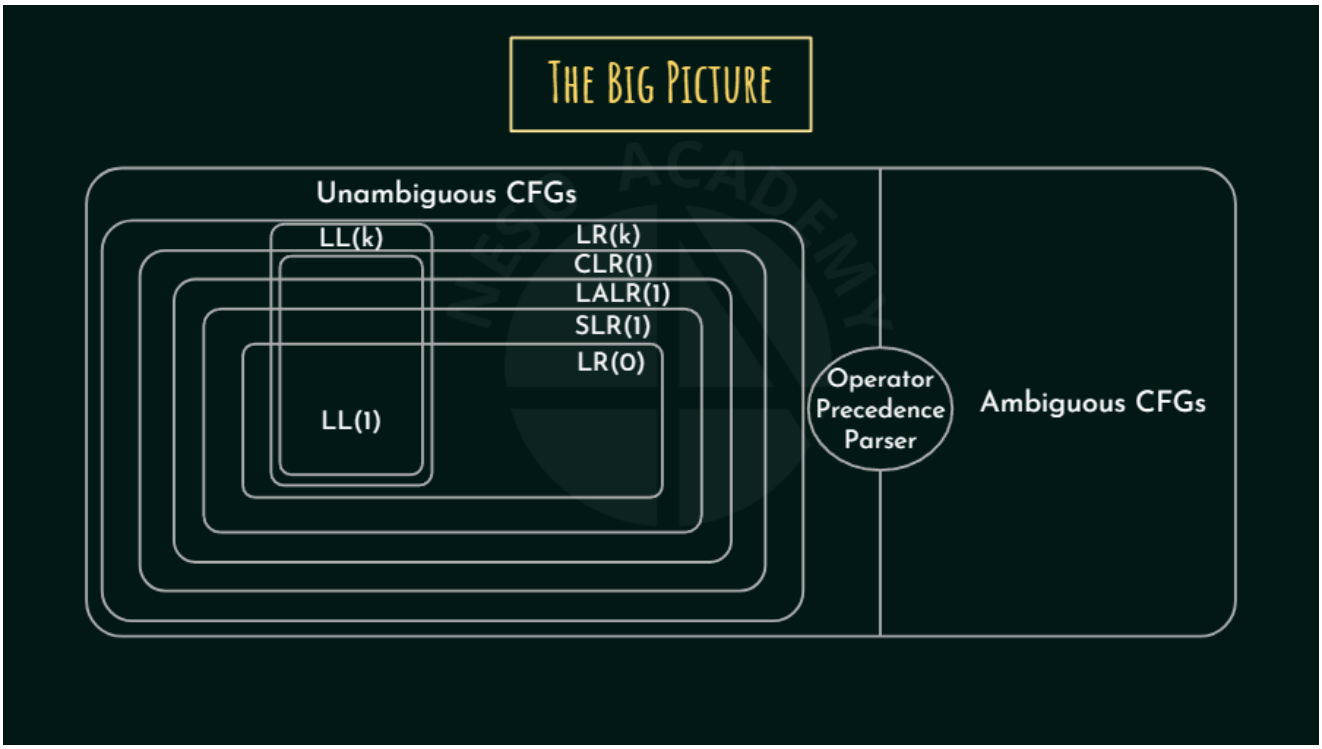
SLR(1)LALR(1)CLR(1)The Big PictureUnambiguous CFGsLR(0)LL(1)S → AaAb I BbBaA → εB → ε



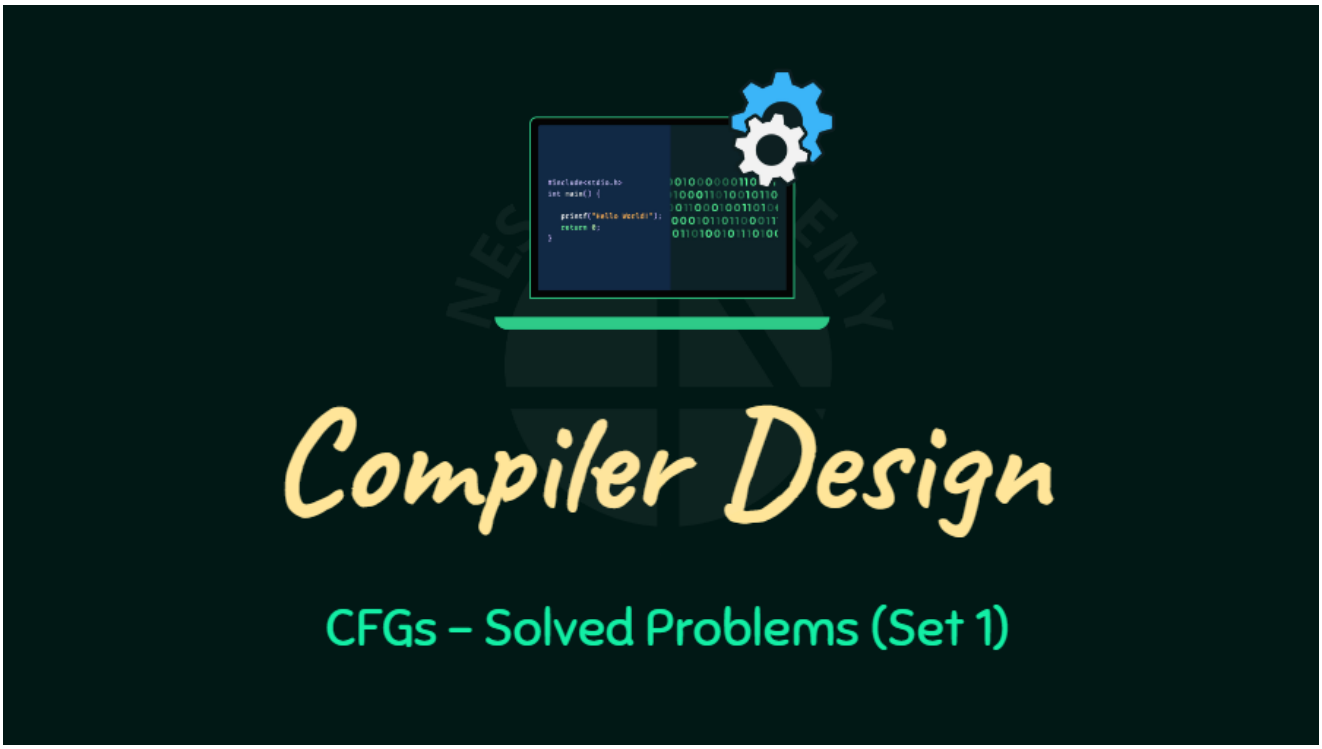
SLR(1)LALR(1)CLR(1)The Big PictureUnambiguous CFGsLR(0)LL(1)S → Aa I bAc I dc I bdaA → d



SLR(1)LALR(1)CLR(1)The Big PictureUnambiguous CFGsLR(0)LL(1)S → Aa I bAc I dc I bdaA → dS → Aa I bAc I Bc I bBaA → dB → d

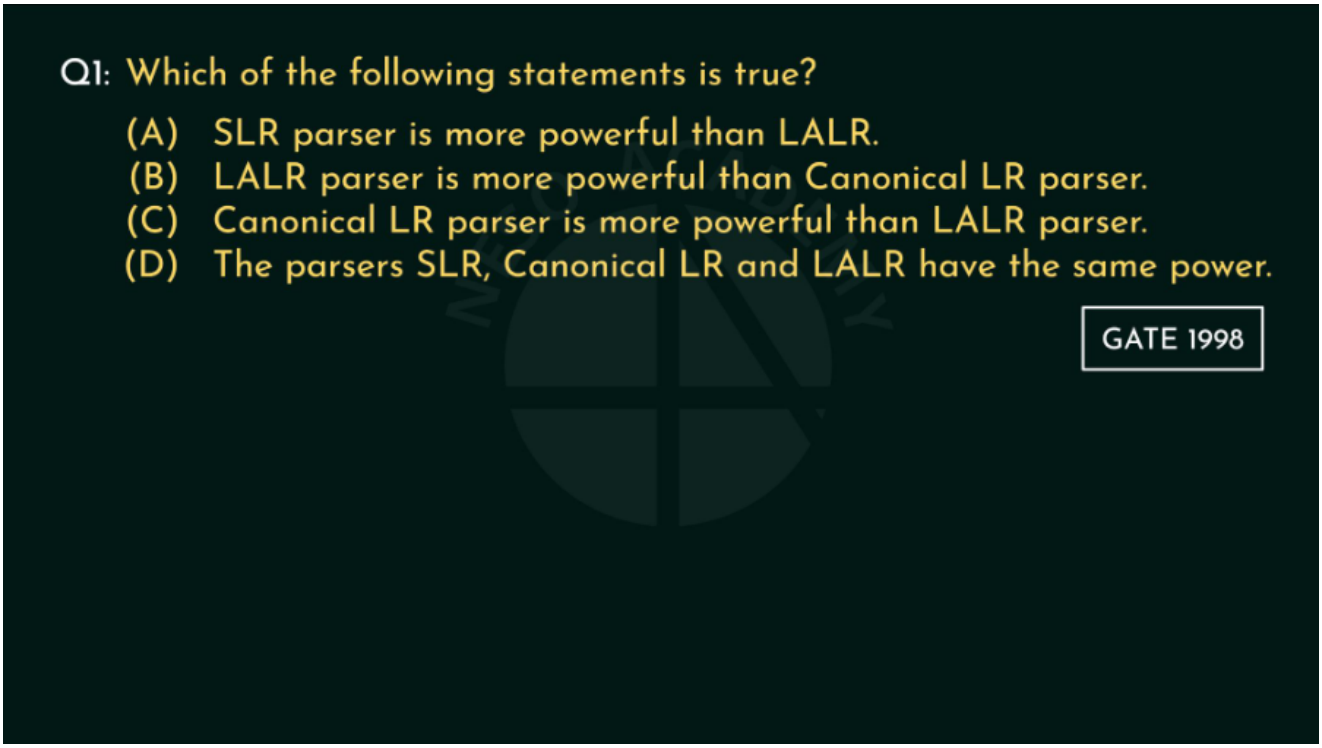


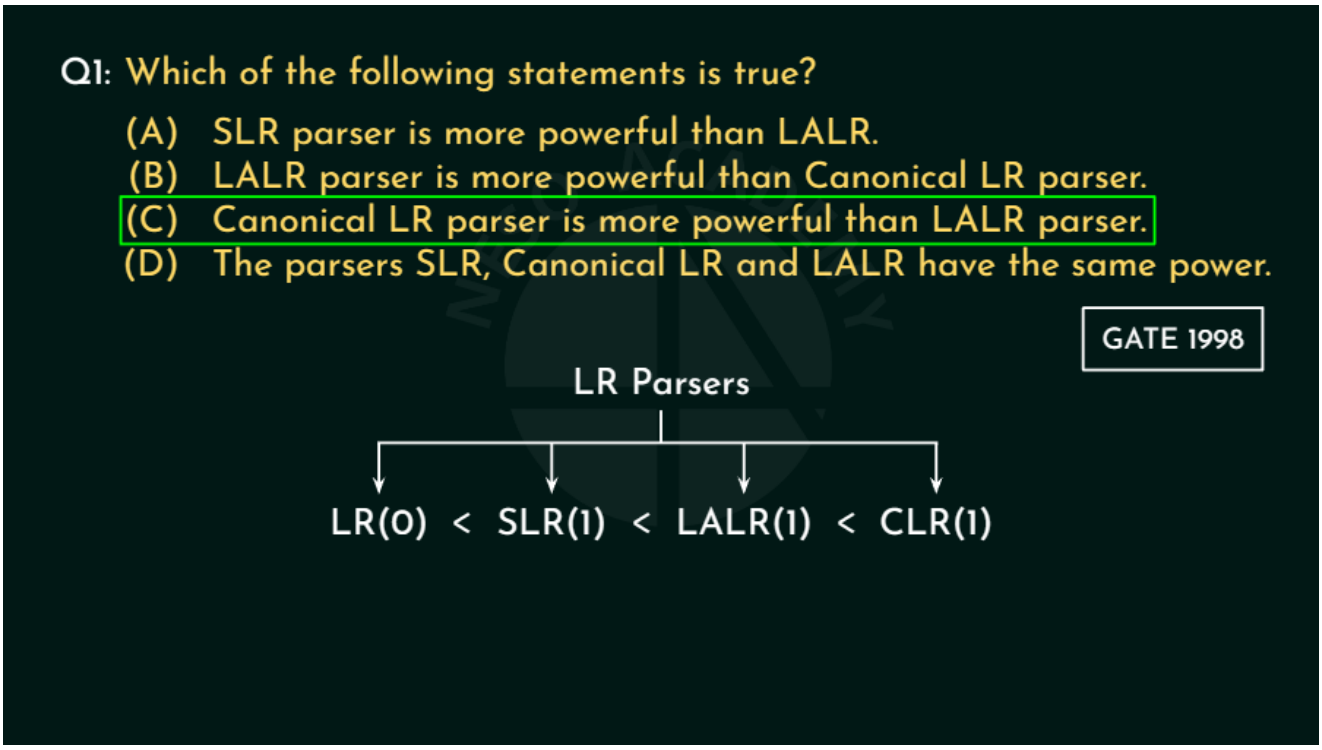
CLR(1)LR(k)The Big PictureUnambiguous CFGsAmbiguous CFGsOperatorPrecedenceParserLR(0)SLR(1)LALR(1)LL(1)LL(k)



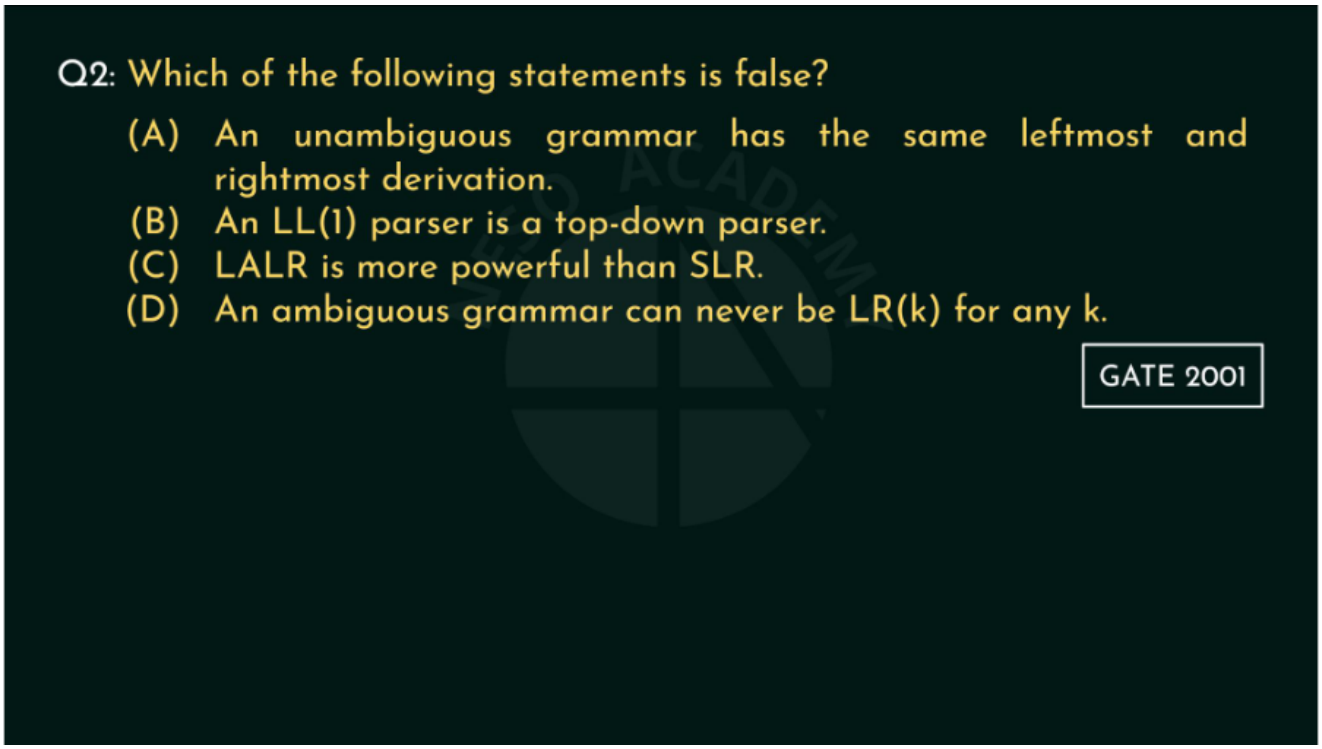
Compiler Design CFGs - Solved Problems (Set 1)

Outcome☆Four solved problems on CFGs.





Q1:Which of the following statements is true?(A)SLR parser is more powerful than LALR. (B)LALR parser is more powerful than Canonical LR parser.(C)Canonical LR parser is more powerful than LALR parser.(D)The parsers SLR, Canonical LR and LALR have the same power.GATE 1998 LR ParsersLR(0)CLR(1)LALR(1)SLR(1)<<<

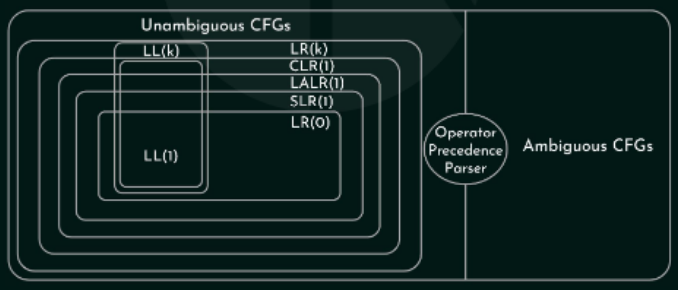


Q2: Which of the following statements is false?

(A) An unambiguous grammar has the same leftmost and rightmost derivation.
(B) An LL(1) parser is a top-down parser.
(C) LALR is more powerful than SLR.
(D) An ambiguous grammar can never be LR(k) for any k.

THE BIG PICTURE

GATE 2001



Q2:Which of the following statements is false?(A)An unambiguous grammar has the same leftmost and rightmost derivation.(B)An LL(1) parser is a top-down parser.(C)LALR is more powerful than SLR.(D)An ambiguous grammar can never be LR(k) for any k.GATE 2001

Q3: Considering the SLR(1) & LALR(1) parsing tables of a CFG, find out the false statement.

GATE 2003

(A) GOTO of both tables may be different.
(B) Shift entries are identical in both tables.
(C) Reduce entries in tables may be different.
(D) Error entries in tables may be different.

Q3: Considering the SLR(1) & LALR(1) parsing tables of a CFG, find out the false statement.

GATE 2003

(A) GOTO of both tables may be different.
(B) Shift entries are identical in both tables.
(C) Reduce entries in tables may be different.
(D) Error entries in tables may be different.

Comparison:

$S' \rightarrow S$
i. $S \rightarrow AA$
ii. $A \rightarrow aA$
iii. $A \rightarrow b$

States	Action			GO TO	
	a	b	$	A	S
0	S_3	S_4		2	1
1			Accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	r_{iii}	r_{iii}	r_{iii}		
5			r_i		
6	r_{ii}	r_{ii}	r_{ii}		

SLR(1) Parsing Table

States	Action			GO TO	
	a	b	$	A	S
0	S_{36}	S_{47}		2	1
1			Accept		
2	S_{36}	S_{47}		5	
36	S_{36}	S_{47}		89	
47	r_{iii}	r_{iii}	r_{iii}		
5			r_i		
89	r_{ii}	r_{ii}	r_{ii}		

LALR(1) Parsing Table

Q3:Considering the SLR(1) & LALR(1) parsing tables of a CFG, find out the false statement. (A)GOTO of both tables may be different.(B)Shift entries are identical in both tables. (C)Reduce entries in tables may be different. (D)Error entries in tables may be different.GATE 2003

Q4: Consider the grammar
$S \rightarrow (S) \mid a$
Let the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3, respectively. The following relationship holds good:

GATE 2005

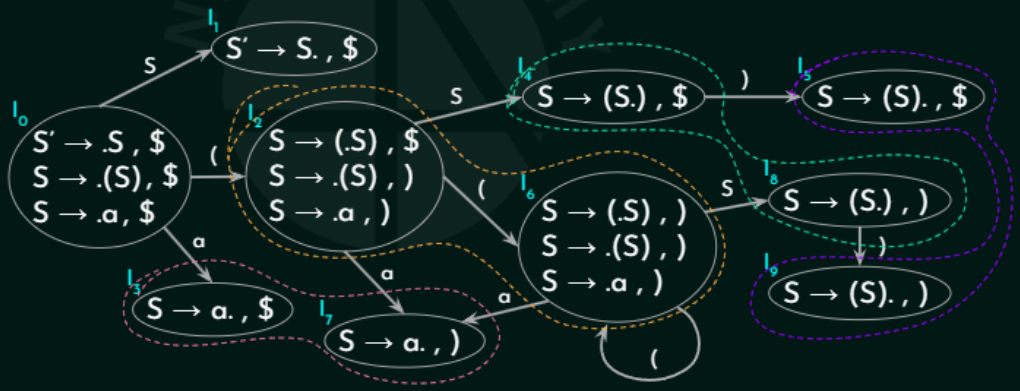
(A) $n1 < n2 < n3$
(B) $n1 = n3 < n2$
(C) $n1 = n2 = n3$
(D) $n1 \geqslant n2 \geqslant n3$

Q4: Consider the grammar

$S \rightarrow (S) \mid a$

Let the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3, respectively. The following relationship holds good:

Sol. $S' \rightarrow S$
$S \rightarrow (S)$
$S \rightarrow a$



I2S → (.S) , $S' → .S , $I0, $, $Q4:Consider the grammarS → (S) I aLet the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3, respectively. The following relationship holds good:S' → SS → (S) S → aS → .(S)S → .aS' → S. , $I1S,),)S → .(S)S → .a(S → a. , $I3aS → (S.) , $I4S → (S). , $I5S)I6S → (.S) ,),),)S → .(S)S → .a(aS → a. ,)I7S → (S.) ,)I8S → (S). ,)I9S)a(Sol.

Q4: Consider the grammar

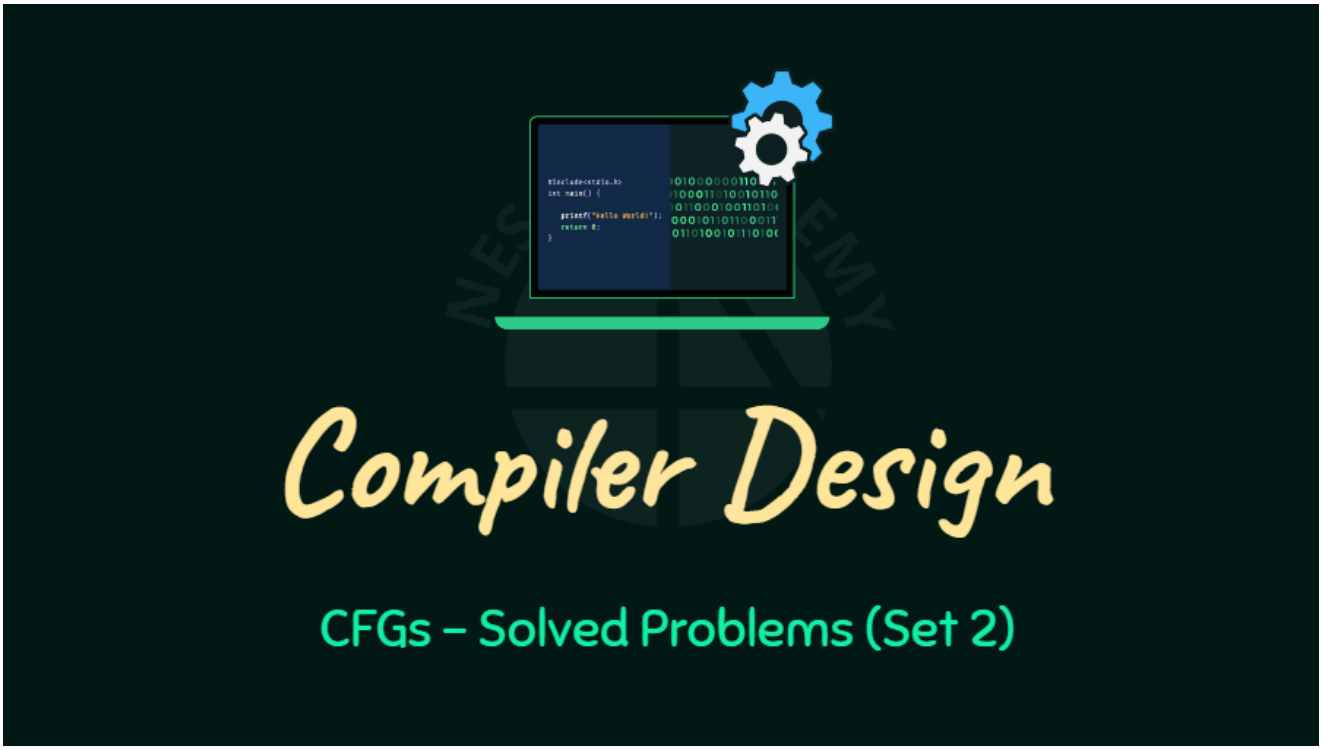
GATE 2005

$S \rightarrow (S) \mid a$

Let the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3, respectively. The following relationship holds good:

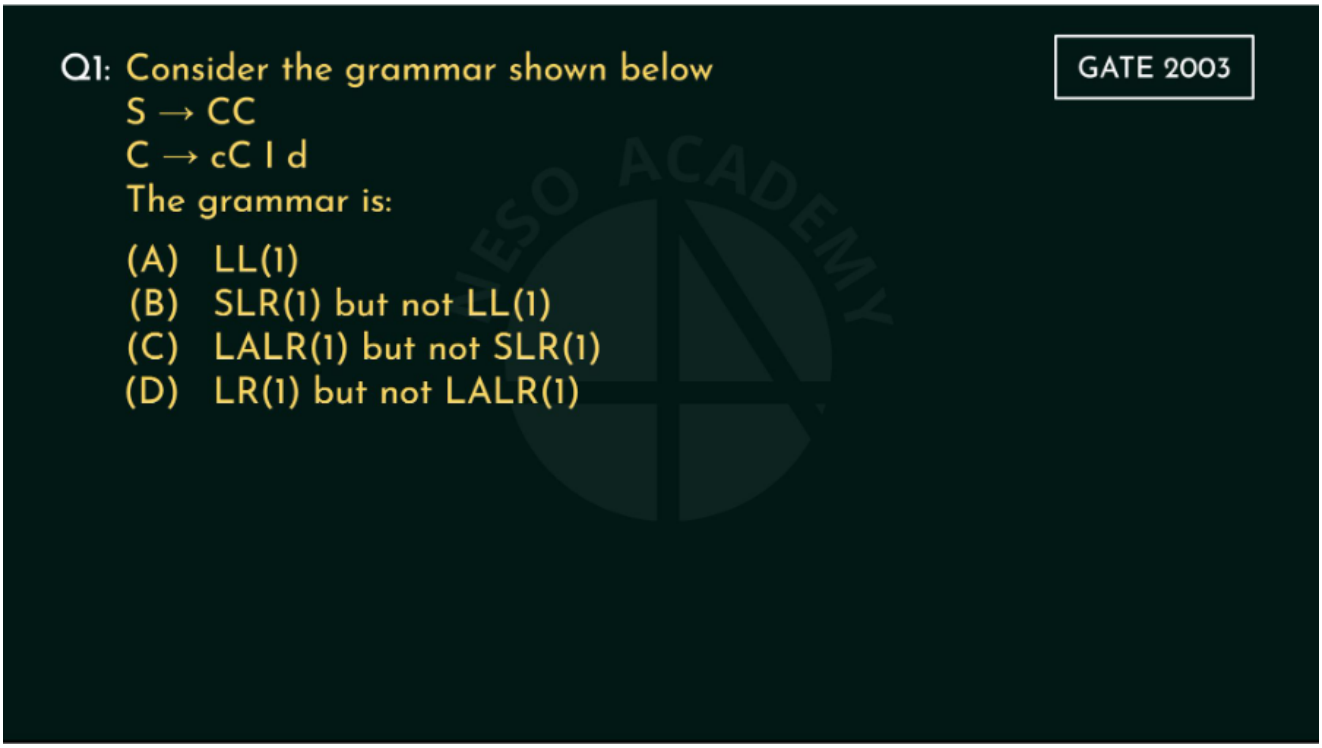
(A) $n1 < n2 < n3$
(B) $n1 = n3 < n2$
(C) $n1 = n2 = n3$
(D) $n1 \geqslant n2 \geqslant n3$

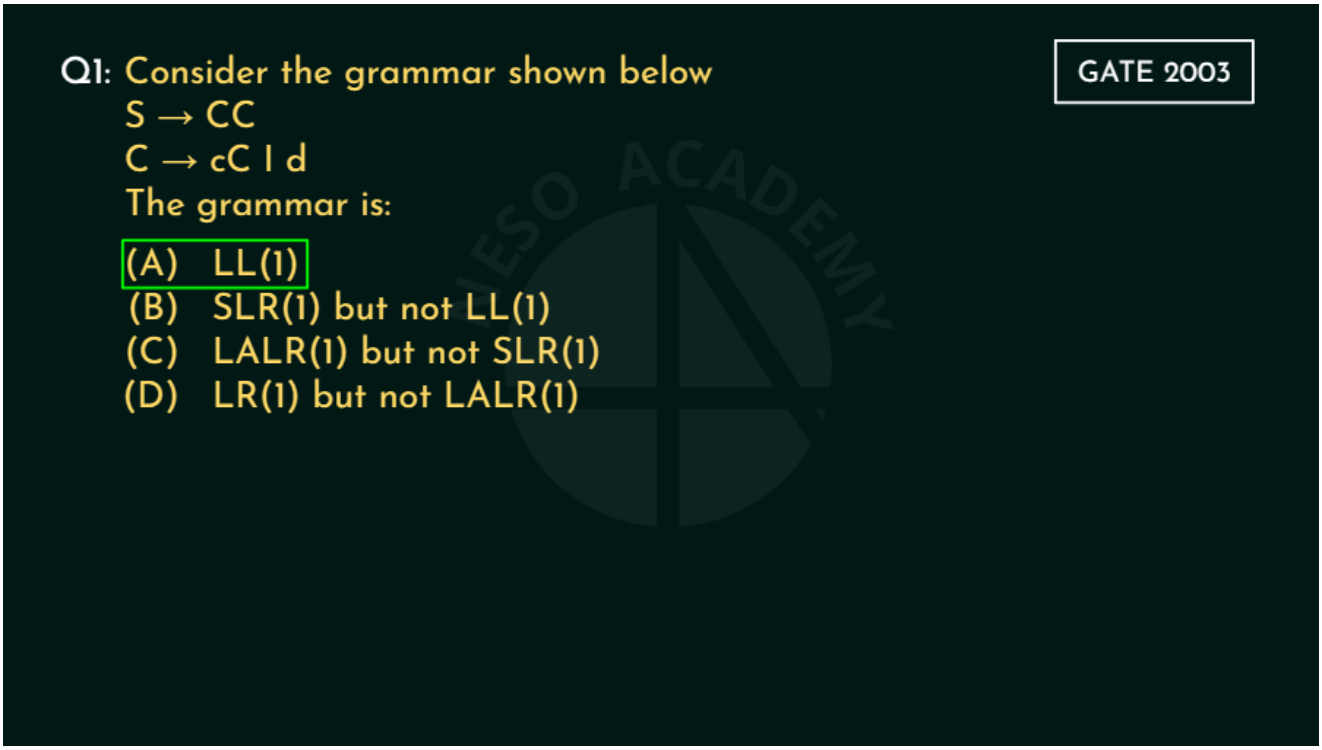
Q4:Consider the grammarS → (S) I aLet the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3, respectively. The following relationship holds good:(A)n1 < n2 < n3(B)n1 = n3 < n2(C)n1 = n2 = n3(D)n1 >= n2 >= n3GATE 2005



Compiler Design CFGs - Solved Problems (Set 2)

Outcome☆Four solved problems on CFGs.





Q1:Consider the grammar shown belowS → CC C → cC I dThe grammar is:(A)LL(1) (B)SLR(1) but not LL(1)(C)LALR(1) but not SLR(1)(D)LR(1) but not LALR(1)GATE 2003

Q1: Consider the grammar shown below

$S \rightarrow CC$

$C \rightarrow cC \mid d$

The grammar is:

Sol.

	FIRST	FOLLOW
$S \rightarrow CC$	{c, d}	{$}
$C \rightarrow cC \mid d$	{c, d}	{c, d, $}

	c	d	$
S	$S \rightarrow CC$	$S \rightarrow CC$	
C	$C \rightarrow cC$	$C \rightarrow d$	

LL(1) Parsing Table

LL(1) Parsing TablecSC$dQ1:Consider the grammar shown belowS → CC C → cC I dThe grammar is:, $}{c, d}{$}S → CCC → cC I dFOLLOWFIRST{c, d}{c, dS → CCC → cCS → CCC → dSol.

Q2: Consider the following grammar

GATE 2006

$S \rightarrow S * E \mid E$

$E \rightarrow F + E \mid F$

$F \rightarrow id$

Consider the following LR(0) items corresponding to the grammar:

(i) $S \rightarrow S * .E$

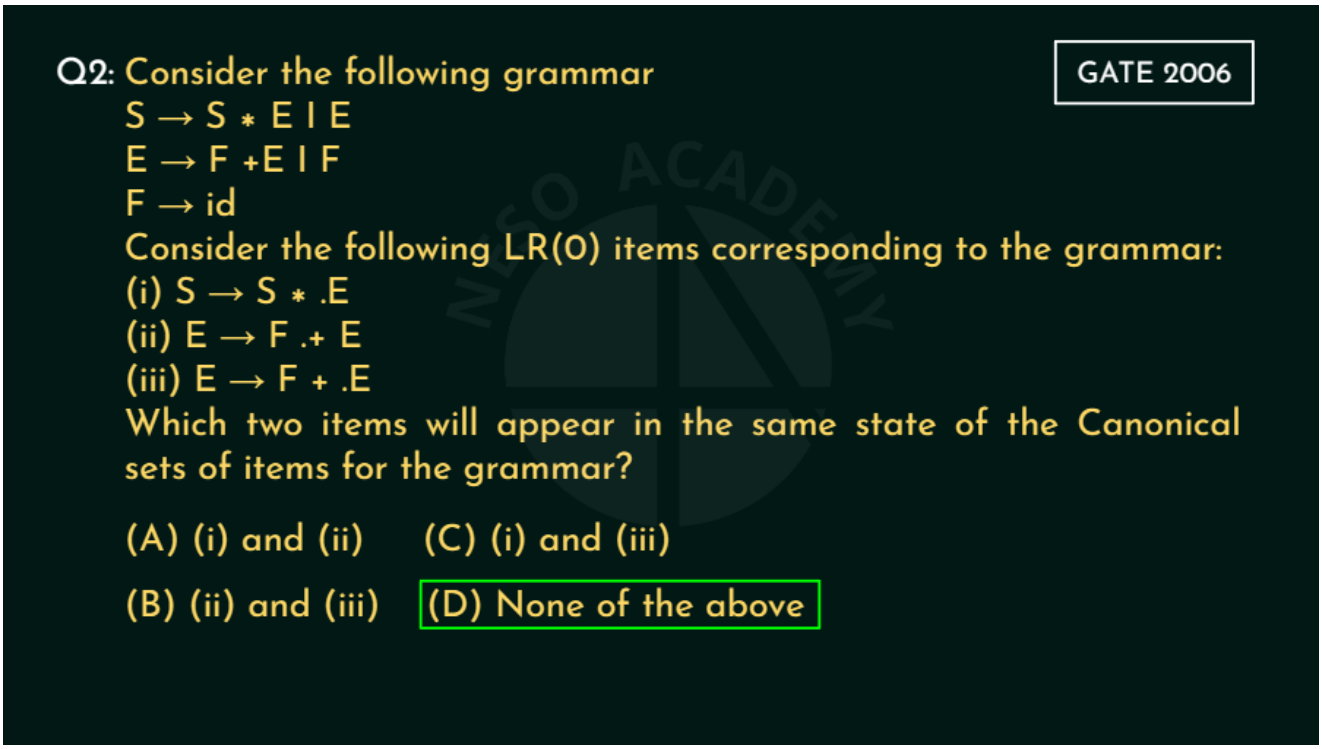
(ii) $E \rightarrow F .+ E$

(iii) $E \rightarrow F + .E$

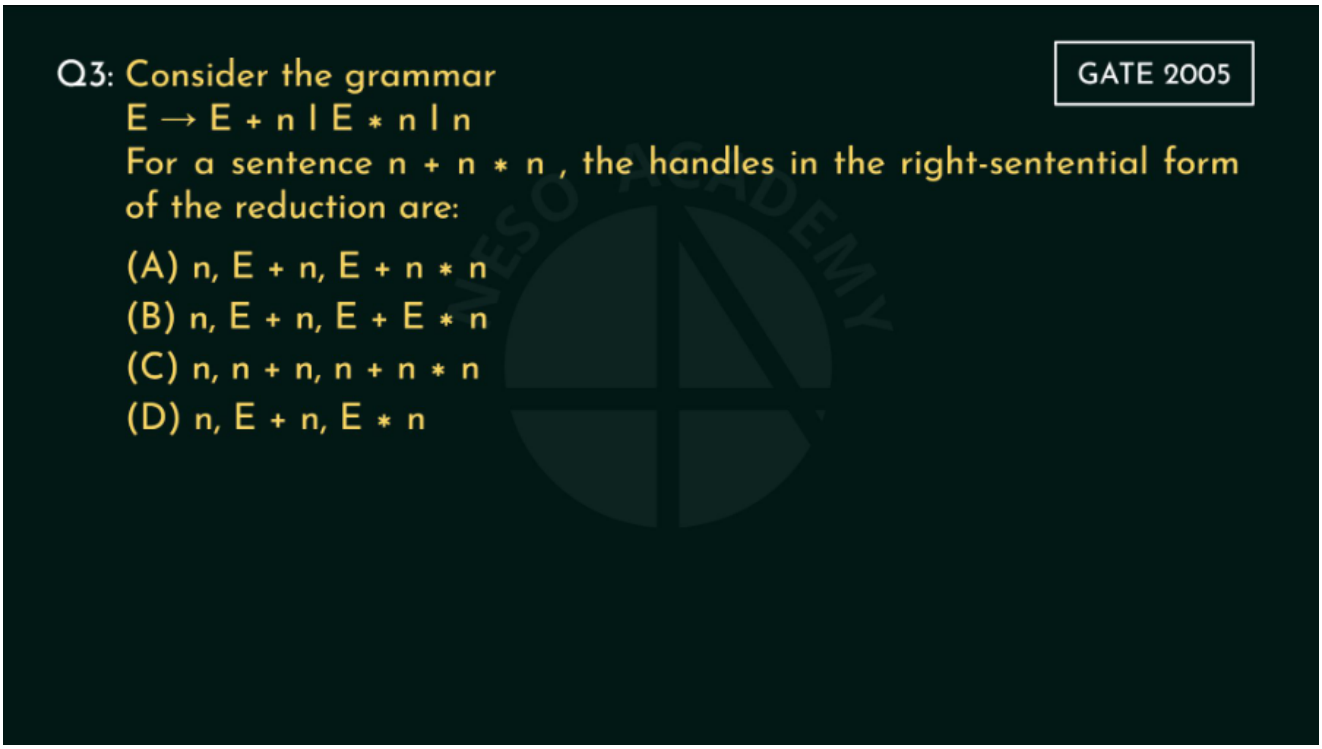
Which two items will appear in the same state of the Canonical sets of items for the grammar?

(A) (i) and (ii) (C) (i) and (iii)

(B) (ii) and (iii) (D) None of the above



Q2:Consider the following grammarS → S + E I EE → F +E I FF → idConsider the following LR(0) items corresponding to the grammar:(i) S → S + .E(ii) E → F .+ E(iii) E → F + .EWhich two items will appear in the same state of the Canonical sets of items for the grammar?(A) (i) and (ii) (C) (i) and (iii)(B) (ii) and (iii) (D) None of the aboveGATE 2006 **



Q3: Consider the grammar

GATE 2005

$E \rightarrow E + n \mid E * n \mid n$

For a sentence $n + n * n$, the handles in the right-sentential form of the reduction are:

(A) n, E + n, E + n * n

(B) n, E + n, E + E * n

(C) n, n + n, n + n * n

(D) n, E + n, E * n

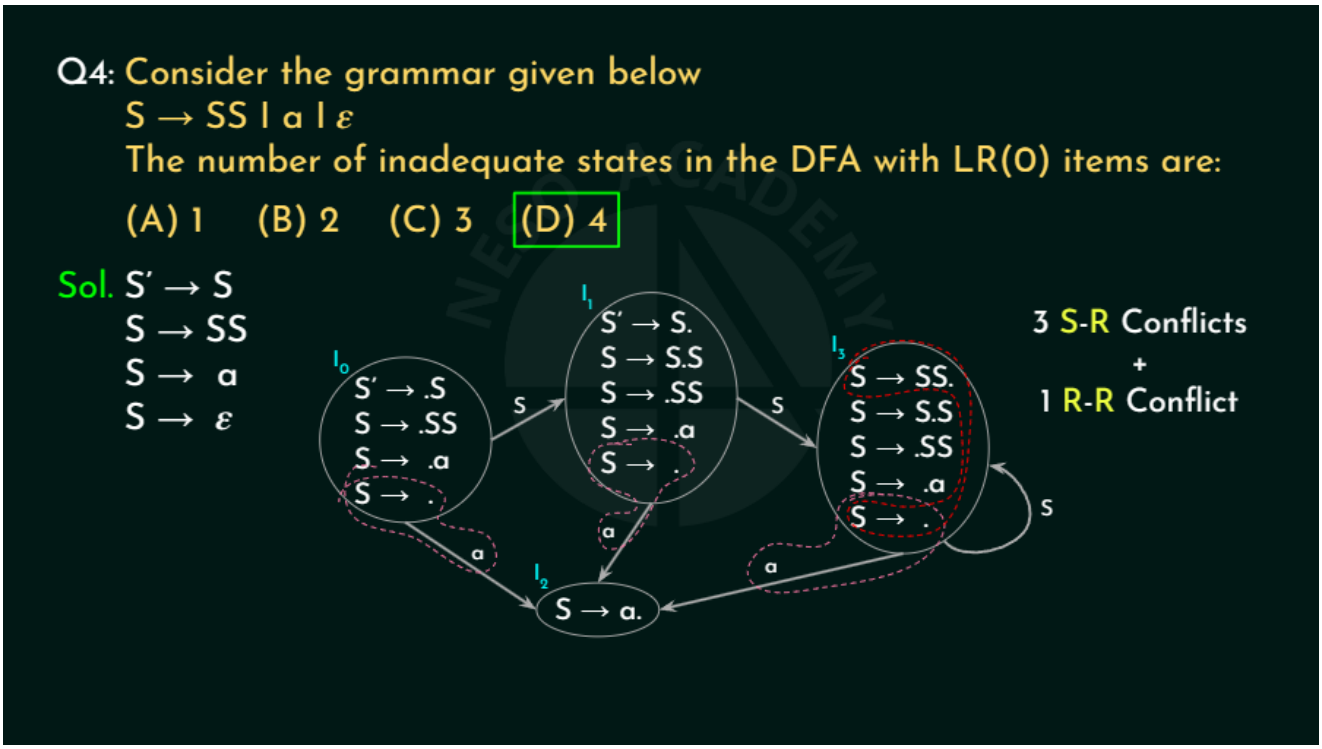
(A) n, E + n, E + n + n (B) n, E + n, E + E + n (C) n, n + n, n + n + n (D) n, E + n, E + n Q3:Consider the grammarE → E + n I E + n I nFor a sentence n + n + n , the handles in the right-sentential form of the reduction are:GATE 2005 ******E E n + n + n * E

Q4: Consider the grammar given below

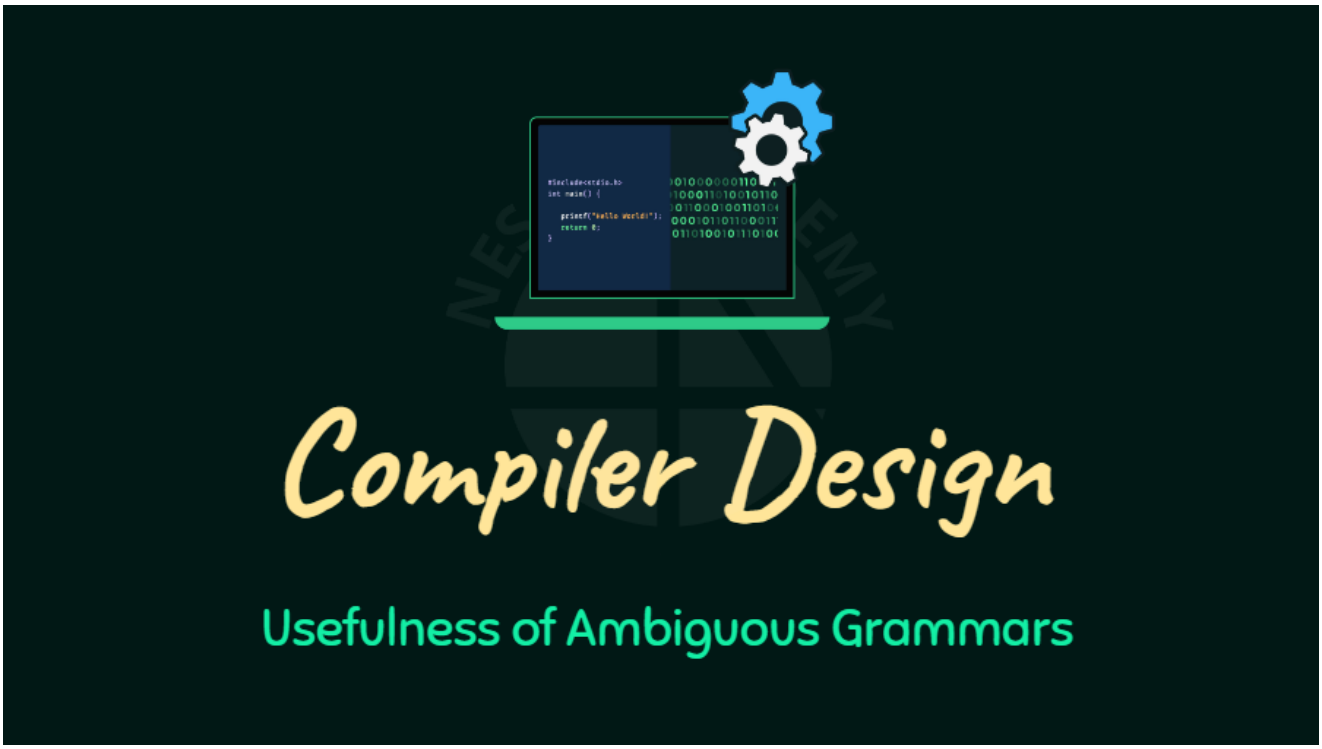
$S \rightarrow SS \mid a \mid \varepsilon$

The number of inadequate states in the DFA with LR(0) items are:

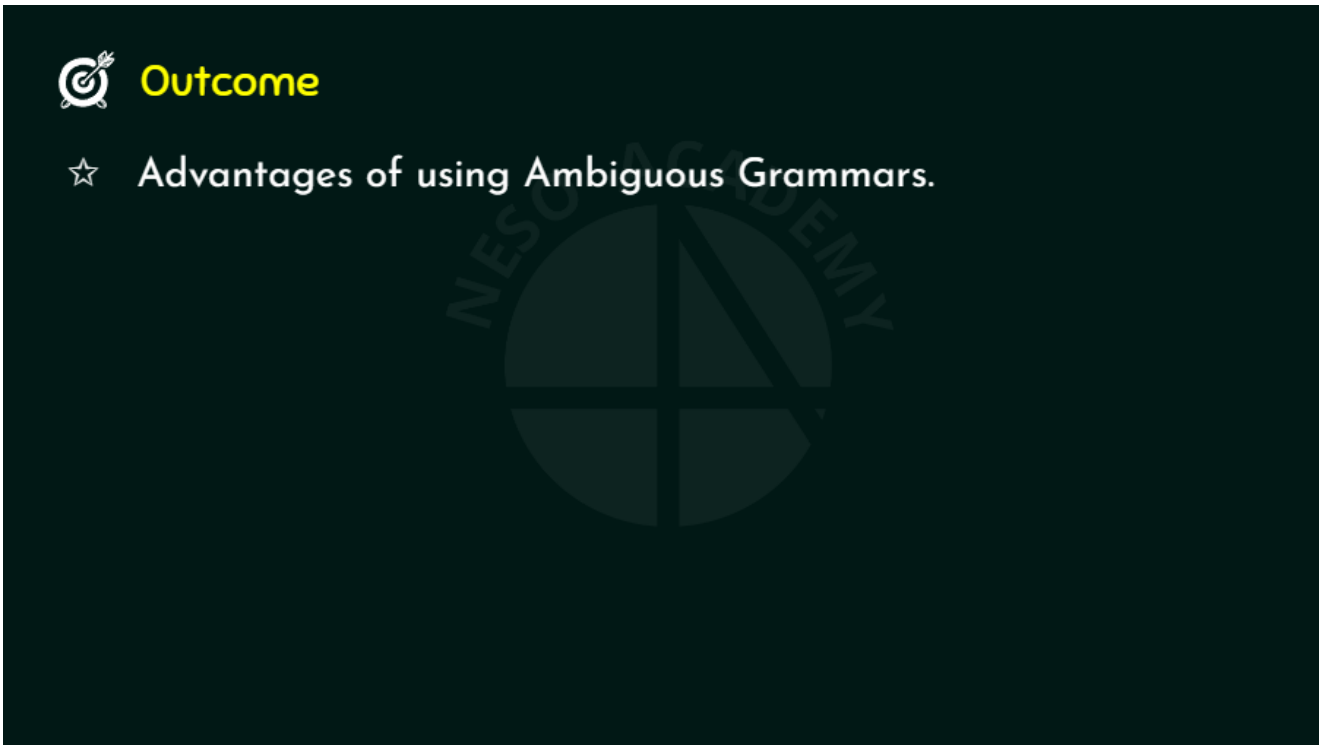
(A) 1 (B) 2 (C) 3 (D) 4



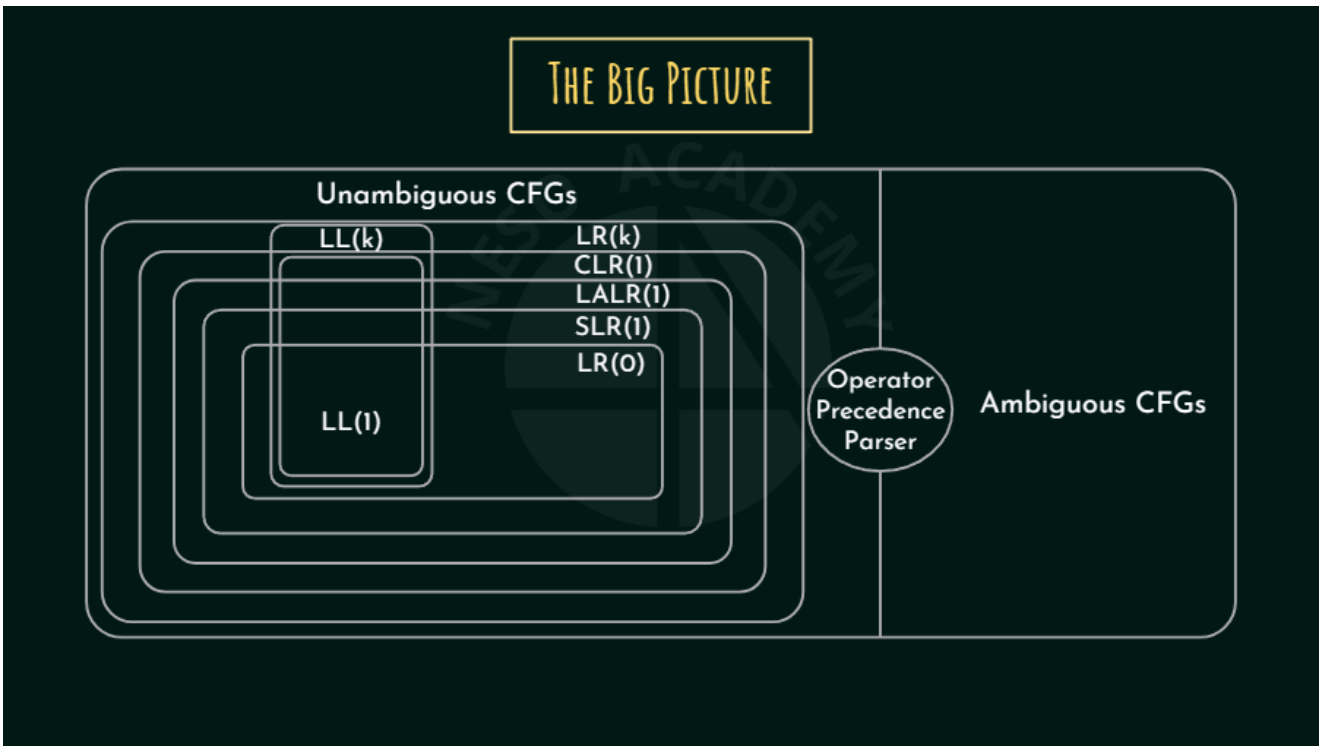
S' → .SI0S → SS.S → S.SI3(A) 1(B) 2(C) 3(D) 4Q4:Consider the grammar given belowS → SS I a I εThe number of inadequate states in the DFA with LR(0) items are:Sol.S' → SS → SS S → a S → εS → .SS S → .a S → .S' → S.S → S.SI1S → .SS S → .a S → .S → .SS S → .a S → .S → a.I2SaSaa3 S-R Conflicts+1 R-R ConflictS



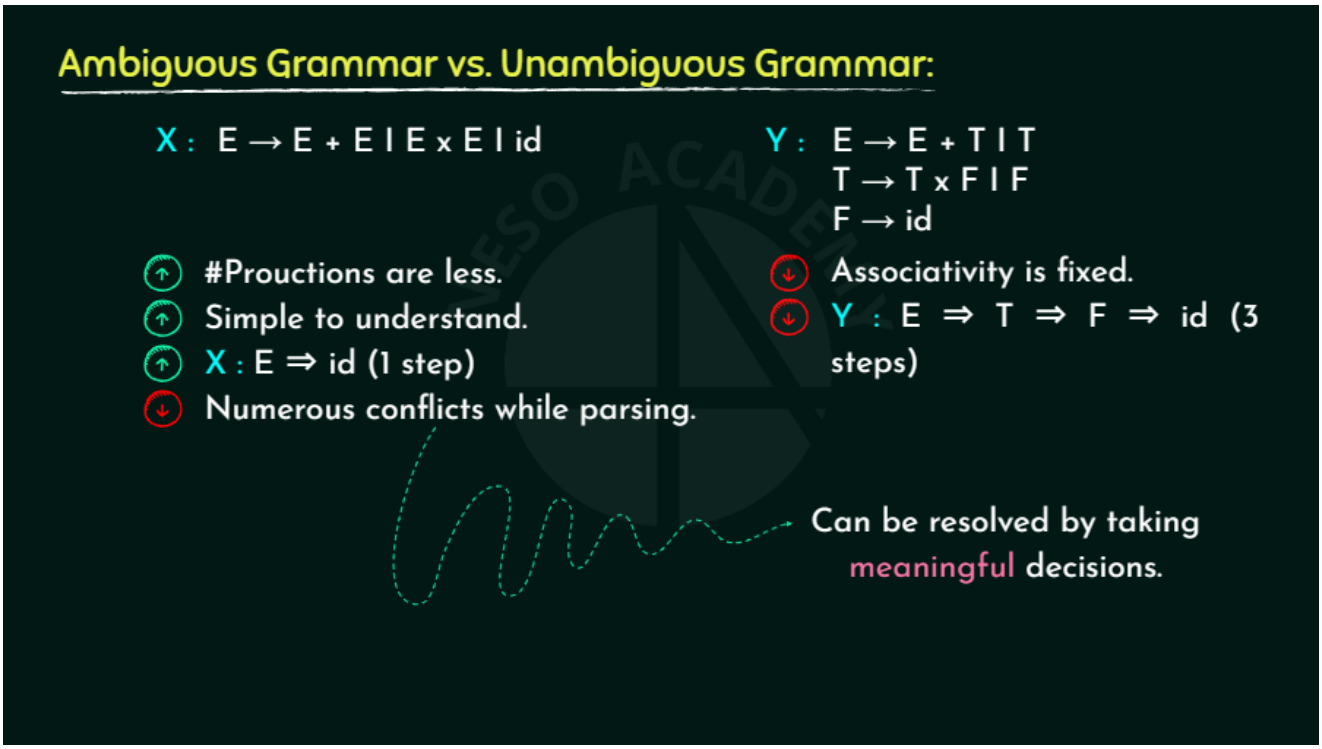
Compiler Design Usefulness of Ambiguous Grammars



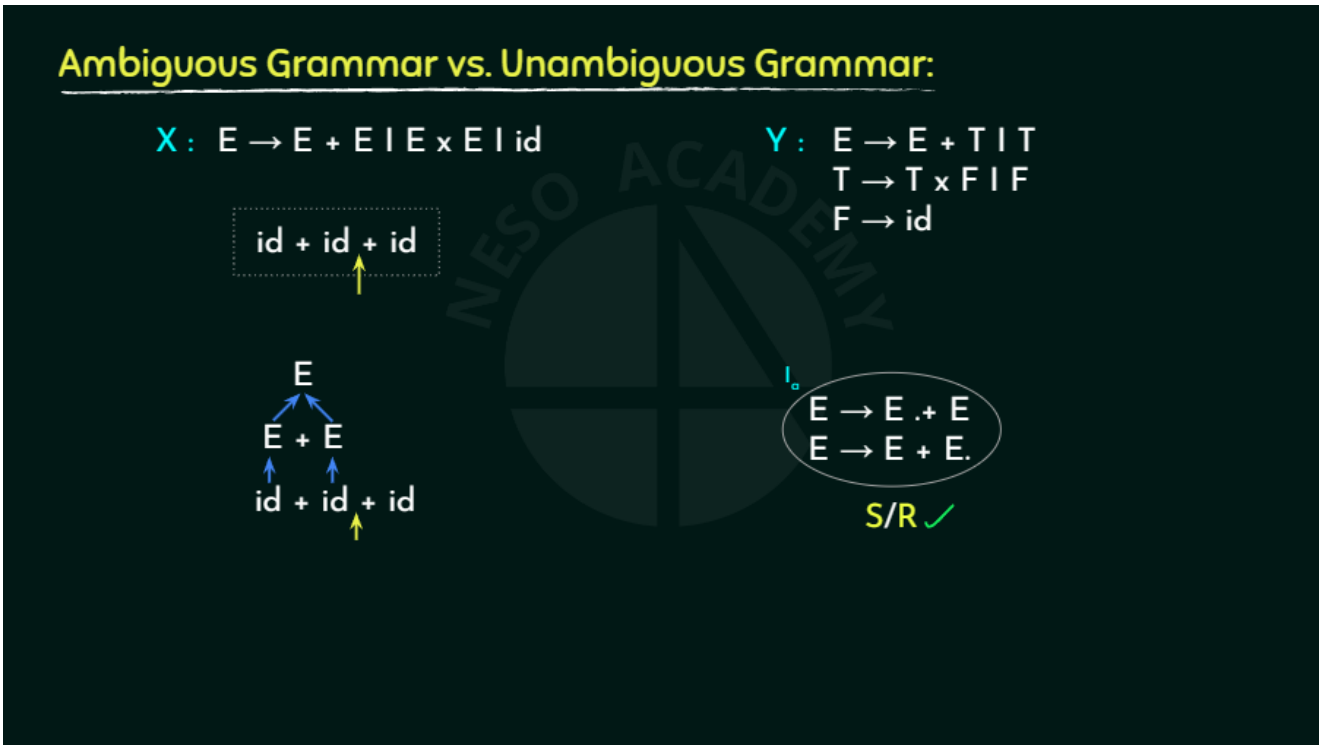
Outcome☆Advantages of using Ambiguous Grammars.



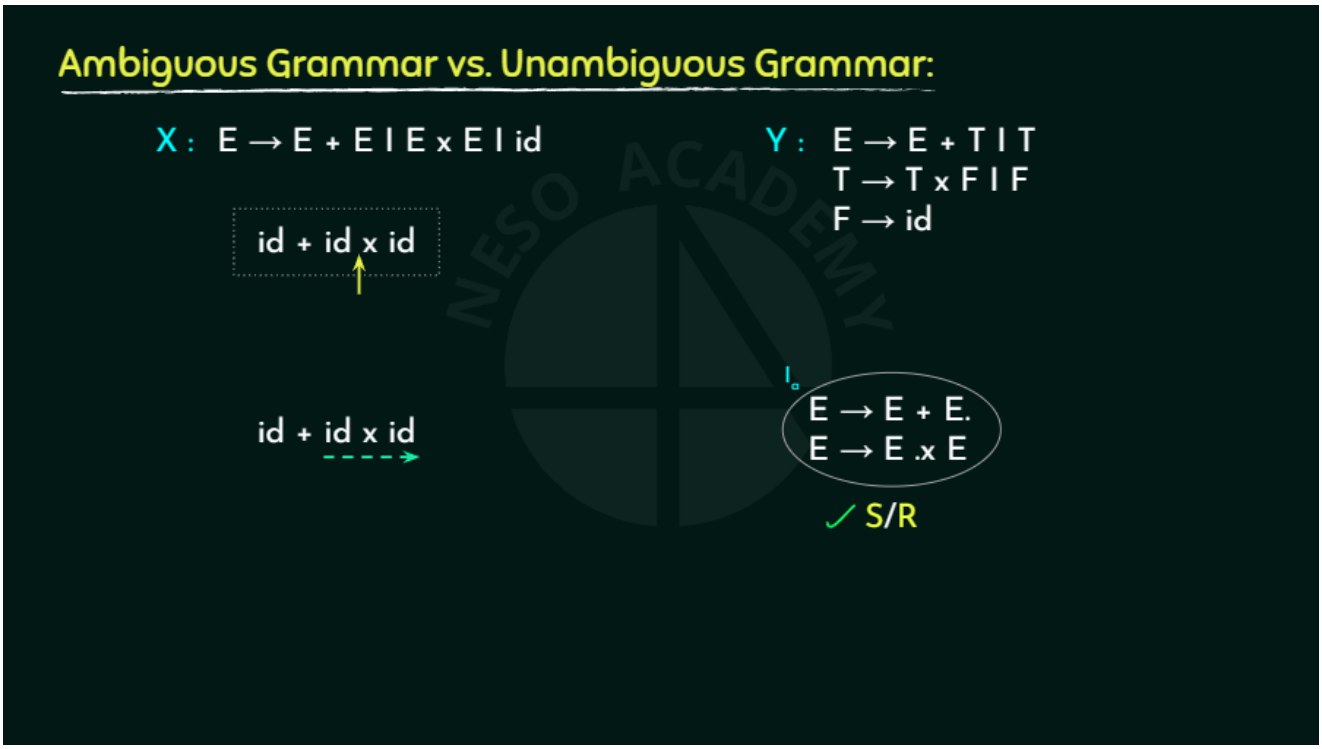
CLR(1)LR(k)The Big PictureUnambiguous CFGsAmbiguous CFGsOperatorPrecedenceParserLR(0)SLR(1)LALR(1)LL(1)LL(k)



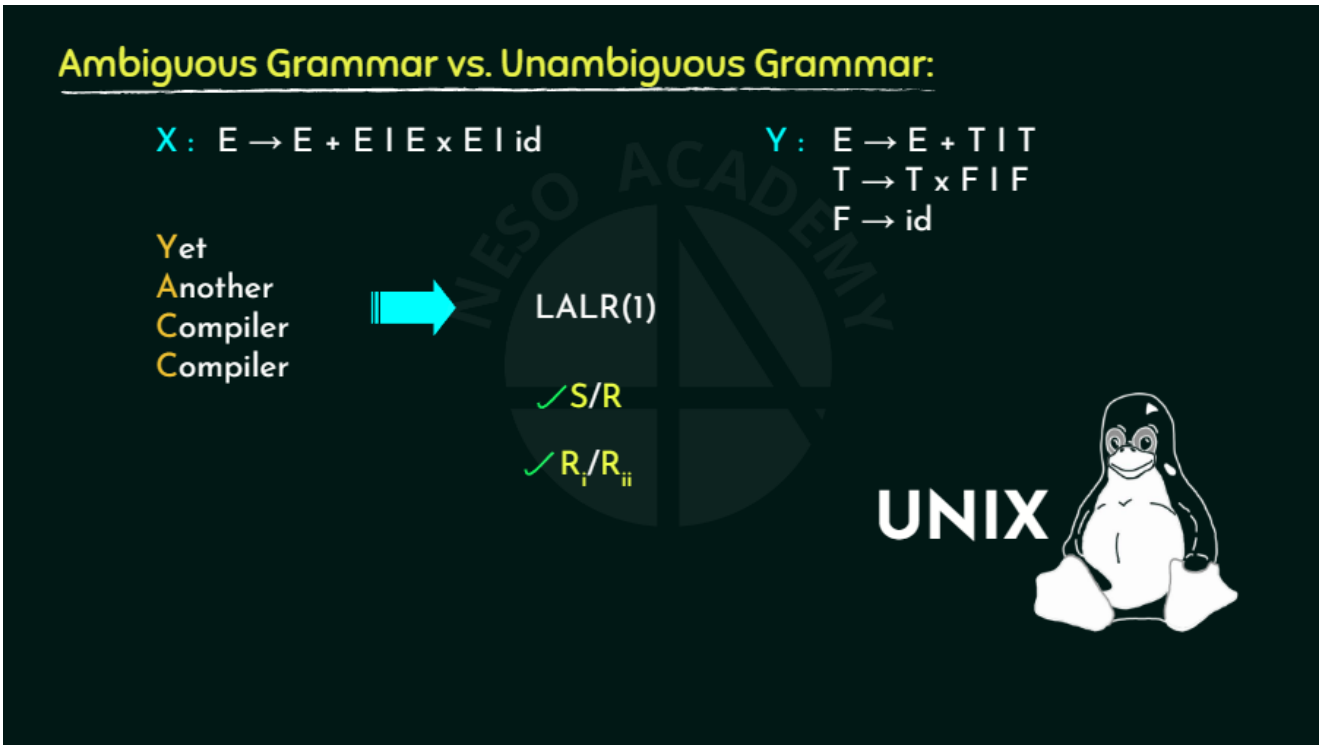
●Associativity is fixed.●Y : E ⇒ T ⇒ F ⇒ id (3 steps) ●#Prouctions are less.●Simple to understand.●X : E ⇒ id (1 step)Ambiguous Grammar vs. Unambiguous Grammar:E → E + T I T T → T x F I FF → idX :Y :E → E + E I E x E I idCan be resolved by taking meaningful decisions.Numerous conflicts while parsing.



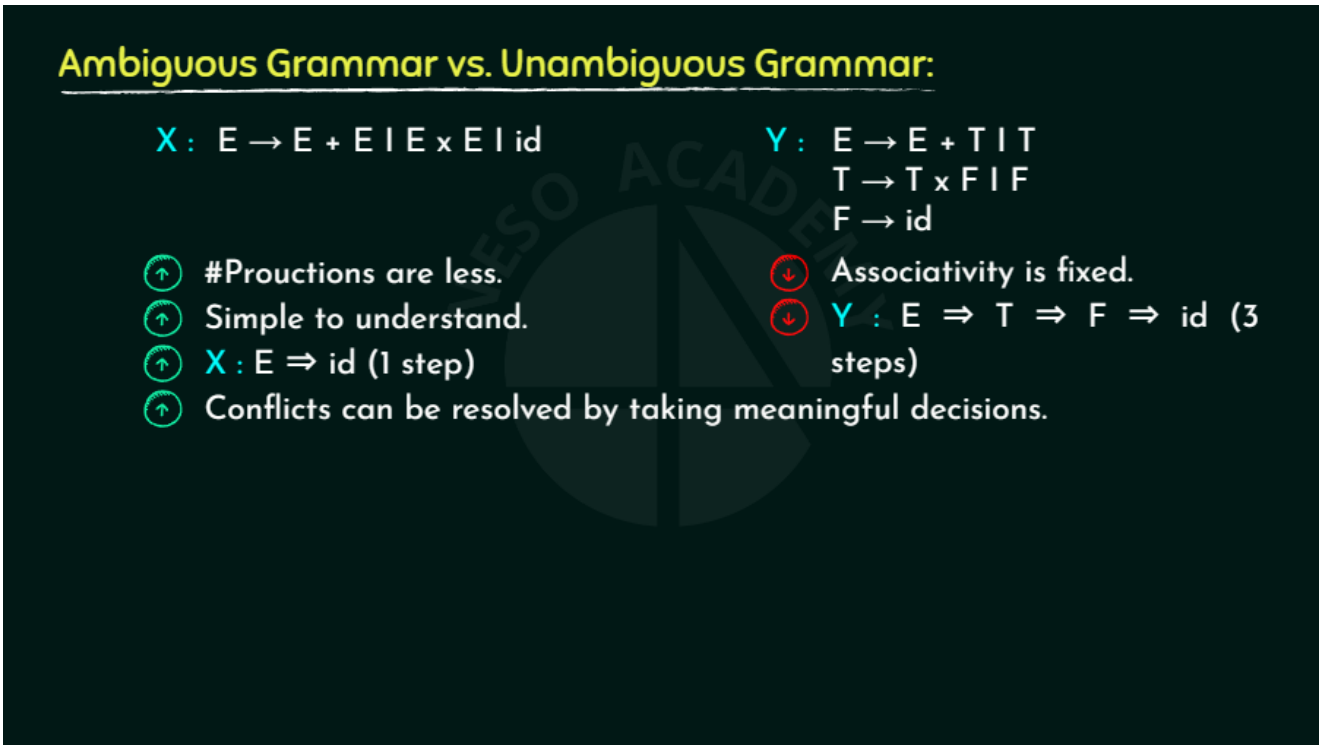
Ambiguous Grammar vs. Unambiguous Grammar:E → E + T I T T → T x F I FF → idX :Y :E → E + E I E x E I idid + id + idid + id + idE + E EE → E .+ E E → E + E. IaS/R



Ambiguous Grammar vs. Unambiguous Grammar:E → E + T I T T → T x F I FF → idX :Y :E → E + E I E x E I idid + id x idid + id x idE → E + E. E → E .x E I0S/R



Ambiguous Grammar vs. Unambiguous Grammar:E → E + T I T T → T x F I FF → idX :Y :E → E + E I E x E I idUNIXYACCetnotherompilerompilerLALR(1)S/RRi/Rii



●Associativity is fixed.●Y : E ⇒ T ⇒ F ⇒ id (3 steps) ●#Prouctions are less.●Simple to understand.●X : E ⇒ id (1 step)●Conflicts can be resolved by taking meaningful decisions. Ambiguous Grammar vs. Unambiguous Grammar:E → E + T I T T → T x F I FF → idX :Y :E → E + E I E x E I id