QUESTIONS AND SOLUTIONS

1. Consider the context free grammar

$S \rightarrow SS+ | SS* | a$

The language generated by the grammar is:

a) L={ Postfix expression consisting of digits , plus and multiplication sign}

b) L={ Prefix expression consisting of digits , plus and multiplication sign}

c) L={ Infix expression consisting of digits , plus and multiplication sign}

d) None of these

Ans :- Option a

Sol:- We can generate postfix expression using above grammar

For eg: The expression aa+a* can be generated which is a postfix expression**.**

SS*→SS+S*→aS+S*→ aa+S*→ aa+a*

2. Left factoring is the process of factoring out

a) Common prefixes that appear in two or more productions of the same non-terminal

b) Predictive parsing

c) Suffixes of the alternative of the grammar rule

d) None of these

Ans :- Option a

Sol :- Consider the non-deterministic grammar given below:

$$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \alpha\beta_3$$

We shall left factor it (NOTE: α is the common prefix)

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 | \beta_2 | \beta_3$$

The above grammar is deterministic grammar and left factored.

We can also tell that left factoring is the process of factoring the common prefixes of the alternatives of the grammar rule.

3. Which parser detects error faster?

a) LR(0) **c)** SLR(1)

b) LALR(1) **d)** None of these

Ans:- Option b

Sol :- The error detecting capability of LALR(1) parser is the highest among the given options (because of more blank entries in the LALR tables).

4. Which one of the following grammar generates the language which depicts right associative list of identifiers separated by commas

a) expr $\rightarrow$ expr, id
expr $\rightarrow$ id

b) expr $\rightarrow$ id, expr
expr $\rightarrow$ id

c) expr $\rightarrow$ id

d) None of these

Ans:- Option b

Sol :- expr $\rightarrow$ id, expr

expr $\rightarrow$ id

expr consists of one or more commas separated id' s

The comma is right associative in other words, the list of id's grows to the right

There shall be no comma at the end of the list of id's

5. For the given grammar what is the precedence & associativity of the operators

$$
\begin{aligned}
\mathbf{E} &\rightarrow \mathbf{E*F} \\
&\mathbf{|F+E} \\
&\mathbf{|F} \\
\mathbf{F} &\rightarrow \mathbf{F-F} \\
&\mathbf{|id}
\end{aligned}
$$

a) + and * is of the same precedence & *is the left associative while + is right associative
b) * has higher precedence than + and * is left associative while + is right associative
c) + has higher precedence than * and * and + are both left associative
d) * has higher precedence than + and * and + are both left associative

Ans:- Option a

Sol:- In the grammar since + and * are both in the same level so they have same precedence

'*' is defined as left recursive so it itself associative

'+' is defined as right recursive so it is right associative

6. The difference between LR(0) and SLR(1) is:

a) They differ in placement of both shift and reduce moves.
b) They differ in placement of shift moves.
c) They differ in placement of reduce moves.
d) Both are same.

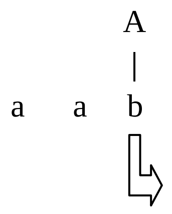
ANS:- Option c

SOL :-

For LR(0) :- Whenever any state has final item then in parsing table for the entire row we place the reduce moves. We don't care what is the next symbol.

For SLR(1) :-

Here we don't place reduce move in the entire row.
We don't blindly reduce whatever is the next symbol.
Reduce a symbol to the LHS of the production only if the next symbol is in the follow of that LHS of the production. Consider the example below for explanation.



Reduce b to A only if the next symbol i.e., a is in the follow of A.

7. The first and follow for the grammar below is :

S→aSbS|bSaS|∈

a) FIRST(S)= {a,b,∈}
FOLLOW(S)= {b,$}
b) FIRST(S)= {a,b}
FOLLOW(S)= {a,b,$}
c) FIRST(S)= {a,b,∈}
FOLLOW(S)= {a,b,$}
d) FIRST(S)= {a,b,∈}
FOLLOW(S)= {a,b}

Ans:- Option c

Sol :- <u>Method To Find FIRST(X) :</u>

FIRST(X) for all grammar symbol of X

(1) If X is a terminal ,then FIRST(X)={X}
(2) If X is a non-terminal and

$$X \rightarrow Y_1Y_2\ldots\ldots\ldots Y_k$$

is a production for some k ≥1, then place 'a' in FIRST(X) if for some i, 'a' is in FIRST(Y_i) and ∈ in all of the FIRST(Y_i)……FIRST(Y_{i-1})

If ∈ is in FIRST(Y_j) for all j=1,2,3…..k then add ∈ to the FIRST(X)

(3) If X → ∈ is a production ,then add ∈ to the FIRST(X)

<u>Method To Find FOLLOW(A) :-</u>

For all non-terminals A, apply the following rules until nothing can be added to any FOLLOW set:

1) Place \$ in FOLLOW(S), where S is the start symbol, and \$ is the input end maker.
2) If there is a production A → αBβ then everything in FIRST(β) except ∈ is in FOLLOW(β)
3) If there is a production A → αB, or a production A → αBβ, where FIRST(β) contains ∈,then everything in FOLLOW(A) is in FOLLOW(B)

   By following the above steps we get :

   FIRST(S)= {a,b,∈}

   FOLLOW(S)={a,b,\$}

8. Consider the following grammar G

G: S → EF

E → a|∈

F → abF|ac

Which of the following is true about the grammar G?

(1) G is a LL(1) grammar

(2) G is a regular grammar

a) 1 only **b)** 2 only **c)** 1 and 2 both **d)** None of these

Ans :- Option d

Sol :- <u>Checking for LL(1)</u>

Find the FIRST:

S → EF	{a}
E → a|∈	{a,∈}
F → abF|ac	{a}∩{a}

We see there is a common term in the FIRST of the production F → abF|ac

Therefore the grammar is not LL(1)

<u>Checking if it is Regular or not:-</u>

The production rules for a regular grammar must be of the form :

S → aA,	
S → Aa and	where A → non-terminal
S → a	a → terminal

Since the given grammar contains production like S → EF , both are non terminal

Therefore, it does not conform to the definition of regular grammar.

(We have discussed about Regular Grammar in TOC)

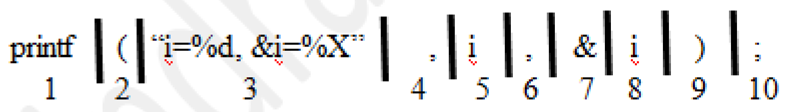
9. The number of tokens in c statement is :

printf("i=%d, &i=%X", i, &i);

a) 3 **b)** 26 **c)** 10 **d)** 21

Ans :- Option c

Sol :- The token can be given as follows :



The number of tokens are :10

10. Suppose that we want to describe Java style class declarations like these using a grammar

class car extends vehicle
public class JavaIsCrazy implements Factory, Builder, Listener
public final class Zergling extends Unit implements Rush

Grammar for this is

1) S→ C
2) C→PF class identifier XY
3) P→ public
4) P →∈

5) F →final
6) X →extends identifier
7) X →∈
8) Y →implements I
9) Y →∈
10) I →identifier J
11) J→ , I (note comma before I)
12) J →∈

For reference the terminals in this grammar are: public, final, class, identifier, extends, implements

The LL(1)parsing table is given below:

	Public	final	Class	extends	implements	identifiers	,	$
S	1	1	E1					
C	2	2	2					
P	3	E2	4					
F		E3	6					
X				7	8			8
Y					E4			10
I						11		
J							E5	13

E1,E2,E3,E4,E5 shall be filled with

a) 2,6,3,9,12 **b)** 1,4,5,9,12

c) 1,4,5,11,12 **d)** 1,4,6,11,12

Ans:-Option b

Sol :- Finding the FIRST:

FIRST(S)= {public,final,class}

FIRST(C)={public,final,class}

FIRST(P)={∈,public}

FIRST(F)={∈,final}

FIRST(X)={∈,extends}

FIRST(Y)={∈,implements}

FIRST(I)={identifier}

FIRST(J)={∈,”,”} (PS :- Here to show comma belongs to FIRST(J) we have used double quotes)

Finding FOLLOW :

FOLLOW(S)= {$}

FOLLOW(C)={$ }

FOLLOW(P)={final,class}

FOLLOW(F)={class }

FOLLOW(X)={implements,$}

FOLLOW(Y)={$}

FOLLOW(I)={$}

FOLLOW(J)={$}

	Public	final	Class	extends	implements	identifiers	,	$
S	1	1	1					
C	2	2	2					
P	3	4	4					
F		5	6					
X				7	8			8

Y					9			10
I						11		
J							12	13

PS: The descriptions given before the grammar in the question plays no role in finding the answer.

11. Consider the following grammar:

G: S→E|e
E→e

The grammar G is :

a) LL(1) **b)** SLR(1) **c)** LR(0) **d)** None of these

Ans :- Option d

Sol :-

<u>For LL(1) finding FIRST</u> :

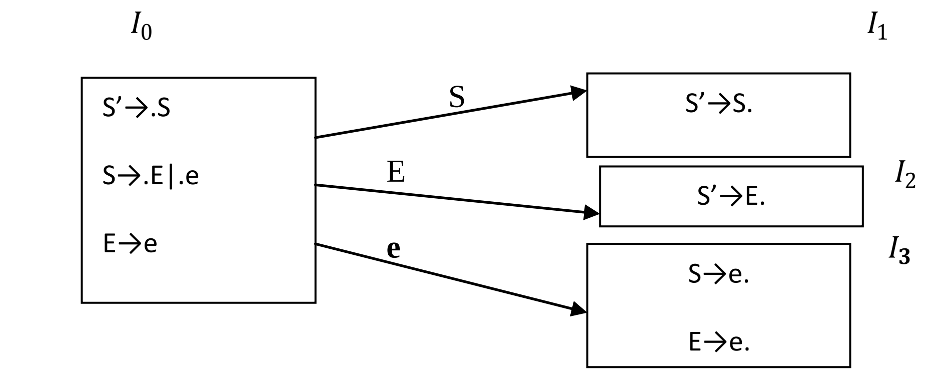
S→E|e	{e}∩{e}←intersection
E→e	{e}

Therefore, Not LL(1)

<u>Checking for LR(0) & SLR(1):</u>

For LR(0): augmented grammar is given below:

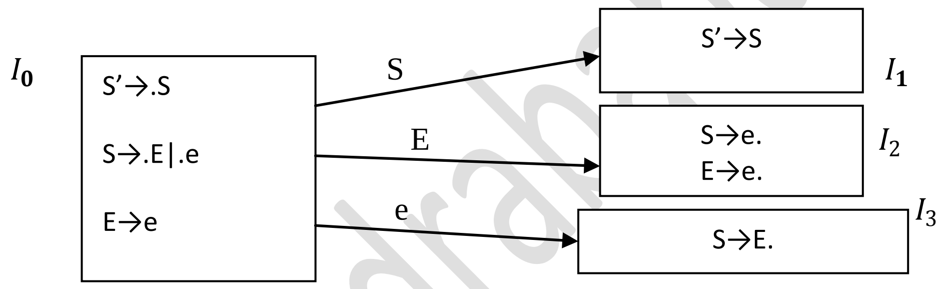
S'→S
S→E|e
E→e



I_3 contains more than one final item so R-R conflict

Therefore, not LL(0)

<u>**Checking for LR(0)**</u>



FOLLOW(S)= {$}, FOLLOW(A)= {$}

I_2 leads to R-R conflict

Therefore, not SLR(1)

The grammar is ambiguous

12. . Identify the correct sequence of parses arranged in decreasing order of their power:

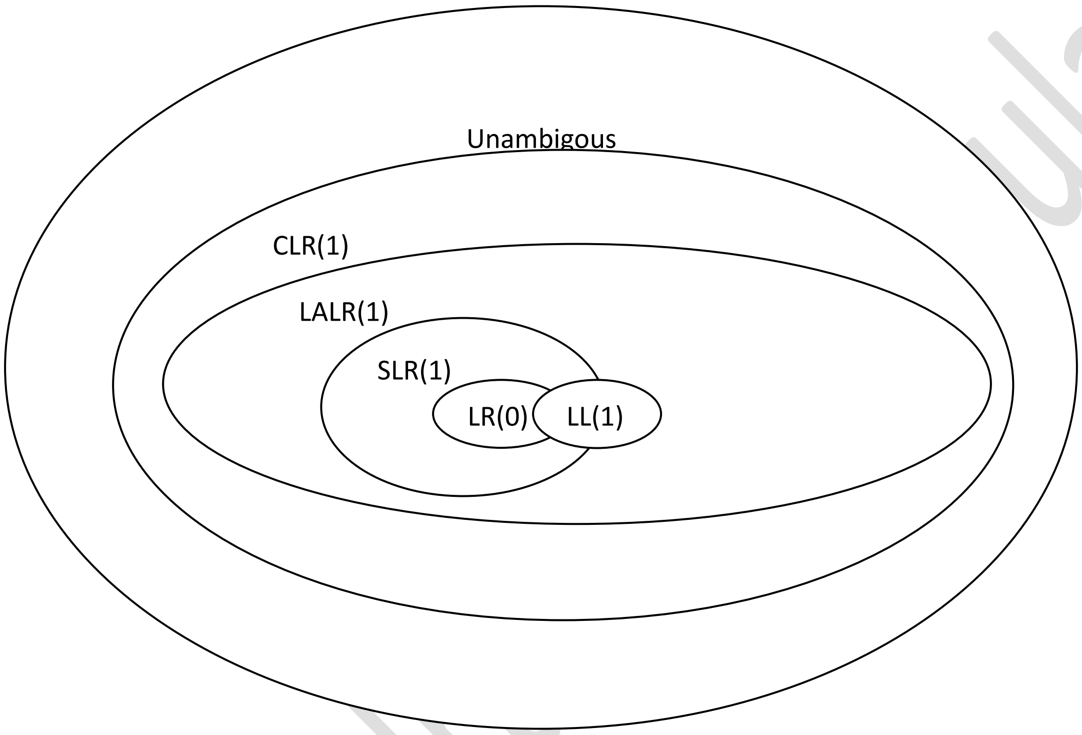
a. CLR(1), SLR(1), LR(0), LALR(1)

b. CLR(1), LALR(1), SLR(1), LR(0)

c. LALR(1), SLR(1), CLR(1), LR(0)
d. LR(0), SLR(1), LALR(1), CLR(1)

Ans :- Option b

Sol:- As we can see from the diagram given below that CLR(1) is the most powerful followed by LALR(1) then SLR(1) and the least powerful is LR(0)



13. Consider the following grammar

S→ABC……..1
S→X…………2
S→∈………….3

Which production of the above grammar violates the condition of operator grammar?

a) 1 only **b)** 1 and 3 **c)** 3 only **d)** 1, 2 and 3

Ans :- Option b

Sol :- Operator grammar should not have two consecutive non- terminals or ∈ on right hand side of productions.

14. Consider the following grammar

S→S(S)| ∈

Which one of the following is true

1. Grammar is ambiguous

2. Grammar is unambiguous

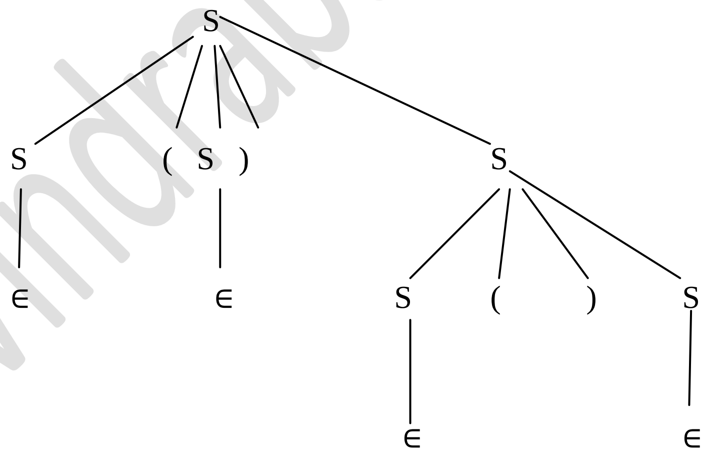
3. The grammar will generate all strings having balanced parenthesis

a) 1 and 3 **b)** 2 and 3 **c)** 1 only **d)** 2 only

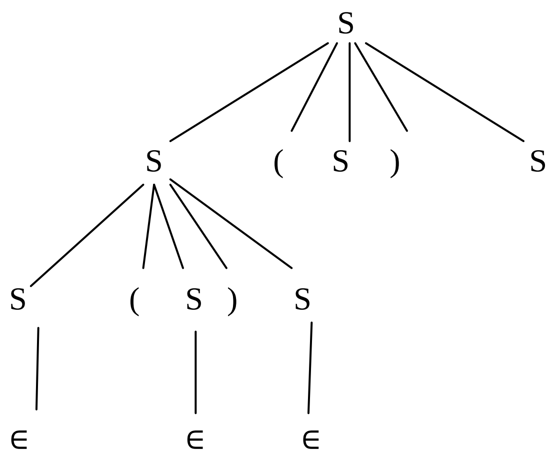
Ans :- Option a

Sol :- Since we get two parse trees so we can say it is ambiguous and also the yield of the parse trees gives us balanced parenthesis

Parse tree 1



Parse tree 2



Common Data Question 15 and 16

Consider the operator precedence relation

	id	+	*	$
id		▪>	▪>	▪>
+	<▪	▪>	▪>	▪>
*	<▪	<▪	▪>	▪>
$	<▪	<▪	<▪	

15. After evaluating id1,id2,id3 give the order in which operator will be evaluated :

a) +,* **b)** *,+ **c)** any operator can be evaluated **d)** none of the above

Ans :- Option a

Sol :- From the given table it is clear that + has higher precedence than * , so + will be evaluated before *

16. Suppose we are evaluating the string id_1+id_2+id_3. Give the order in which it shall be evaluated.

a) id_1+(id_2+id_3)

b) Cannot be said

c) (id_1+id_2)+id_3

d) None of these

ANS :- Option c

SOL :- Given that

+ •> +

Which shows the left side + should be given more precedence

Therefore, it is left associative and the order of evaluation should be:

(id_1+id_2)+id_3

Common Data Question 17 and 18

17. Below is a CFG for strings of balanced parenthesis

1) S→P
2) P→(P)P
3) P→∈

The SLR(1)parsing table is :

	Action			Goto	
	(	)	$	P	S
0	S_3	E1	r_3	2	1
1			E2		
2		r_2	r_2		
3	S_3	E3	r_3	4	
4		S_5			
5	E4	r_3	r_3	6	
6		r_2	r_2		

The follow sets of non terminals will be :

a) FOLLOW(S) = { $ }
FOLLOW(P) = {),$ }
b) FOLLOW(S) = { (, $ }
FOLLOW(P) = {(,), $ }
c) FOLLOW(S) = {), (**}**
FOLLOW(P) = { (,), $}
d) FOLLOW(S) = { $ }
FOLLOW(P) = { $ }

Ans :- Option a

18. The entries E1,E2,E3,E4 will be :

a) r_3, accept, S_5, r_3 **b)** S_3, r_4, r_5, S_3

c) r_3, accept, r_3, S_3 **d)** r_3, accept, r_4, r_3

Ans :- Option c

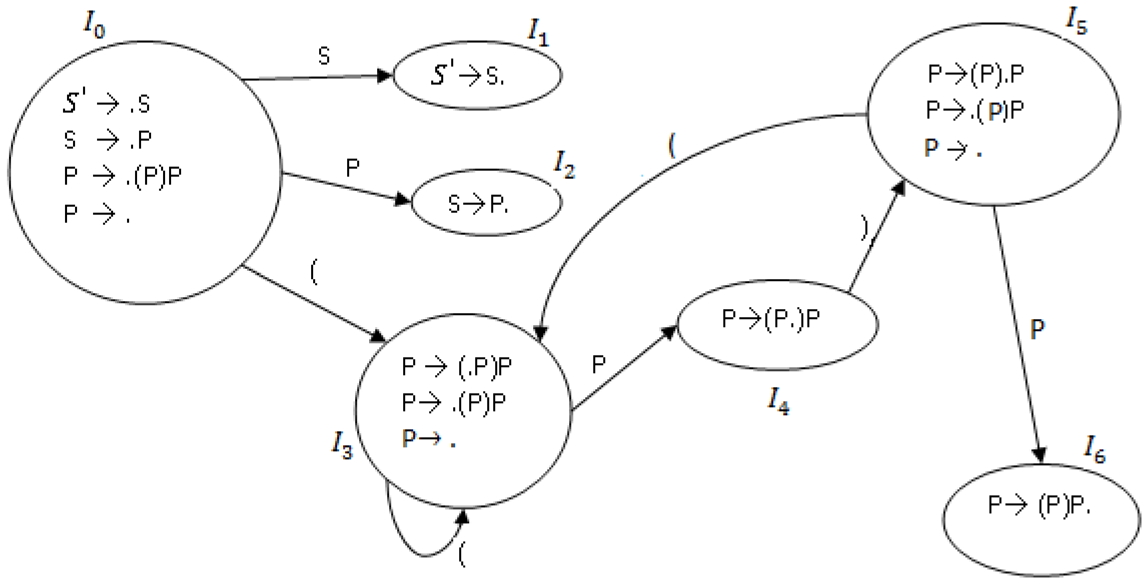
Sol :- We have the given grammar :

$S \rightarrow P$ ……………….. 1

$P \rightarrow (P)P$ ……………….. 2

$P \rightarrow \in$ ……………….. 3

Canonical collection of LR(0) items is given as follows:



The SLR(1) parsing table is:

	Action			Goto	
	(	)	$	P	S
0	S_3	r_3	r_3	2	1
1			accept		
2		r_2	r_2		
3	S_3	r_3	r_3	4	
4		S_5			
5	S_3	r_3	r_3	6	
6		r_2	r_2		

So from here we can see that:

E1 : r_3

E2: accept

E3: r_3

E4: S_3

19. Given the grammar

E→ E + T/T
T→ T * F/F
F→ G ↑ F/G
G→ id

Specify which operator has the highest precedence and whether it is left or right recursive?

a) + highest precedence and left recursive.
b) ↑ highest precedence and right recursive
c) ↑ lowest precedence and left recursive
d) * highest precedence and is right recursive

Ans : Option b

Sol : As among all the operators only ↑ is farthest away from start symbol so ↑ has the highest precedence.

The production $F \rightarrow G \uparrow F/G$ is of the form $A \rightarrow \alpha A/\beta$

Where we can say: F is equivalent to A and G↑ equivalent to α

So we say that it is right recursive.

20. Determine whether the following grammar is :

$$S \rightarrow AS|b$$
$$A \rightarrow SA|a$$

a) LL(1) **b)** LL(0) **c)** SLR(1) **d)** None of these

Ans :- Option d

Sol:- **<u>Checking for LL(1):</u>**

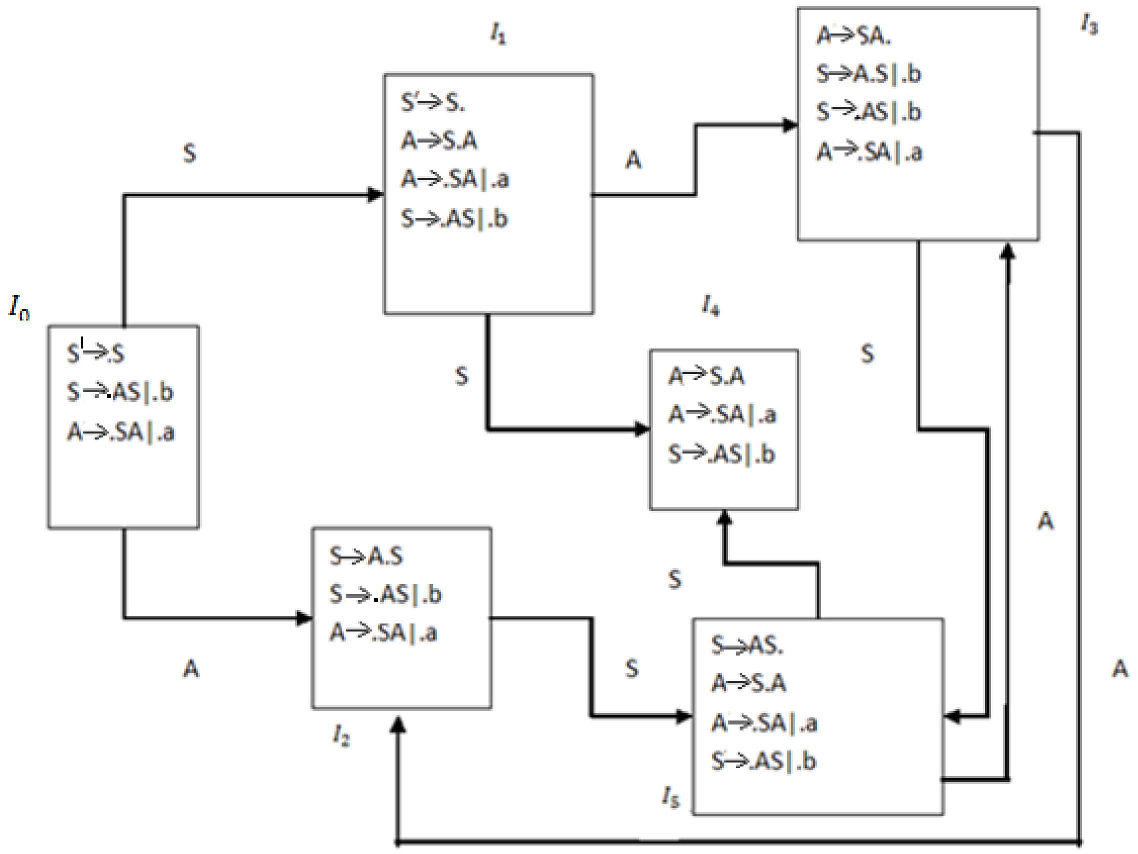
$S \rightarrow AS$………(1)

|b………..(2)
$A \rightarrow SA$………(3)
|a………..(4)

Prod (1) will be placed in { a, b } columns of S row

Prod (2) will be placed in { b } columns of the S row

Since in the S row of Parsing Table at b column there are two entries. So it is **NOT LL(1)**



The above collection of LR(0) items is partially completed.

<u>Checking for LR(0) items:</u>

- I_0, I_2, I_4 do not contain any reduce move so we need not worry about these states.
- I_1 contain S'→S.

  Which is actually S'→S.$ so we won't consider this augmented production.

- I_3 contains reduce move as well as shift move

A→SA**.** ← reduce move
S→A**.**S|**.**b
S→**.**AS|**.**b ← shift move

A→**.**SA|**.**a

This leads to SR conflict, hence the grammar is **NOT LR(0)**

Checking for SLR(1):

Again checking state I_3

A→SA. $\longrightarrow$ This reduce move will have to be placed in the follow of LHS i.e., follow of A which is {a,b}

S→A.S|.b $\longrightarrow$ This shift move will have to be placed in {b}

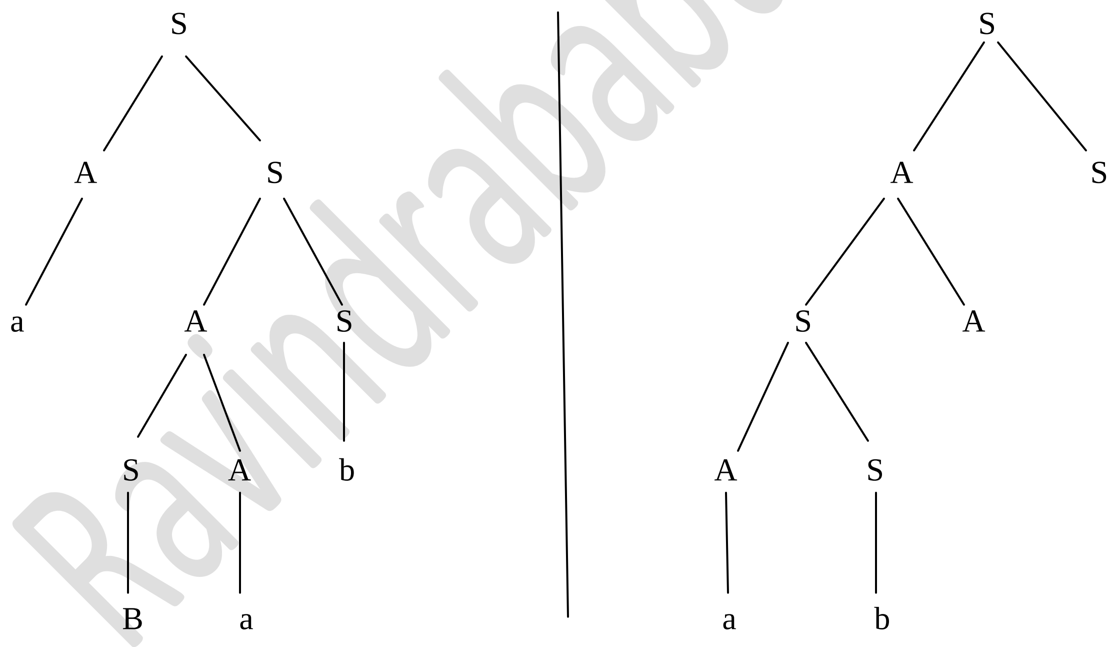
S→.AS|.b

A→.SA|.a

So there will be a conflict, hence it is **NOT SLR(1).**

No parsers can parse this grammar as it is an ambiguous grammar (grammar having more than one parse trees)

For eg : For the string abab the grammar gives 2 parse trees



S
A
S
a
A
S
S
A
b
B
a
S
A
S
S
A
A
S
a
b