* (2) SLR(1) Parser (or) Simple LR (1)

(i) Construction is same as LR(0)

(ii) First set and followset values are required to place the reduce actions.

*** FIRST SET

Set of terminals which appears on the first position on RHS of grammar.

1. First{abc} = {a}

2. $A \rightarrow ax$
   First(A) = {a}

3. $A \rightarrow Bx$ where B does not produce ϵ
   First(A) = First(B)

4. $A \rightarrow BX$
   First(A) = {First(B) − ϵ} ∪ {First(X)}

5. $A \rightarrow \epsilon$
   First(A) = ϵ

Q. (i) $S \rightarrow AB$
$A \rightarrow a/b$
$B \rightarrow c/d$

	First
S	a, b (First(A))
A	a, b
B	c, d

(ii) $S \to AB$
$A \to a$
$B \to b / \epsilon$

	First
S	a
A	a
B	b, ϵ

(iii) $S \to AB$
$A \to a / \epsilon$
$B \to d$

	First
S	a, d
A	a, ϵ
B	d

(iv) $S \to AB$
$A \to a / \epsilon$
$B \to b / \epsilon$

	First
S	a, b, ϵ
A	a, ϵ
B	b, ϵ

(v) $S \to AaAb / BbBa$
$A \to \epsilon$
$B \to \epsilon$

	First
S	a, b
A	ϵ
B	ϵ

(vi) $S \to ACB / Cb / Ba$
$A \to da / BC$
$B \to g / \epsilon$
$C \to h / \epsilon$

	First
S	d, g, h, b, a, ϵ
A	d, g, h, ϵ
B	g, ϵ
C	h, ϵ

(vii) $S \to aAb / bAaB / \epsilon$
$A \to S$
$B \to S$

	First
S	a, b, ϵ
A	a, b, ϵ
B	a, b, ϵ

★ Q. $A \rightarrow A_1 A_2 A_3$

First (A_1) = 3 elements
First (A_2) = 4 elements
First (A_3) = 5 elements

All first sets containing ϵ and all elements are different.

Then First (A) =

(a) 9
✓ (b) 10
(c) 11
(d) 8

$A \rightarrow A_1 A_2 A_3$
$2 + 3 + 4 + \epsilon(1)$
$\Rightarrow 10$

L-18

★★★★ FOLLOW SET

Follow of Nonterminals is the set of terminals which appears immediately on the RHS of that non-terminal in some sentential form

Example -

	LMD	RMD
$S \rightarrow AB$	$S\$$	$S\$$
$A \rightarrow a$	$AB\$$	$AB\$$
$B \rightarrow b$.	$aB\$$	$Ab\$$
	$ab\$$	$ab\$$

Follow (S) = { $ }
Follow (B) = { $ }
Follow (A) = { b }

1. Follow(S) = { $ } Follow of starting symbol is always $

2. If Grammar is
$A \to \alpha B X$ where $x \not\to \epsilon$
Follow(B) = First(X)

3. If Grammar is
$A \to \alpha B$
Follow(B) = Follow(A)

4. If Grammar is
$A \to \alpha B X$ where $x \to \epsilon$
Follow(B) = { First(X) - ϵ } $\cup$ Follow(A)

Q. (i) $S \to AaAb \mid BbBa$
$A \to \epsilon$
$B \to \epsilon$

	Follow
S	{ $ }
A	{ a, b }
B	{ a, b }

(ii) $E \to T + E$
$E \to T$
$T \to a$

	Follow
E	{ $ }
T	{ +, $ }

(iii) $E \to TE'$

$E' \to +TE' / \epsilon$

$T \to FT'$

$T' \to *FT' / \epsilon$

$F \to (E) / a$

	Follow
E	{$,)}
E'	{$,)}
T	{+, $,)}
T'	{+, $,)}
F	{*, +, $,)}

Follow(E') = Follow(E)

Follow(T') = Follow(T)

Follow(T) = First(E')

{+, ε}

Follow(F) = First(T')

*, ε

(iv) $S \to ACB / Cb / Ba$

$A \to da / BC$

$B \to g / \epsilon$

$C \to h / \epsilon$

Follow(A) = First(C)

= First(B)

= Follow(S)

	Follow
S	{$}
A	{h, g, $}
B	{a, h, g, $}
C	{b, g, $, h}

b,

Follow(B) = First(C)

= Follow(A)

Follow(C) = b

= First(B)

= Follow(S)

= Follow A

ϵ never appears in follow set

Page No.
Date : / /201

(v) $S \to aSA \mid \epsilon$
$A \to c \mid \epsilon$

Follow(S) = $

First(A)

Follow(A) = Follow(S)

	Follow
S	{$, c}
A	{$, c}

(vi) $S \to aAbB \mid bAaB \mid \epsilon$
$A \to S$
$B \to S$

	Follow
S	{$, a, b}
A	{a, b}
B	{$, a, b}

Follow(A) = a, b
Follow(B) = Follow S
Follow(S) = $
Follow(A)
Follow(B)

Q. Consider the following grammar :

$P \to xQRS$
$Q \to yz \mid z$
$R \to w \mid \epsilon$
$S \to y$

What is the follow(Q) ?

(a) {R} (b) {w} ✓(c) {w, y} (d) {w, $}

GATE PYQ

Q. Consider the following grammar given below:

$S \rightarrow Aa$

$A \rightarrow BD$

$B \rightarrow b / \epsilon$

$D \rightarrow d / \epsilon$

Let a, b, d and \$ be indexed as follows:

a	b	d	\$
3	2	1	0

Compute the follow set of the non-terminal B and write the index values for the symbols in the follow set in the descending order.
(For example, if the follow set is {a, b, d, \$}, the answer should be 3210).

$\Rightarrow$

Follow(B) = First(D) $\rightarrow d$

Follow(A) $\rightarrow a$

Follow(A) = a

$\therefore$ Answer = 31

Q. $S \to aAbB \mid bAaB \mid \epsilon$

$A \to S$

$B \to S$

First (A) = {a, b, ε}

First (B) = {a, b, ε}

Follow (A) = {a, b}

Follow(B) = Follow(S) = {$, a, b}

Follow(S) = {$, a, b}

Q. $S \to iEtSS' \mid a$

$S' \to eS \mid \epsilon$

$E \to b$

	Follow
S	{$, e}
S'	{e, $}
E	{t}

Follow(E) = t

Follow(S) = First (S') ∪ Follow (S')

First(S') = {e, ε}

Follow (S') = Follow S

Page No.
Date : / /201

Q. Find First and Follow

$S \rightarrow L=R \mid R$

$L \rightarrow *R \mid a$

$R \rightarrow L$

	First		Follow
S	{*, a}	S	{$}
L	{*, a}	L	{=, $}
R	{*, a}	R	{$, =}

Follow (S) ⇒ $

Follow (L) ⇒ = and Follow (R)

Follow (R) ⇒ Follow (S)

Follow (L)

L-19

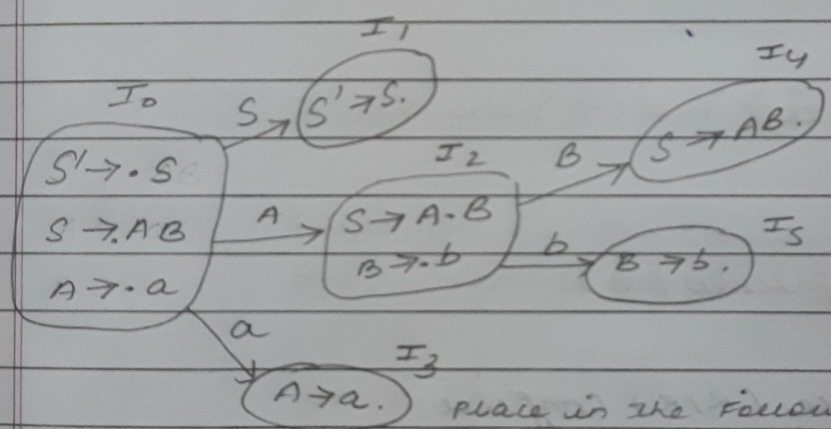
31-07-2021

SLR(1) Parsing ~~table~~ Algorithm

1. Augmented Grammar
2. Construct LR(0) items DFA → Closure()
   → Goto
3. Reduce actions are filled as per symbol of follow value

Example -

	Follow:-
$S \to AB$	{$}
$A \to a$	{b}
$B \to b$	{$}



Place in the Follow of LHS.

	Shift			GOTO		
	a	b	$	S	A	B
I_0	S_3			1	2	
I_1			Accept			
I_2		S_5				4
I_3		r_2				
I_4			r_1			
I_5			r_3			

Difference between LR(0) and SLR(1) is that the reduce actions may be different and Error actions may be different.

Shift and GOTO actions are same.

SLR(1) Grammar Detection

Any Grammar in SLR(1) parsing table contains Shift/Reduce conflict (OR) Reduce/Reduce Conflict then that grammar is not SLR(1) grammar.

* Conflict States for SLR(1) parser

S/R Conflict

$A \rightarrow \alpha . a X$

$B \rightarrow Y.$

if $a \in$ follow(B)

Reduce/Reduce (R/R) Conflict

$A \rightarrow X.$

$B \rightarrow Y.$

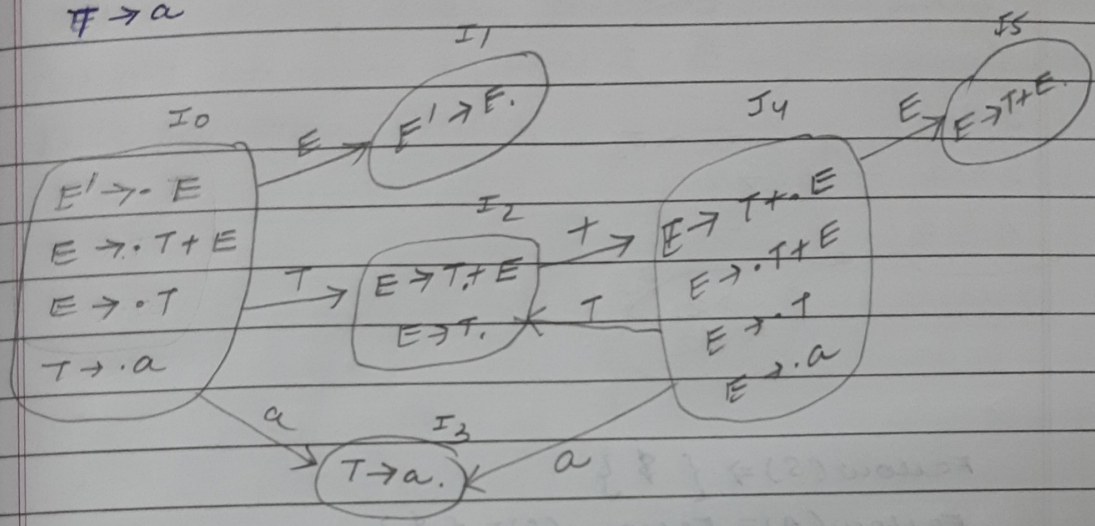
Follow(A) $\wedge$ Follow(B) $\neq \phi$

Q. Verify given grammar is SLR(1) or not ?

(i) $E \to T+E$

$E \to T$

$T \to a$



Follow(E) = { $ }

I_2 42 Shift (1) & '+' 42

E → T. reduce at $.

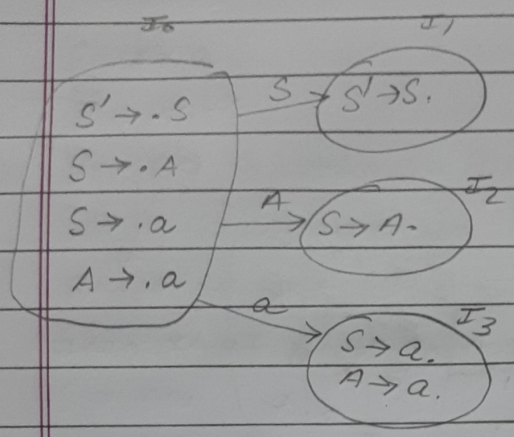
NO conflict

∴ the grammar is SLR(1)

(ii) $S \to A$

$S \to a$

$A \to a$



Follow(S) ⇒ { $ }

Follow(A) = Follow (S) = { $ }

Reduce/Reduce conflict at I_3

∴ Grammar is not SLR(1)

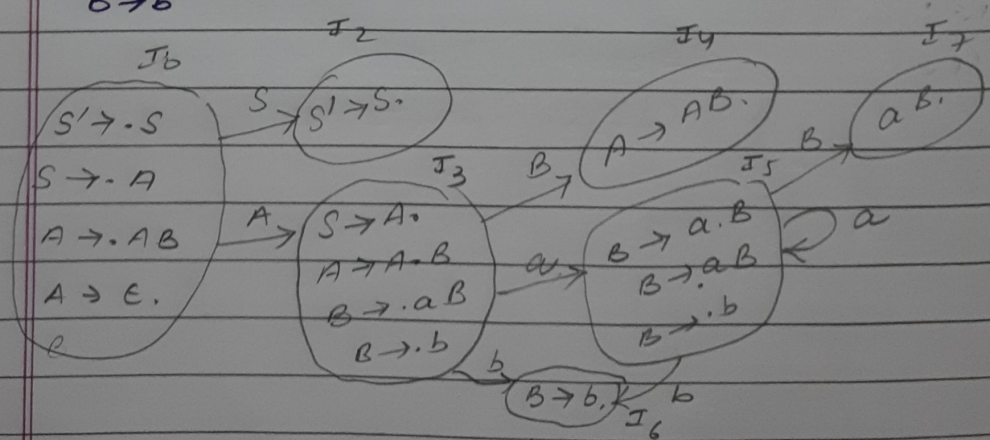
(iii) $S \to A$

$A \to AB$

$A \to \epsilon$

$B \to aB$

$B \to b$



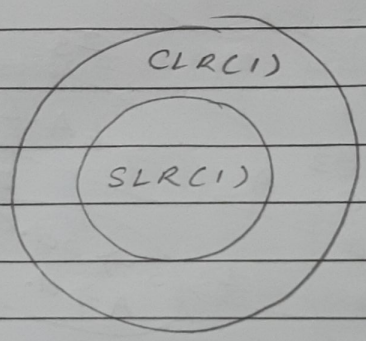
Follow(S) = { $ } ≠ a or b

⇒ NO conflict, ∴ grammar is SLR(1).

L-20

Drawback of SLR(1)

SLR(1) parsing table contains unnecessary reduce actions compared to CLR(1); hence, SLR(1) is less powerful than CLR(1).



Q. verify the following grammar is SLR(1) or not?

$S \to AaAb$

$S \to BbBa$

$A \to \epsilon$

$B \to \epsilon$

I_0

$S' \to .S$

$S \to .AaAb$

$S \to .BbBa$

$A \to \epsilon.$

$B \to \epsilon.$

आगे करने की आवश्यकता नहीं है।

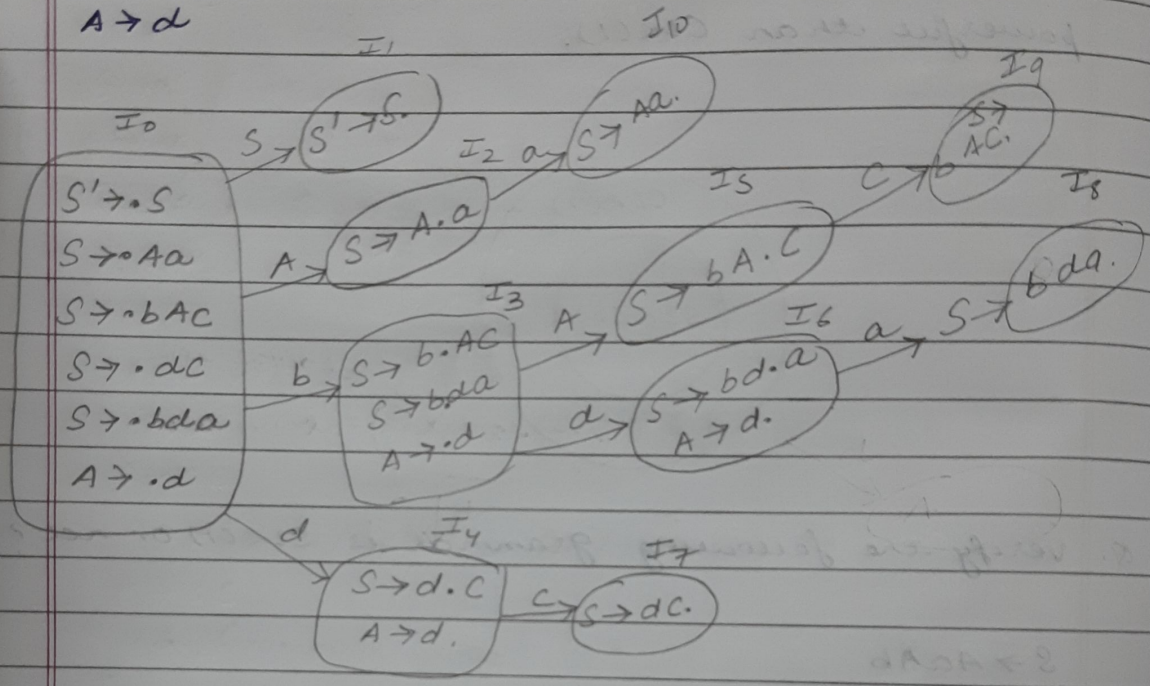
Follow(A) ⇒ {a, b}

Follow(B) = {b, a}

Reduce-Reduce (R/R) Conflict ∴ Not SLR(1)

*3 Q. verify the given grammar is SLR(1) or not?

$S \to Aa$
$S \to bAc$
$S \to dc$
$S \to bda$
$A \to d$



Follow(A) = { a, c }

S/R conflict in I_4 and I_6
∴ grammar is not SLR(1)

Total conflicting states = 2
[at I_4 and I_6]

(3) CLR(1) Parsing

Caronical LR(1)

CLR(1) table construction.

1. Construct Augmented Grammar
2. Construct LR(1) items DFA by using closure() and goto()
3. Construct parsing table

LR(1) items = LR(0) items + Lookahead

Example -

$S \to AB$

$A \to a$

$B \to b$

$A \to .a, X$ → lookahead

LR(1) item

Calculation of lookaheads is done using closure.

$S \to A.B, Y$

$B \to .b$, first(Y) ⇒ .B के आगे जो कुछ है उसका first निकालो।

Page No.
Date : / /201

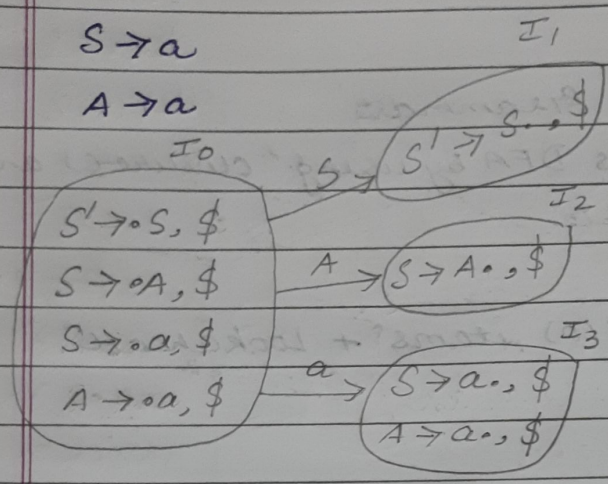
Q. Construct LR(1) items DFA for the following grammar.

(i)

$S \to A$

$S \to a$

$A \to a$



(ii)

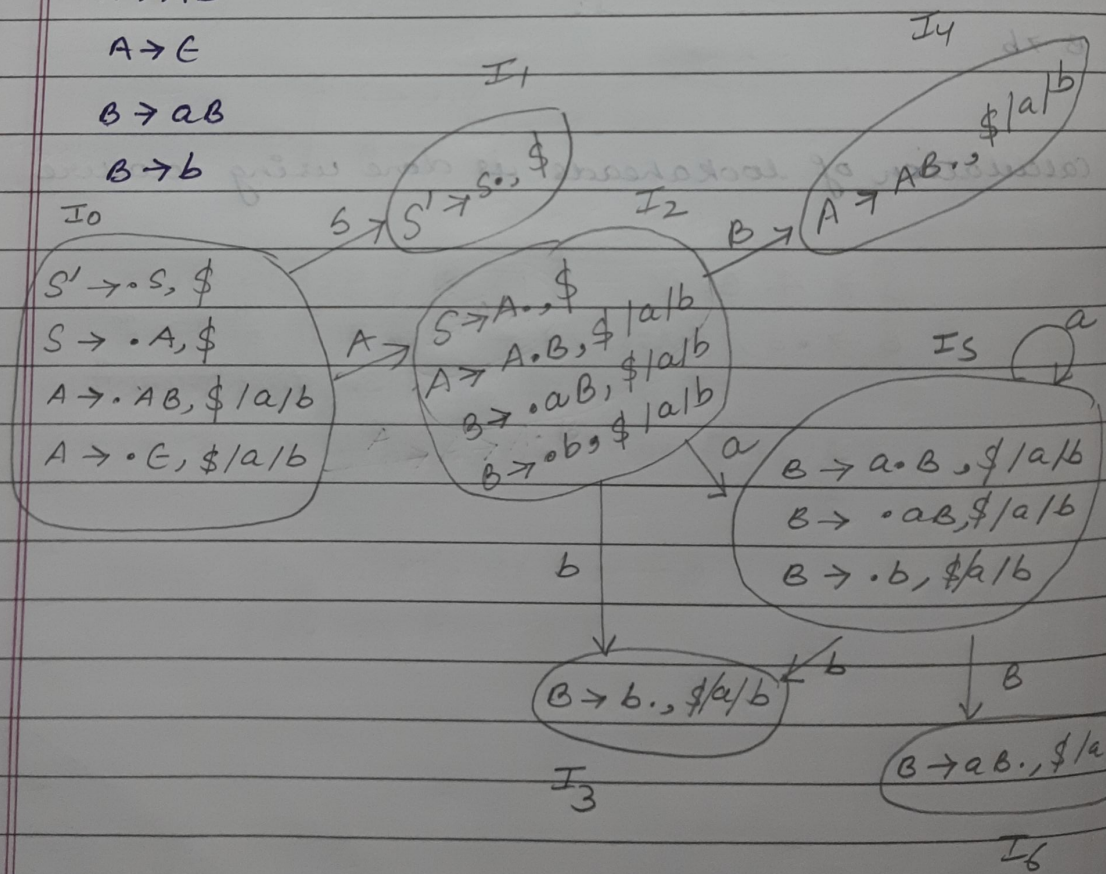
$S \to A$

$A \to AB$

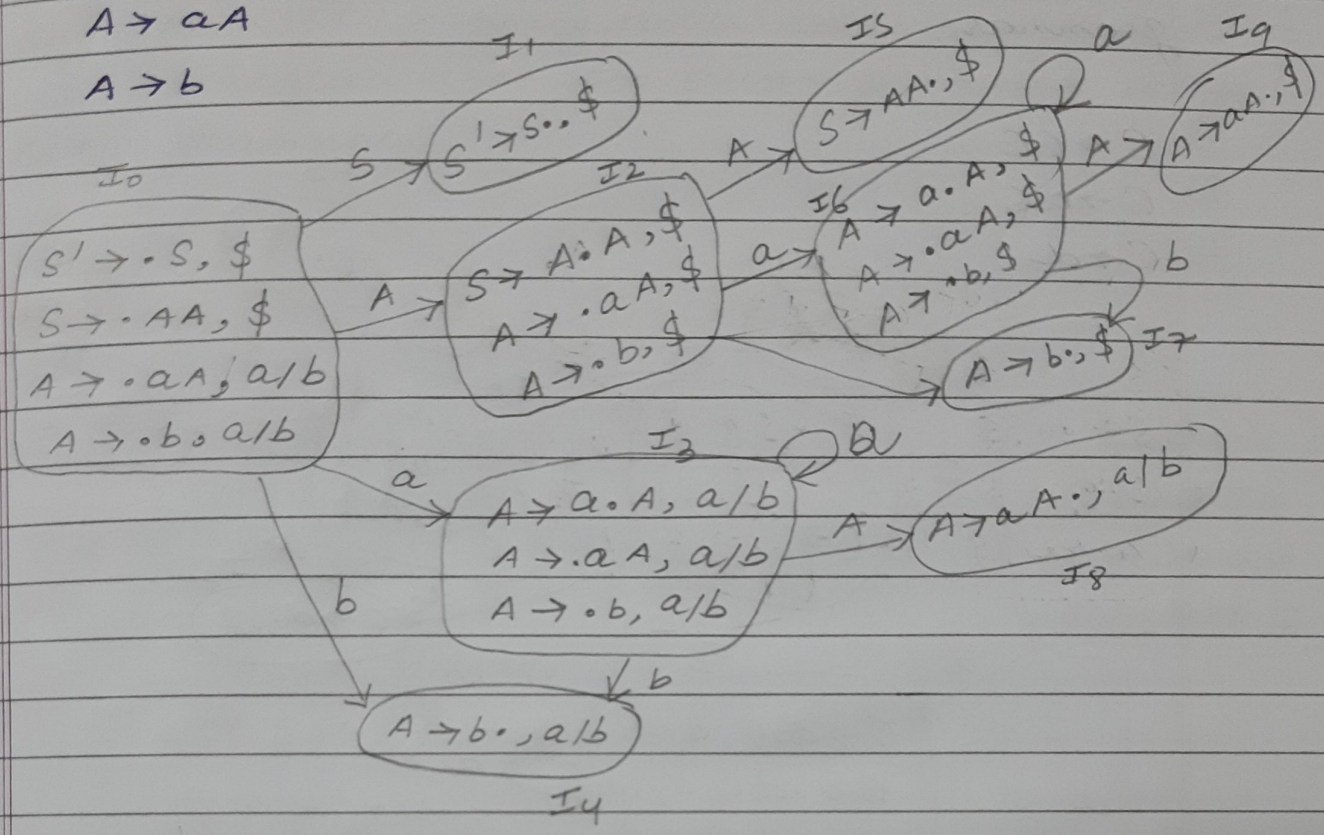
$A \to \epsilon$

$B \to aB$

$B \to b$



(iii) $S \to AA$
$A \to aA$
$A \to b$



L-21

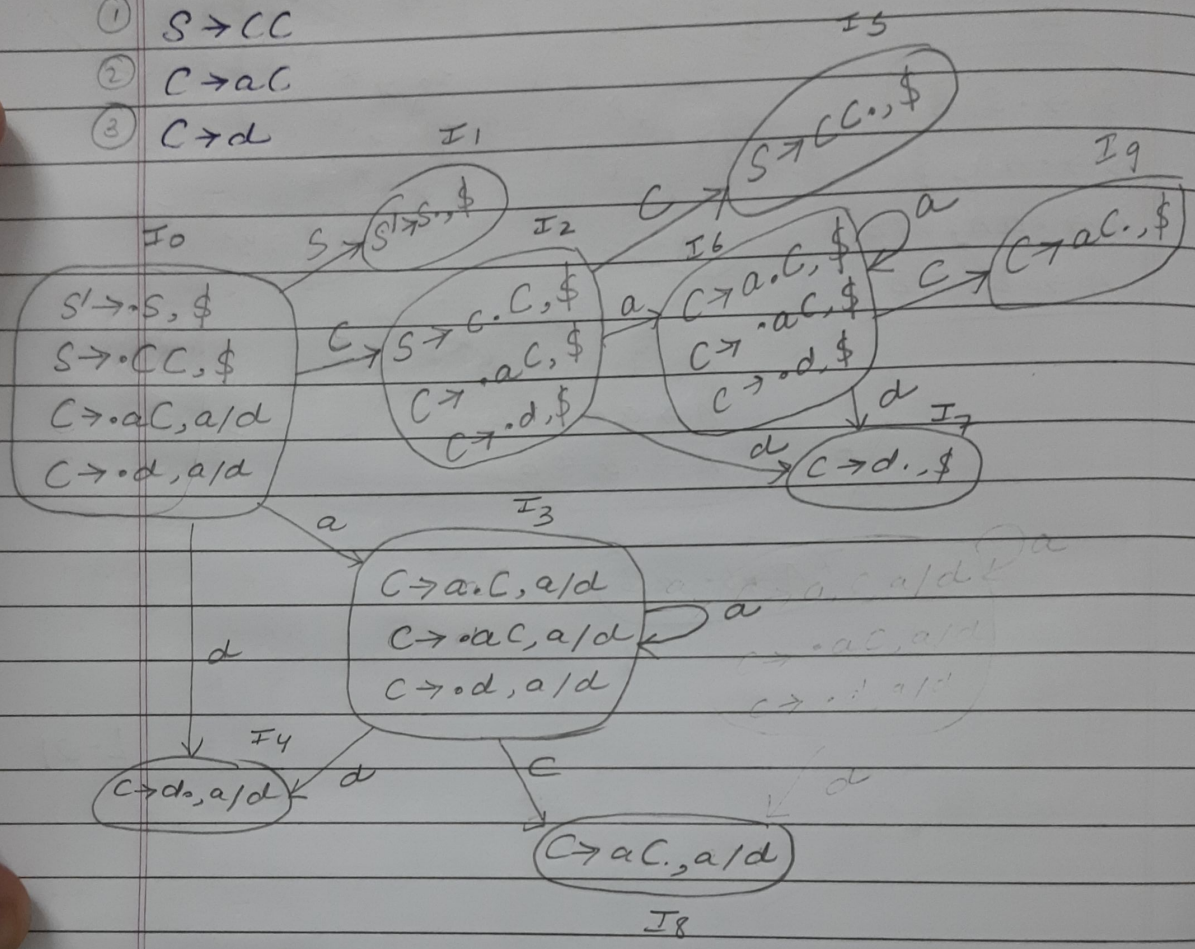
१ अगस्त २०२१, रविवार

Page No.
Date : / /201

01-08-2021

Q. Construct CLR(1) parsing table for the following grammar.

① $S \to CC$
② $C \to aC$
③ $C \to d$



	Action			Goto	
	a	d	$	S	C
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_6	S_7			5
I_3	S_3	S_4			8
I_4	r_3	r_3			
I_5			r_1		
I_6	S_6	S_7			9
I_7			r_3		
I_8	r_2	r_2			
I_9			r_2		

CLR(1) grammar detection

Any CFG CLR(1) parsing table contains S/R conflict or R/R conflict then the given grammar is not CLR(1) grammar.

Note:-
Default LR(1) parser is CLR(1)

Conflict states

1) S/R Conflict -

$A \to \alpha . a X, t_1$
$B \to Y., t_2$

if $a \in t_2$ then S/R conflict

2) R/R conflict -

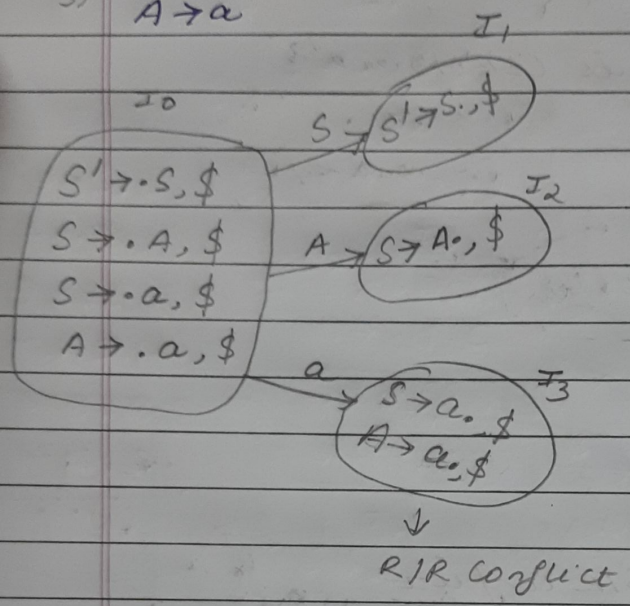
$A \to X., t_1$
$B \to Y., t_2$

$\{t_1\} \cap \{t_2\} \neq \phi$

Any LR(1) items DFA contains either S/R conflict or R/R conflict that grammar is not CLR(1)

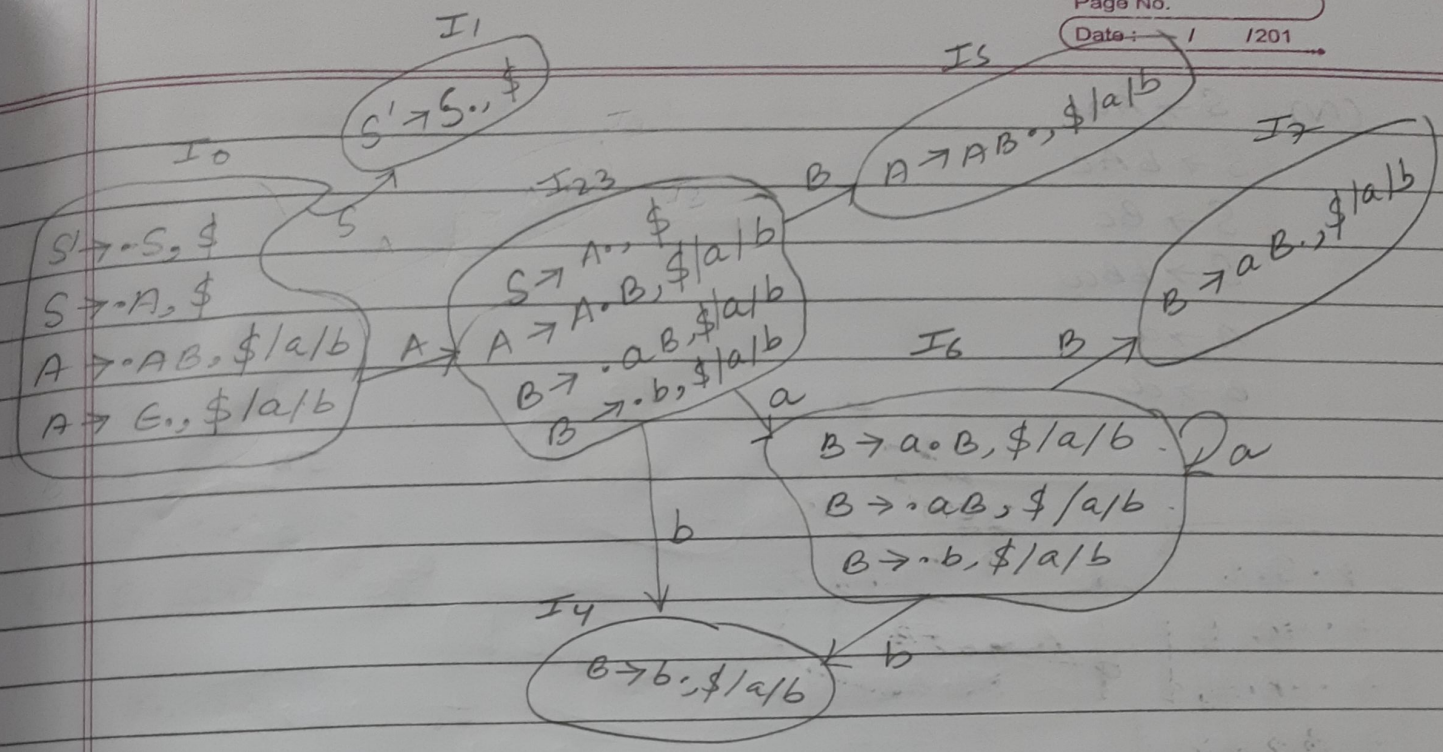
Q. Verify following grammar is CLR(1) grammar or not?

(i) 1) $S \to A$
2) $S \to a$
3) $A \to a$



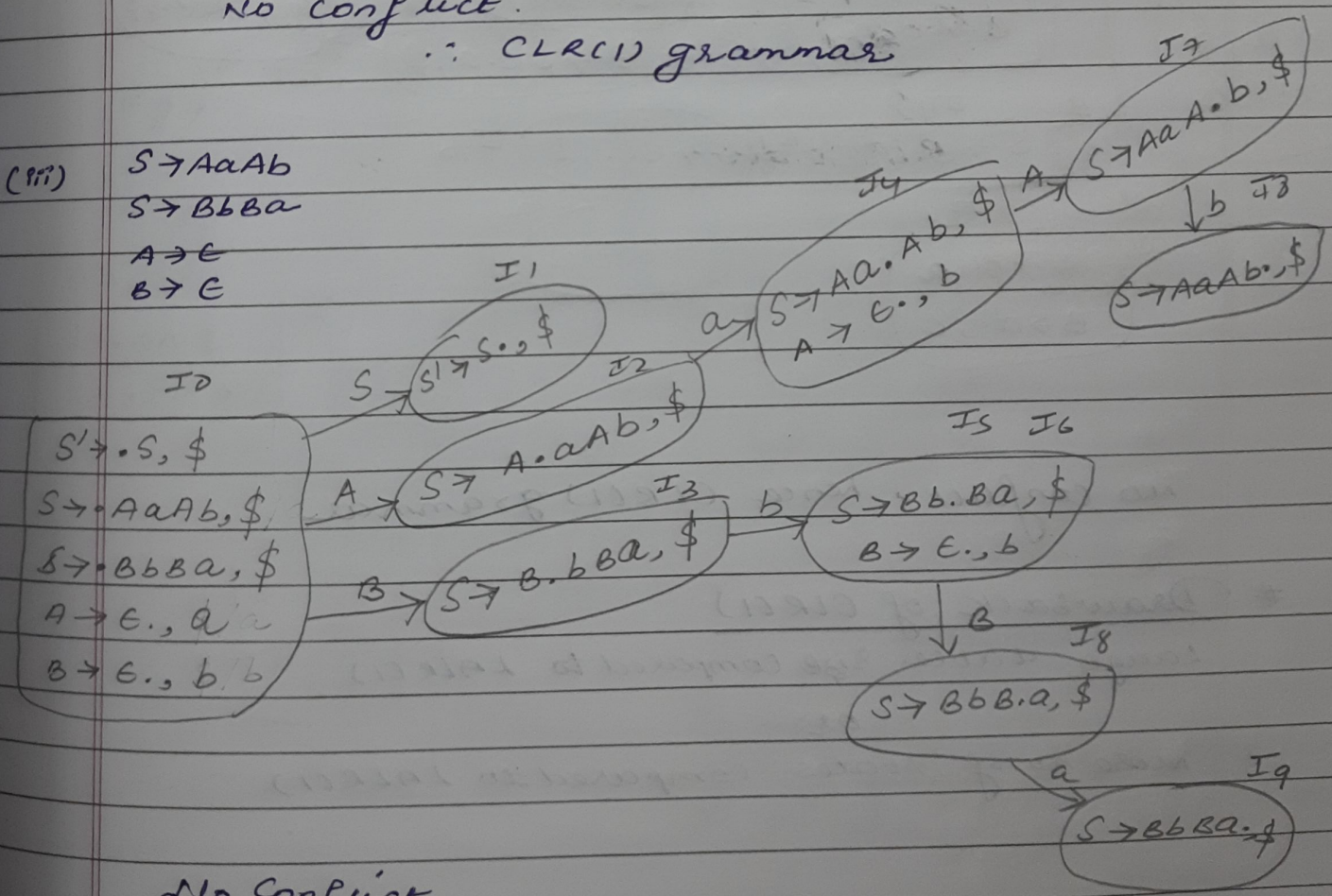
∴ the grammar is not CLR(1)

(ii) $S \to A$
$A \to AB$
$A \to \epsilon$.
$B \to aB$
$B \to b$.



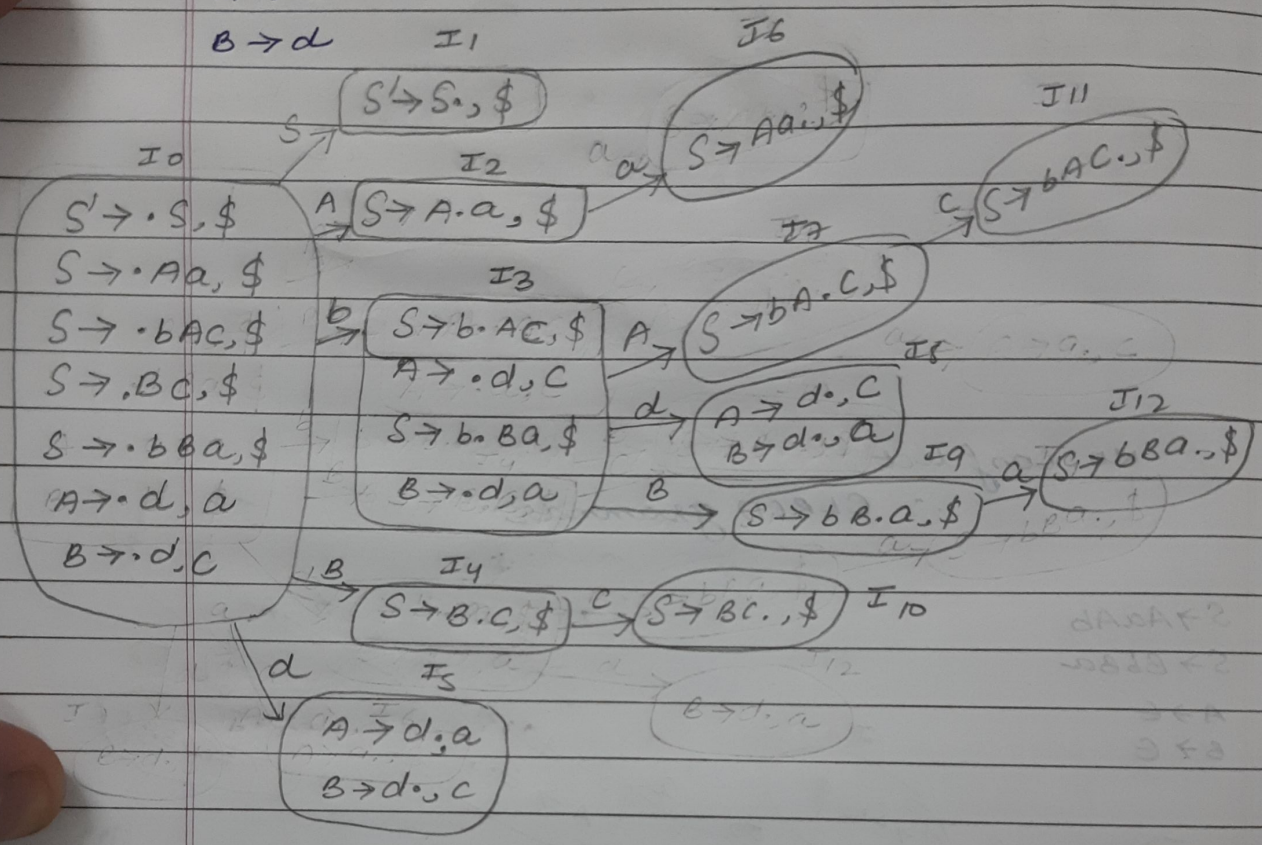
No conflict.
∴ CLR(1) grammar

(iii) $S \to AaAb$
$S \to BbBa$
$A \to \epsilon$
$B \to \epsilon$



No Conflict
∴ CLR(1)

(PV) $S \to Aa$
$S \to bAc$
$S \to Bc$
$S \to bBa$
$A \to d$
$B \to d$



no conflict, hence CLR(1) grammar

Drawback of CLR(1)

Large table size compared to LALR(1)

or

More no. of states compared to LALR(1)

(4) Look Ahead LR(1) or LALR(1)

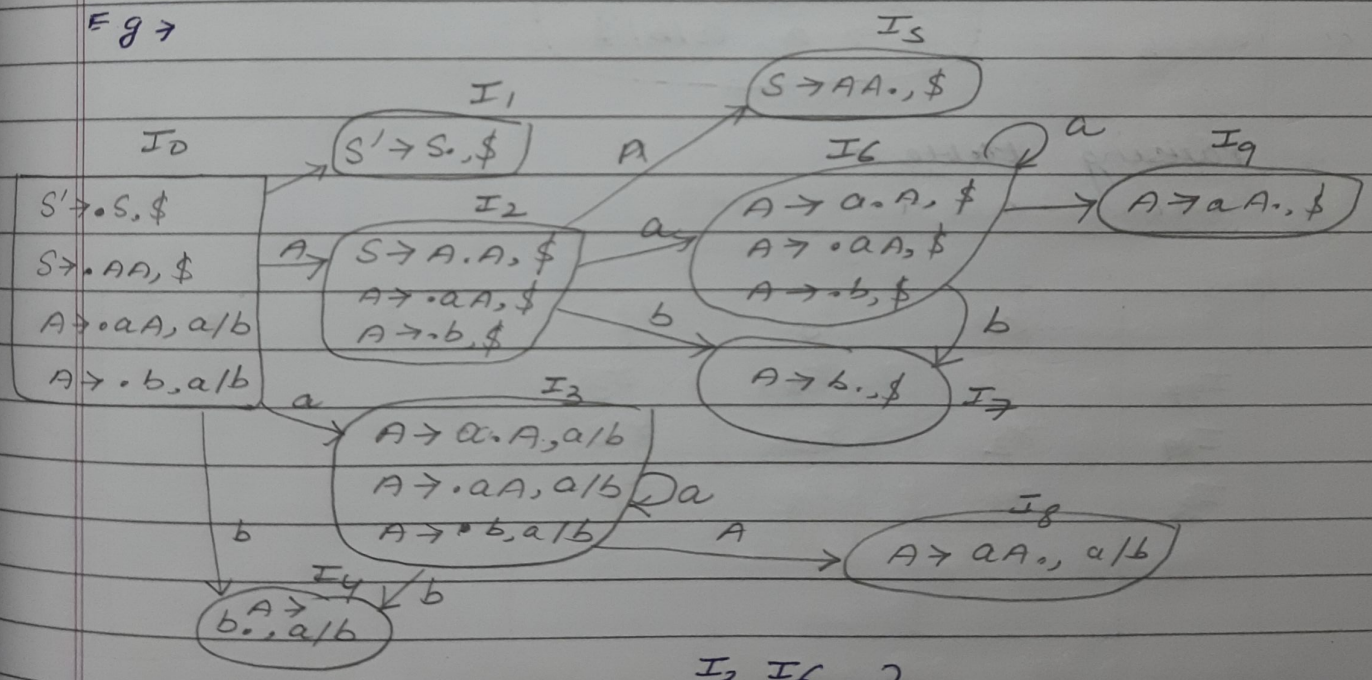
★ LALR(1) table construction :-

(1) Construct Augmented grammar
(2) Construct LR(1) items DFA by using closure() and goto()
(3) merge the similar states into single state
(4) Construct parsing table.

~~In LR(1)~~ : items DFA any state having same LR(0) part and different lookahead part then merge them into a single state

$S \to AA$ (1) $A \to aA$ (2) $A \to b$ (3)

Eg →



I_3 I_6
I_4 I_7
I_8, I_9 } same LR(0) part but different lookahead part

No. of states in
LALR(1) ⇒ 7
CLR(1) → 10

I_{47}

$A \to b\cdot, a/b/\$$

I_{89}

$A \to aA\cdot, a/b/\$$

I_{36}

$A \to a\cdot A, a/b/\$$
$A \to \cdot aA, a/b/\$$
$A \to \cdot a, a/b/\$$

Parsing table :-

	a	b	$	S	A
I_{47}	r_3	r_3	r_3		
I_{89}	r_2	r_2	r_2		
I_5			r_1		

Conflict states of LALR(1) is same as CLR(1)

CLR(1)

I_m: $A \to X \cdot, a$; $B \to Y \cdot, b$

No conflict (R/R)

I_n: $A \to X \cdot, b$; $B \to Y \cdot, a$

No conflict (R/R)

LALR(1)

I_{mn}: $A \to X \cdot, a/b$; $B \to Y \cdot, a/b$

R/R conflict

Not LALR(1)

CLR(1)

I_m: $A \to \alpha \cdot aX, t_1$; $B \to Y \cdot, b$

No Conflict (S/R)

I_n: $A \to \alpha \cdot aX, t_2$; $B \to Y \cdot, C$

No conflict (S/R)

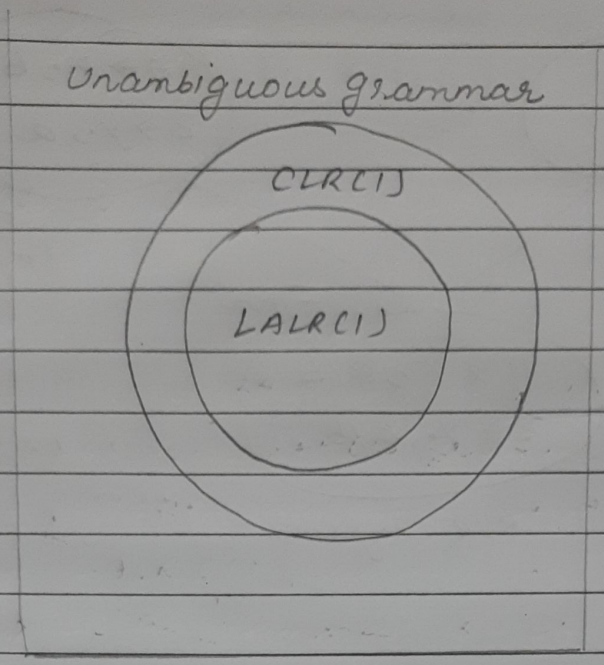
LALR(1)

I_{mn}: $A \to \alpha \cdot aX, \{t_1/t_2\}$; $B \to Y \cdot, b/C$

No conflict (S/R)

If no S/R conflict in CLR(1) then no chance of having S/R conflict in LALR(1).

If no R/R conflict CLR(1) then there might be R/R conflict in LALR(1).



CLR(1) ≠ LALR(1) because of R/R Conflict

Q. verify the following grammar is

a) CLR(1) or not ?
b) LALR(1) or not ?

(i) Grammar :-

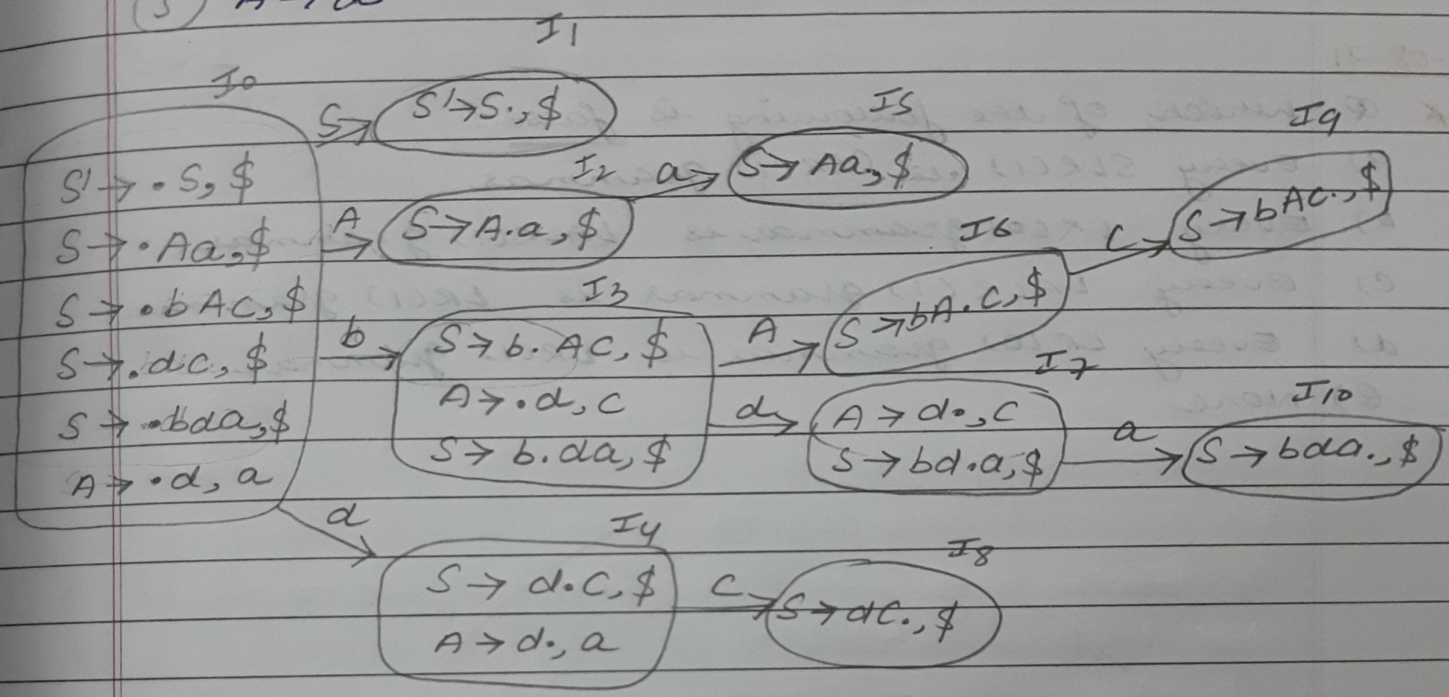
$S \to Aa$
$S \to bAc$
$S \to Bc$
$S \to bBa$
$A \to d$
$B \to d$

CLR(1) ✓ LALR(1) ✗

I_{58}
$A \to d., a/c$
$B \to d., a/c$

(MP)

1. $S \to Aa$
2. $S \to bAC$
3. $S \to dc$
4. $S \to bda$
5. $A \to d$



NO conflict, hence CLR(1)

Even after merging, no conflict, hence LALR(1)
(No similar states in the above DFA).

★★★# Number of states relation

CLR(1) ≥ LALR(1) = SLR(1) = LR(0)

L-23

02-08-21

★ Q. Which of the following is false?

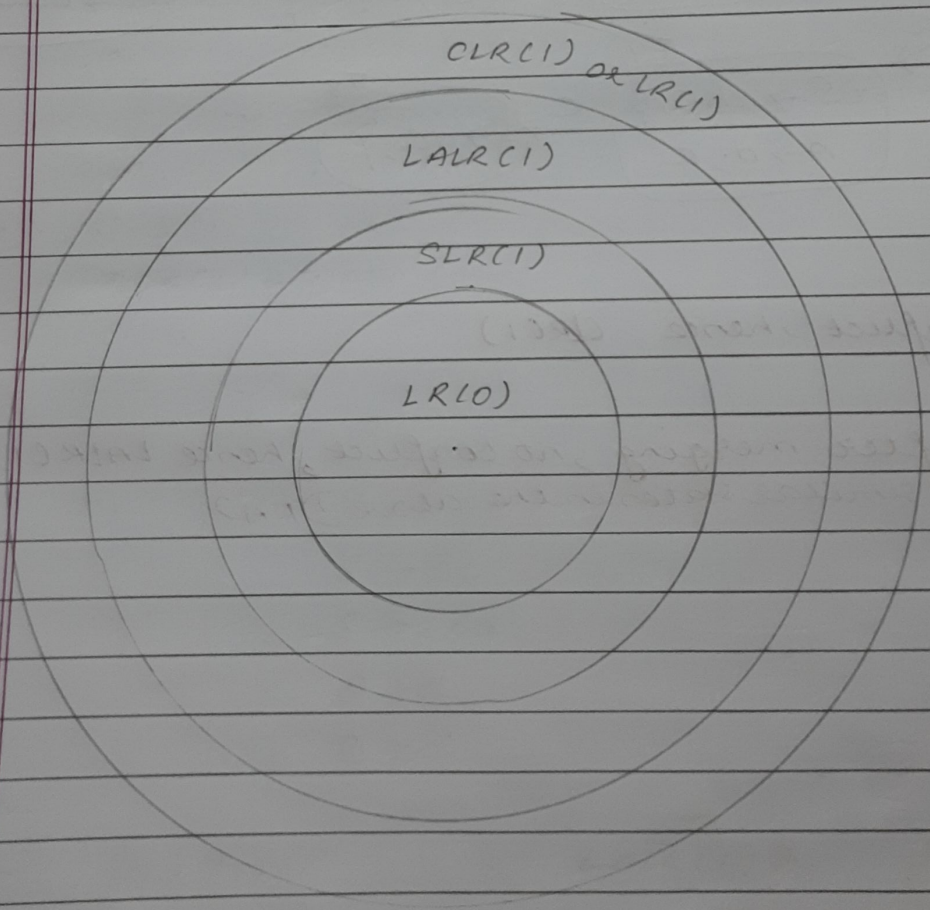
a) Every SLR(1) is CLR(1) grammar

b) Every LR(0) grammar is LALR(1) grammar

c) Every LALR(1) grammar is LR(1) grammar

d) Every LR(0) grammar is LR(1) grammar

✓e) None



Power of the parser

No. of grammars suitable for particular parser is known as power of parser.

LR(0) < SLR(1) < LALR(1) < CLR(1)

Less powerful — More powerful

GATE PYQS

Q. Which of the following is most easy to design?

a) LR(0)

b) SLR(1)

c) CLR(1)

d) LALR(1)

Q. 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, CLR and LALR have the same power.

Q. Assume that the SLR parser for a grammar G has n_1 states and LALR parser for G has n_2 states. The relationship between n_1 & n_2 is

(a) n_1 is necessarily less than n_2

✓(b) n_1 is necessarily equal to n_2

(c) n_1 is necessarily greater than n_2

(d) None of the above.

Q. Consider the grammar

$$S \rightarrow (S) \mid a$$

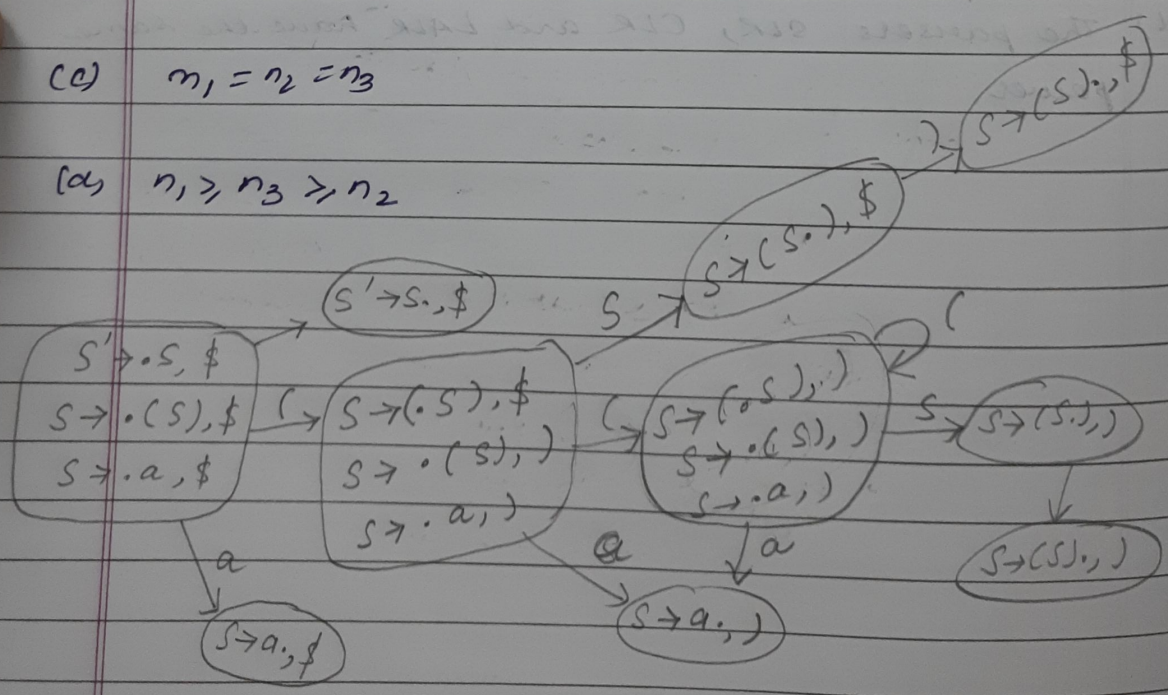
Let the no. of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n_1, n_2 & n_3 respectively. The following relationship holds good.

(a) $n_1 < n_2 < n_3$

✓(b) $n_1 = n_3 < n_2$

(c) $n_1 = n_2 = n_3$

(d) $n_1 \geq n_3 \geq n_2$



Q. Consider the following grammar

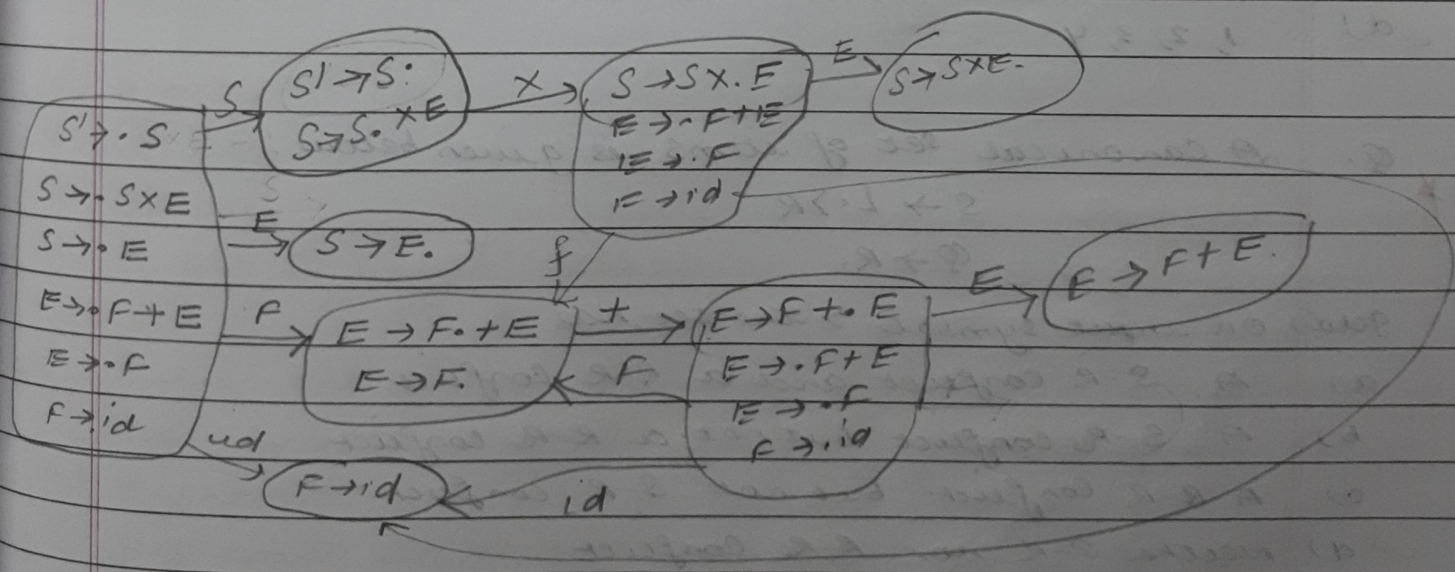
$S \to S \times E$
$S \to E$
$E \to F + E$
$E \to F$
$F \to id$

Consider the following LR(0) items corresponding to the grammar above.

(i) $S \to S \times . E$
(ii) $E \to F . + E$
(iii) $E \to F + . E$

Given the items above, which two of them will appear in the same set in the canonical sets-of-items for the grammar?

a) (i) and (ii) (b) (ii) and (iii)
c) (i) and (iii) ✓ (d) None



Gate PYQ

2 marks

Q. Consider the following two sets of LR(1) items of LR(1) grammar.

$X \to c.X, c/d$	$X \to c.X, \$$
$X \to .cX, c/d$	$X \to .cX, \$$
$X \to .d, c/d$	$X \to .d, \$$

Which of the following statement related to merging of two sets in the corresponding parser is/are FALSE?

1. Cannot be merged since look aheads are different
2. Can be merged but will result in S-R conflict
3. Can be merged but will result in R-R conflict
4. Cannot be merged since goto on c will to two different ~~states~~? sets.

a) 1 only
b) 2 only
c) 1 & 4 only
✓ d) 1, 2, 3, 4

Q. A canonical set of items is given below :-

$S \to L.>R$

$Q \to R.$

goto() on input symbol > the set has

a) A S-R conflict and a R-R conflict
b) A S-R conflict but not a R-R conflict
c) A R-R conflict but not S-R conflict
✓ d) Neither S-R nor R-R conflict

$S \to L \cdot > R$
$\Phi \to R \cdot$ $\Rightarrow$ $S \to L > \cdot R$

इस state को पार कर गई है।

Q. Among simple LR (SLR), canonical LR, and look-ahead LR (LALR), which of the following pairs identify the method that is very easy to implement and the method that is most powerful in that order?

a) SLR, LALR
b) CLR, LALR
✓c) SLR, CLR
d) LALR, CLR

Q. Which of the following statements about parser is/are CORRECT?

I Canonical LR is more power than SLR
II SLR is more powerful than LALR
III SLR is more power than CLR.

✓a) I only
b) II only
c) III only
d) II and III only

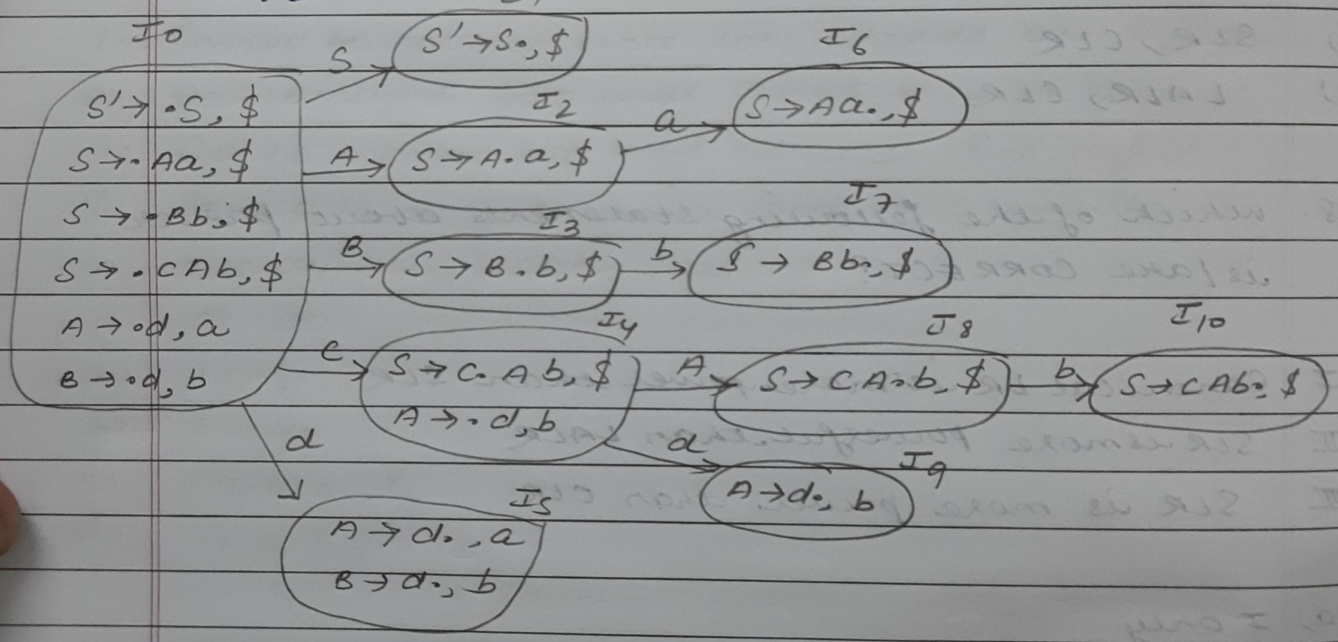
Q. Which one of the following kinds of derivation is used by LR parser?

a) Rightmost
✓b) Rightmost in reverse
c) Leftmost
d) Leftmost in reverse

Q. verify following grammar is

(1) CLR(1) or not?
(2) LALR(1) or not?
(3) SLR(1) or not?
(4) LR(0) or not?

$S \to Aa / Bb / CAb$
$A \to d$
$B \to d$



For SLR(1) ⇒ Is common
Follow(A) ⇒ a, b
Follow(B) ⇒ b

CLR(1) ✓
LALR(1) ✓
SLR(1) ✗
LR(0) ✗

No. of States ⇒ CLR(1) = 11
LALR(1) = 11
SLR(1) = 11
LR(0) = 11

Page No.
Date : / /201

NOTE: Ambiguous grammar :

* Any Ambiguous grammar is -
  a) not CLR(1)
  b) not LALR(1)
  c) not SLR(1)
  d) not LR(0)

Suitable for ambiguous as well as unambiguous grammars

Operator-Precedence Parser

(i) very easy to design
(ii) It is S/R parser
(iii) It can handle ambiguous grammar.
(iv) It is suitable only for operator Grammar.

→ An operator precedence parser is a bottom-up parser that interprets an operator grammar. This parser is only used for operator grammars.

→ Ambiguous grammars are not allowed in any parser except operator precedence parser.

A grammar that is used to define mathematical operators is called an operator grammar or operator precedence grammar.

** Operator grammar -
It is a CFG in which
(1) no ε should be there
(2) no adjacent non-terminal allowed

E.g ⇒

$E \to E+E \mid E*E \mid id$

$S \to SAS \mid a$
$A \to bSb \mid b$ } Not an Operator grammar

↓ converted

$S \to SbSbS \mid SbS \mid a$

Q. Which of the following violates rules of operator grammar?

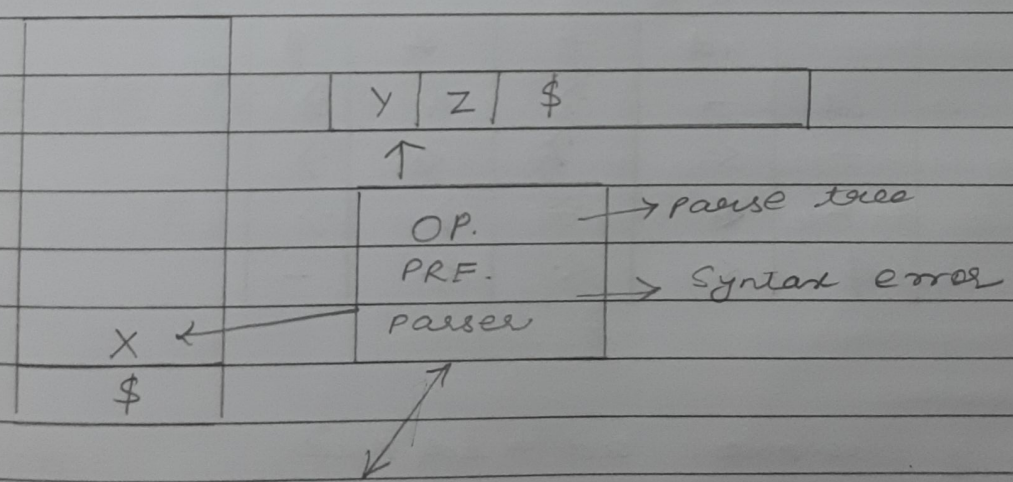
a) $E \to E+E / E-E / a$

b) $E \to E+T / T$
$T \to a$

c) $S \to AB$
$A \to a$
$B \to b$

d) None

OPERATOR PRECEDENCE PARSING ALGORITHM



	a	+	$
a		$\circ\!>$	$<\!\circ$
+	$<\!\circ$	$\circ\!>$	$\circ\!>$
$	$<\!\circ$	$<\!\circ$	accept

1. $X \lessdot Y$, $X \doteq Y$ } Shift

2. $X \gtrdot Y$ } Reduce

3. $[X, Y]$ = accept

4. $[X, Y]$ = Blank } Syntax Error

Example:-

Consider the grammar

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$

	id	+	*	$
id	–	$\gtrdot$	$\gtrdot$	$\gtrdot$
+	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
*	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$	–

input: id + id * id

$

Stack		input buffer	Action
			$ + id
$	<·	id + id * id $	Shift (id +)
$ | id	·>	+ id * id $	Reduce
$ | E	($ +) <·	+ id * id $	Shift
$ | E | +	<·	id * id $	Shift
$ | E | + | id	·>	* id $	Reduce
$ | E | + | E	<·	* id $	Shift
$ | E | + | E | *	<·	id $	Shift
$ | E | + | E | * | id	·> (* $)	$	Reduce
$ | E | + | E | * | E		$	Reduce
$ | E | + | E		$	Reduce
$ | E		$	—

Note:-

Time Complexity of LR Parser ⇒ $O(n)$

Time Complexity of Operator Precedence Parser ⇒ $O(n)$

Time Complexity of CYK algorithm ⇒ $O(n^3)$

03-08-2021

LR is more powerful than Operator Precedence Parser.

Ambiguous grammars are also suitable for operator precedence parser.

Ambiguity is undecidable problem.

Ambiguity may or maynot be removed from an ambiguous grammar.

$E \rightarrow E+E$	$E \rightarrow E+T/T$
$E \rightarrow E*E$	$T \rightarrow T*F/F$
$E \rightarrow id$	$F \rightarrow id$
ambiguous	unambiguous

OPERATOR PRECEDENCE PARSING TABLE CONSTRUCTION

1. Parse tree method is used to determine precedence and associativity of operators.
2. \$ is always at lesser precedence than other operators
3. Identifier is always at higher precedence over the operators.
4. Size of OPP table $\rightarrow (T+\$)^2$
   $T \rightarrow$ terminals

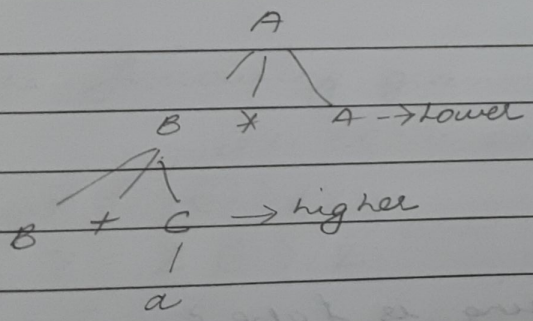
Q. Construct operator Precedence table for the following grammar.

(i) $A \to B * A / B$ → Right Recursion Right associativity

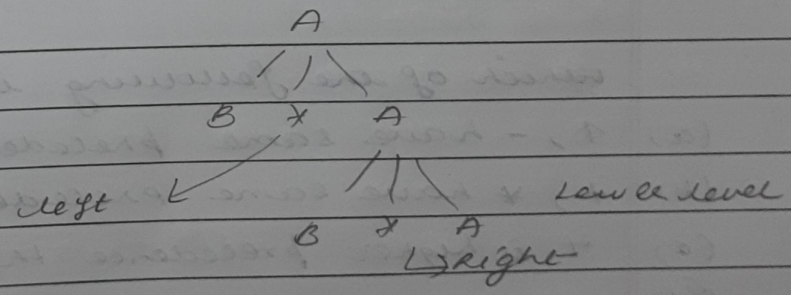
$B \to B + C / C$ → Left Recursion Left associativity

$C \to a$

	a	*	+	$
a		·>	·>	·>
*	<·	<·	<·	·>
+	<·	·>	·>	·>
$	<·	<·	<·	accept



Lower level ⇒ higher Precedence



∴ Right associativity

Remember ⇒ Right recursion
Right associative.

(ii) $A \to B \uparrow A \mid B$

$B \to B \,\&\, C \mid C$

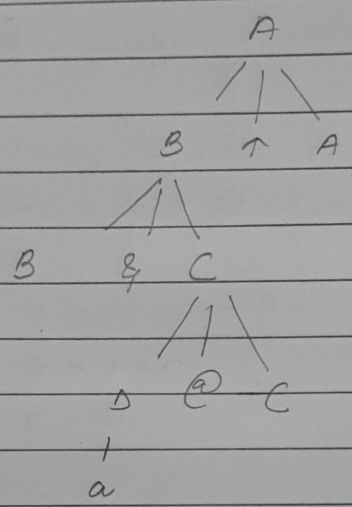
$C \to D @ C \mid D$

$D \to a$

$\uparrow \Rightarrow$ Right associativity

$\& \Rightarrow$ Left ass.

$@ \Rightarrow$ Right asso.



$@ > \uparrow$ $\quad$ $\uparrow \lessdot \&$

$@ \cdot\!> \&$

$\& \lessdot @$

(iii) $E \to E \uparrow T \mid T - E$

$T \to F + E \mid F * T$

$F \to a$

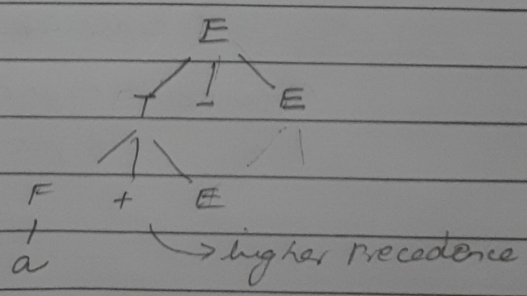
Which of the following is false?

(a) $\uparrow$, $-$ have same precedence

(b) $+$, $*$ have same precedence

(c) $+$ is higher precedence than $-$

(d) None.



Gate PYQ

Q. Consider the grammar defined by the following production rules, with two operators * & +

2 marks

$S \rightarrow T * P$

$T \rightarrow U \mid T * U$ → Left Recursion ⇒ Left associative

$P \rightarrow Q + P \mid Q$ → Right Recursion ⇒ Right associative

$Q \rightarrow Id$

$U \rightarrow Id$

Which of the following is True?

a) + is left associative, while * is right associative

b) + is right associative while * is left associative.

c) Both + & * are right associative.

d) Both + & * are left associative.

Gate PYQ

Q. In the following grammar

2 marks

$X = X \oplus Y / Y$ → Left Recursion ⇒ Left associative

$Y = Z \odot Y / Z$ → Right Recursion ⇒ Right associative

$Z = id$

Which is true?

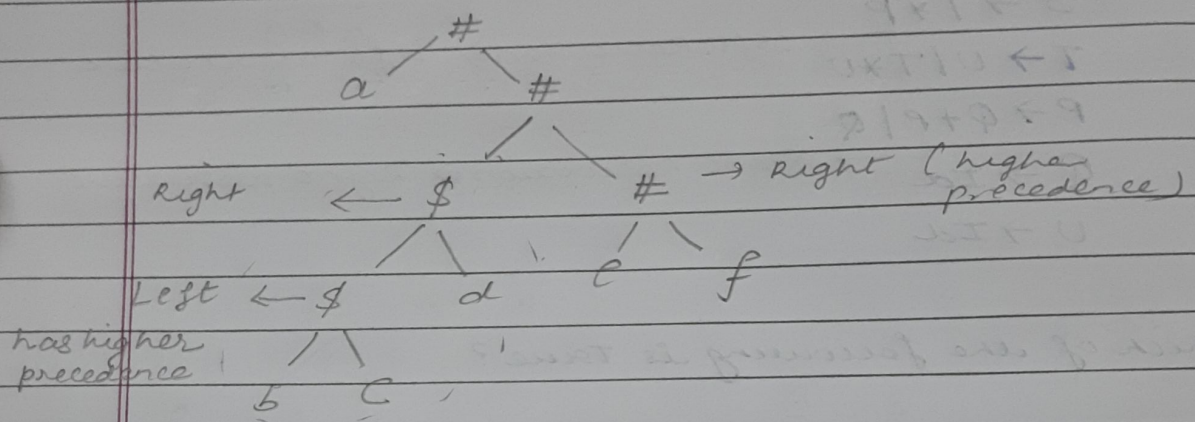
a) $\oplus$ is left associative while $\odot$ is right associative

b) Both $\oplus$ & $\odot$ is left associative

c) $\oplus$ is ~~left~~ right associative & $\odot$ is left associative

d) None.

Q. Consider the following parse tree for the expression a # b $ c $ d # e # f



Which one of the following is true

✓ a) $ has higher precedence and is left associative; # is right associative.

b) # has higher precedence & is left associative; $ is right associative.

c) $ has higher precedence & is left associative # is left associative.

d) # has higher precedence & is right associative; $ is left associative.

जिस $ का higher precedence होगा वही associativity तय करेगा।

Q. Production Rule | Semantic action

Production Rule	Semantic action
$D \to TL$	$X_1.type = X_2.type$
$T \to int$	$T.type = int$
$T \to float$	$T.type = float$
$L \to L_1, id$	$X_3.type = X_4.type$ $addType(id.entry, X_5.type)$
$L \to id$	$addType(id.entry, X_6.type)$

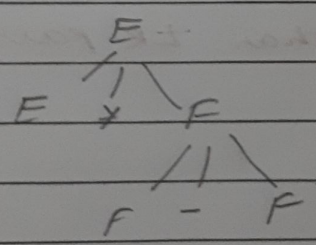
Which one of the following are the appropriate choices X_1, X_2, X_3, X_4 ?

a) $X_1 = L, X_2 = T, X_3 = L_1, X_4 = L$
b) $X_1 = L, X_2 = L, X_3 = L_1, X_4 = T$
c) $X_1 = T, X_2 = L, X_3 = L_1, X_4 = T$
d) $X_1 = T, X_2 = L, X_3 = T, X_4 =$

Q. $E \to E * F \mid F + E \mid F$
$F \to F - F \mid a$

Which of the following is true ?

a) $*$ has higher precedence than $+$
b) $-$ has lesser precedence than $+$
c) $+, -$ has same precedence
✓ d) $-$ has higher precedence than $*$



In any operator grammar if both left Recursion & Right recursion occurs, then the grammar is ambiguous.

E.g ⇒ $E \to E+E \mid E*E \mid n$

Ambiguous

	a	+	*	$
a		·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	

Follow general approach for ambiguous grammars
Don't go by the production rules.

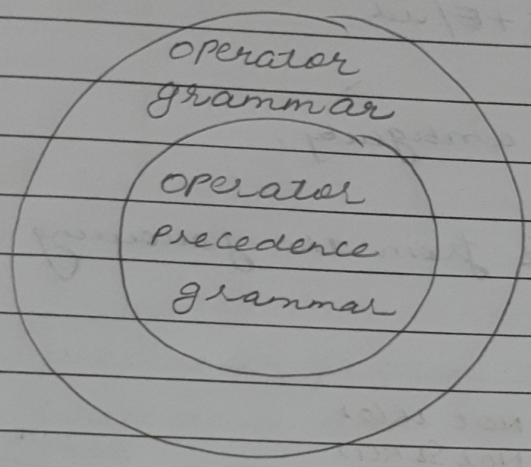
This way the parser also parser ambiguous grammars also.

General approach ⇒ जो C में follow होती है।
left Precedence

Drawback:-
Less powerful ~~by~~ than LR parsers.

Operator Precedence Grammar

It is an operator Grammar and its parsing table should not contain multiple precedence relation ($\rightarrow$ & $\leftarrow$)



$E \rightarrow E*E / E+E / n$

operator grammar but not operator precedence grammar

Drawback of Operator Precedence Parsing

1. Less powerful than LR parser
2. OPP is not suitable for - operator

$-2-3$

↑ unary ↑ binary

Cannot differentiate b/w unary '-' & binary '-'.

Handling Ambiguous Grammar in LR Parsing

Expression = b + c * d;

$S \to id = E;$

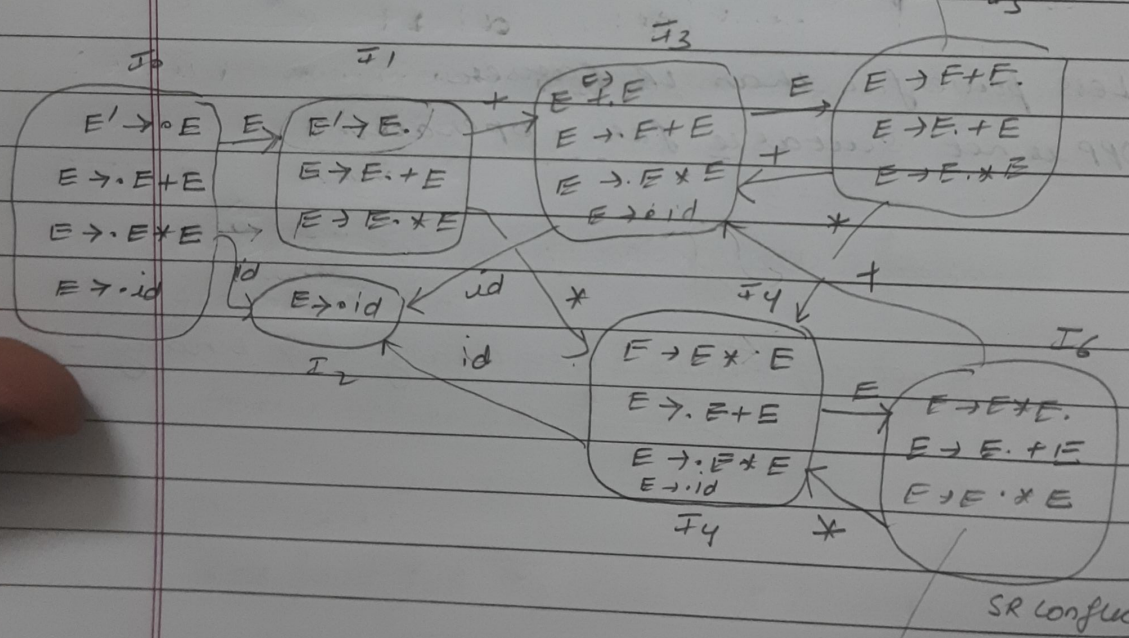
$E \to E+E / E+E / id$

Elimination of ambiguity.

Q. Resolve conflicts from the following ambiguous grammar.

① $E \to E+E$
② $E \to E*E$
③ $E \to id$

} Not LR(0), Not SLR(1), Not CLR(1), Not LALR(1)



Reduce और E → E.*E
शक साथ है। तो Shift करो

इसको देखो।
r_2 और shift actions है।
यदि Reduce और E → E.+E
Shift शक साथ आते है
तो * > + तो Reduce
का साथ दो

	action				goto
	id	+	*	$	E
I_0	S_2				1
I_1		S_3	S_4	accept	
I_2		r_3	r_3	r_3	
I_3	S_2				5
I_4	S_2				6
I_5		S_3/r_1	S_4/r_1	r_1	
I_6		S_3/r_2	S_4/r_2	r_2	

r_2 is maintained → r_2 → S_4 is maintained

Conflicts present in parsing table are resolved by using precedence and associativity of operators.

r_1 is maintained to maintain left associativity

in the stack

id	+	id	*	id	$

Further reduce this or shift (* id) to maintain higher precedence of * we make shift move

0
$

LR Parsing

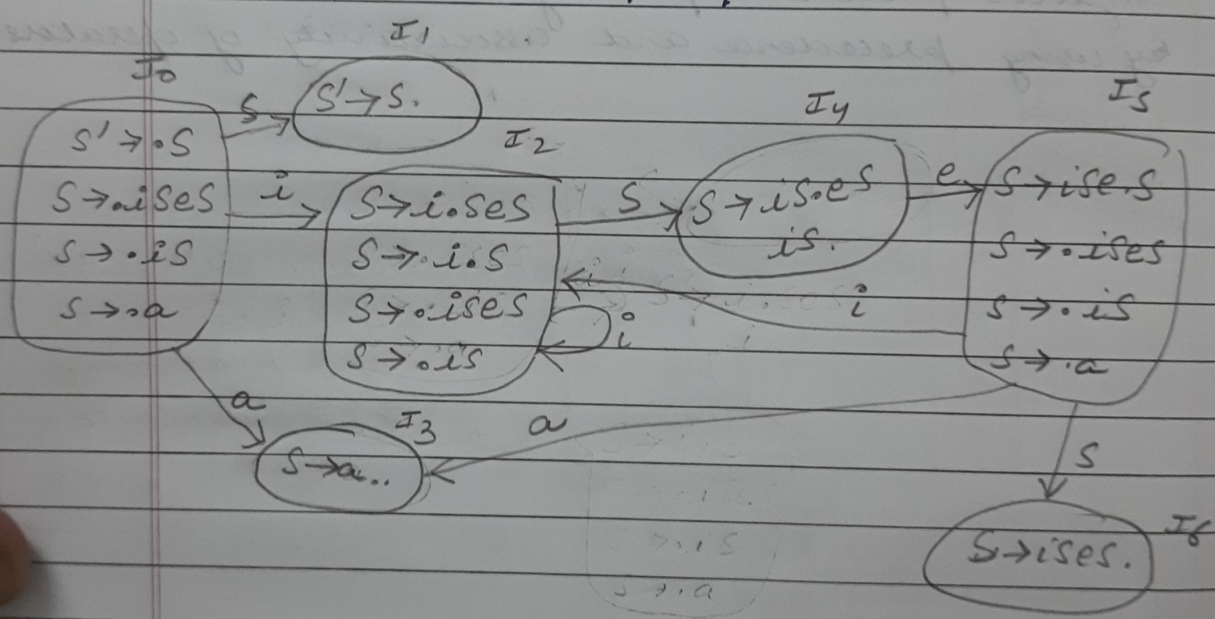
Q. $S \to AA$
$A \to aA / a$ } No operators.

How to resolve the conflicts?

Conflict	Conflict
S/R	R_1/R_2
give priority to shift	give priority to first reduce action.

Q. Resolve conflict for following ambiguous grammar

$$S \to \underset{①}{iSeS} \mid \underset{②}{iS} \mid \underset{③}{a}$$



	action				Goto
	e	i	a	$	S
I_0		S_2	S_3		1
I_1				accept	
I_2		S_2			4
I_3	r_3			r_3	
I_4	S_5/r_2			r_2	
I_5		S_2	S_3		6
I_6	r_1			r_1	

give priority to shift

04-08-2021

TOPDOWN PARSING

1. Topdown parser constructs parse tree starts from the root and proceeds towards leaf.

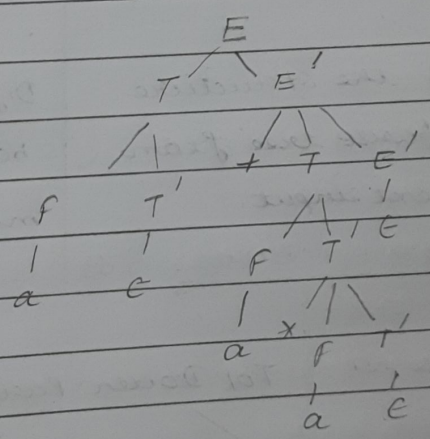
2. Topdown parser uses leftmost derivation.

NOTE:-
Bottomup parser is more powerful than topdown parser.

E.g ⇒

input ⇒ a+a*a

$E \to TE'$
$E' \to +TE' / \epsilon$
$T \to FT'$
$T' \to *FT' / \epsilon$
$F \to a$



Difference between Top-Down Parser and Bottomup Parsing

	Top-Down Parsing	Bottom-up Parsing
1.	Parse tree is constructed from root to leaves.	Parse tree is constructed from leaves to root.
2.	Traversal of parse trees is a preorder traversal	Traversal of parse tree is a postorder traversal.
3.	Tracing leftmost derivation.	Tracing rightmost derivation.
4.	Guess the structure of the parse tree from the next input.	Different structures and backtracks if it does not matched the input

Top Down Parsers

- With Full Backtracking
  - Bruteforce method
- without Backtracking
  - Recursive Descent
  - Non-Recursive Descent LL(1) ★★

1 Backtracking [BRUTE FORCE PARSING]

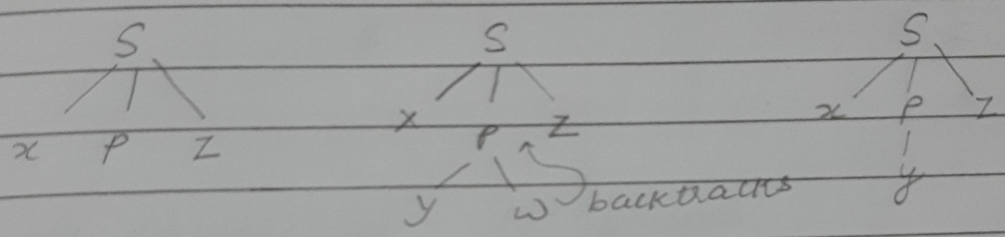
It is a technique in which for expansion of non-terminal symbol we choose one alternative and if some mismatch occurs then we try another alternative if any.

E.g -

$S \rightarrow xPz$

$P \rightarrow yw / y$

String xyz



Time Complexity $\Rightarrow O(2^n)$

Practically not suitable to design compiler

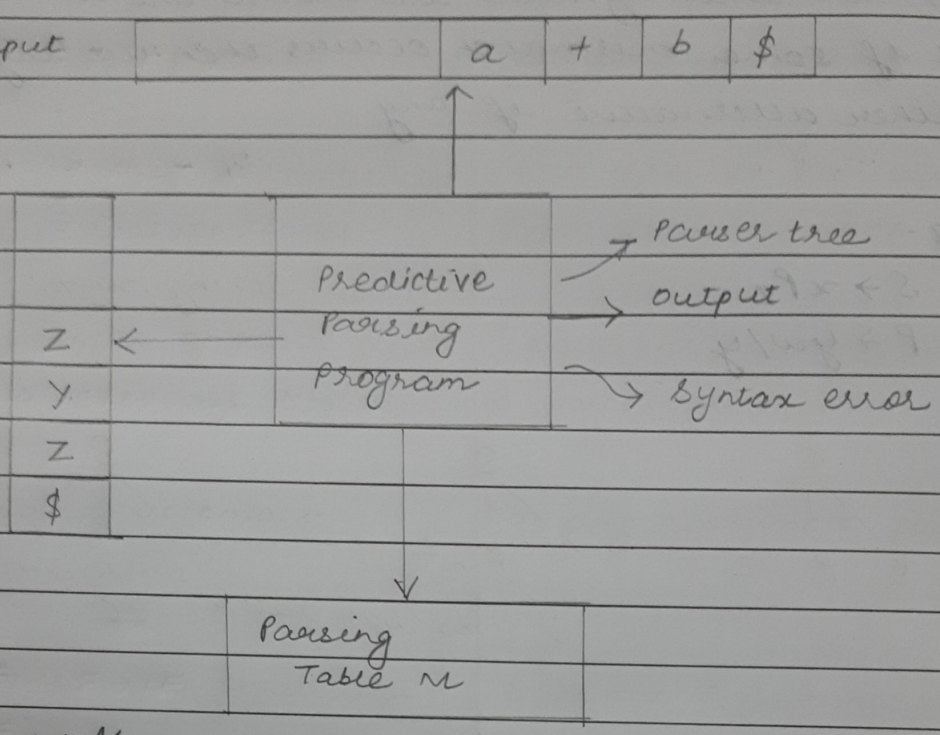
Most powerful parsing method, but time complexity is too high.

Top Down parser uses NPDA

★★★★ 2. LL(1) Parsing

→ LMD → Lookahead symbol 1

↳ Left to Right scan



Lookahead symbol

Algorithm – X = Symbol on top of stack

1. If X = a = $ Parsing success → input valid
2. If X = a ≠ $ Pop X from stack & Increment Lookahead
3. X is a Non-terminal on top of the stack
   T[X, a] = X → uvw { table देखो }
   X is replaced by uvw in reverse order in the stack.
4. T[X, a] = Blank } Syntax error

E.g

S → aBa

B → bB / ε

LL(1) Parsing table

	a	b	$
S	S → aBa		
B	B → ε	B → bB	

Stack	Input	Output
Starting symbol		
$S	abba $	S → aBa
$aBa	abba $	Pop a & Inc LA
$aB	bba $	B → bB
$aBb Reverse Order	bba $	Pop b & Inc LA
$ aB	ba $	B → bB
$ aBb	ba $	Pop b & Inc LA
$ aB	a $	B → ∈
$ a	a $	Pop a
$	$	Accept

Time complexity = O(n)

L-28

E.g.

$E \rightarrow TE'$

$E' \rightarrow +TE' / \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' / \epsilon$

$F \rightarrow (E) / id$

LL(1) Parsing table

NON T.	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Input :- id + id * id $

Page No.
Date : / /201

reverse

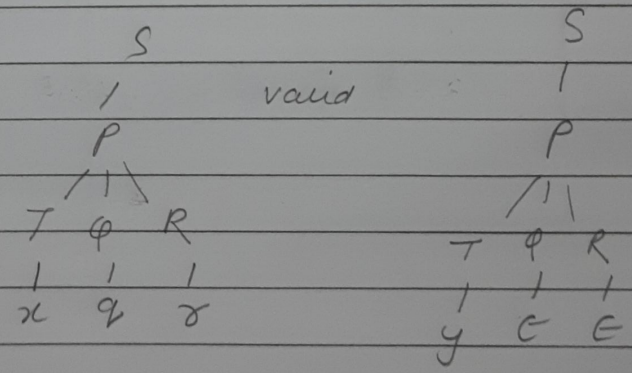
	Stack	Input	Output
1)	$ E	id + id * id $	E → TE'
2)	$ E' T	id + id * id $	T → FT'
3)	$ E' T' F	id + id * id $	F → id
4)	$ E' T' id	id + id * id $	pop id & incLA
5)	$ E' T'	+ id * id $	T' → ε
6)	$ E'	+ id * id $	E' → +TE'
7)	$ E' T +	+ id * id $	pop +
8)	$ E' T	id * id $	T → FT'
9)	$ E' T' F	id * id $	F → id
10)	$ E' T' id	id * id $	pop id
11)	$ E' T'	* id $	T' → *FT'
12)	$ E' T' F *	* id $	pop *
13)	$ E' T' F	id $	F → id
14)	$ E' T' id	id $	pop id
15)	$ E' T'	$	T' → ε
16)	$ E'	$	E' → ε
17)	$	$	accept

Q. Consider the following LL(1) Parsing table:

NonT	x	q	r	y	$
S	S→P	S→P	S→P	S→P	S→P
P	P→TQR	P→QR	P→QR	P→TQR	P→QR
Q		Q→q	Q→ε		Q→ε
R			R→r		R→ε
T	T→x			T→y	

Which of the following string generates syntax error by LL(1) parser with start symbol?

a) y
b) xqr
✓c) xyq
d) all of these



Construction of LL(1) Parsing Table

Algorithm

For each production rule $A \to \alpha$ of a grammar G

1) for each terminal a in First(α)
   add $A \to \alpha$ to $M[A, a]$

2) If ϵ in First(α)
   For each terminal a in follow(A)
   Add $A \to \alpha$ to $M[A, a]$

3) If ϵ in First(α) and \$ in Follow(A)
   add $A \to \alpha$ to $M[A, \$]$

Q. Construct LL(1) Parsing table for :-

	First	Follow
$S \to aABC$	a	\$
$A \to a \mid bb$	a, b	a, b, \$
$B \to a \mid \epsilon$	a, ϵ	b, \$
$C \to b \mid \epsilon$	b, ϵ	\$

NON T.	a	b	\$
S	$S \to aABC$		
A	$A \to a$	$A \to bb$	
B	$B \to a$	$B \to \epsilon$	$B \to \epsilon$
C		$C \to b$	$C \to \epsilon$

1) Row is decided by Non-Terminal

2) Column is decided by First of RHS

Eg $S \to (aAB) \Rightarrow a$

3) First (RHS) having ϵ then follow (LHS NT) is used to decide column.

Gate PYQ
2 marks

Q. Construct LL(1) Parsing table for -

	First	Follow
$S \to aAbB / bAaB / \epsilon$	a, b, ϵ	$\$, a, b$
$A \to S$	a, b, ϵ	a, b
$B \to S$	a, b, ϵ	$\$$

	a	b	$
S	$S \to aAbB$ ($S \to \epsilon$)	$S \to bAaB$ ($S \to \epsilon$)	$S \to \epsilon$
A	$A \to S$	$A \to S$	
B	$B \to S$	$B \to S$	$B \to S$

Not LL(1)

★ LL(1) Grammar

If parsing table contains multiple entries / productions in any box then that grammar is not LL(1).

		First	Follow
Q.	$S \to A$	b, c, ε, a, d	$
	$A \to BC / DBC$	b, c, ε, a, d	$
	$B \to Bb / \epsilon$	b, ε	b, c, $
	$C \to c / \epsilon$	c, ε	$
	$D \to a / d$	a, d	b, d

An LL(1) parsing table is constructed for the grammar as

	a	b	c	d	$
S	$S \to A$	$S \to A$	$S \to A$	$S \to A$	$S \to A$
A	$A \to DBC$	$A \to BC$	$A \to BC$	$A \to DBC$	$A \to BC$
B	$B \to Bb$	$B \to Bb$ $B \to \epsilon$	$B \to \epsilon$		$B \to \epsilon$
C			$C \to c$		$C \to \epsilon$
D	$D \to a$			$D \to d$	

Let 'x', 'y', 'z' & 'u' represent the no. of entries in rows of non terminals S, A, B & C, then $x+y+z+u$ is 16.

LL(1) Parsing table = Predictive parsing table

Page No.
Date : / /201

05-08-2021

Q. Consider the grammar shown, below

Gate PYQ 2 marks

$S \to iEtSS' / a$

$S' \to eS / \epsilon$

$E \to b$

In the LL(1) parsing table M, of this grammar, the entries M[S', e] and M[S', $] respectively are

a) {S' → eS} and {S' → ε}

b) {S' → eS} and { }

c) {S' → ε} and {S' → ε}

✓ d) {S' → eS, S' → ε} and {S' → ε}

	e	$
S'	S' → eS, S → ε	S → ε

Follow(S') = Follow(S) ⇒ {e, $}

Not LL(1) grammar

Gate PYQ 2 marks

Q. Consider the following grammar

$S \to FR$

$R \to *S / \epsilon$

$F \to id$

Follow{R} ⇒ Follow(S) ⇒ $

In the Predictive parser table, M, of the grammar the entries M[S, id] and M[R, $] resp.

✓ a) {S → FR} and {R → ε}

b) {S → FR} and { }

c) {S → FR} and {R → *S}

d) {F → id} and {R → ε}

Page No.
Date : / /201

Q. $S \to aAbB / bAaB / \epsilon$

$A \to S$

$B \to S$

FOLLOW(S) ⇒ FOLLOW(A) ∪ FOLLOW(B)

a, b, $

	a	b	$
S	$S \to aAbB$ $S \to \epsilon$	$S \to bAaB$ $S \to \epsilon$	$S \to \epsilon$
A	$A \to S$	$A \to S$	Error
B	$B \to S$	$B \to S$	$B \to S$

LL(1) grammar Detection

* 1. Single production grammar is LL(1)

* 2. $A \to \alpha_1 / \alpha_2 / \alpha_3 / \cdots / \alpha_n$

$First(\alpha_1) \cap First(\alpha_2) \cap \cdots \cap First(\alpha_n) = \phi$

Then LL(1)

* 3. $A \to \alpha / \epsilon$

$First(\alpha) \cap Follow(A) = \phi$

then LL(1)

* 4. Left recursive grammars are not LL(1)

* 5. Ambiguous grammars are not LL(1)

* 6. Common prefix grammar not LL(1)

solution = Left factoring

Eg

$$\left.\begin{array}{l} S \to AB \\ A \to a \\ B \to a \end{array}\right\}$$ single production

Hence, LL(1)

Q. The grammar $S \to aSa / bS / c$ is

$First(aSa) \cap First(bS) \cap First(c)$

$\{a\} \cap \{b\} \cap \{c\} = \phi$

$\therefore$ LL(1).

Q. $S \to FR$

$R \to *S / \epsilon \to \{\{*\} \cap Follow(R)\}$

$F \to id$ $\quad \{*\} \cap \{\$\} = \phi$

$\therefore$ LL(1)

Q. $S \to iEtSS' / \alpha$

$S' \to eS / \epsilon \to \{e\} \cap Follow\{S'\} = Follow(S) = First\{S'\}$

$E \to b$

$\{e\} \cap \{\$, e\} \neq \phi$

$\therefore$ not LL(1)

Q. Which of the following grammars is LL(1)?

a) $S \to aSbS / bSaS / \epsilon$

b) $S \to aSa / bSb / \epsilon$

✓c) $S \to AaAb / BbBa$ $\quad \{a\} \cap \{b\} = \phi$

$A \to \epsilon$ $\quad$ Follow(A)

$B \to \epsilon$ $\quad$ Follow(B)

d) None

Page No.
Date : / /201

Q. Which of the following is LL(1)?

a) $S \to SS/a$ Ambiguous grammar
$A \to aA/a$

b) $E \to E+T/T$
$T \to a$ Left Recursive

c) $S \to ab/abc$

✓ d) None

L-30

Q. Which of the following is LL(1) grammar?

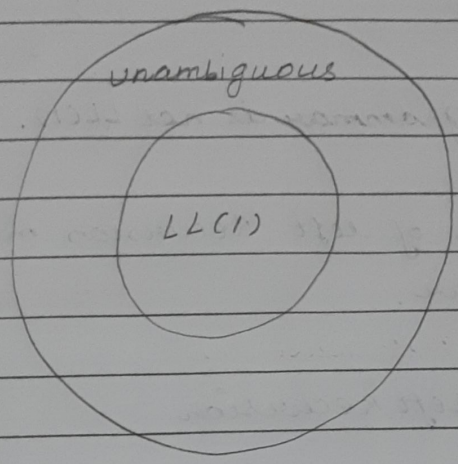
a) $S \to SS/a$ Left Recursion

b) $E \to E+E/E*E/a$ Ambiguous

c) $S \to iEtS/iEtSeS/a$ common prefix

✓ d) None

Note:-



If a grammar is LL(1) it is definitely unambiguous

If a grammar is not LL(1) then it may or may not be unambiguous

Q. Which of the following grammars is Ambiguous?

a) $S \to aSa / bSb / C$ LL(1)

b) $S \to aSbS / bSaS / c$ LL(1)

c) $S \to AaAb / BbBa$ LL(1)
$A \to \epsilon$
$B \to \epsilon$

✓ d) None

If LL(1) then unambiguous

Eliminating Left Recursion is Decidable problem

Page No.
Date : / /201

Left Recursion

* Left Recursive grammar is not LL(1).

* But elimination of left Recursion may result in LL(1) grammar.

** Elimination of Left Recursion

Recursive Grammar

$S \to Sa/b$	$S \to aS/b$	$S \to aSb/b$
Left Recursion	Right Recursion	Middle Recursion
NOT LL(1)	LL(1)	LL(1)

(i) $S \to Sa/b$

$S \to bS'$ = bS^* $\Rightarrow \left\{ \begin{array}{l} S \to bS'/b \\ S' \to aS'/a \end{array} \right\}$

$S' \to aS'/\epsilon$

(ii) $A \to A\alpha/\gamma$

$A \to \gamma A'$

$A' \to \alpha A'/\epsilon$

(iii) $E \to E+T / T$
$T \to T*F / F$
$F \to a$

$E \to TE'$
$E' \to +TE' / \epsilon$
$T \to FT'$
$T' \to *FT' / \epsilon$
$F \to a$

(iv) $A \to A\alpha_1 / A\alpha_2 / \dots / A\alpha_2 / \beta_1 / \beta_2 / \dots / \beta_n$

$A \to \beta A' / \beta_2 A' / \dots / \beta_n A'$
$A' \to \alpha_1 A' / \alpha_2 A' / \dots / \alpha_n A' / \epsilon$

(v) $S \to Sa / Sb / c / d / e$

$S \to cS' / dS' / eS'$
$S' \to aS' / bS' / \epsilon$

Elimination of ϵ productions

$S \to aS_1 b$ $\quad$ $S \to aS_1 b / ab$
$S_1 \to aS_1 b / \epsilon$ $\quad$ $S_1 \to aS_1 b / ab$

Q. Consider the following expression grammar G.

$E \rightarrow E-T / T$
$T \rightarrow T+F / F$
$F \rightarrow (E) / id$

which of the following grammars is not left recursive, but is equivalent to G ?

$E \rightarrow TX$
$X \rightarrow -TX / \epsilon$
$T \rightarrow FY$
$Y \rightarrow +F / \epsilon$
$F \rightarrow (E) / id$

Left Recursion

Direct Left Recursion

$A \rightarrow A\alpha / X$

Not LL(1)

Indirect Left Recursion

$A \rightarrow Ba / b$
$B \rightarrow Ab / a$

Not LL(1)

PYQ

Q. Which of the following grammars is not left recursive ?

a) $S \to AB$
$A \to Aa/b$
$B \to c$
Direct Left Recursion

b) $S \to Ab/Bb/c$
$A \to Bd/a$
$B \to e$

c) $S \to Aa/B$
$A \to Bb/Sa/b$
$B \to d$
Indirect Left Recursion

d) $S \to Aa/Bb/c$
$A \to Bd/a$
$B \to Ab/a$
Indirect left Recursion

Elimation of Indirect Left Recursion

$A \to Ba/b$
$B \to Ab/a$

1) $A \to Ba/b$

2) $B \to Ab/a$
$B \to Bab/bb/a$

$B \to bbB'/aB'$
$B' \to abB'/\epsilon$

$\Rightarrow$ $A \to Ba/b$
$B \to bbB'/aB'$
$B' \to abB'/\epsilon$

Q. Eliminate Left Recursion from the given grammar

$S \to Aa / b$
$A \to Ac / Sd / a$

Both Direct & Indirect Left Recursion is present

$S \to Aa / b$ इसको वैसे ही रखेंगे।

$A \to Ac / Aad / bd / a$ इसमें replace करो।

$A \to bdA' / aA'$
$A' \to cA' / adA'$

$\Rightarrow$ $S \to Aa / b$
$A' \to cA' / adA'$
$A \to bdA' / aA'$

After removing Left Recursion a grammar need not necessarily be LL(1).

Left Factoring

$S \to ab / ac$ common Prefix

$S \to aS'$
$S' \to b / c$

Common prefix grammar are NOT LL(1).
But left factoring of the grammar may result into LL(1) grammar.

Gate PYQ
Q. Which of the following suffices to convert an arbitrary CFG to an LL(1) grammar?

a) Removing left recursion alone
b) Removing left recursion & left factoring both.
c) Factoring the grammar alone
d) None.

Q. Left factor the grammar

$S \to iEtS / iEtSeS / a$
$E \to b$

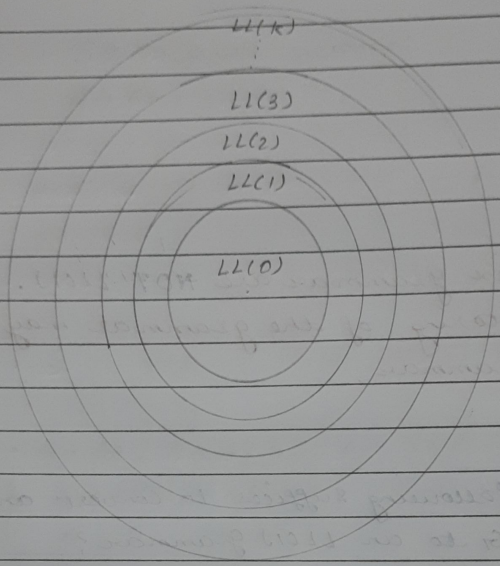
$S \to iS'$
$S' \to ES''$
$S'' \to tS'''$
$S''' \to ST$
$T \to eS / a$
$E \to b$

X

$S \to iEtSS' / a$
$S' \to eS / \epsilon$
$E \to b$

Not LL(1)

07-08-2021



Power :-

LL(k) > LL(3) > LL(2) > LL(1) > LL(0)

All are unambiguous

Q Which of the following is true ?

a) Every LL(2) is LL(1)
b) Every LL(3) is LL(2)
c) Every LL(k) is LL(k-1)
✓ d) None

Page No.
Date : / /201

Note!-

Predictive Parser is not suitable for

1. Left Recursive Grammar
2. Common Prefix Grammar

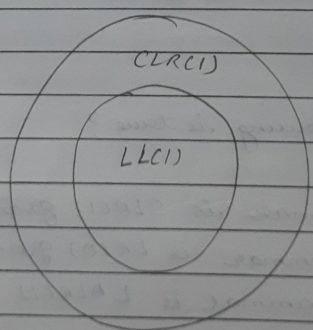
Solution!-

a) Removal of left Recursion
b) Left Factoring

} May or may not be LL(1)

Q. The grammar $S \rightarrow aSa / bS / c$ is

a) LL(1) but not LR(1)
b) LR(1) but not LL(1)
c) Both LL(1) and LR(1)
d) Neither LL(1) nor LR(1)



Every LL(1) grammar is CLR(1).

Q. Which of the following is not CLR(1)?

a) $S \to (S) / a$ LL(1)

b) $S \to CC$ LL(1)
$C \to aC / d$

c) $S \to aSbS / bSaS / c$ LL(1)

✓ d) None

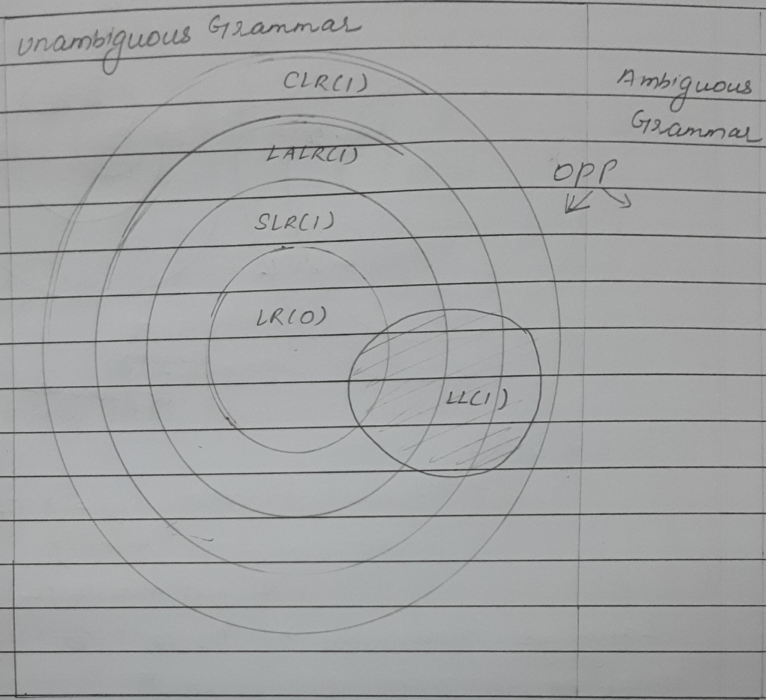
If a grammar is not LL(1) we will have to construct CLR(1) DFA to check.

Q. Which of the following is true?

a) Every SLR(1) grammar is LL(1) grammar
b) Every LALR(1) grammar is LL(1) grammar
c) Every LL(1) grammar is ~~CLR(1)~~ LALR(1) grammar
✓ d) None

Q. Which of the following is true?

a) Every LL(1) grammar is SLR(1) grammar
b) Every LL(1) grammar is LR(0) grammar
c) Every LL(1) grammar is LALR(1) grammar
d) All of the above
✓ e) None of the above.



Gate PYQ 2marks

Q. Consider the following two statements

P: Every regular grammar is LL(1)

Q: Every regular ~~grammar~~ set has LR(1) grammar.

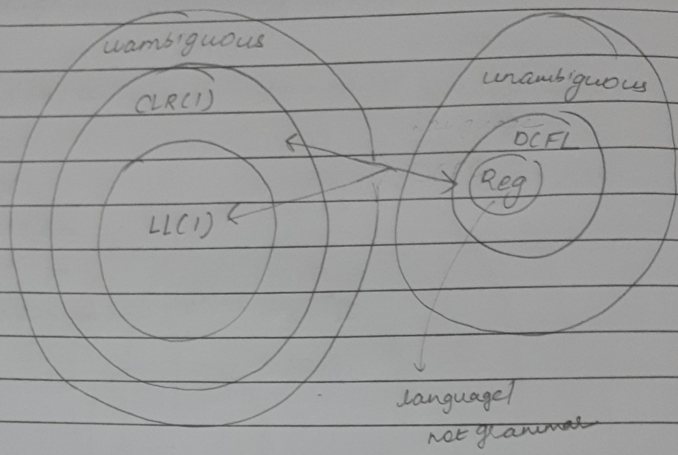
Which of the following is true?

a) Both P & Q are true

b) P is true and Q is false

✓ c) P is false and Q is true

d) Both are False



Parser → DPDA
(DCFL)

Every reg = DCFL

Language Unambiguous → { G_1, G_2, ... G_3 ... }
↓
Unambiguous Grammar (atleast one)

कोई से कोई एक unambiguous grammar exist करती है language unambiguous

Atleast one U.G should exist

For every DCFL } → LR(1) more powerful exists
LL(1)

1) For every DCFL LR(1) exist but LL(1) may or may not exist
2) For every regular set LR(1) exist LL(1) may or may not exist.

An unambiguous grammar has different leftmost & rightmost derivatives

Page No.
Date : / /201

Example :-

$S \to aS / a$

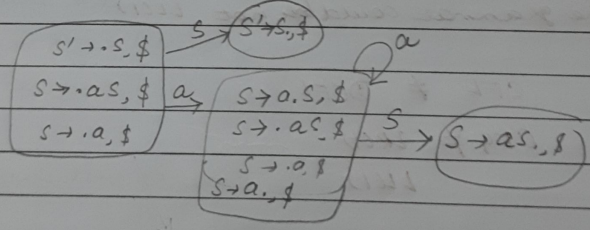
Regular but not LL(1)

Every regular set = DCFL = LR(1)

Every DCFL produces CLR(1)

For Regular language & DCFLs many grammars may exist but atleast one unambiguous grammar must exist

$S \to aS / a$



Shift move not present in look ahead of reduce move no conflict, hence CLR(1)

An ambiguous grammar can never be LR(K) for any value of K.

Note:-

LR parser and Operator Precedence parser both are S/R parsers.

Bottom up parsers are used RMD in reverse order.

CFL → DCFL, CFL

CFG ⇒ CFL

if → LL(1) = DCFL grammar

but

$S \to aSa / bSb / \epsilon$

$L = \{ ww^R \mid w \in (a, b)^* \}$

CFL, but not DCFL

After removing left Recursion & doing Left factoring, grammar maybe CFG and CFL but not DCFL. Since LL(1) accepts only DCFLs so the grammar could not LL(1).

CFL ≠ DCFL

CFG ≠ LR(1)

LL(1)

For this reason even after removing left Recursion & Left factoring the grammar still not be LL(1).

08-08-21

Drawback of Parsers

1. Declaration statement — int a, b, c;
2. Expression statement — a = b + c * d;
3. I/O statement — printf(x);
4. Flow control statement — for, while, if-else
5. Functions — sum(a, b);

Q Which of the following C programs has syntax error?

a)
```
main()
{
 int a, b, a, b;
}
```
Compile time error

b)
```
main()
{
 int a, b;
  c = a + b;
}
```
Compile time error

c)
```
main()
{
 int a, b;
 char c[] = "Hello";
  a = b + c;
}
```
Compile time error

d) None

If parse tree can be generated, then no syntax error.

Here compile time errors are semantic errors.

Page No.
Date : / /201

Following are some of the semantic errors not detected by Parser.

1. Using a variable without declaration.
2. Incompatible types in expression statement.
3. Multiple times declaration of same variable in same scope.
4. Array index out of bound.
5. Division by 0.
6. Mismatch of formal & actual parameters.
7. Type mismatch of formal and actual parameters.

SEMANTIC ANALYSIS – (Type Checker)

(Context sensitive Analysis)

Mathematical Model used –

1. Context Sensitive Grammar
2. Linear bound Automata.

- Structure has already been verified by syntax analysis.
- Semantic Analysis phase checks/verifies meaning of the statement.

Attribute Grammars (CSG Simplest Form)

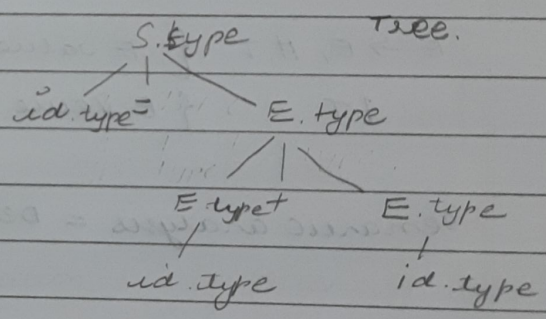
attribute = type.

E.g

$S \to id = E$
$E \to E + E$
$E \to id$

} CFG

Annotated Parse Tree.



CFG + attribute information is called attribute grammar.

Eg

attribute grammar.

$S \to id = E.type$
type type

$E.type \to E.type + E.type$
$E.type \to id.type$

} CSG

Page No.
Date : / /201

Necessary for semantic error checking

Syntax Directed Translation (SDT)
(bring attributes to the grammar)

SDT is a mathematical model where each CFG production is associated with set of semantic rules.

SDT = CFG + semantic rules → calculate attributes

Semantic rules are attribute calculation rules in the annotated parse tree.

E.g

E.g of SDTs

$$\begin{cases} E \rightarrow E_1 * T & \{ E.value = E_1.value * T.value \} \\ \quad | \; T & \{ E.value = T.value \} \end{cases}$$

semantic analysis = Designing of SDT

★ Application of SDT

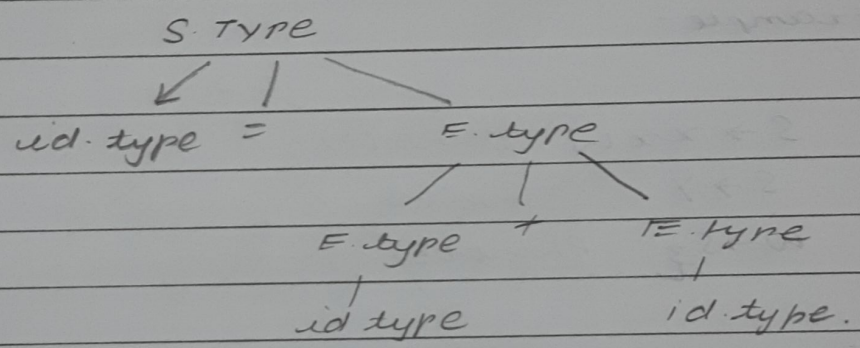
1. Detection of semantic error
2. Construction of symbol table
3. For type casting (Implicit)
4. Generation of Intermediate Code
5. Construction of Syntax tree

Complete construction of symbol table happens in Semantic Analysis phase.

Semantic Errors -

1. Type mismatch in expressions
2. Multiple declaration.
3. Undeclared variable
4. Array index Out of Bound
5. Division by zero

Annotated Parse tree - Parse tree with attribute information.



	Rule Representation	Accepting machine
1. Lexical Analysis →	Regular Expression →	Finite Automata
2. Syntax Analysis →	CFG →	PDA (parser)
3. Semantic Analysis →	SDT →	Parser (LCA) 2 stacks 1 For syntax other for attributes. [Same parser with 2 stacks]

Points to Remember

1. Design SDT (Making Semantic Rules)
2. Getting Output from SDT

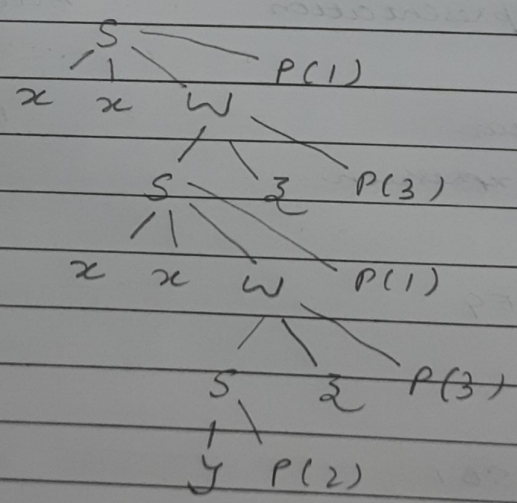
Note -

If SDT is given to Bottom up Parser for reduce action the semantic rule attached to the production is executed.

Example -

$S \rightarrow xxW$ Print(1)
$S \rightarrow y$ Print(2)
$W \rightarrow Sz$ Print(3)

Input = $xxxxyzz$



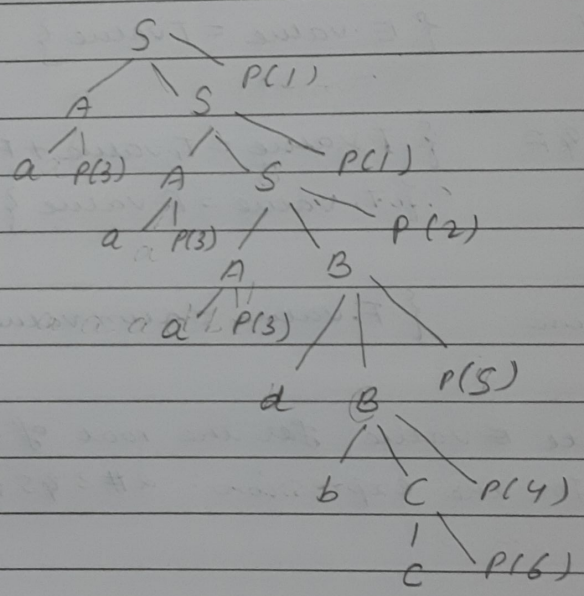
S
x x W
x x S z
x x x x W z
x x x x S z z
x x x x y z z

output → 23131

Page No.
Date : / /201

Q. $S \to AS$ {Print (1)}
$S \to AB$ {Print (2)}
$A \to a$ {Print (3)}
$B \to bC$ {Print (4)}
$B \to dB$ {Print (5)}
$C \to c$ {Print (6)}

This SDT is carried out by BUP then what is the output for the input ⇒ aaadbc



Output ⇒ 3 3 3 6 4 5 2 1 1

1	S
2	AS
5	AAS
4	AAAB
6	AAAdbC
3	AAAdbc
3	AAadbc
3	Aaadbc
	aaadbc

Evaluation of values from SDT

1. Construct parse tree for the given input
2. Construct annotated parse tree
3. Complete attribute information in annotated parse tree.

Q. Consider the grammar with the following translation rules & E as the start symbol.

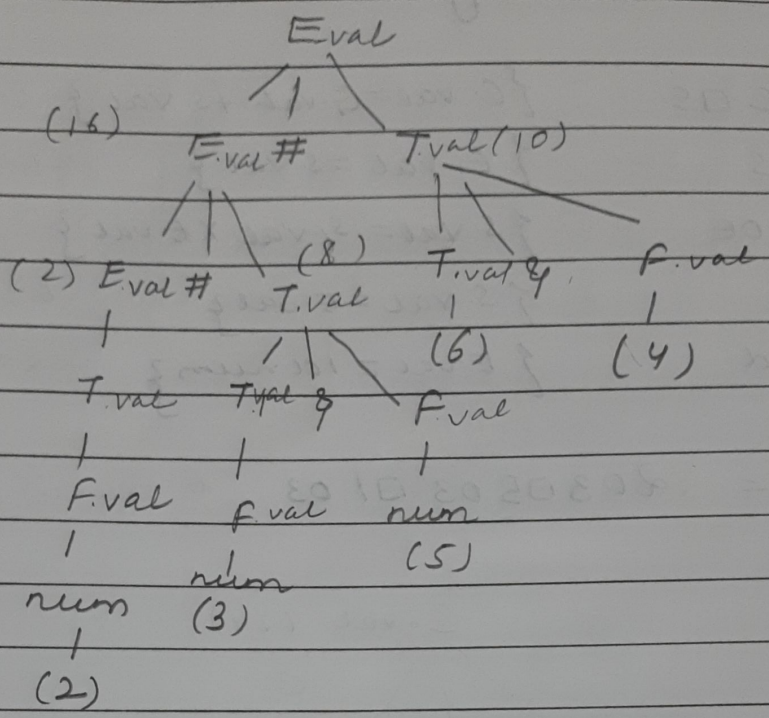
$E \rightarrow E_1 \# T$ { E.value = E_1.value * T.value }
$\quad / T$ { E.value = T.value }

$T \rightarrow T_1 \& F$ { T.value = T_1.value + F.value }
$\quad / F$ { T.value = F.value }

$F \rightarrow num$ { F.value = num.value }

Compute E.value for the root of the parse tree for the expression: 2#3&5#6&4

a) 200
b) 180
✓ c) 160
d) 40



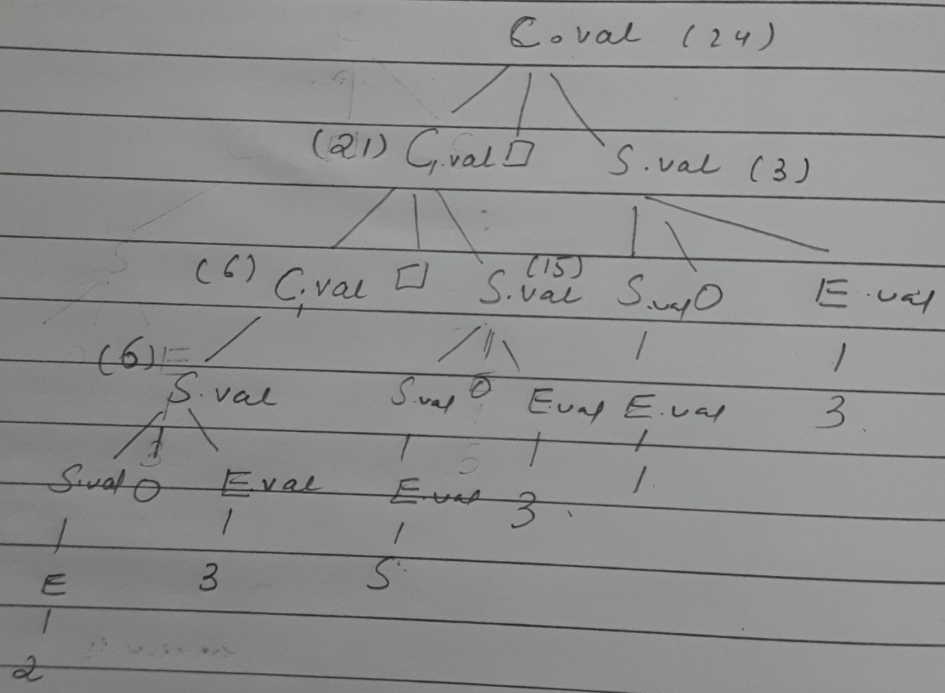
$\Rightarrow 16 \times 10 = 160$

Page No.
Date : / /20

Q Consider the following SDT.

$C \to C \square S$	$\{C.val = C_1.val + S.val\}$
$C \to S$	$\{C.val = S.val\}$
$S \to S \circ E$	$\{S.val = S_1.val \times E.val\}$
$S \to E$	$\{S.val = E.val\}$
$E \to id$	$\{E.val = id.num\}$

Input = 2 0 3 □ 5 0 3 □ 1 0 3

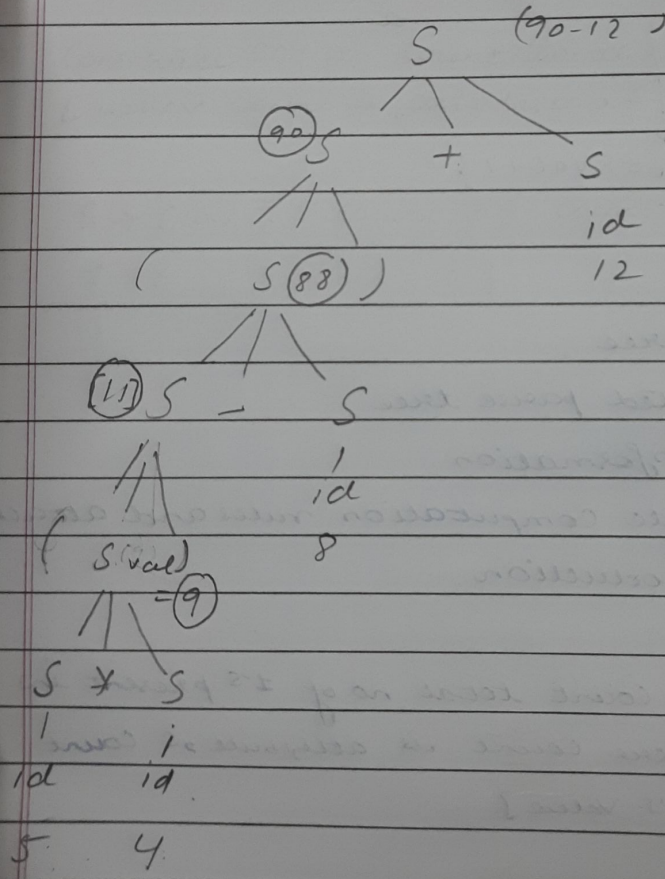


⇒ 24

Q. Consider the following SDT:

$S \to S + S$ $\{S.val = S_1.val - S_2.val\}$
$S \to S * S$ $\{S.val = S_1.val + S_2.val\}$
$S \to S - S$ $\{S.val = S_1.val * S_2.val\}$
$S \to (S)$ $\{S.val = S_1.val + 2\}$
$S \to id$ $\{S.val = id\}$.

Input ⇒ ((5*4)-8)+12



⇒ 78 answer

Venkata Rao Sir 9364444446

09-08-2021

Page No.
Date : / /201

Construction of SDT

Q. Construct SDT that computes total no. of reduce actions taken by bottom-up parser [val is attribute, E.val gives total no. of reduction value]

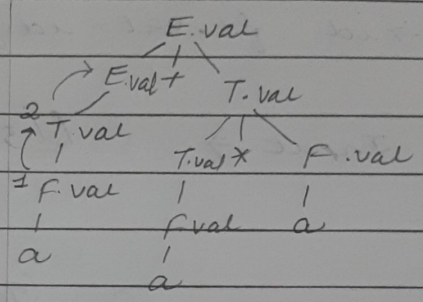
$E \to E + T$ {E.val = E1.val + T.val + 1}

$E \to T$ {E.val = T.val + 1}

$T \to T * F$ {T.val = T.val + F.val + 1}

$T \to F$ {T.val = 2} or

$F \to a$ {F.val = 1}



{T.val = F.val + 1}

Rules:-

1. Construct parse tree
2. Construct annotated parse tree
3. Compute attribute information
4. Generalize attribute computation rules and attach to grammar production.

Q. Construct SDT to count total no. of 1's present in binary string [where val count is attribute, S.count gives total no. of 1's value].

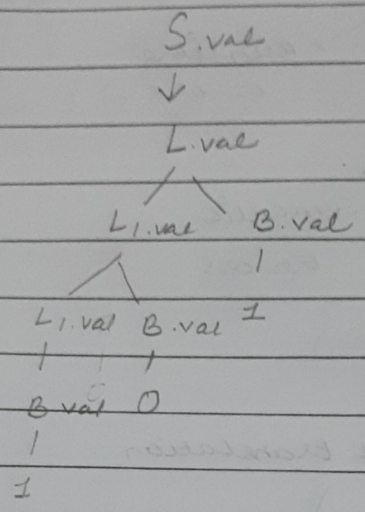
$S \to L$ {S.val = L.val}

$L \to LB$ {L.val = L.val + B.val}

$L \to B$ {L.val = B.val}

$B \to 0$ {B.val = 0}

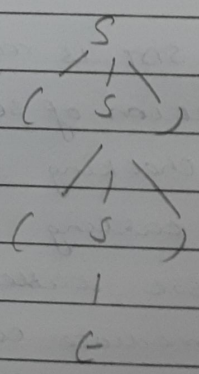
$B \to 1$ {B.val = 1}



Q. Construct SDT to count total no. of balanced paranthesis [where count is attribute, S.count gives total value]

$S \rightarrow (S) \{ S.count = S_1.count + 1 \}$

$S \rightarrow \epsilon \{ S.count = 0 \}$



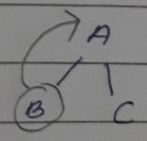
Errors → Runtime Errors
Errors → Compile time
Compile time → Lexical phase Errors
Compile time → Syntax Errors
Compile time → Semantic Errors

Syntax Directed translation
→ Bottom up parsing
→ Top-down parsing
Bottom up parsing → LR parsing
Bottom up parsing → Operator Precedence Parsing
Top-down parsing → LL(1)
Top-down parsing → Recursive Descent

Why SDT is required?

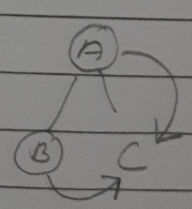
1. Detection of semantic errors
2. Type checking
3. Type casting
4. Symbol table construction
5. Intermediate code.

★ SDT uses Attributes
→ Synthesized attributes
→ Inherited attributed



Parent ⇐ Children

A → BC



computed from Parent or Left sibling

Invalid (R to L)

Page No.
Date : / /201

Q. $S \to TL$ { T.type = L.type } Neither synthesized nor inherited

$T \to int$ { T.type = int } Synthesized

$L \to L_1, id$ { L_1.type = L.type } Inherited

$L \to id$

{ L.Type = T.Type }

Inherited

Inherited

Q. $S \to ABC$ { C.x = B.x }

$B \to CA$ { B.x = A.x } Synthesized

$A \to a$ { "

$B \to b$ { B.x = b.y } Synthesized

$C \to d$

L attributed definition

** Type of attributes

S-attributed Definition

Using only synthesized attributes

L-attributed Definition

Using both synthesized and inherited attributes

Q. $S \to ABC$ $\{S.x = B.x\}$ Synthesized
$A \to BC$ $\{A.x = C.x\}$ Synthesized

S attributed Definition

Q. $S \to AB$ $\{S.x = B.x\}$
$A \to BC$ $\{C.x = B.y\}$

L-attributed Definition

Q. $S \to AB$ $\{B.x = A.x\}$ Inherited
$A \to BC$ $\{C.x = B.x\}$ Inherited

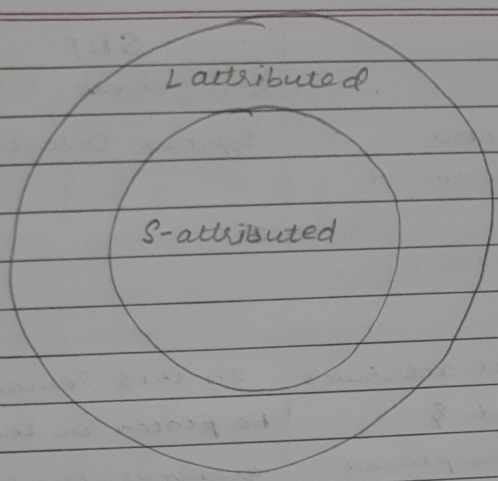
Since synthesized attributes not present
∴ Neither L nor S-attributed
Definition.

Invalid

Q. $S \to AB$ $\{S.x = A.x\}$ Synthesized
$B \to AC$ $\{A.y = C.y\}$ Invalid

Invalid SDT

10-08-2021



Q. $S \rightarrow AB \{ S.x = A.x \}$ Synthesized
$A \rightarrow BC \{ A.x = B.x \}$ Synthesized

a) S-attributed
b) L-attributed
✓c) Both
d) None

Because every S-attributed is L-attributed.

L attributes

L & S
(only synthesized)

L but not S
(synthesized & inherited both)

SDD	SDT (or Translation scheme.)
Syntax Directed definition. → S-attributed, L-attributed	Syntax Directed translation.
SDD in which all attributes (Both Synthesized & Inherited) are placed in right most end of grammar production.	In this Semantic rules can be placed in the middle grammar symbols.
E.g	E.g
$R \rightarrow +T\{Printf'+'\}$	$R \rightarrow +T\{Printf(+)\}R \mid \epsilon$

Translation scheme -

1. Semantic rules of synthesized attributes are placed at right most end of grammar production.

2. Semantic rules of Inherited attributes are placed in between the grammar symbols.

$$A \rightarrow a\{\underbrace{\text{Semantic Rule}}_{\text{Inherited}}\}\, b\{\underbrace{\text{Semantic Rules}}_{\text{Synthesise}}\}$$

E.g.

SDD

Synthesized

$S \to AB \quad \{ S.x = A.x \}$

$A \to BC \quad \{ C.x = B.x \}$

Inherited

2. Translation scheme

$S \to AB \{ S.x = A.x \}$

$A \to B \{ C.x = B.x \} C$

Q. $S \to AB \{ S.x = A.x \}$ Synthesised

$A \to BC \{ A.y = B.y \}$ Synthesized

Translation scheme is same as above SDD.

General Structure -

1. Contruct SDD
2. Translation scheme

Evaluation of Semantic rules from Translation scheme

1. Construct Parse tree
2. Construct edge for every semantic rule by assuming it as grammar symbol.
3. Performs DFS traversal.

Q. Consider the translation scheme shown below

$S \rightarrow TR$

$R \rightarrow +T \{ printf('+'); \} R \mid \epsilon$

$T \rightarrow num \{ printf(num.val); \}$

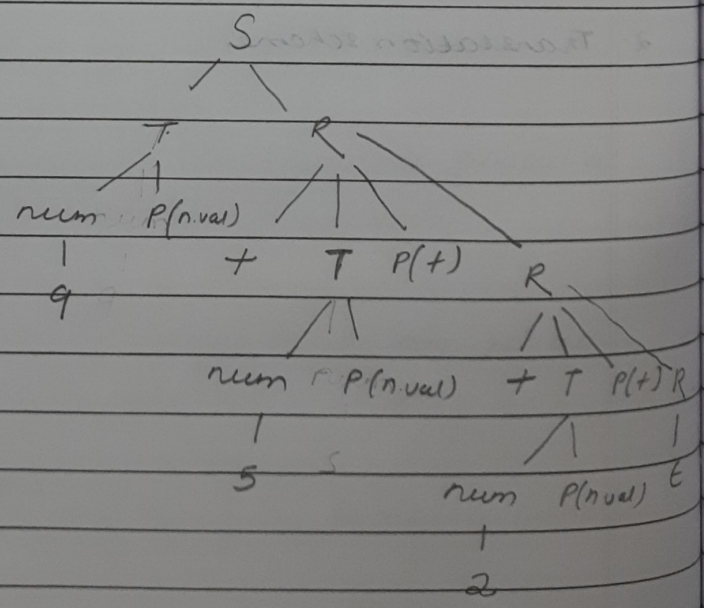
Here num is a token that represents an integer and num.val represents the corresponding integer value. For an input string '9+5+2', this scheme will output.

a) 9+5+2

✓ b) 95+2+

c) 952++

d) ++952



DFS ⇒

95+2+

Q. Consider the following translation scheme

$S \to ER$

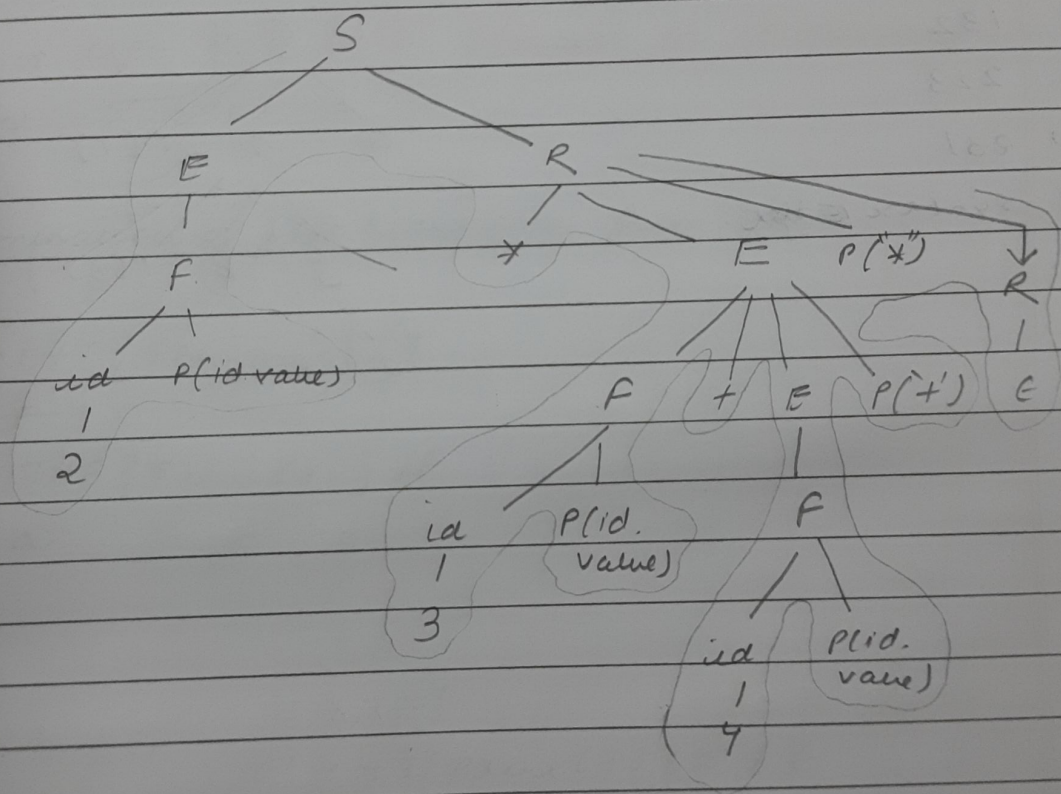
$R \to *E\{printf('*')\}; R \mid \epsilon$

$E \to F + E\{printf('+');\} \mid F$

$F \to (S) \mid id\{print(id.value);\}$

I/P ⇒ $2 * 3 + 4$

This translation scheme prints



DFS ⇒

$234 + *$

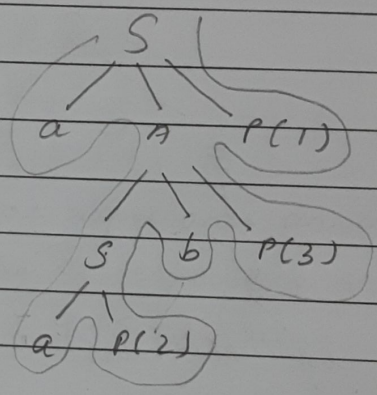
GATE 2016

Consider the following SDT scheme with non-termi. {S, A} and terminals {a, b}.

$S \to aA$ {printf(1)}
$S \to a$ {printf(2)}
$A \to Sb$ {printf(3)}

Using the above SDTS, the o/p printed by a bottomup parser, for the input aab is

a) 132
b) 223
✓ c) 231
d) Syntax Error.



L-38

Translation Scheme with Parsing

Top Down Parsing (Parser) (LL(1))
(2 stacks)

If translation scheme carried out with Topdown parser then semantic rules are evaluated by using DFS traversal from Left to Right manner.

If translation schemes contains left Recursion then it ~~cannot be~~ isn't possible to carry out with Topdown Parser.
Hence left Recursion should be removed from Translation scheme.

Elimination of Left Recursion from Translation Scheme

$E \to E+T\{Print(+)\}/T$

~~$E \to T$~~ NOT Possible for LL(1)

$T \to T*F\{Print(*)\}/F$

$F \to a$

$$\left.\begin{array}{l} E \to TE' \\ E' \to +T\{Print(+)\}E'/\epsilon \\ T \to FT' \\ T \to *F\{Print(*)\}T'/\epsilon \\ F \to a \end{array}\right\} \text{LL(1) Possible}$$

Page No.
Date : / /201

Q. $E \rightarrow TE'$

$E' \rightarrow +T \{print(+)\} E' \mid \epsilon$

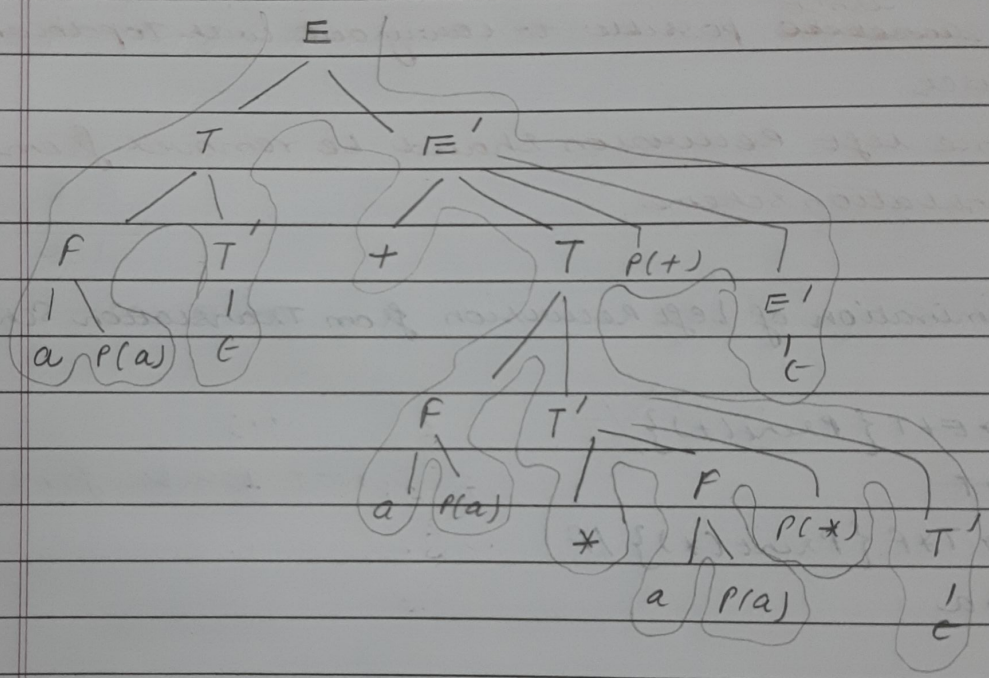
$T \rightarrow FT'$

$T' \rightarrow *F \{print(*)\} T' \mid \epsilon$

$F \rightarrow a \{print(id.val)\}$

what O/P it prints for input

$a + a * a$



DFS ⇒ aaa*+

Translation Scheme with Bottomup Parser

NOTE:-

If translation scheme is carried out with Bottom up parser for every reduce action corresponding semantic rule of the production is executed.

E.g

$S \to aA$ {Print 1}
$S \to a$ {Print 2}
$A \to Sb$ {Print 3}

I/P ⇒ aab

S	P(1)
a A	P(3)
a Sb	P(2)
a a b	

If rules are in the middle, handle detection cannot be performed.

Note -

If translation scheme directly given to B.U.P then handle detection problem arises. Hence, we have to modify translation scheme before giving to Bottom up parser

Modification example -

Not suitable for B.U.P

$S \to TR$
$R \to +T\{print('+');\}R \mid \epsilon$
$T \to num\{print(num.val);\}$

Suitable for B.U.P

$S \to TR$
$R \to +TMR \mid \epsilon$
$M \to \epsilon\{print('+')\}$
$T \to num\{print(num.val);\}$

Note:-

If translation scheme is given to Top down parser (or) Bottom up parser output is same for the given input

Drawback of Semantic Analysis

- Static Semantic Error
  - SDT
- Dynamic Semantic Errors / Exception / Runtime error
  - Semantic Analysis Can't detect
  - { Not part of the Computer }

11-08-2021

Q. Consider the following SDT.

$A \rightarrow A * B \{ A.val = A_1.val * B.val \}$ Synthesised
$A \rightarrow A + B \{ A.val = A_1.val + B.val \}$ Synthesised
$A \rightarrow id \{ A.val = id.num \}$ Synthesised

The given SDT is
a) L-attributed but not S-attributed
b) Only S-attributed
✓c) Both S-attributed & L-attributed
d) None.

Q. Consider the following SDT :

$A \rightarrow A * B \{ A.val = A_1.val * B.val \}$
$A \rightarrow A + B \{ A_1.val = B.val \}$ Neither Synthesised nor Inherited
$A \rightarrow id \{ A.val = id.num \}$

The given SDT is
a) L-attributed but not S-attributed
b) Only S-attributed
c) Both S-attributed & L-attributed
✓d) None

Q Consider the following SDT:

$C \to C \square S$ $\{C.val = C_1.val + S.val\}$

$C \to S$ $\{C.val = S.val\}$

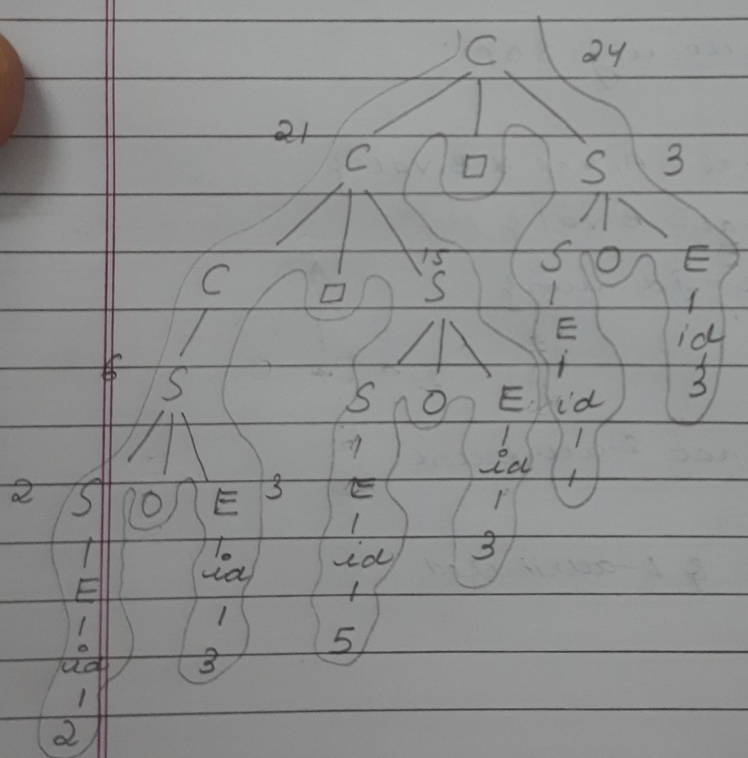
$S \to S \bigcirc E$ $\{S.val = S_1.val \times E.val\}$

$S \to E$ $\{S.val = E.val\}$

$E \to id$ $\{E.val = id.num\}$

Find the value of expression

2 0 3 □ 5 0 3 □ 1 0 3



⇒ 24

Q. Consider the syntax directed translation scheme:

$S \to A$	$\{ S.val = A.val \}$
$A \to A+B$	$\{ A.val = A_1.val \times 2 + B.val \}$
$A \to B$	$\{ A.val = B.val \}$
$B \to 1$	$\{ B.val = 1 \}$
$B \to 0$	$\{ B.val = 0 \}$

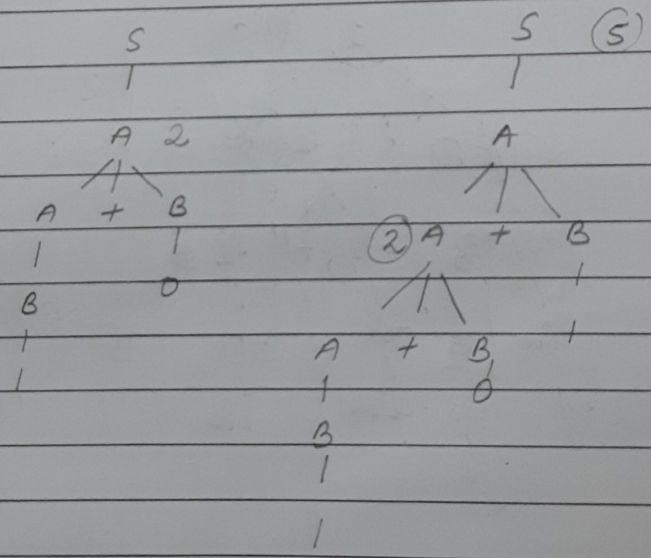
The value of S is

✓ a) The value of the binary input in decimal form.

b) The no. of bits in the input string.

c) The value of the binary is octal form.

d) None of these.



Q. Consider the given grammar with semantic actions given below:

$S \rightarrow S_1 + S_2$ $\{S.sign = S_2.sign\}$

$S \rightarrow S_1 * (S_2)$ $\{S.sign = S_1.sign * S_2.sign\}$

$S \rightarrow S_1 / S_2$ $\{$ if $(S_1.sign == 0)$ then $S.sign = 1$ else $S.sign = 0\}$

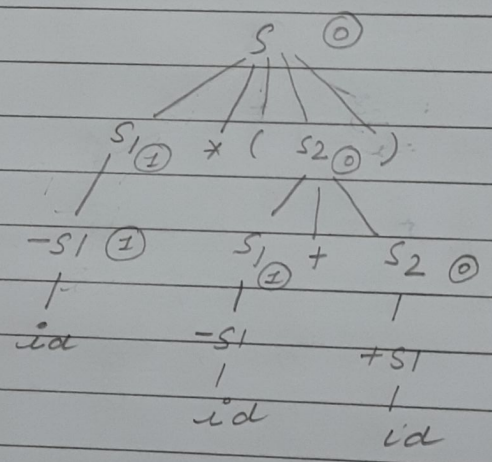
$S \rightarrow +S_1$ $\{S.sign = 0\}$

$S \rightarrow -S_1$ $\{S.sign = 1\}$

$S \rightarrow id$ $\{S.sign = 0\}$

The attribute value at the root S for input $-id * (-id + id)$ is _____

(S_1 & S_2 are same as S)



✓ A) 0

B) 1

C) 5

D) 7

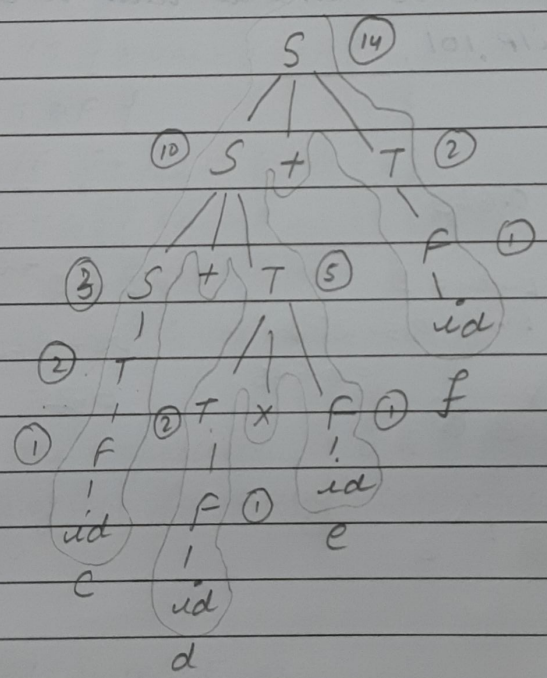
Only synthesized attributes

Page No.
Date : / /201

Q. Consider the following SDT scheme:

$S \to S + T$ $\{S.val = S.val + T.val + 2\}$
$S \to T$ $\{S.val = T.val + 1\}$
$T \to T \times F$ $\{T.val = T.val + F.val + 2\}$
$T \to F$ $\{T.val = F.val + 1\}$
$F \to id$ $\{F.val = 1\}$

For an input c+d xe+f, this translation scheme will give output _____



A) 10
B) 14
C) 18
D) 12

Q. Consider the following grammar with corresponding attributes.

$F \to .L$ $\{F.val = L.val\}$

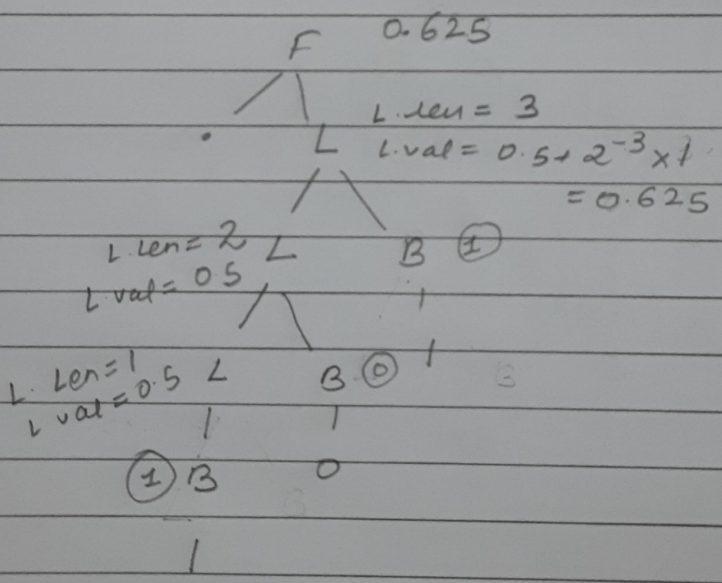
$L \to LB$ $\{L.len = L_1.len + 1,\ L.val = L_1.val + 2^{-L.len} \times B.val\}$

$L \to B$ $\{L.len = 1,\ L.val = B.val/2\}$

$B \to 0$ $\{B.val = 0\}$

$B \to 1$ $\{B.val = 1\}$

If "F.val" gives the value of the binary fraction generated by F in the above grammar then that will be the value of F.val on i/p, 101.



$0.5 + 2^{-2} \times 1$

$0.5 + 0.125$

0.625

✓ A) 0.625

B) 0.521

C) 0.525

D) None

Q. In a SDT scheme, if the value of an attribute of a node is a function of the values of the attributes of its children, then it is called a

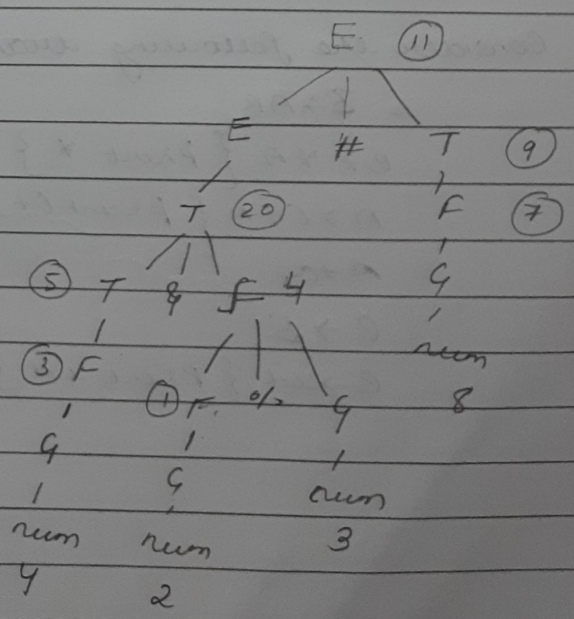
(A) Synthesized attribute
(B) Inherited attribute
(C) Canonical
(D) None of the above

Q.

Production Rule	Translation Rule
$E \to E \# T \{ E.val = E_1.val - T.val \}$	I
$/ T \{ E.val = T.val \}$	II
$T \to T \& F \{ \quad \}$	III
$/ F \{ T.val = F.val + 2 \}$	IV
$F \to F \% G \{ F.val = F.val + G.val \}$	V
$/ G \{ F.val = G.val - 1 \}$	VI
$G \to num \{ G.val = num.val \}$	VII

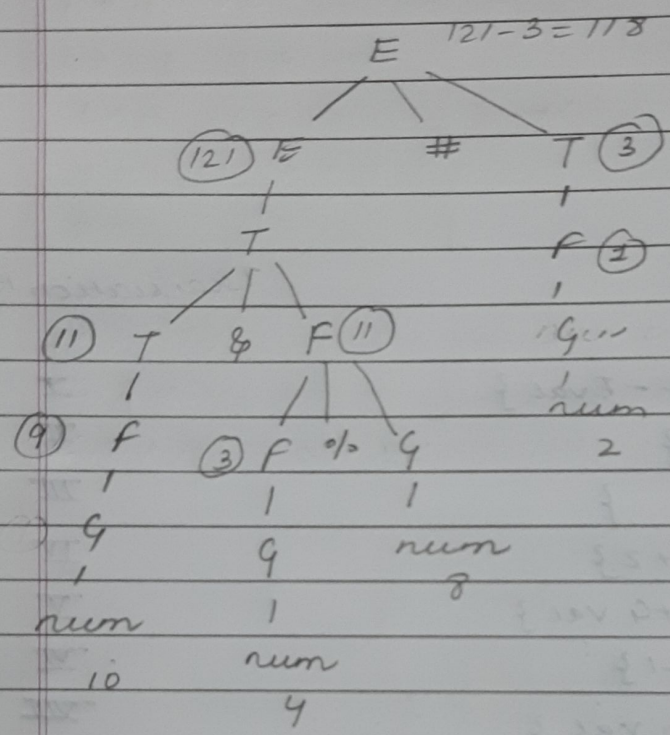
If the expression 4 & 2 %3 #8 is 11 then which one of the following correctly represent the rule III ?

a) T.val = T_1.val × F.val
b) T.val = T_1.val + F.val
c) T.val = T_1.val − F.val
d) None



Q. Answer for the expression

10 & 4 % 8 # 2



Answer ⇒ 118

Q. Consider the following translation scheme

$S \to AB$

$B \to *A \{ print * \} B / \epsilon$

$A \to C + A \{ print (+) \}$

$A \to C$

$C \to S$

$C \to id \{ Print\text{-}id \}$

Page No.
Date : / /201

What is the output it produces for input

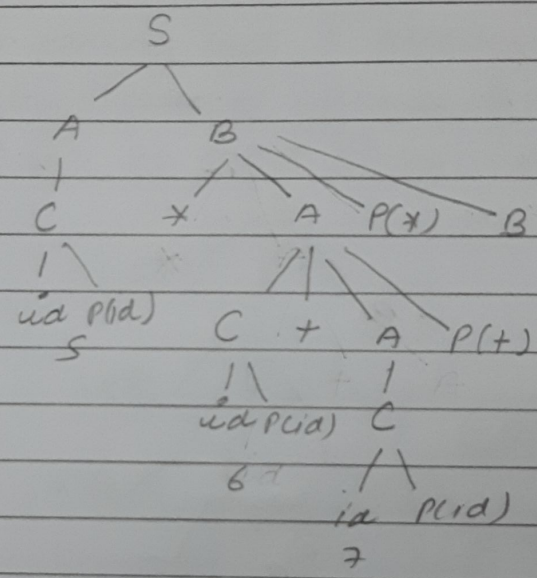
5 * 6 + 7

A) 37

B) 5 * 6 + 7

C) 56 * 7 +

d) 567 + *

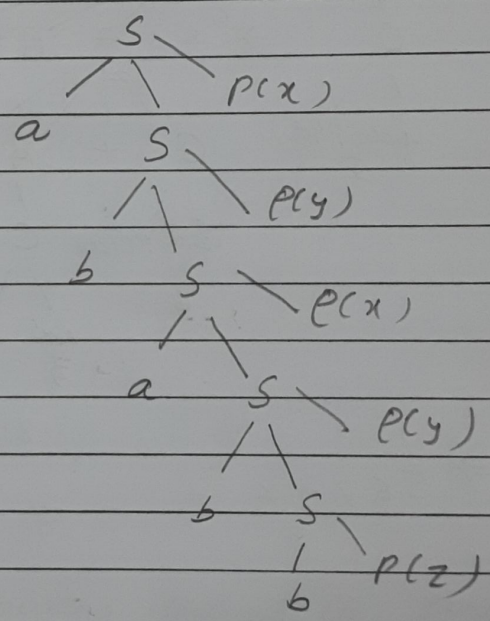


⇒ 567 + *

Q. $S \to aS$ {print(x)}
$S \to bS$ {print(y)}
$S \to a$ {print(z)}
$S \to b$ {print(z)}

SDT is carried out with Bottomup parser. What is the output for input. → ababb

a) ZXYXY
b) ZYXYY
✓ c) ZYXYX
d) ZYYXX



⇒ ZYXYX

Q. Consider the following SDT :

$E \to E + T \quad \{E.x = E.x \times T.x\}$

$E \to T \quad \{E.x = T.x\}$

$T \to T * F \quad \{T.x = T.x - F.x\}$

$T \to F \quad \{T.x = F.x = 1\}$

$F \to id \quad \{F.x = 5\}$

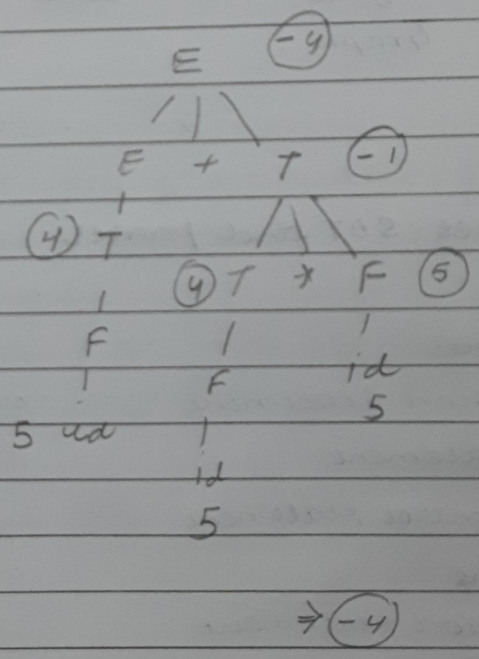
If bottomup parsing uses S-attributed definition then that is the value of attribute evaluated at root side E for an input string "id+id*id"?

a) 10

b) 4

c) 2

✓ d) None



$\Rightarrow -4$