65

Page No.
Date : / /

compiler design

Lexical Analysis [Regular Expression, Finite automata]
parsing [CFG, DPDA]
sementic Analysis [context sensitive grammer]
Intermidiate code genration [Attribute grammer]
code optimization
code generational and Runtime Enviroment.

Grammar

Def :- Set of rule used to describe string of language.

G = (V, T, P, S)
V ⇒ Set of variable (Non terminals)
T ⇒ Set of terminal
P ⇒ No. of production
S ⇒ Starting symbol

\# For every language their exits a grammer and every grammar represent rules of some language.

Page No.
Date : / /

$\alpha = \{a^n b^n \mid n \geq 1\}$

$\{ab, aabb, aaabbb \ldots\}$
Recursion

then grammar for this,

$S \rightarrow ab \mid aSb$
Recursion representation

$\alpha = \{a^n b^n c^m \mid n, m \geq 1\}$
$\{abc, aabbc, aaabbbc\}$

$S \rightarrow AB$
$A \rightarrow ab \mid aAb$
$B \rightarrow c \mid cB$
production

$V = \{S, A, B\}$ P
$T = \{a, b, c\}$

$\alpha = \{a^n b^m c^m d^n\} \; m, n \geq 1\}$

$S \rightarrow a A d \mid aSd$
$A \rightarrow bc \mid bAc \; \{b^m c^m\}$

Page No.
Date : / /

$\alpha = \{a^m b^n \mid m > n\}$

$\{aab, aaabb, aaaabbb\}$

$S \rightarrow aSb / aAb$
$A \rightarrow aA / a$ { Minm String }

$\alpha = \{a^i b^j \mid i \neq j\}$

$S \rightarrow aSb / aAb \mid aBb$
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b$

find the grammar, that all, string of a and b where each string is odd length palindrom.

$\alpha = \{a, b, aba, bab, aaa \ldots bbb \ldots\}$

$S \rightarrow aSa \mid bSb \mid a \mid b$

Find the grammar that generet odd length palindrom string of english language. and how many production will be there.

$S \rightarrow \underbrace{aSa \mid bSb \mid cSc \mid \cdots zSz}_{26} \mid \underbrace{a \mid b \mid c \mid d \cdots}_{26}$

$= 52$ A

Page No.
Date : / /

\# Find the grammar that represent all palindrome of english language

$$S \to \underbrace{aSa \mid bSb \mid cSc \mid \cdots zSz}_{26} \mid \underbrace{a \mid b \mid c}_{26} \mid \underbrace{aa \mid bb \mid cc}_{26}$$

$= 78.$

\# Identify ~~grammar~~ language generated by this grammar.

$$S \to aS \mid bS \mid a \mid b$$

Derivation : process of generating strings from the given grammar by replacing left hand side grammar by it's ~~co~~ corresponding right side part.

$$\alpha = \{a, b, aa, bb, ab, ba\}$$

$$= (a+b)^+$$

\# $a^n b^n c^n \geq 1$ — it can't be generated by type 2 grammar.

$S \to aSBC$
$S \to abC$
$cB \to Bc$
$bB \to bb$

Page No.
Date :

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

{ε, aab, aa, bb, baaa, bbab

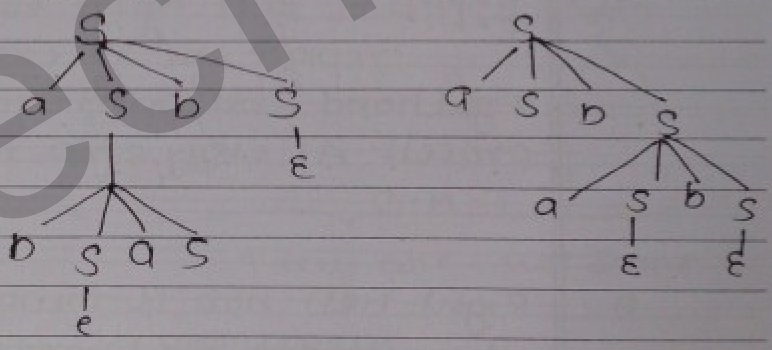
{ab, ba, ~~abab~~, aabb, abab, baba, bbaa

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

parse tree

The tree representation of derivation.

abab



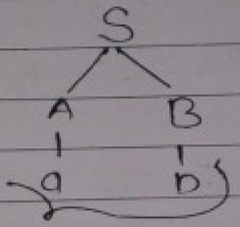
context free grammer said to be, ~~contex~~ ambigous if it contain more then '1' parse tree, More then 1 left most derivation and 1 right most derivation and '1' left most and right most derivatio

Page No.
Date: / /

Q1 $S \to AB$

$A \to a$

$B \to b$



unambigous

We can replace A B with a, b any one first.

if we replace A by a first then it is called left Most derivation else it is right most derivatio

*1 L·M·D :- Left most non terminal replace, by it corresponding right hand side part. In the derivation process. At every step is known as L·M·D

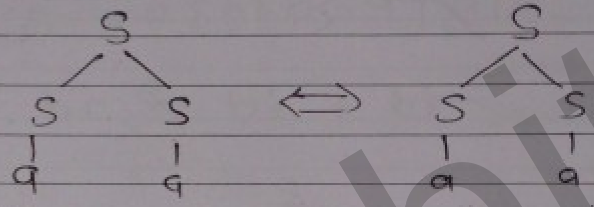
*1 Right Most non terminal replace by its crosponding right hand side part at every step in derivation process is R·M·D.

The derivation may be left most for right most derivation or random derivation.

But in parsing we will use left most derivation or right most derivation.

same: a = 

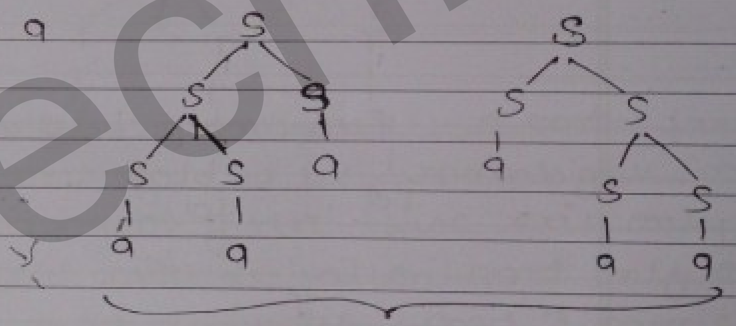
S → SS|a



left   rightmost

⇒ ~~Sa~~

⇒ a a a



Trees are not same

so it is unambiguous.

S → aSb
S → ab
S → SS } well formed parenthesis.

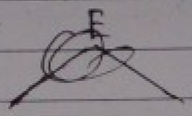
Page No.

Date: / /

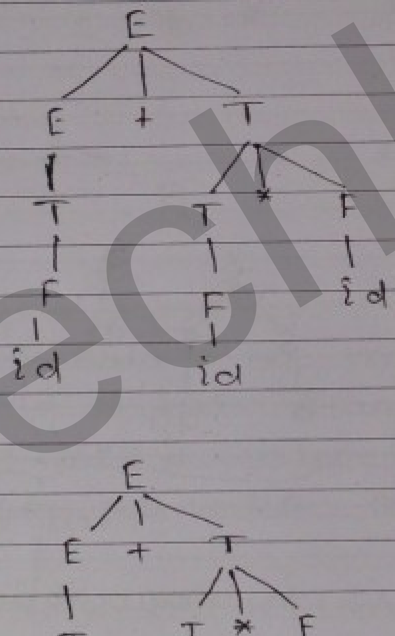
Q1 ~~E→T~~ $E \rightarrow E + T \mid T$

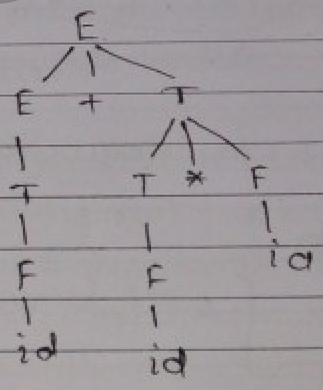
$T \rightarrow T * F \mid F$

$F \rightarrow id$



id + id * id

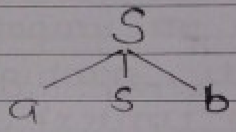




Note :- To check the ambugity in context free grammar there is no algorithm hence ambiguity problem is "undecidable" problem.

$$S \rightarrow aSb \mid bSa \mid a \mid b$$

~~Small~~
Small string = a



unambigous grammer :- A context free grammar said to be unambigous for "all string of given" there exit only one left most derivation and only one right most derivation and 1 parse tree

For one string one left most derivation and one right most derivation exist. both have to produce only one parse tree.

Page No.
Date : / /

$S \to aSb \mid ab$ grammer is unambiguous are not.

Ans => undecidable

\# **NOTE** :- It is impossible to eliminate, ambiguity from every ambigous context free grammar because, there is no generalized algorithm hence, elimination of ambiguty problem is also unddecidable problem.

\# A ambigous grammar for which, elimination of ambiguty is not possible, it called as inherently ambigous grammar.

$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

$L = \{a^n b^n c^m \mid n,m \geq 1\} \cup \{a^n b^m c^m \mid n,m \geq 1\}$

\# Fortunatly No programming language are inherently ambigous.

Page No.
Date : / /

$S \rightarrow S_1 | S_2$

$S_1 \rightarrow AB$
$A \rightarrow aAb | ab$
$B \rightarrow cB | c$

$S_2 \rightarrow CD$
$C \rightarrow cC | a$
$D \rightarrow bDc | bc$

Types of grammer

- R.L.G, L.L.G → Regular (Type 3)
  - $A \rightarrow xB | x$
  - $A \rightarrow Bx | x$
- (Type 2 G) CFG
  - $A \rightarrow \alpha$, $\alpha \in (V+T)^*$
- (Type 1 G) CSG sensetive
  - $\alpha \rightarrow \beta$, $|\alpha| \leq |\beta|$, $\alpha, \beta \in (V+T)^+$, $aA \rightarrow bb$
- (Type 0) unrestricted grammer

Linear grammer:- In any grammer, left hand side only one non terminal right hand side at most one non terminal is present, that is linear grammer.

Page No.
Date : / /

linear grammer may be left lenear or right lenear or middle linear.

If the linear grammer is either left linear or right linear then it is regular.

All regular grammer are linear grammar but all linear grammar need not be linear.

$S \to aSb \mid ab$
linear but not regular.

<u>CFG</u>
eg $A \to BCD$
$S \to aSb \mid ab$: linear not regular.

<u>CSG</u> $aA \to bb$ left context
$S \to AB$
$aA \to ab$
$B \to a$
] CSG

Page No.
Date : / /

Regular grammer

$S \to aS \mid bS \mid a$

$S \to lA$
$A \to lA \mid dA \mid \epsilon$ } RG

$S \to CS \mid SS \mid \epsilon$ = CFG

CSG

$a\,A\,b \to a\alpha b$

replace A by α if only it has a and b as left context & right context.

$S \to aSb \mid ab$ = CFG

becoz it is regular we can have either
$A \to xB$;
or $A \to Bx$;

Page No.

Date : / /

#

1) Regular grammars not have enough power, to represent syntex of programming - language. (Structure) hence to represent syntactic structure of programming - languages the suitable grammers are context free grammar.

2) context free grammar, doen't have - enough power to represent semantic (meaning) of programming language hence. To represent symentic of programming - languages suitable grammar are ~~conte~~ contex sensetive grammar that language contex sensetiv language.
Hence C language, c++ Java this languages called as context sensative language.

#

$S \rightarrow aSb \mid aAb$: $aSb\ b$

$A \rightarrow bA \mid b$ $\Rightarrow abb$

b, bb - -

abb

$\boxed{a^n b^n \quad n<m}$

Page No.
Date : / /

#1 checking the string is member of grammar or not. It called as parsing.

$S \to AB \mid BC$
$A \to BA \mid a$
$B \to CC \mid b$ b
$C \to AB \mid a$ a

Ans

First calculation first (First set)

=) 1) The First of a non terminal, gives information regarding that non terminal right hand side available first terminal information.

$A \to a\alpha$
↳ a is ans.

$First(A) = \{a\}$

2)

$A \to B\alpha$
$first\ of\ (A) = First\ of\ (B)$
$A \to B\alpha$
$B \to \epsilon$
$First(A) = \{First(B) - \epsilon\} \cup \{First\ \alpha\}$

Page No.

Date : / /

#1

$S \to aAB$

$A \to b$

$B \to c$

calculate first of every non terminal of this grammer.

S	{a}
A	{b}
B	{c}

#1

$S \to AB$

$A \to a \mid \epsilon$

$B \to b$

$S \to A$

$A \to a$

S	{a, b}
A	{a, ε}
B	b

#

$S \to AB$

$A \to a$

$B \to b \mid \epsilon$

S	{a}
A	{a}.
B	{b, ε}

Page No.

Date : / /

#1 $\cdot E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow id$

$E = \{id\}$

$E' = \{+, \epsilon\}$

$T = \{id\}$.

$T' = \{*, \epsilon\}$

$F = \{id\}$

#1 $S \rightarrow \overset{\epsilon}{A}a\,Ab \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

$S \rightarrow \cancel{A} = \{\epsilon\} = \{a\}$

$S \rightarrow \cancel{B} = \{\epsilon\} = \{b\}$

$S \rightarrow \{a, b\}$

$S = \{\cancel{\epsilon}\, a, b\}$

$A = \{\epsilon\}$

$B = \{\epsilon\}$

#1 $S \rightarrow ACB \mid Cb \mid Ba$ {

$A \rightarrow da \mid BC$ $\{d\}$ $\{d, g, h, b, a, \epsilon\}$

$B \rightarrow g \mid \epsilon$ = $\{g, \cancel{\epsilon})$ $\{d, g, h, \epsilon\}$

$C \rightarrow h \mid \epsilon$ = $(\cancel{h, \epsilon})$ {

$S = \{dg$

Page No.
Date : / /

Q) $A \to A_1 A_2 A_3$

first of (A_1) contain 5 elements.
First of (A_2) contain 4 element.
First of (A_3) contain 3 element.
All first sets contain (ϵ) and
all element are different. The how
many element contain

$A_1 \to 5$ (1 'ϵ' then take 4)
$A_2 \to 4$ (3)
$A_3 \to 3$ (2)
and last of $A_3 \to \epsilon$ include ϵ

$= 4+3+2+1$
$= 10.$

Page No.
Date : / / .

1) follow (Starting symbol) = {$}

2) $A \to \alpha B \beta$
follow(B) = First(β)

3) $A \to \alpha B$
follow(B) $\supseteq$ follow(A)

$A \to \alpha B \beta$
$\beta \to \epsilon$

[NOTE:- follow set doesn't include 'ϵ']

follow (B) = { First(β) - ϵ } $\cup$ follow (A) }

$E \to T E'$
$E' \to + T E' | \epsilon$
$T \to F T'$
$T' \to * F T' | \epsilon$ {*, ϵ}
$F \to id$

E = ~~{E}~~ Follow = {$
follow = E' = {$}.
follow(T) = {+, $}

$T \to E' \to \epsilon$ generate (ϵ)
then take first of E'
= {+, ϵ}
exclude => {+, $}

Foll (T') = {+, $}
follow(F) = {*, +, $}

Page No.
Date : / /

#1 $S \to AaAb \mid BbBa$

$A \to \epsilon$

$B \to \epsilon$

$S = \{ \$ \}$

$A = \{ a, b \}$

$B = \{ b, a \}$

#1 $S \to ACB \mid Cb \mid Ba$

$A \to da \mid BC$

$B \to g \mid \epsilon \quad = \{ g, \epsilon \}$

$C \to h \mid \epsilon \quad = \{ h, \epsilon \}$

$S = \{ \$ \}$

$A = \{ h, g, \$ \}$

$B = \{ \$, a, h, g \}$

$C = \cancel{\{ g, \$ \}} \; \{ g, \$, b, h \}$

Page No.
Date: / /

$S \rightarrow aAbB \mid bAaB \mid \epsilon$

:) $A \rightarrow S$

$B \rightarrow S$

First(S) = {a, b, ε}

First(A) = {a, b, ε}

First(B) = {a, b, ε}

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

Follow(A) = {a, b}

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

What is the regular expression that represent all the substring of string delhi.

	Substring of delhi	TOC
5	d, e, l, h, i ✓	T, O, C — 3
4	de, el, lh, hi ✓	TO OC — 2
3	del, elh, lhi ✓	TOC — 1
2	delh, elhi ✓	
1	delhi ✓	

Total $\boxed{\frac{n(n+1)}{2} + 1}$

Page No.
Date : / /

Q1 what is regular expression that generates letter and digit. And every string is staring with letter and length of string is atmost "32".

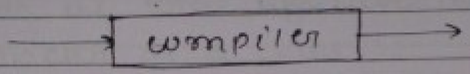
Ans

$\Sigma = \{a, b\}$ of lengths 5

$(a+b+\epsilon)^5$

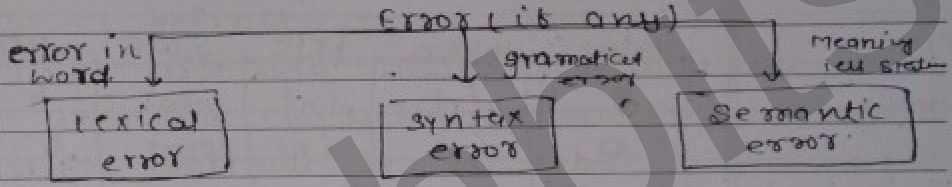
A) $l \cdot (l+d+\epsilon)^{31}$

compiler :- Is a program that translate a program written in one language into an equivalent program in another language.



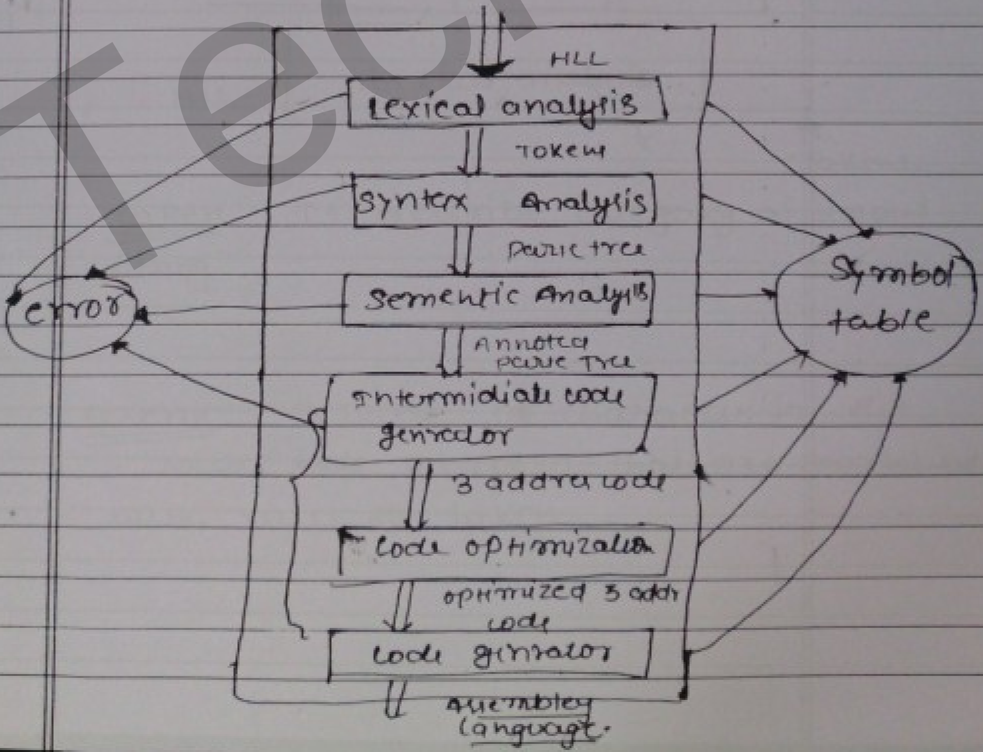
Javac → Byte code

C → Assembly lang



; printf("Hi").

int a,b
c= a+b;



Page No.
Date : / /

Symbol table is a data structure that contain information regarding variable and constant of the program alog with its attribute values.

Symbol table

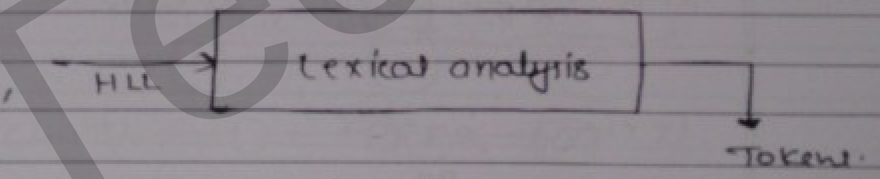
value	token	type	scop	Memory
a	id	int	local	4
b	id	int	local	4

Lexical analysis

Functionalites

1) It reads the total high level language program one character at a time.

2) It breaks the program into token.
3) It defines lexical error.
4) It eliminate blanks comment lines new line character present in the program
5) It construct Symbol table
6) It maintain line number of the program.



Token:- It describe category of input, String

Lexeme:- Sequence of character in the source program, that are matched with, rules of token.

Page No.
Date : / /

int a, b;
↓
Token (keyword)

Keyword :- if, else, for, while.

Identifier :- xIy, Ix

constant 20, 50.6

operator :- +, -, ++, --;

punctuation :-
; , ( ) { }

int a , b ;

Kew Id PU Id PU = lexeme

lexical error

pu

int 2x , 1y ;

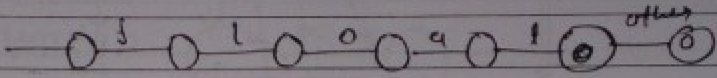
Kew

punctuation

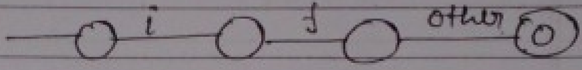
lexical error

int =

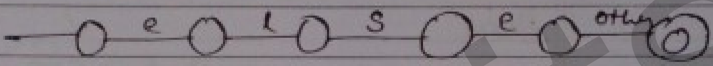
float



if :



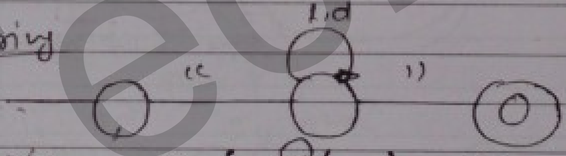
else:



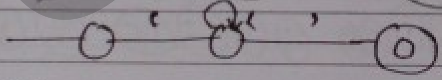
Identifiers

R.E $l(l+d)^*$

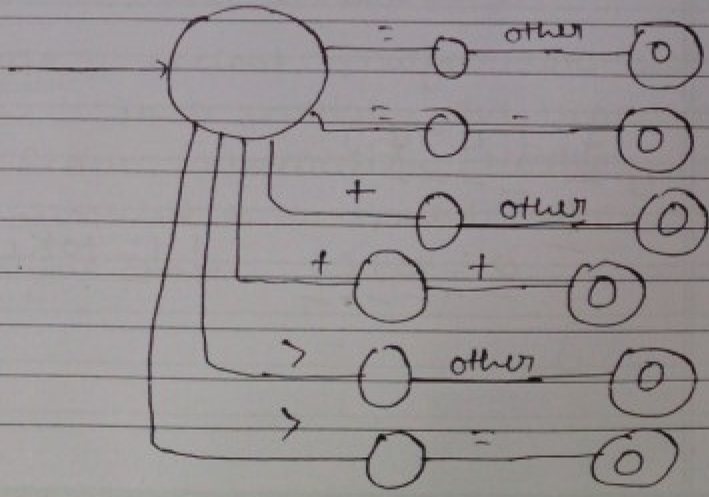
String



Char

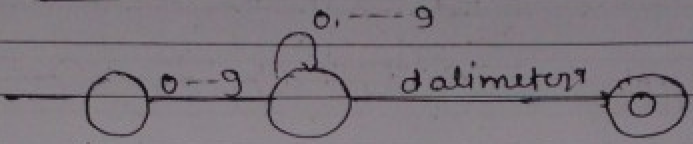


operator



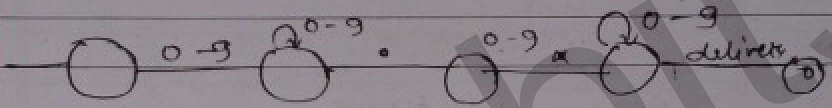
Page No.
Date : / /

constant



Real

$(0+1+2+ - - 9)^{+} (0+1+2 - - +9)^{1}$



int xy3123 , xabc ;
other other

for (i = 0 ; i <= 10 ; i ++)

= Total = 13 token.

#1 if (a > b)

{

a = b + c ;   14 = token

}

Page No.
Date: / /

int a   int123 a , b ;

Id   Id   pn   Id   Id

# if (a > b)
{
a = b + c ; /* additi. */
}

= 14 ~~keyword~~ Token

white spaces automata also implemented

int ☐ a , b ;

Key   Id   pun   Id   pun

Symbol table construction

NOTE :- Lexical analyzer, construct Symbol table partially. The type information Scope information, Memory size will be allocated to the symbol table, into semantic analysis phase.

lexical error

pu id op id op id

✓ 1) ;a=b+c = NO

✓ 2) c = a + "Hello"; NO
id op id op string

✓ 3) int a = 1, b = 2; NO
key id op id

× 4) String S = "Hello"; No operator, Not punctuation, No keywo

× 5) char a = 'ab; charactar atuomatg give err

× 6) char b = ab';

× 7) String J = compiler'' ; error

✓ 8) int a = 908; NO

✓ 9) float c = 9.7; NO

~~10~~ float d = 9...8; lexical error

11) int1 a, b;

~~12~~ int 1a, b; error

13) 1int a, b;

14) Int a, b; NO

Page No.
Date : / /

No of token

1) int a, b, c; ⇒ 7

2) printf("compiler"); ⇒ 5
1 2 3 4 5

3) printf("%d %d", a, b); = 9
1 2 3 4 5 6 7 8 9

4) if (a > b)
id
{
a = a + c;
}
= 14 token

5) ; a = b + c 6 token

6) int findMax(int a, int b)
{
if (a > b)
return a; = 24 (Token)
else
return b;
}

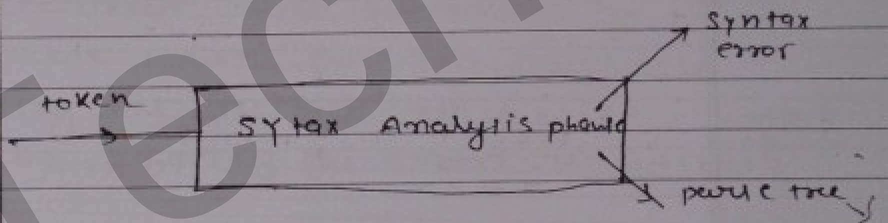
Page No.
Date : / /

\# main()
{
fro(i=0 ; i<=10; i++)
}

compiler reponse for this code

1) lexical error : [Fro identifier]
2) syntex error
3) lexical and syntex
4) None of these.

\#



Sytax errors

1) Lack of tokens to frame Statment

2) More no of token then actually required for a statment

3)

Page No.
Date : / /

\# $a = b + c * d$

$S \rightarrow id = E$

~~E → E+E|E*E|id~~

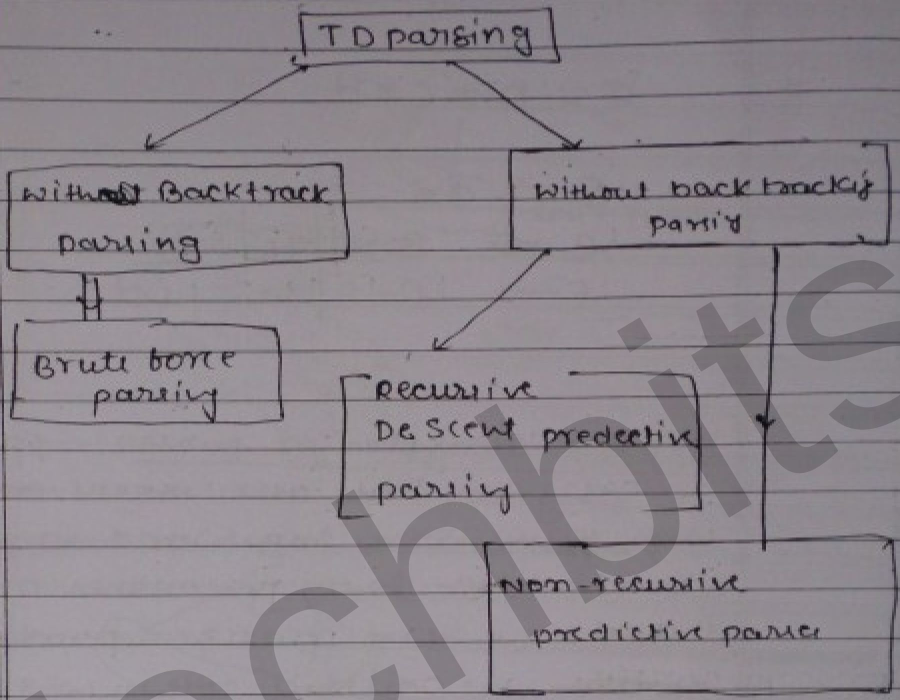
$E \rightarrow E+E \mid E*E \mid id$

<u>Syntax analyzer or parser</u>:- parser is a program that takes tokens and context free grammar as input and varifies tokens are derivible from grammer or not. If derivable, then parser produces parse tree ~~parsing~~ as output otherwise it written syntax error to the error handler.

\# The parsing can be done is top down, bottom up approach.

\# If any parser uses top down parsing technique then it follows left most derivation in the construction of parse tree.

\# If any parser uses bottom up parser technique then it uses reverse of right most derivation in construction of parse tree.



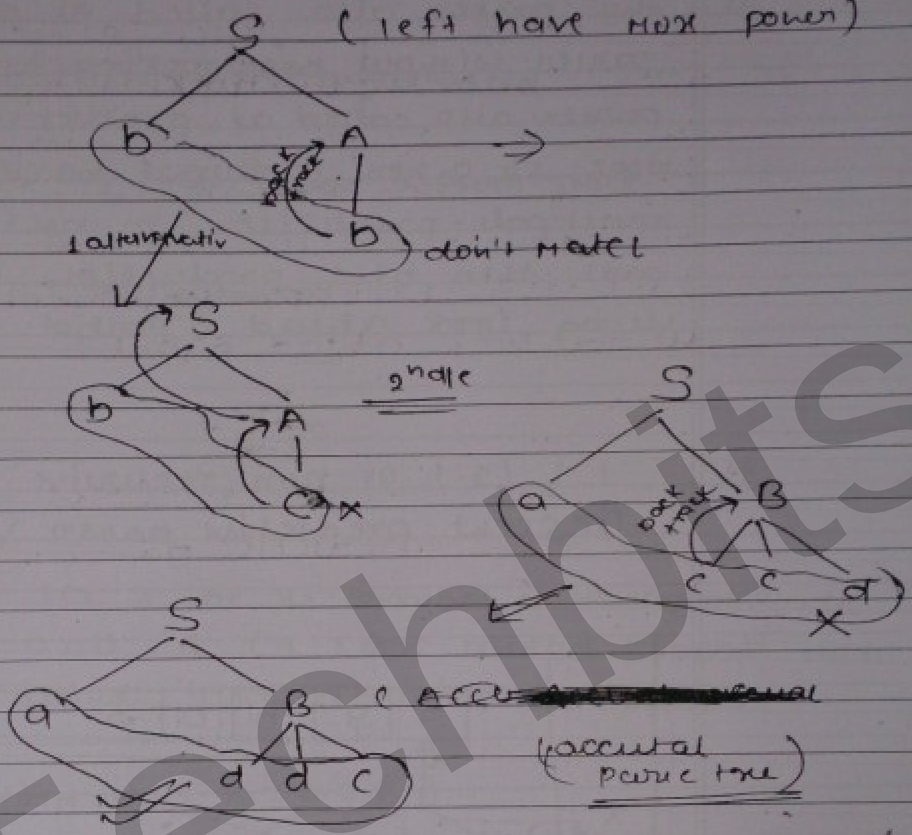
Q1

addc

$S \rightarrow bA \mid aB$

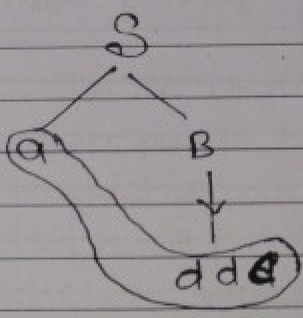
$A \rightarrow b \mid c$

$B \rightarrow ccd \mid ddc$



$O(2^n)$

NOTE left have higher priority



a d d c

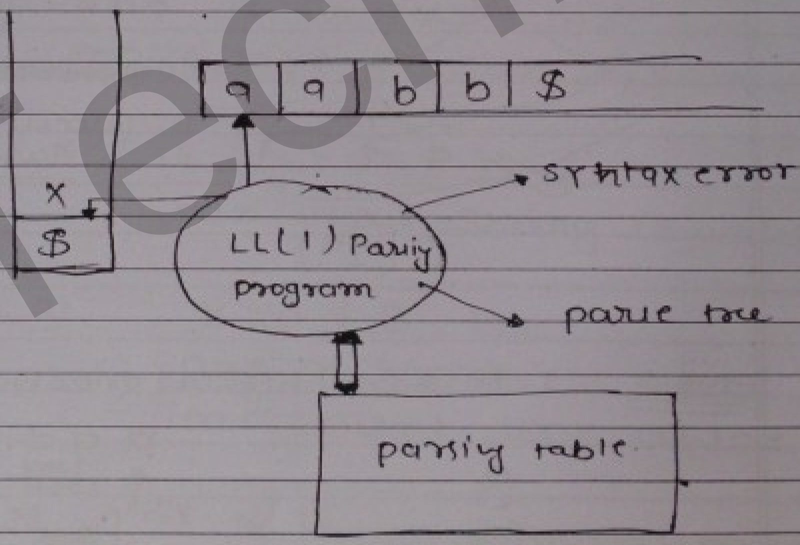
1st, 2nd look ahead

we check, which side give a first

Page No.
Date : / /

#) This parser also called as predictive parser. without backtracking topdown - parser also called as predictive parse. bcoz it a non terminal having, - multipale production on right hand - part then best production is selected using look ahead symbol.

#) LL(1) or non recursive ~~parser~~ Descent predictive parser



Page No.
Date : / /

if

1) $x = a = \$$ } input valid

2) $x = a \neq \$$ } pop X from stack

3) X is non terminal

automata
cowgera.com

Let x be the symbol at the top of stack and 'a' is look ahead symbol then parser take decision as follows.

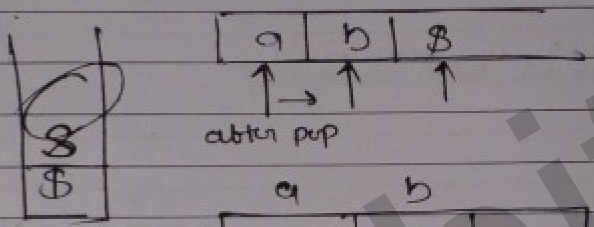
1) if $(x = a = \$)$
then input string is valid

2) $x = a \neq \$$ the
pop x from the stack
increment look ahead.

3) If x is non terminal on the stack then parser take decision form predictive parsing table. In the table if $[x, a] = X \longrightarrow U, V, W$ x were UVW production ~~such UVW present at~~ then replace x such the U apper at the top of stack.

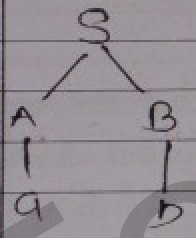
Page No.
Date : / /

[x, a] = blank entry in the table then there is syntax error, and written it do error handler.

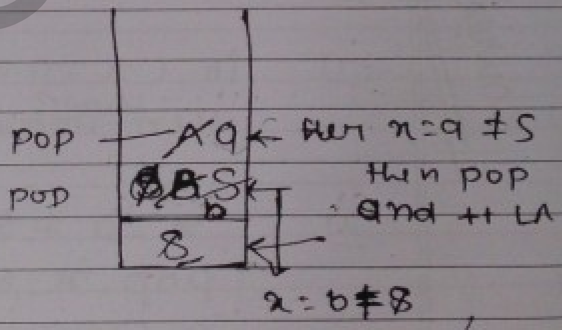


algo

Making wit S



	a	b	
S	S→AB		
A	A→a		
B		B→b	



x

Page No.

Date: / /

construction of LL(1) parsing table for the following grammar. And verify grammar is LL(1) grammar or not. also check wether it is ambigous or not.

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow id$

Size of table (LL(1)) = (No. of non terminal) × (Terminal + 1)

First and follow

$E = \{id\}$

$T' = \{*, \varepsilon\}$

$T = \{id\}$

$F = \{id\}$

$E' = \{+, \varepsilon\}$

Follow E

$E = \{\$\}$

$E' = \{\$\}$

$T' = \{+, \$\}$

T

F

Page No.
Date : / /

	id	+	*	)	$
E	$E \to TE'$				
E'		$E' \to +TE'$			$E' \to \epsilon$
T	$T \to FT'$				
T'		$T' \to \epsilon$	$T' \to *FT'$	$T' \to \epsilon$	
F	$F \to id$				

All LL(1) grammar are unambiguous grammar. But all unambiguous grammar Need Not be LL(1)

Follow: $S = a$

$A = a$

$B = \{a, \epsilon\}$

$C = \{b, \epsilon\}$

$S \rightarrow aABC$

$A \rightarrow a \mid bb$

$B \rightarrow a \mid \epsilon$ $\quad = \{a, \epsilon\}$

$C \rightarrow b \mid \epsilon$ $\quad \{b, \epsilon\}$

First	Follow
$S = \{a\}$	$S = \{\$\}$
$A = \{a, b\}$	$A = \{a, b\}$
$B = \{a, \epsilon\}$	$B = \{$
$C = \{b, \epsilon\}$	

construct LL1 parsing table for the following grammar

$S \rightarrow aABC$

$A \rightarrow a \mid bb$

$B \rightarrow a \mid \epsilon$

$C \rightarrow b \mid \epsilon$

[For, ε calculate follow then choose position where to place.]

	a	b	\$
S	$S \rightarrow aABC$		
A	$A \rightarrow a$	$A \rightarrow bb$	
B	$B \rightarrow a$	$B \rightarrow \epsilon$	$B \rightarrow \epsilon$
C		$C \rightarrow b$	$C \rightarrow \epsilon$

Page No.
Date : / /

$S \to a A B e$
$A \to A b c \mid b$
$B \to d$

first

$S = \{a\}$
$A = \{b\}$
$B = \{d\}$

Follow

$S = \{\$\}$
$A = \{d, b\}$
$B =$

	a	b	c	d	e	$
S	$S \to aABE$					
A		$A \to Abc$, $A \to b$				
B				$B \to d$		

NOTE

\# If grammar contain left recursion then there is multipale entry in table.

Page No.
Date : / /

NOTE

power of parser :- The number of grammar handed by a paticular parser, is know as power of that parser.

$S \rightarrow aS|b$ $\quad$ $S \rightarrow Sa|b$ $\quad$ $S \rightarrow Sa$, $S \rightarrow b$

	a	b
S	$S \rightarrow aS$	$S \rightarrow b$

	a	b
S		$S \rightarrow b$, $S \rightarrow Sa$

NOTE :- If a grammar contain left-recursion then its pridictive parsing table contain multipal entries. Hence left recursive grammer are not LL(1).

→ We can make this grammar are LL(1) by removing, left recursion for that.

Removing of left recursion

$S \rightarrow Sa/b$

generetation string = ba^*

⇓ Removal

$$S \rightarrow bS' \quad = b$$
$$S' \rightarrow aS' | \epsilon \quad = a^*$$
$$= ba^*$$

There is no more left recursive

Page No.

Date : / /

\# $A \to A\alpha_1 \mid A\alpha_2 \cdots \mid B_1 \mid B_2 \mid B_3 \mid \cdots$

$A \to B_1A' \mid B_2A' \mid B_3A' \cdots$

$A' \to \alpha_1A' \mid \alpha_2A' \mid \cdots \alpha_nA' \cdots$

\# Eliminate left recursion from grammar.

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id$

$$\left.\begin{aligned} &E \to TA' \\ &A' \to +ET / \epsilon \\ &T \to FB' \\ &B' \to *FT / \epsilon \\ \end{aligned}\right\}$$ Removed left recursion

$F \to id$

Page No.
Date : / /

Eliminate left recursion form following grm

$$S \rightarrow Sa \mid Sb \mid c \mid d \mid e$$

$$\cancel{S \rightarrow cA' \mid dB' \mid ec'}$$

$$A' \rightarrow aA' \mid \epsilon$$
$$B' \rightarrow dB' \mid \epsilon$$
$$c' \rightarrow e$$

$$S \rightarrow cS' \mid dS' \mid eS'$$

$$S' \rightarrow aS' \mid bS' \mid \epsilon$$

<u>More problem</u>

\#

$S \rightarrow aB \mid aA$ { string ab
$A \rightarrow a \mid b$ { ambiguous there problem
$B \rightarrow d \mid c$ create . problem for LL(1)
}

Solⁿ :- using left factoring we remove ambiguity problem.

$S \rightarrow aB \mid aA$
$A \rightarrow a \mid b$ $\Rightarrow$
$B \rightarrow d \mid c$

$S \rightarrow a(\underbrace{B + A}_{Z'}) \Rightarrow S \rightarrow aZ'$
$Z' \rightarrow B \mid A$
$A \rightarrow a \mid b$
$B \rightarrow d \mid c$

Removed ambiguity.

Page No.
Date : / /

$A \to \alpha B_1 | \alpha B_2 | \alpha B_3 | \dots \alpha B_n$

$A \to \alpha B$
$B \to B_1 | B_2 | \dots B_n$

left factor the grammar

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

$S \to iEtS(\epsilon + eS) | a$
$S \to iEtSZ' | a$
$Z' \to \epsilon | eS'$
$E \to b$

Doubt
* # NOTE GATE prob
It is impossible to convert into into LL(1) grammar

Follow(A)

Page No.

Date : / /

check whether this grammar (LL1) or not.

$S \rightarrow iEtSZ' \mid A$

$Z' \rightarrow \epsilon \mid eS$

$E \rightarrow b$

	i	t	B	a	b	$
S						
S'						
E						

Q! $S \rightarrow a\underline{S}\hat{\underline{A}} \mid \epsilon$ } creat table

$A \rightarrow c \mid \epsilon$

condition check (LL(1))

1) single production is LL(1)
2) left recursion gram not LL(1)
3) Non left functioning gram not LL(1)
4) $A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \alpha_n$

First $(\alpha_1) \cap$ First$(\alpha_2) \cap$ Firs$(\alpha_n) = \phi$ } LL(1)

5) $A \rightarrow \alpha \mid \epsilon$

First$(\alpha) \cap \{$~~First (A)~~ $\}= \phi$ } LL(1)

Follow(A)

Page No.
Date : / /

$S \to AB$
$A \to a$
$B \to b$
} single producti

LL(1)

Q) $S \to aABe$
$A \to \underline{A}bc / b$ (left recursion)
$B \to d$

Not LL(1)

\# $S \to ab / aa$

common prefix so Not LL(1).

\# $S \to AB$
$A \to a/b$
$B \to b/c$
[calculate first of a, b
i.e a, b]

and $\cap$ is ϕ so LL(1)

Page No.
Date : / /

$E \rightarrow TE'$ ✓ S·P

$E' \rightarrow +TE' \mid id$ { First of $+TE'$ id
$+ \cap id = \phi$

$T \rightarrow FT'$ ~ S·P

$T' \rightarrow *FT' \mid \epsilon \Rightarrow$ calcute Follow T'

$F \rightarrow id$ ✓ S·P ie

\# $S \rightarrow aSA \mid \epsilon$

$A \rightarrow c \mid \epsilon$ First (a) for this calculate foll of S

$S \Rightarrow$ First A $\{c, \epsilon\}$ then remove A $\Rightarrow$

$\{c, \$\} \cap$ First$(aSA) = \phi$

First of $(c) = c$ $\cap \neq \phi$ the

follow $(A) = \{c, \$\}$

Not LL(1)

\# $E \rightarrow TE'$

$E' \rightarrow +T$

Page No.
Date : / /

\# $S \to \underset{a}{aSA} \mid \underset{''}{\epsilon}$ $\{c, \epsilon\, \$\}$

$A \to \underset{c}{c} \mid \epsilon$

S

Q) $S \to \underset{a}{aSbS} \mid \underset{b}{bSaS} \mid \underset{a}{\epsilon} = \phi$

⇒

$[S, a] \Rightarrow$

follow $S = (b, a, \$)$

Not LL(1) :

$[S, a] = S \to aSbS$
$[S, a] = S \to \epsilon$

$[S, b] = S \to bSaS$
$[S, b] = S \to \epsilon$

Page No.
Date : / /

Q1 $S \to A \mid a$ $[S,a] = S \to a$ } Multiple
$[S,a] = S \to A$ } Not LL(1)
$A \to a$

Q) $S \to 1ABb \mid \varepsilon$
$A \to 1Ab \mid bC$
$B \to aB \mid b$
$C \to a$

Yes LL(1)

Q) $S \to AaAb \mid BbBa$ ~~A~~ $= \phi$
$A \to \varepsilon$
$B \to \varepsilon$
LL(1) A

Q) $E \to E+n \mid n$ Not LL(1)

Which of the following condition sufficient to convert CFG with LL(1)

1) Removing left recursion above
2)

Page No.
Date : / /

consider the following grammar,

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$ (e, $)

$E \rightarrow b$

(S', e)

pridctive parsing table entries (S', E)
(S', e)

follow

Q1 consider following grammar and pridictive parsing table.

$S \rightarrow aAbB \mid bAaB \mid \epsilon$

$A \rightarrow S\ b$

$B \rightarrow S$

	a	b	$
S	S → ε S → aAbB	S → ε S → bAaB	S → ε
A	A → S	A → S	
B	B → S	B → S	B → S

Page No.
Date : / /

~~First :~~

~~First = S {a, b, E}~~

~~Follow = S ⇒ A ⇒ b~~

~~Follow S ⇒ B →~~

First(S) = (a, b, E) | Follow(S) = {a, b, \$}

First(A) → First(S) ⇒ (a, b, E) ⇒ for E

follow(A) ⇒ {b, a} ✓

First(B) → (a, b, E)

follow(B) ⇒ {a, b, \$} ✓

#

$S \to AB$

$A \to aa \mid ab$

$B \to ba \mid bb$

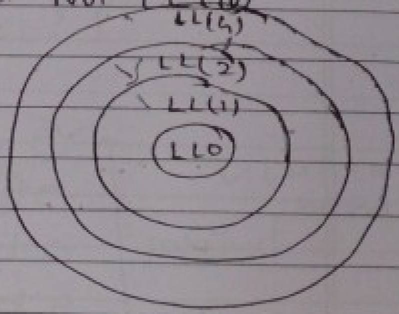
LL(1) but Not LL(0)

#

$S \to AB$

$A \to a \mid b$

$B \to b \mid a$



#

$S \to AB$

$A \to a$

$B \to b$

NOTE : All LL(K) grammer are unambigous but all unambigous grammer need not be LL(K) where K is the length of look ahead

Page No.
Date : / /

Q) consider the following grammer

$S \rightarrow FR$ (S, id)
$R \rightarrow *S \mid \epsilon$ (*, $)
$F \rightarrow id$

in predictive parsing table to entri

[S, id] [R, $] ✓
⇓ [R,
(FR) ($R \rightarrow \epsilon$) ✓

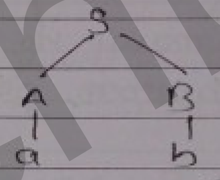
Page No.
Date : / /

NOTE

Top down parser can't work for, left recursive grammar and non deterministic grammar. Even if eliminate the, left recursion from the grammar loses precedence, and associativity rules and readablity also. Hence we need a parsing technique that can for "left recursive grammer" i.e Bottom up-parsing.

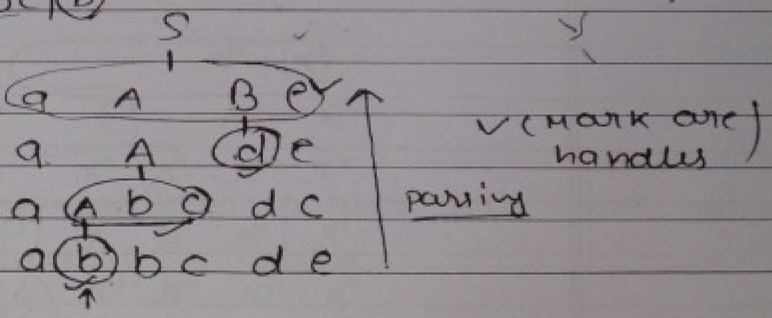
ab string

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$



\# $S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

abbcde



NOTE Bottom up parser construct the parse tree starting from, giving string and proceeds until, starting symbol of grammer.

Bottom up parser uses reverse of right most derivation, in construction of parse tree.

GATE-2008

Handle:- Handle is substring in the sentencial form ie is matching with complete right hand side part of given grammar

Q1 How many no of handle are detected by bottom parser tree?

$S \rightarrow xxW$ $\quad$ $xxxxyZZ$

$S \rightarrow y$

$W \rightarrow SZ$

S

xxW

$xxSZ$ $\quad$ There are handles

$xxxxWZ$

$xxxxSZZ$

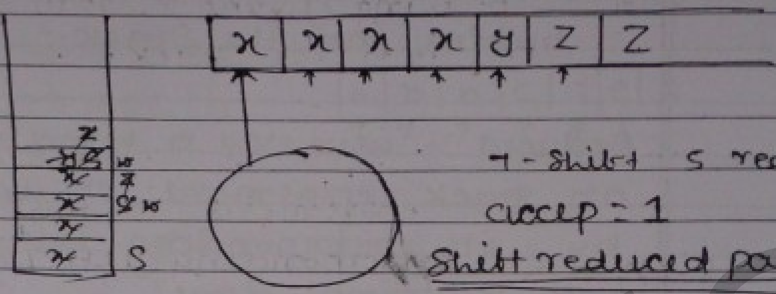
$xxxxyZZ$

5 No of handel are done.

Page No.
Date : / /

Q2)

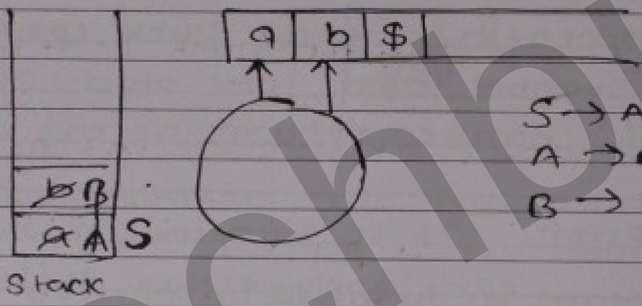
S R



7 - shift 5 reduced

accep = 1

Shift reduced parser



$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

S	R
1	1
1	1
	1

\# All the bottom up parser will work based on following 4 action
1) Shift.
2) reduce
3) accept.
4) error.

Page No.
Date : / /

Shift :- pushing symbol from input buffer into the stack.

Reduced :- whenever a handle is detected on stack replacment of that handle by its corrosponding left hand is non terminal done.

accept :- with the stack contain starting symbol and look ahed symbol is "$" parser return accept action.

Error :- It is situation in which parser can't apply either shift action or reduced action not even accept also.

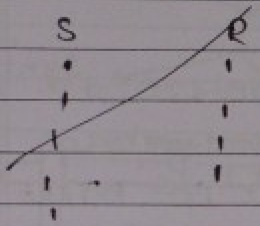
Q1 How many shift action reduced action How many total action taken by shift reduced parser to parse

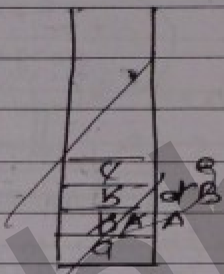
Shift = 7
Reduced = 5
accep = 1

$\left.\begin{array}{l}\text{Shift} = 7\\ \text{Reduced} = 5\\ \text{accep} = 1\end{array}\right\} = 13$ Q2A

$S \rightarrow aABe$
$A \rightarrow Abc / b$
$B \rightarrow d$

a	b	b	c	d	e	\$

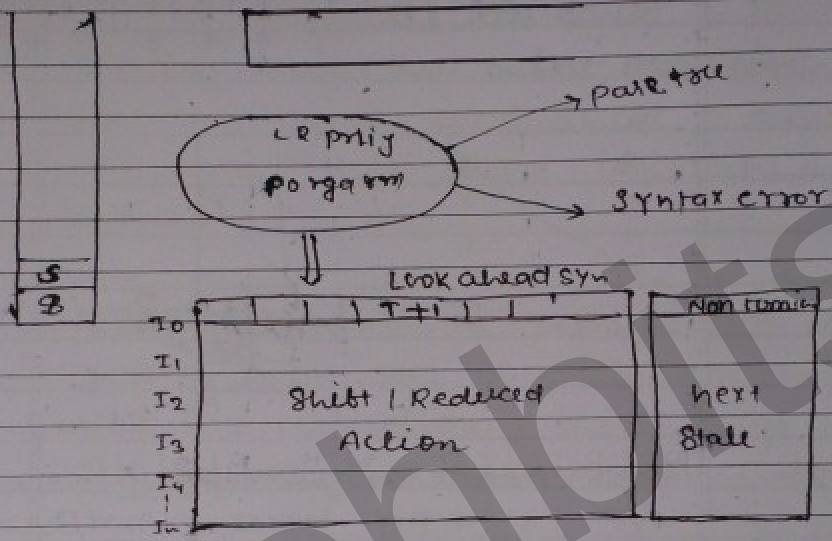




\# Two problem with B.UP

1) Reduced Reduced & conflict
2) when to do reduced and , shift

LR parsing



DFA

1) Let S be the State on the top of Stack "a" is the look ahead Symbol then parser take decision regarding Shift reduced action from the parsing table.

⇒ following are the possible action in the parsing table

Page No.
Date : / /

if Action [S,a] is shift j in the parsing table, then shift 'a' into the stack and shift j on the top onto stack and increment look ahead symbol

If action[S,a] is reduce in the parsing table and A → B is the production used for reduction, then pop 2 × |B| (length of B) symbol from the stack and push 'A' onto the stack and then push goto(i, A) onto the top of stack where 'i' is previous state onto the stack.

5. If action [S,a] is accept and ~~stop~~ parser halt and announces success.

If action [S,a] is error (black in the table entry) parser halt, return syntax error to the error handler.

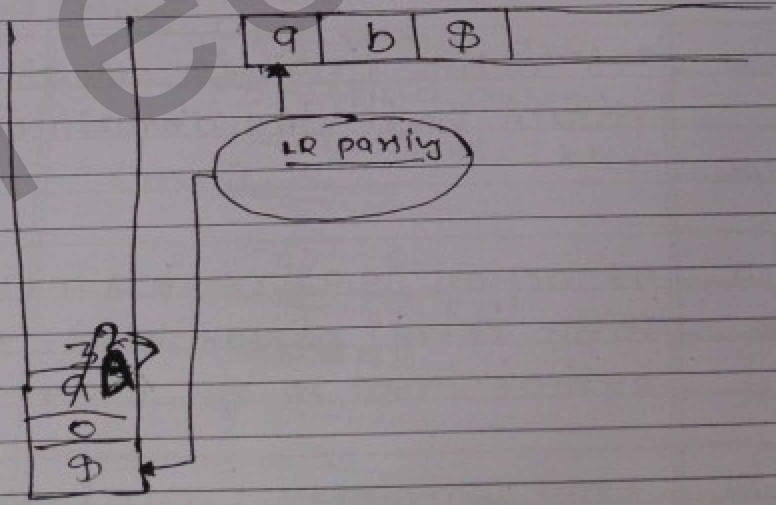
Time complexity = O(n) where n is length of input string.

Page No.
Date : / /

	Action a	Action b	Action $	goto S	goto A	goto B
I0	S3			1	2	
I1			accept			
I2		S5				4
I3		r2				
I4			r1			
I5			r3			
I6						
I7						

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$

a b



Page No.
Date : / /

Stack	input buffer	Action
$ 0	a b $	[0,a] = shift 3
$ 0 a 3	a b $	[3,b] = Reduce (A→a)
$ 0 A 2	a b $	[2,b] = shift 5
$ 0 A 2 b 5	$	[5,$] = Reduce B→b
$ 0 A 2 B 4	$	[4,B] = Reduce S→AB
$ 0 S 1	$	[1,S] = Accept

Note any parsing done on table known as table driven parser. hence LL(1) and LR(1) parsing are example

Recursive decent parser is not table driven.

Page No.
Date : / /

How many reduced action how many, shift and total action taken by LR process to parse input string (i+i) by considering following parser table.

$E \to T+E$

$E \to T$

$T \to i$

	i	+	$	E	T
I_0	S_3			1	2
I_1			accept		
I_2		S_4	r_2		
I_3		r_3	r_3		
I_4	S_3			5	2
I_5			r_1		

0
$

Page No.
Date : / /

Stack	Input buffer	Action
| $ | 0 |	| $ | ~~i~~ | i | $ |	
| $ | 0 | i | 3 |		

+ | i | $

~~$0~~

$0

$0 i 3 | | [0, i] = S3

$0 i 3 + | | [3, +] = R3 (T → i)

$0 i T 2 + 4 | | [2, +] = S4

$0 i T 2 + 4 | | [4, i] = S3

$0 i T 2 + 4 i 3 | | [3, $] = r3 T → i

$0 i T 2 + 4 T 2 | | [2, $] = r2 E → T

$0 i T 2 + 4 E 5 | | [5, $] = r1 E → T + E

~~$0 i T 2~~ +

$0 i

Stack	input buffer	
\$ | 0	i | + | i | \$	[0,i] = S3
\$ | 0 | i | 3	i | + | i | \$	[3,+] = R3 (T→i)
\$ | 0 | T | 2	i | + | i | \$	[2,+] = S4
\$ | 0 | T | 2 | + | 4 | i | 3	i | + | i | \$	[3,\$] = R3 (T→i)
\$ | 0 | T | 2 | + | 4 | T | 2	i | + | i | \$	[2,\$] R2 (E→T)
\$ | 0 | T | 2 | + | 4 | E | 5	i | + | i | \$	[5,\$] R1 (E→T+E)
\$ | 0 | E | 1	i | + | i | \$	[1,\$] = Accept

Page No.
Date : / /

construction of LR(0) parsing table

1) Agumented grammar:- Adding a New production
$S' \rightarrow S$ to the orignal grammar.

2) This agumented production helps the parser to show accept action ie whenever parser tries to redues S by S' it lead to accept action.

3) compute LR(0) item :-

LR(0) item is nothing but a context free grammar production having a "•" on ~~the~~ RHS part.

$A \rightarrow \cdot XYZ$ parser has seen 'x' and ready to see "YZ".
$A \rightarrow X \cdot YZ$
$A \rightarrow XY \cdot Z$
$A \rightarrow XYZ \cdot$

$S' \rightarrow \cdot S$ (parse ready to see right hand side part)

Page No.
Date : / /

closer $S' \rightarrow \cdot S$
$S \rightarrow \cdot AB$
$A \rightarrow \cdot a$

goto mean's move "•" 1 mov right place.

closure:- closure function add non terminal production and it will put "•" on right hand part of the production.

Goto :- Goto function move "•" 1 position ahead.

Q) construct LR(0) parsing table, for the following grammar. And verify grammar is LR(0) or not. check it is ambigous or, unambigous.

Page No.

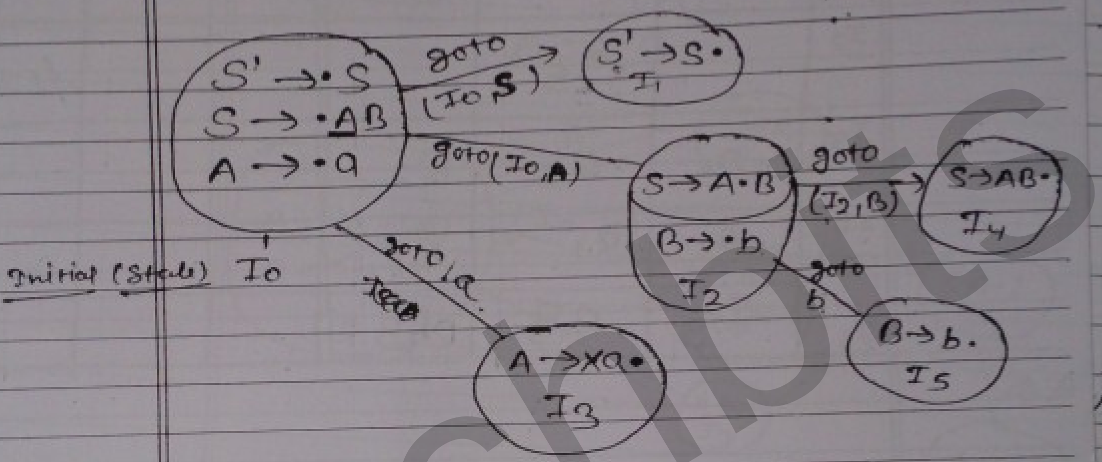
Date : / /

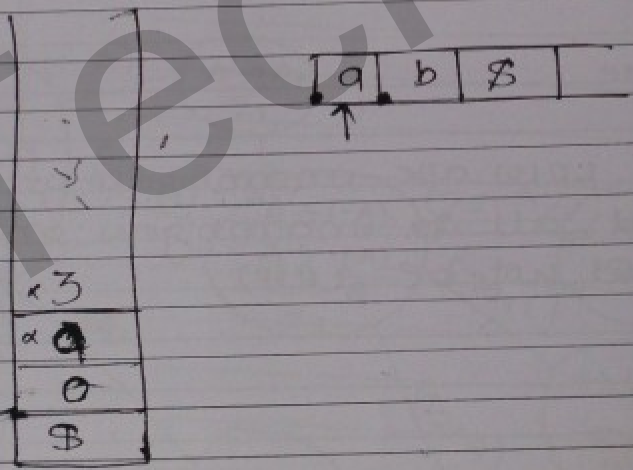
Ans

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

construct agument grammar →

Accp $S \to S'$
R_1 $S' \to AB$
R_2 $A \to a$
R_3 $B \to b$





Page No.
Date : 6/0/0

Shift

	a	b	$	S	A	B
I_0	S_3			1	2	
I_1			Accept			
I_2		S_5				4
I_3	R_2	R_2	R_2			
I_4	R_1	R_1	R_1			
I_5	R_3	R_3	R_3			

Note

All LR(0) are unambigous grammar but all ~~as~~ unambigous grammar need not be LR(0)

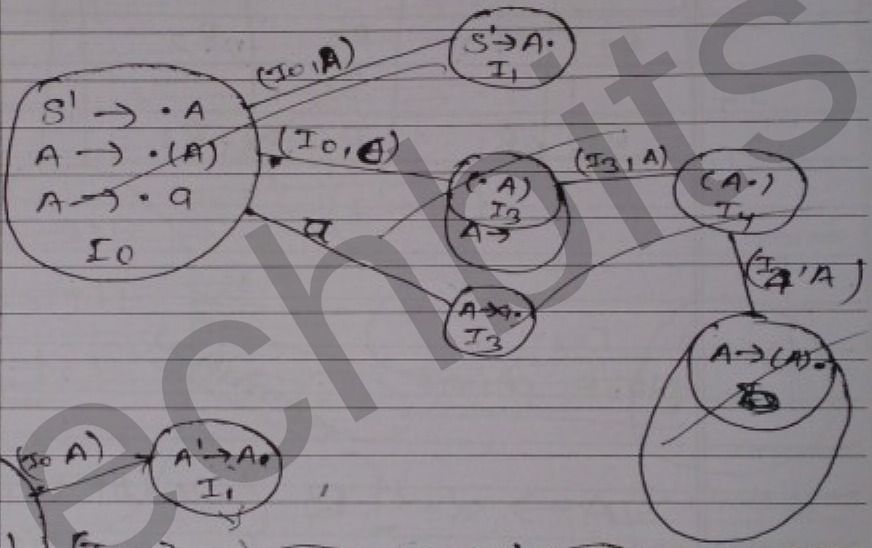
Page No.
Date : / /

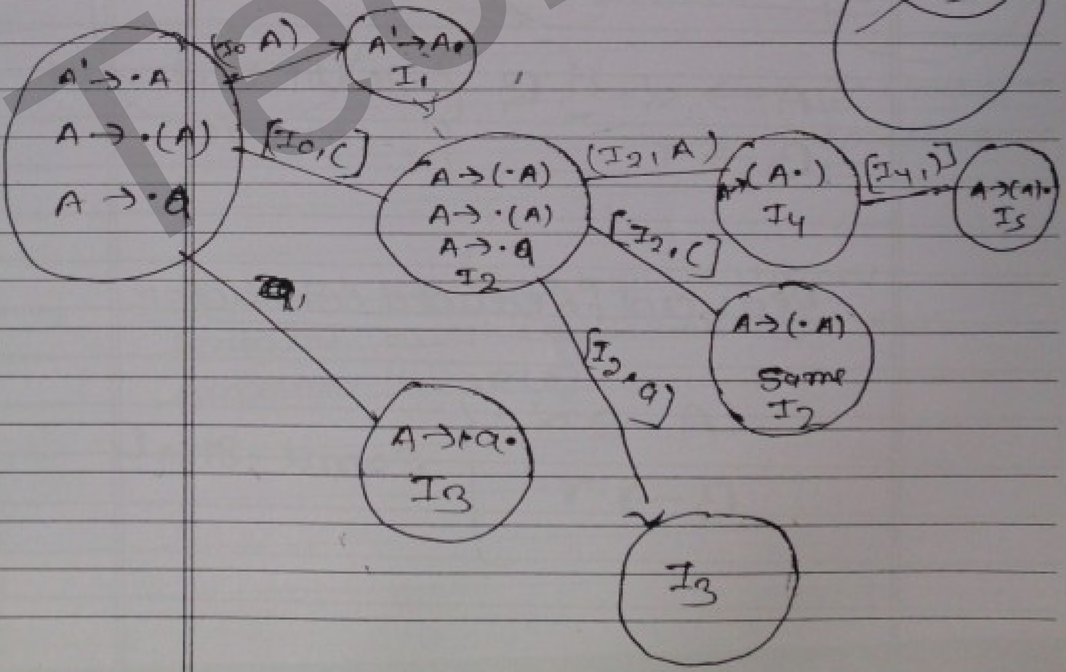
Q1

$A \rightarrow (A)$
$A \rightarrow a$

~~$S \rightarrow S'$~~
~~$S' \rightarrow$~~

$S' \rightarrow A$
$A \rightarrow \cdot(A)$
$A \rightarrow \cdot a.$





Page No.
Date : / /

	a	(	)	$	A
I_0	S3	S2			1
I_1					
I_2	S3	S2			4
I_3	R2	R2	R2	R2	
I_4			S5		
I_5	R1	R1	R1	R1	

NOTE point LR(0)

$$\left.\begin{array}{l} A \rightarrow \alpha \cdot \overset{\text{terminal}}{a}\, B \\ B \rightarrow \gamma \cdot \end{array}\right\}$$ Shift reduced conflict

Reduced / Reduced conflict

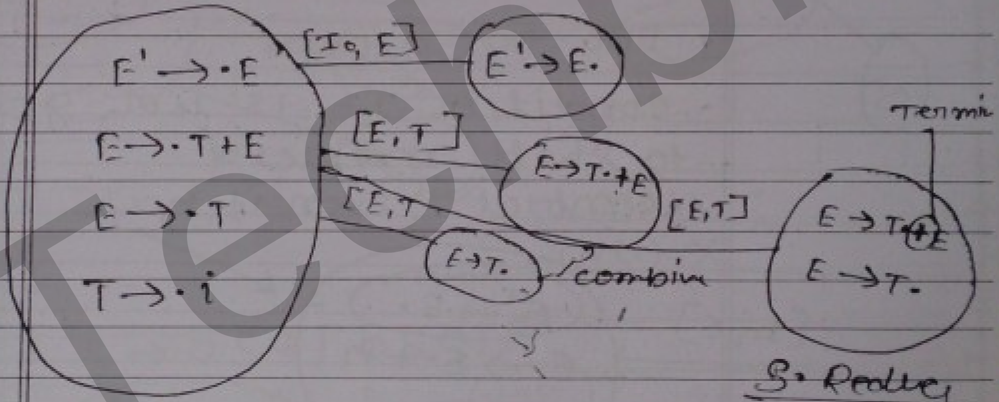
$$\left.\begin{array}{l} A \rightarrow \alpha \cdot \\ B \rightarrow \gamma \cdot \end{array}\right\}$$ Same state

Page No.

Date : / /

$E' \to \cdot E$

$E \to T+E$		$E' \to \cdot E$
$E \to T$	$\Rightarrow$	$E \to \cdot T+E$
$T \to i$		$E \to \cdot T$
		$T \to \cdot i$



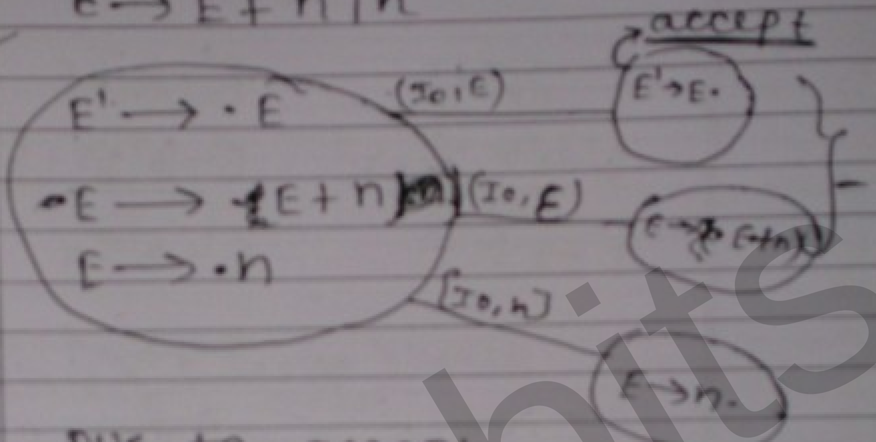
S-R Realte conflict

Note:- If LR(0) item automata contains atleast inadequate sate the grammar to LR(0)

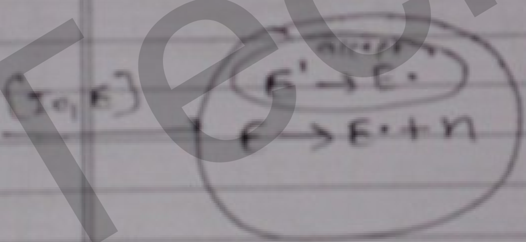
Q1

GATE

$E \rightarrow E + n \mid n$



Due to accept condition it is not going to shift reduced conflict even it has



Page No.

Date : / /

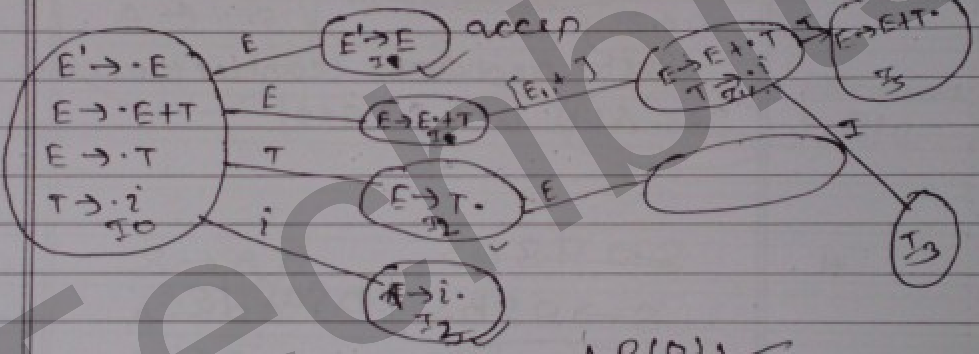
Q1 check wether following grammar is LR(0) or not.

$E \to E+T$ $\Rightarrow$ $E' \to \cdot E$

$E \to T$ $\quad$ $E \to \cdot E+T$

$T \to i$ $\quad$ $E \to \cdot T$

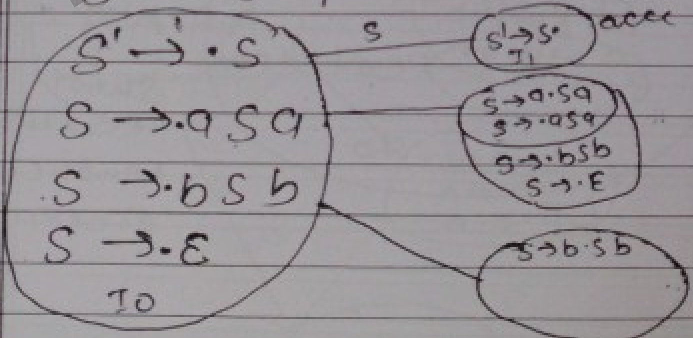
$T \to \cdot i$



LR(0) ✓

LR(1) X

$S \to aSa \mid bSb \mid \varepsilon$



Page No.
Date : / /

$S \to aSa \mid bSb \mid \epsilon$

$S' \to \cdot S$
$S \to \cdot aSa$
$S \to \cdot bSb$
$S \to \epsilon$ } Reduce

I_0

Shift reduced conflict occur so it is not LR(0)

$S \to aSbS$
$S \to aSbS \mid bSaS \mid \epsilon$

$S' \to \cdot S$
$S \to \cdot aSbS$
$S \to \cdot bSaS$
$S \to \cdot \epsilon$ Reduced

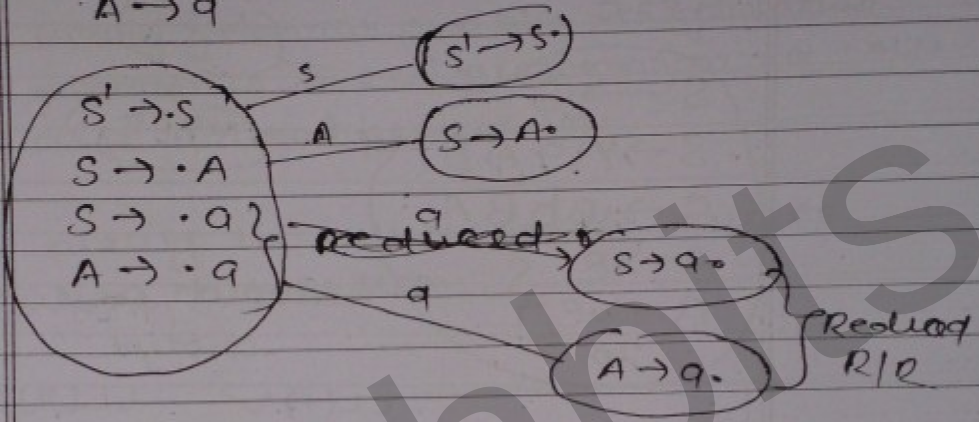
I_0

Not LR(0) Shift Reduced conflict occur.

#1

$S \to A \mid a$

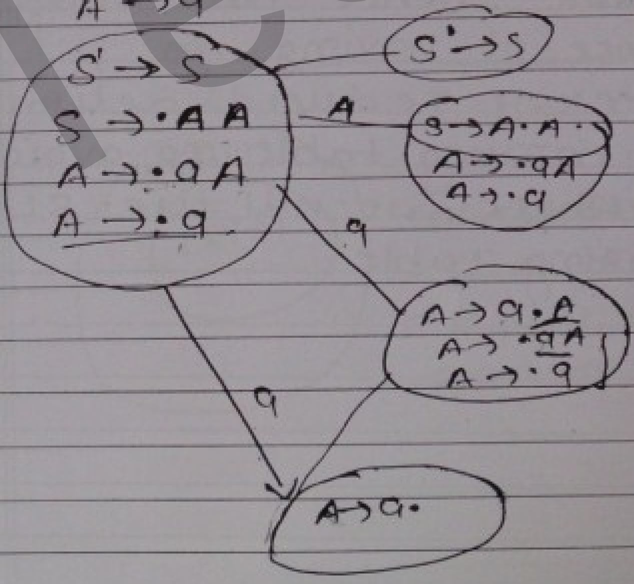
$A \to a$



Q1 $S \to AA$

$A \to aA$

$A \to a$

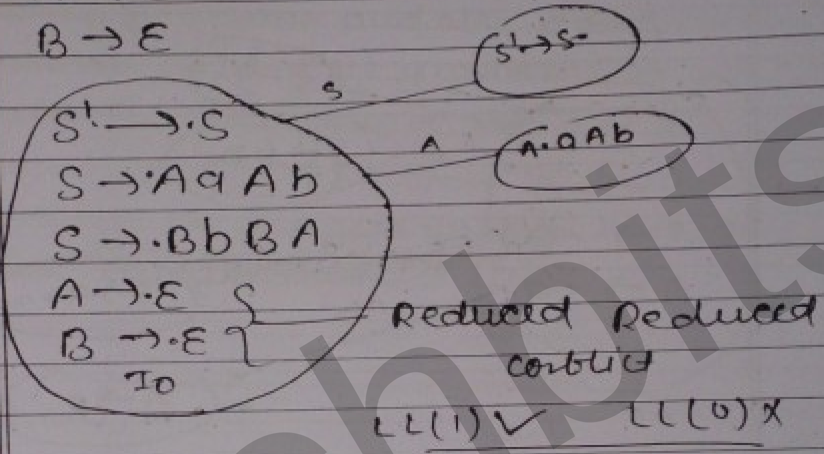


Page No.
Date : / /

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$



<u>NOTE</u> :- In LR(0) "0" indicate no look ahead concept is used to place the reduced action. Hence there may be chance of having unecessary reduced action in the parsing table. To avoid this drawback. we will use SLR(1) parsing table.

Page No.
Date : / /

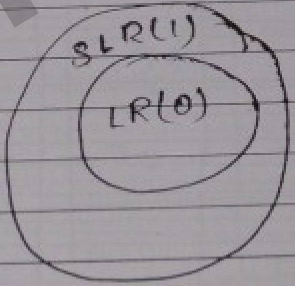
SLR(1) = simple LR(1)

construct SLR(1) parsing table for the following grammar and also verify whether grammar is SLR(1) grammar or not. Also check it ambigous or not.

As placing the reduced enetry base on Follow calculation.

I_0
I_1
I_2
I_3
I_4
I_5

#



power SLR(1) > power LR(0)

and it is unambigous grammar.

Page No.

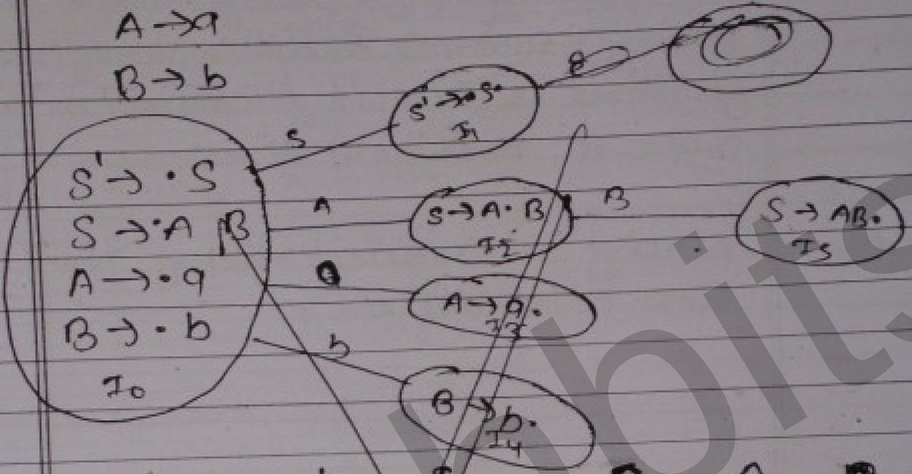
Date : / /

Q1 Check whether grammar is SLR(1)

$S \to AB$

$A \to a$

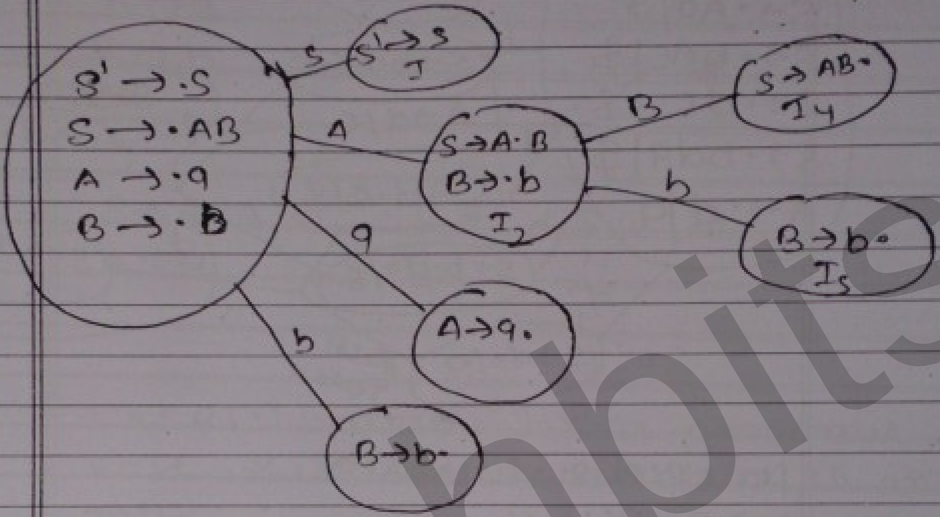
$B \to b$



	a	b	$	S	A	B
I0	A→a	B→b·			2	3
I1						
I2						5
I3						
I4						
I5						

Page No.
Date : / /

S → AB
A → a
B → b

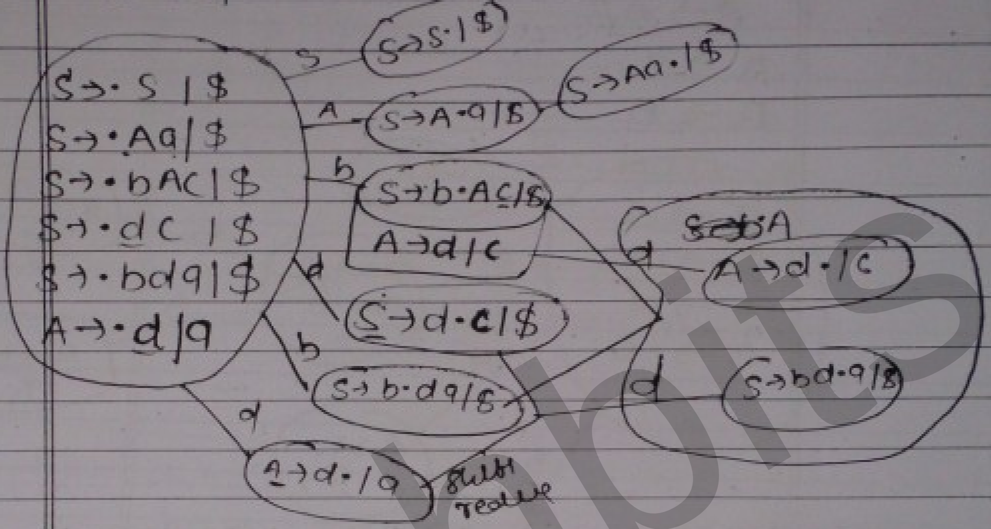


	a	b	$	A	B
I_0	S3			1	2
I_1					
I_2		S5			4
I_3		r2			
I_4			r4		
I_5			r3		

Page No:
Date : / /

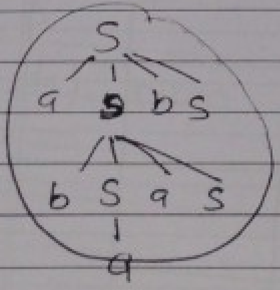
NOTE:- context free are not regular but set of all LR(0) item are regular.

Q) $S \to Aa \mid bAc \mid dc \mid bda$
$A \to d$



for LR(0) shift reduce
for SLR(1) ×
So

$S \to aSbS \mid bSaS \mid \epsilon$ (abab



ambiguous gramma

Not LR(0) ×
LL(1) ×
SLR(1) ×
CLR(1) ×
LALR(1) ×

Page No.
Date : / /

Q) $S \to AaAb$
$S \to BbBa$
$A \to a$
$B \to b$ check for LL(1)

Ex $S \to A / a$
$A \to a$

$\Rightarrow$ S
|
a

$\Rightarrow$ S
|
A
|
a

Q) LALR(1) parser, for a grammer have shift reduce conflict, if an only.

1) LR(1) grammar reduce reduce conflict
2) LR(0) grammar have reduce Reduce conflict
3) LR(1) have shift reduce
4) Non of this

Page No.
Date : / /

Q) consider the following grammar and following LR(0) item

$S \to S*E$

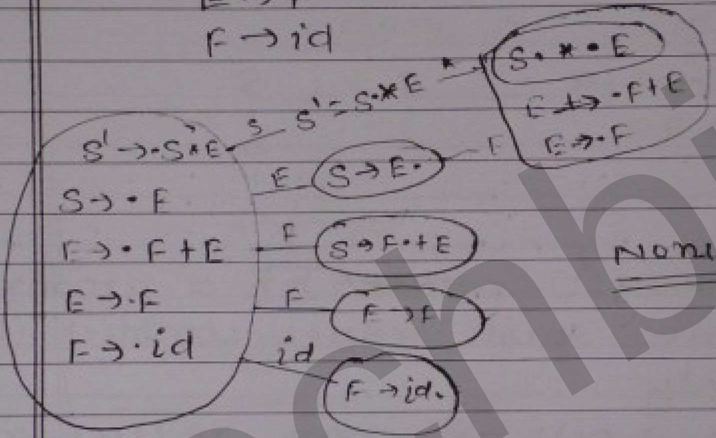
$S \to E$

$E \to F+E$

$E \to F$

$F \to id$

1) $S \to S*E$
2) $E \to F+E$
3) $E \to F+\cdot E$



None of them A

Q) consider the following grammar which of following is true.

$S \to CC$

$C \to cC \mid a$

1) This is LL(1)
2) This is LR(1) but not LL(1)
3) This is LL(1) but not LALR(1)
4) This is neither

Page No.

Date : / /

Q) consider the following grammar

$E \to E+n \mid E*n \mid n$

which of the following are the handel in the right sential form.

A $n+n*n$

$$\underbrace{\underbrace{\underbrace{\underbrace{n}+n}\times n}}_{E}$$

$E \times n$

$E + n \times n$

$n + n \times n$

handle

$n, E+n, E\times n$

Q)

Page No.
Date : / /

	LR(0)	SLR(1)	CLR(1) and LALR(1)
Shift Reduce	$A \to \alpha a B$ $B \to \gamma$	$A \to \alpha\, a\, B$ $B \to \gamma$ $follow(B) = \{ a \}$	$A \to \alpha a B$ $B \to \gamma\ \{ t_1 \}$ $if\ a = \{t_1\}$
Reduce Reduce	$A \to \alpha.$ $A \to B.$	$A \to \alpha$ $B \to \gamma$ $follow(A) \cap follow(B) \neq \phi$	$A \to \alpha, \{t_1\}$ $B \to \gamma\ \{t_2\}$ $Follow(t_1) \cap Follow(t_2) \neq \phi$

LR(1) by default CLR(1)

NOTE :- If we remove the ambiguity from the grammar lower readblity.

Removal ~~of~~ ambiguity is not advisable. Hence to handle ambiguous grammar, the suitable parsing technique is operator parsing.

Page No.
Date : / /

Operator precedence parsing

1) Ambigous grammer and unambigous

2) Simple to design.

-) we can constructor operator precedence parsing for the operator precedence grammar

operator precedence grammar is nothing but operator grammar having atmost one precedence relation b/w any pair of terminals.

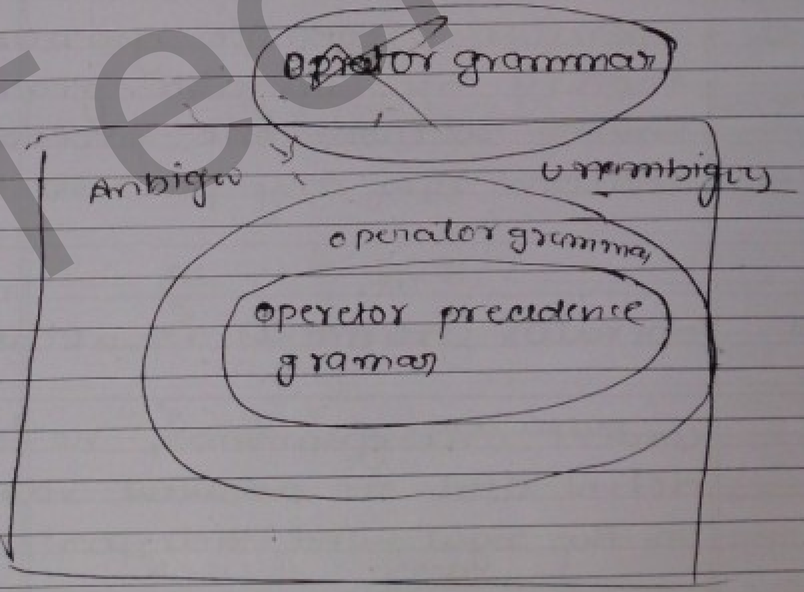
operator grammar is nothing but context free grammar, no "ε" rules define and no adjacent variable on the right hand side part.

Q) Which the following two rule, violate the requirement of operator grammar.

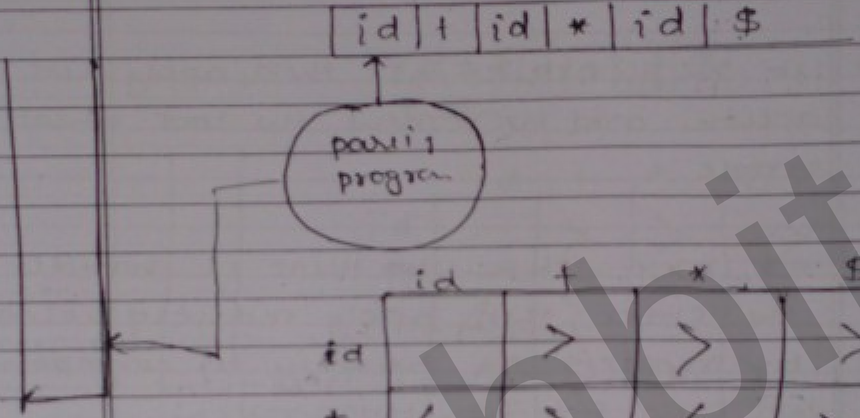
1) $P \rightarrow QaR$
2) $P \rightarrow \varepsilon$ x
3) $P \rightarrow QbRa$
4) $P \rightarrow QR\alpha$

\# ~~pre~~

Every operator precedence grammar is operator grammar but, operator grammar need not be operator precedence grammar



operator preciden parsing algo



	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	accept

operator precedence parsing table

$x < y$ } shift
$x = y$ }

$x > y$ } Reduce action

$x = y = \$$ accept }
$[x, y]$ = blank = error }

Page No.
Date : / /

Let 'x' is the topmost terminal on the stack, and 'y' is current look ahead symbol then, parser will take action based on the parsing table.

1) if $x < y$ or $x == y$ then, apply shift action and increment the look ahead single.

2) If $x > y$, then there is handle in the stack, then apply reduced action by replacing the handling by corresponding left hand side non terminal.

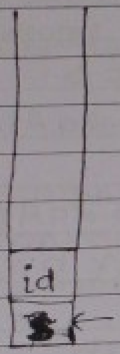
3) $x = y = \$$ input string is accepted

4) if x and y blank entry in parsing table then there is syntax error written into error handler.

Q1 How many shift action, reduce action and how many total action taken by parser.

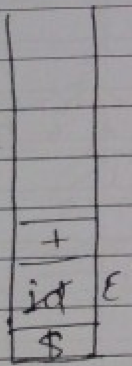
id+id*id

id	+	id	*	id	$



	id	+	*	$
id	.	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	acc

Stack	Input str	Action
$	id | + | id | * | id	Shift
$ | id	id | + | id | * | id	Redy
$ | E	id | + | id	Shb
Non termig		
$ | E | +	id | + | id | * | id	Shift



Construction of table.

Ambigous:

$+ < *$

$* \equiv /$

$- < *$

$\uparrow \; > *$

Unambigous gramar

1) $a = b$

$S \rightarrow \alpha a A b \beta$

2) $S \rightarrow \alpha a A \beta \Rightarrow A < b$

$A \rightarrow \gamma b S$

3) $a > b$

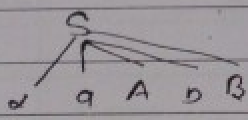
$S \rightarrow \alpha A b \beta$

$A \rightarrow \gamma a S$

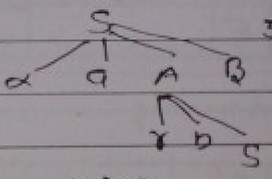
Note

Dollar($) :- is less precedence.

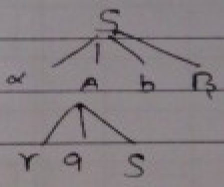
Parse tree

1) 

$a = b$

2) 

$b > a$

3) 

$a > b$

Page No.
Date : / /

If any operator grammar contains both left recursion or right recursion — it is ambigous

If operator grammar contain either left recursion or right recursion every operator present at one non terminal only the grammar is unambigous

grammar is unambigous, parsing table is constructed based on the given — grammar

If the grammar is ambigous, genral programming language precedence use to construct parsing table.

Q) construct operator precedence table
$A \rightarrow A * A \mid A + A \mid id$

Note
1) identifier is always higher precedence over the other operator
2) \$ is always lower precedence.

3) identifier, identifier ~~no~~ preceden available
[id, id] — no

Page No.

Date : / /

Page No.

Date : / /

Q) [$, $] is accep.

	id	+	*	$
id		>	>	>
+	>	>	<	>
*	>	>	>	>
$	>	<	<	accep

Q) $A \to B * A \mid B$
$B \to B + C \mid C$
$C \to D \mid C / D$
$D \to id$

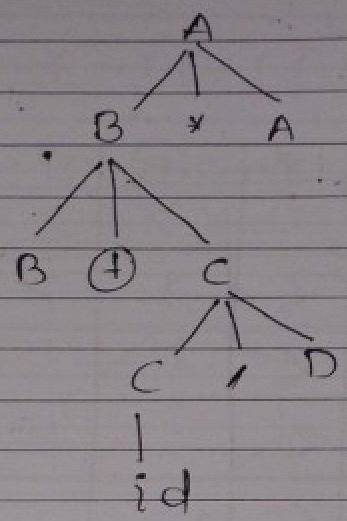
	id	+	*	/	$
id		>	>	>	>
+	<	>	>	<	>
*	<	<	<	<	>
/	<	>	>	<	>
$	<	<	<	<	accept

Page No.

Date: / /

Page No.

Date: / /



+1 A → B↑T/B

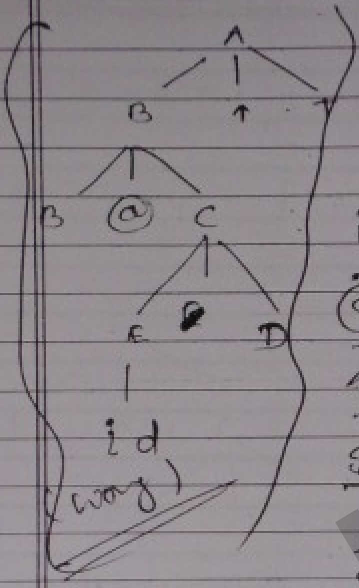
B → B@C|C

C → D&C|D

D → E-D|E

E → id

	↑	@	~~$~~	$
↑				
@				
+				
$				

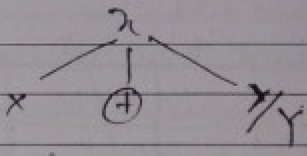


	id	@	↑	&	-	$
id		>	>	>	>	>
@	<	>	>			>
↑	<					>
&	<					>
-	<					>
$			<	<	<	ac

o) $x \to x \oplus Y / Y$

$Y \to Z^* \; Y/Z$

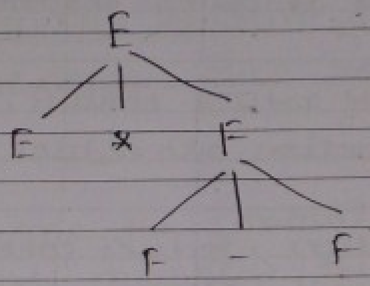
$Z \to id$



Page No.
Date : / /

Q1 $E \rightarrow E * F \ / \ F + E \ / \ F$

$F \rightarrow F - F \ / \ id$



"−" has highest precedence then + and *.

Q11 It will not worke for partical compiler.

<u>Draback</u>

Not usefull for (−ve) operator
~~[illegible]~~ unable to diffriate uniary (−) and binary (−)
(−3)(−4) assume no operator

power of this technique is very less
it suitable for very

$E \rightarrow E+E \mid E*E \mid id$

[redacted]

<u>Note</u>: LALR(1) parser tool, handle the ambiguous grammar by resolving the conflict in the following manner,

In case of shift reduce conflict, priority is given to shift action over reduce action.

In case of reduce - reduce conflict priority is given for first reduce, production over other reduce production

<u>Q)</u> consider the following grammar.

$E \rightarrow E+E$ all is the above grammar
$E \rightarrow E*E$ fed to, LALR(1) parser
$E \rightarrow id$ tool, then which on of the following is true about the action take by the parser

1) it detect recusion an elimnate recure

2) it detect, reduce - reduce conflict, and resolve the conflict by reducing the first production over other production

Page No.

Page No.
Date: / /

3) it detect shift reduce conflict and resolve the conflict, in favour of - reduce over, shift action

4) it detect shift reduce conflict and resolve the conflict in favour of shift action over reduce action.

Q2) assume the conflict are resolved in above manner, then what precedence and associativity rules parser realize and what is the value of the expression to be evalu

3*2+1

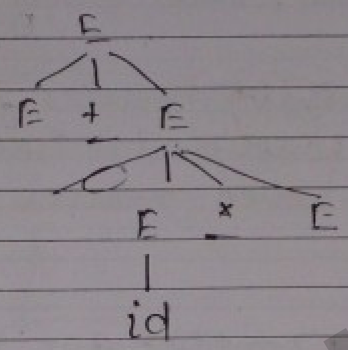
1) both + and * are same precedence and left associative expression value is '7'

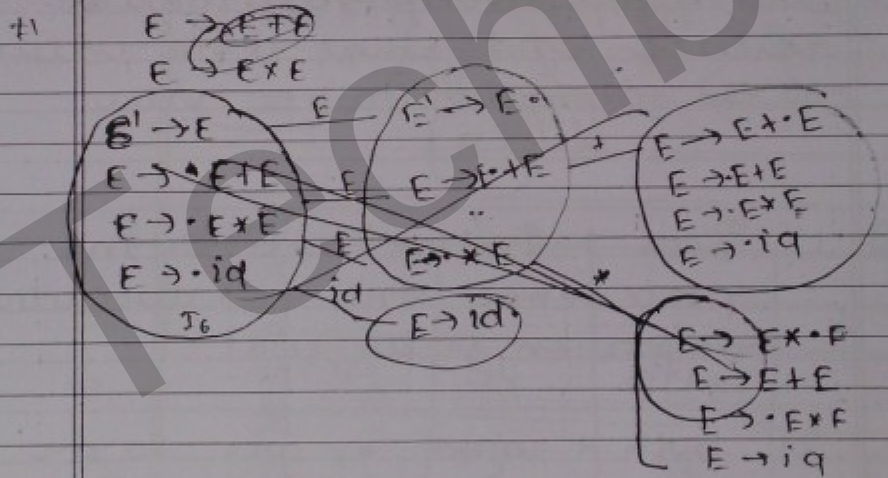
2) both + and * are same precedence and expression value is right associativ is '9'.

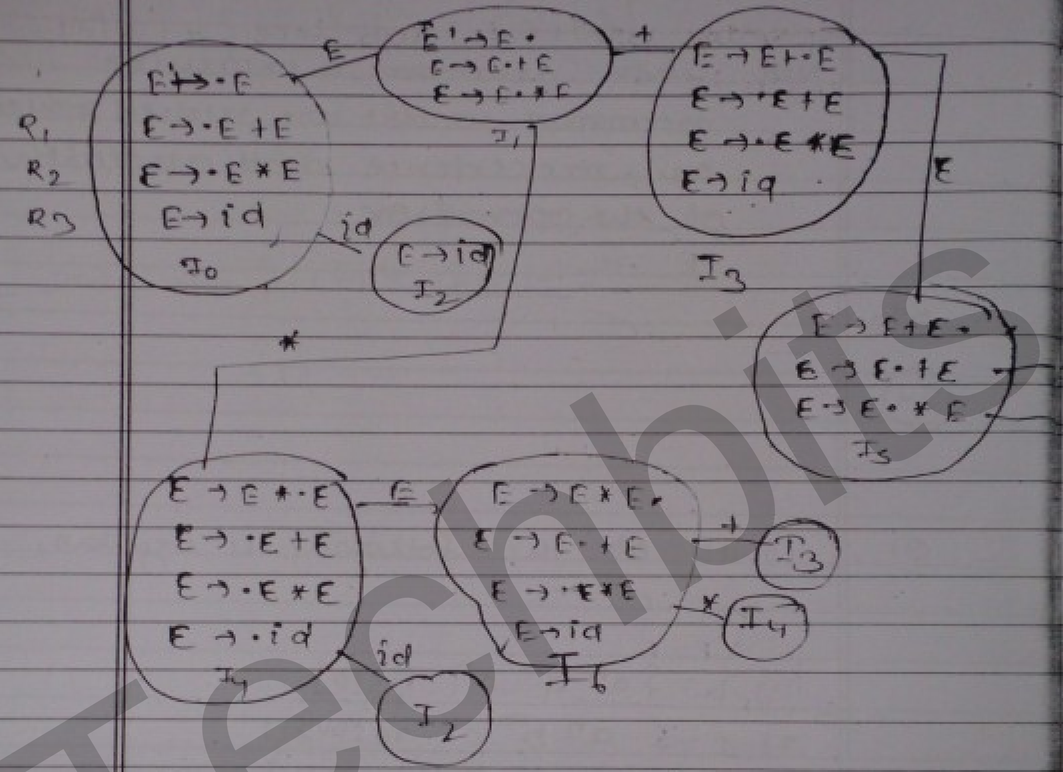
3) precedence of '*' is higher then '+' both are left associative value is '7'.

4) + , higher precedence then '*' and

operator are left associative expression value is '9'.







	id	+	*	$	E
I0	S2				1
I1		S3	S4	accept	
I2		r3	r3	r3	
I3	S2				5
I4	S2				6
I5	r1	r1	r1	r1	
I6					

Page No.
Date : / /

Note given grammar is expression grammar conflict are resolved based on, precedence and associativity of the operator.

$\to I_3$
$+$
$*$ I_4

Q) which of the following is regular,

1) $L = \{a^n b^n c^n \; n \le 10\}$
2) $L = \{a^n b^m c^n \; m, m \ge 1$
3) $L = \{a^n b^n \mid n - m = 4\}$
4) $L = \{a^n b^m \mid n + m$ is odd
5) $L = \{a^n b^m \mid n * m$
6) $L = \{a^n b^m \; n/m = 4\}$
7) $L = \{a^{2^n} \mid n \ge 1\}$
9) $L = \{a^{n!} \mid n \ge 1$
8 $L = \{a^{2^n} \mid n \ge 1\}$
10) $L = \{a^k \mid k$ is odd no$\}$
11) $L = \{a^P \mid P$ is even number$\}$
12 $L = \{w x w^R \mid w \in \{a, b\}$
$x \in \{a, b\}^*$

$a_1, 3, a_2$

Page No.

Date : / /

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

14) $L = \{ w x$

20) $L = \{1, 2, 4, 8, \ldots 2^n \ldots\}$

all

Page No.
Date : / /

#) Every finte language are regular.

Any infinite language contain comparision b/w symbol ~~to~~ non regular

In any language symbol power not in "a.p" non regular.

Page No.
Date : / /

```
# include <stdio.h>
Main()
{
printf(" compiler");

}
```

#1

preprocessor

⇓ — used for support the Macro

compiler

⇓ Assembly language

Assembler

⇓

Linker  [All Module are combine]

⇓

Loader

preprocessor :- preprocessor take high label code as input, and it evaluate All the macro present in the program. which include # include, # define in - c language and input statement in Java language

compiler :- compiler take preprocessor high level languag and produces assembly language as output.

Page No.
Date : / /

Assembler :- Assembler take assembley language as input and produce relocatable machine code (The machine code which can be moved in different place in memory).

Linker :- All the diffrent module of program are compile seprately since to get output of our program. All different module can be made into single module the task is done by the linker. Hence linker is a program that resolve all external refrences required for the program to execute. Which include machine code of predefine function (printf, scanf)

Loader :- Loader is a program that loads executable machine code from secondary memory into main memory.

Page No.
Date : / /

#

```
Main()
{
  printf("Hello");
}
```

Diff compile

C compiler is a translator i.e it translate one kind of language into another type of language

I Interpreter is a language processor which take the language as input and execution done directly.

C Total program is translate at a time in compiler.

Line by line execution will be done in interpreter.

compiler takes less time in compilation of program compare to interpreter.

where as interpreter take more time in the interpretation of language bcoz line by line execution done.

HLL → [pass 1] → [pass 2]
⇓
o/p

H **pass** :- one complete scan of the given ~~high~~ high lable language called pass.

H All Modern, compiler are multipass compiler beacuse in multipass optimization of program perform

H **phase** :- The Sub module of each pass is known as phase.

Page No.
Date : / /

Front end:- The phase of compiler which depend on source language and independent on target machine.

Backend:- The phase of compiler depend on target machine or independent on source.

Back:- code optimization, code generation

Front:- lexical, syntax, semantic, intermediate

Q:2)
7) b), 8) b) 9) b, 10 a) 12) b)

13) b, 14) b, 15) c, 16) b), 17) a)

Page No.
Date : / /

\# $S \to aABe$
$A \to Abc \mid b$
$B \to d$

a AcBe
ε, a A, aA, Ade a A Be
ε, a, abcde a Abcde
viable prefixe a b bcde
ε, a, ab

The prefix of handle left hand side in right sentetial form are called as "viable prefix".

\#

$S \to AB$
$S \to CA$
$B \to BC$
$B \to AB$
$A \to a$

\# NOTE

⇒ In SLR(1), LALR shifted entries are same.
⇒ Go to part is same and

reduced entries may be different

New Chapter

Page No.
Date : / /

the following are the semantic error

1) using variable, without declaration

2) Re-declaration of multiple variable

3) scope variable rule.

4) in compatible type expression

10) Mismatch of formal actual parameter.

11) Type mismatch in formal and actual parameter.

Context Sensitive grammers

↳ Attribute grammer will used.

CFG + {Sementic rule}

used to handle the Sementic analysis phase.

Page No.
Date : / /

Syntax directed translation is formalization in which each grammar production is associated with set of sementic rules.

1) functionality of (SDT)

1) It identifies all sementic errors and issues error message to error handeler

2) It helps ~~the construction of~~ in construction of intermidiate table

3) Symbol table construction

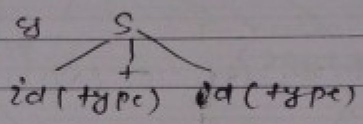
4) Construction of syntax ~~state~~ tree.

Designin SDT

1) construct parse tree
2) contruct annoted parse tree
3) compute attribute values in annoted
parse ~~(4) compute annoted value~~ tree.
5) The genralize rules of attribute computation attached to grammar prodution which known as sementic rules.

Page No.
Date : / /

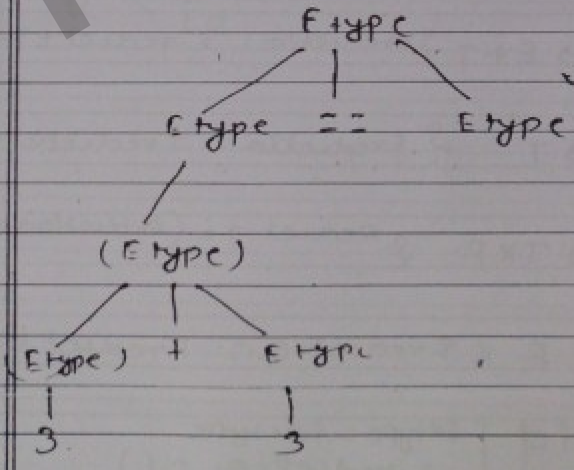
Annoted parse tree : Decorate parse tree.



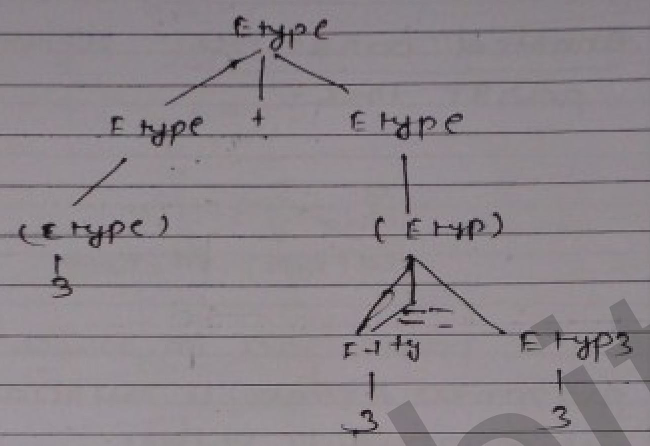
⇒ Is a parse tree in which each grammar symbol is associated with set of attribute values.

E → E + E	[if (E₁.Typ == E₂.Type) && (E₁.Type : int) then E.type = int else other()]
E → (E)	[E.Type = E₁.type]
E → E == E	[if (E₁.Type = E₂.Type) && E₁.Type = int / boolean then, E.type = boolean]
E → Num	[E.TYPE = num type]
E → true	[E.Type = boolean]
E → false	[E.Type = boolean]

(3+3) == 6



Q1 3+(3==3) K



H Construct the SDT that ~~w~~ would give total no. of reduction take by bottom up parser, in construction of parse tree.

red is attribute and E.red = Total no. of reduction take by the parser. the construct SDT. by follow grammar.

E → E+T {E.reduction = E.redect + T.redect + 1}

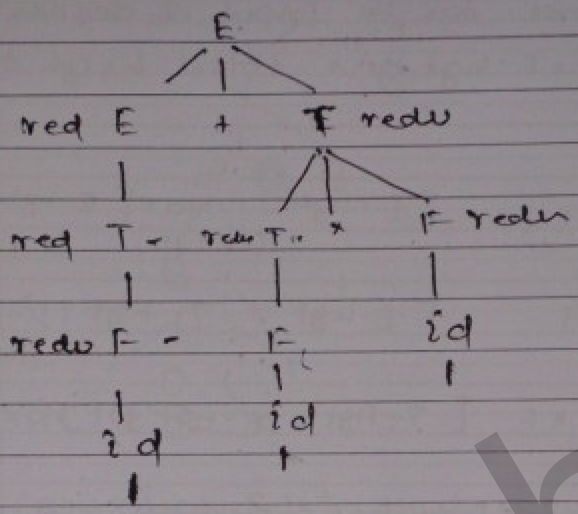
E → T {E.reductio = T.reduction + 1}

T → T*F {Redect T = (Reduction T + ReductioF) + 1}

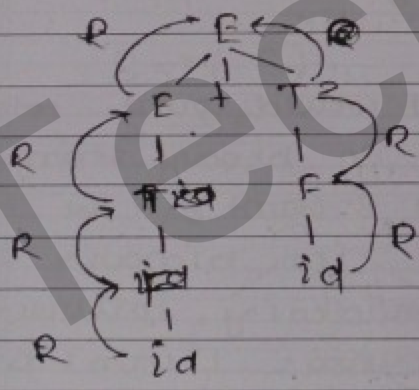
T → F (F.reduction = F.redect + 1)

F → id [E.type = id.type
F.reduction = 1)

Page No.
Date : / /



#1 Reduct to a+b



Page No.
Date : / /

H construct SDT that would calculate height of parse tree. for the input is derives. where E.hgt, E.hgt give total heigh of parse m

E → E+T [E.hgh = Max[E.hgh, T.hgt]+1]

E → T [E.hgt = T.hgt+1)

T → T×F [T.hgt = ~~F.hgt+1~~ Max[T.hgh, F.hgh]+1

T → F [T.hgt = F.hgt+1

F → id [F.hgt = 1

Q) construct that would count, total no. of 1's present in binary string. where count is attribute S.count give the total no. of 1's in binary string by considering following grammar.

[S.count = L.count] S → α

α → αB [L.count = L.count + B.count]

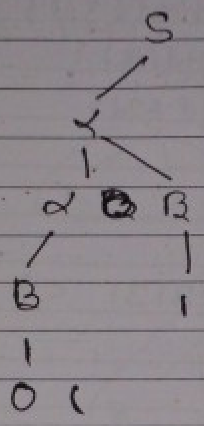
α → B [α.count = B.count + ~~0~~,

B → 1 [B.count = 1)

B → 0 [B.count = 0

11001

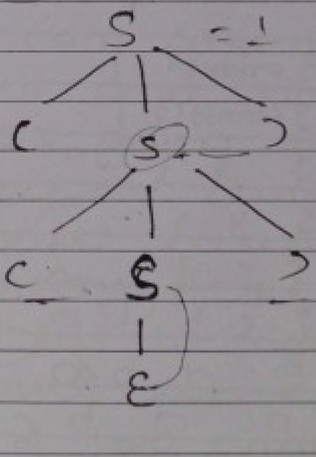
Page No.
Date : / /



Q) Construct SDT that would give total no. of balanced parenthesis present in input string

$S \to (S)$: $S.count = S.count + 1$

$S \to \varepsilon$: $S.count = 0$



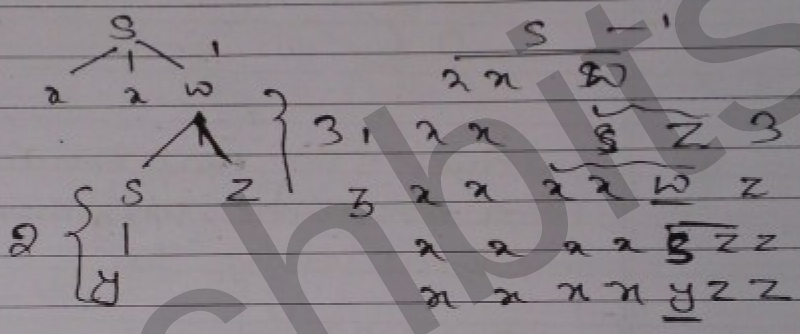
((()

Page No.
Date : / /

\# $S \to xxW$ { printf (1) }
$S \to Y$ { printf(2) }
$W \to Sz$ { printf (3) }

xxxxyzz

a) 12313 ~~b)~~ 23131 c) 13112 d) None



2
2 3 1 3 1

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

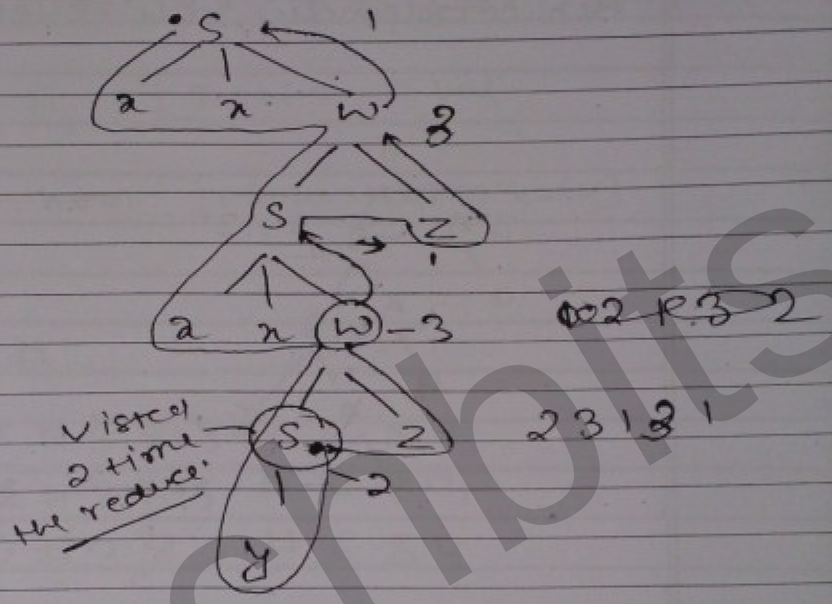
a) 666453211
b) 33364521
c) 333641251
d) None

a a a d b c
A A A d b c a a a d b c
A A a d b c = 3
A a a d b c = 3
a a a d b c

Page No.

Page No.
Date : / /

Top down pars (left most)



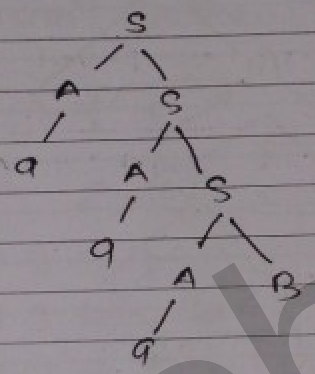
parse tree creat:

Note

SDT can give to top dow parser and botton up parser if SDT given to top down parser, top down parser construct parse tree by traversing "DFS" traversal from left to right. In DFS traversal when ever node is visited two time corresponding rule is executed.

Page No.
Date : / /

if SDT is given to bottom up parser, when ever reduced action it ~~sta~~ happing then corresponding rule is executed.



Page No.
Date : / /

Q) $E \rightarrow E_1 \# T$ {$E.val = E_1.red + T.val$}

$E \rightarrow T$ {$E.val = T.val$}

$T \rightarrow T_1 \& F$ {$\cancel{T}.val = T_1.val * F.val$]

2 $T \rightarrow F$ [$T.val = F.val$

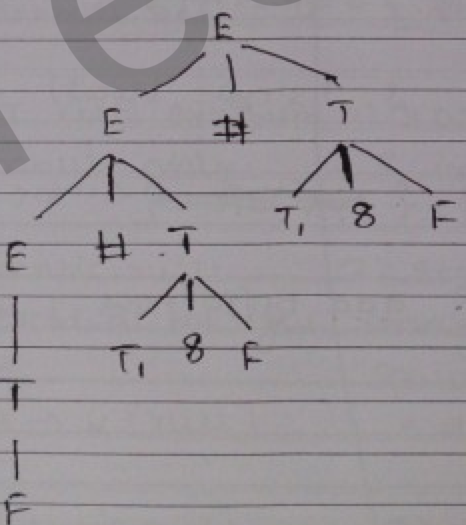
2 $F \rightarrow num$ [$F.valu = \cancel{num}.value$]

~~2~~ #

T # 3 8 5 # 6 8 4

E # 3 8 5 # 6 8 4

2 2 # 3 8 5 # 6 8 4



Page No.
Date : / /

Synthesized attribute :- The attribute value at a node is computed in terms of its children in the annoted parse tree such type of attribute known as synthesized attribute
(well suitable for = bottom up)

Inherited attribute :- The attribute value at a node in the anoted parse tree, is computed either from parent, or from the left sibling is known as inherited attribute.
(suitable = ~~post~~ top down)

#

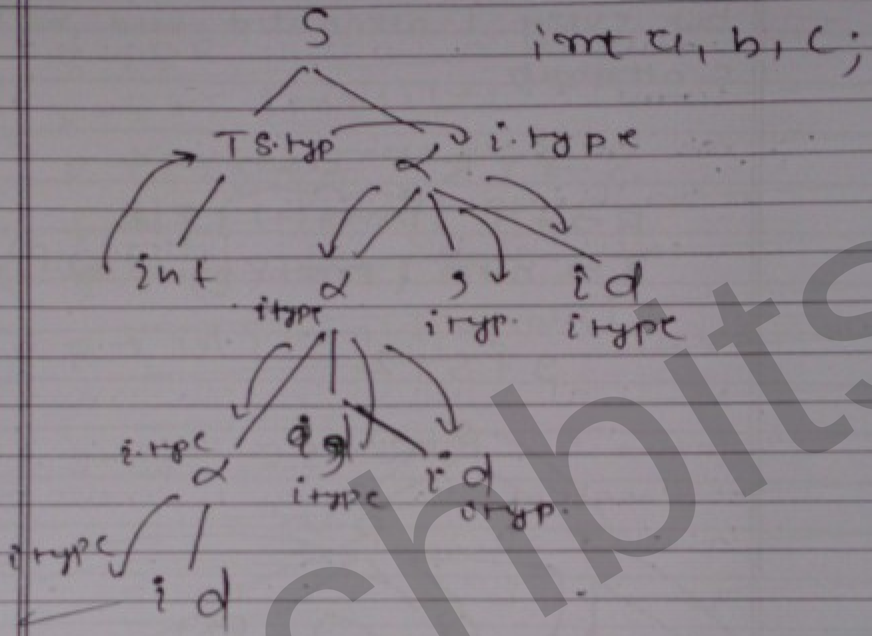
S → T L { L.itype = T.stype }

T → int { T.stype = int }

T → float { T.stype = float)

L → L₁, id { ~~L1.itype~~ { L₁.itype = L.itype }

L → id { add ty(id.name, L.itype



SDTS

Q)

	S-attributed Def	L-attributed
1)	SDT uses synthesized attribute only	1) SDT uses synthesized & inherited
2)	Sementic rules present at the right end $A \to \alpha$ [rule]	2) Semantic rules may present in the middle also. $A \Rightarrow \alpha$ {rule} B
3)	Sementic rule evaluates in bottom-up parser	3) Rules are evaluated in DFS.

Page No.

Date : / /

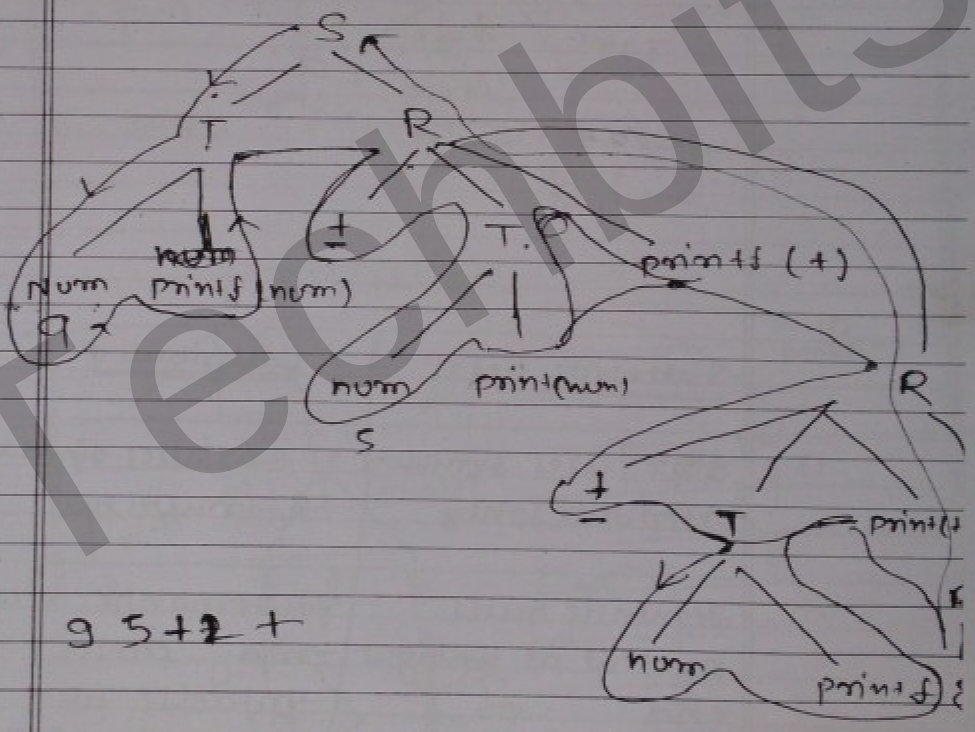
Q) Every S'attributed Def is L-attributed Def but every L-attributed need not to be S attribute.

S → T R $\underbrace{\text{L-attributed.}}$

R → + T { print (+) } R | E

T → num { printf (num value)}

9 + 5 + 2



9 5 + 2 +

Page No.

Date : / /

Consider the following SDT

$S \to E R$

$R \to * E \{print(*)\} R \mid \epsilon$

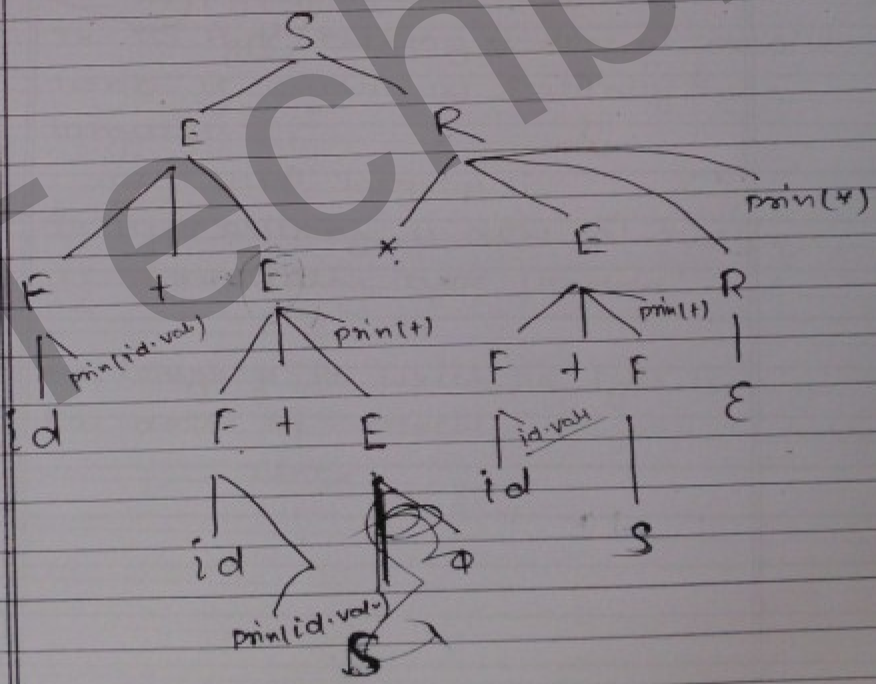
$E \to F + E \{print(+)\}$

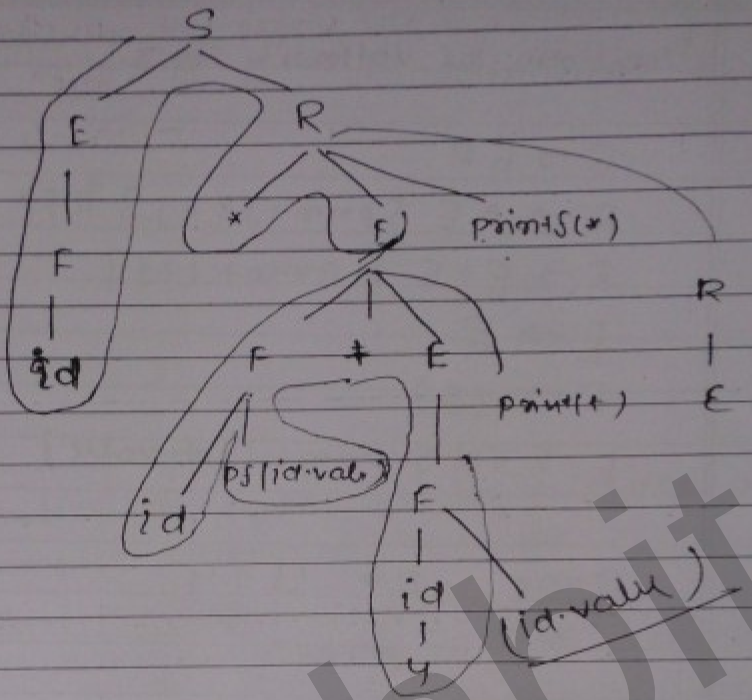
$E \to F$

$F \to (S)$

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

String 2* 3+4




S
E
R
F
id
*
F
printf(*)
R
F
+
E
printf(+)
E
id
pf(id.val)
F
id
y
(id.val)

Page No.
Date: / /

Construct the STD that perform type conversion by consider. type conversion done in integer and real value.

E → E+T
E → T

Statically type checking will done at compile time and runtime.
eg C language

If the type checking is done at compile time ie known as statically type-language.
eg Lisp language

If type checking is done at runtime, ie dynamically type language.

The language in which no type checking will done. eg = Machine language, Assembly language