कुशाग्र : मिन्तल :

Page No.
Date : / /201

ॐ

21-07-21

COMPILER DESIGN

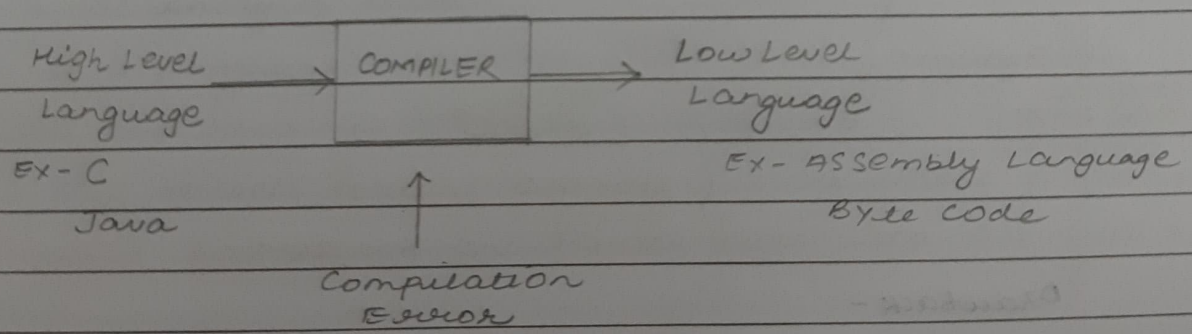
[6-8 MARKS]

Lexical Analysis → 10%
Parsing OR Syntax Analysis → 60%
Syntax Directed Translation → 20%
Intermediate Code Generation
Code Optimisation (Data flow Analysis)
Runtime Environment
Local Optimisation
Data flow Analysis
Constant Propogation
Liveness Analysis
Common Subexpression Elimination.

} 10%

Introduction to Compiler Design

Compiler is a software which converts a program written in high level language (source language) to low level language (Object / Target / Machine Language).



Tasks:-

Translation

Error Detection

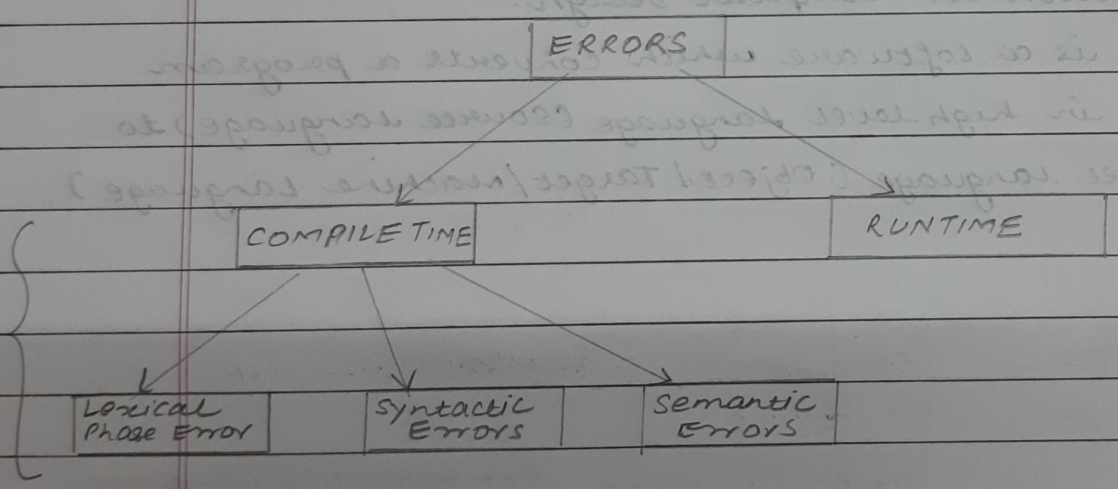
Other translators:

H.LL → Pre Processors → PreProcessor HLL

Assembly → Assembler → Machine Code

Machine Code → Linker → Executable Code

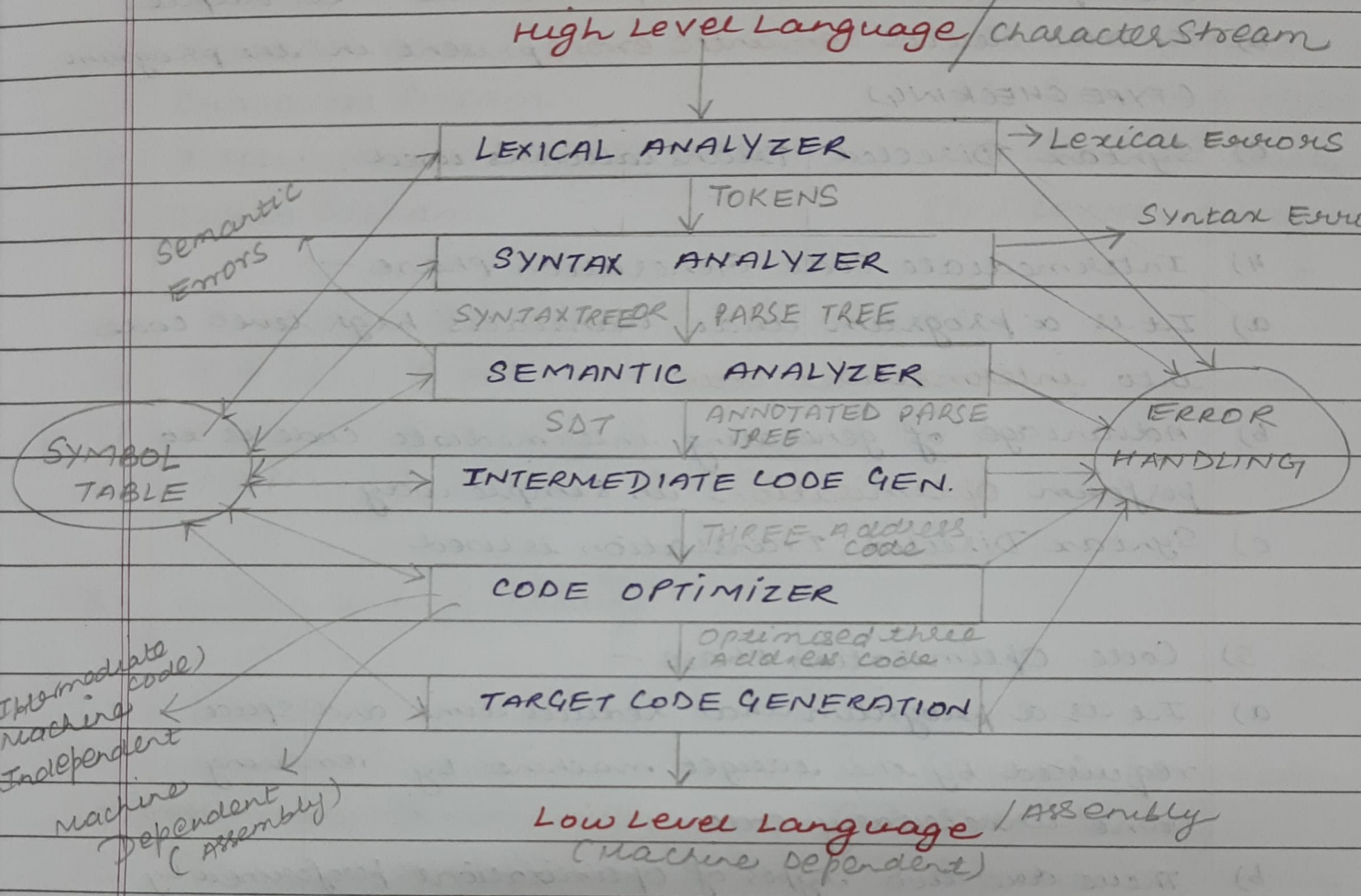
★ Errors detected by Compilers



Drawback -

Compiler cannot detect runtime errors

PHASES/MODULES OF COMPILER



* 1) Lexical Analysis Phase -

a) It is a program that takes high Level Language as input and produces tokens as output

b) It also detects lexical errors present in the programs

c) Regular Expression and Finite Automata are used to design.

* 2) Syntax Analysis Phase -

a) It is a program that takes tokens as input and produces parse tree as output.

b) It detects Syntax errors present in the program.

c) CFGs and PDA is used.

* 3) Semantic Analysis phase –

a) It is a program that takes parse tree as input and produces annotated parse tree as output.

b) It also detects Semantic errors present in the program (TYPE CHECKING)

c) Syntax Directed Translation is used.

* 4) Intermediate Code Generation Phase –

a) It is a program that translates high level code into intermediate code.

b) Advantage of generating intermediate code is to perform optimisation in simple way.

c) Syntax Directed Translation is used

* 5) Code Optimisation Phase –

a) It is a program that reduces time and space required by the target machine by removing some unnecessary code. Data Flow Analysis is used

b) There are two types of optimizations performed by compilers known as machine independent optimisation and machine dependent optimisation.

Intermediate code

Assembly language

* 6) Code generation Phase –

It is a program that translates optimised intermediate code into assembly language or target code. Standard Algorithms are used.

NOTE!–

PHASES 1, 2, 3 ⇒ Error Detection Phases

PHASES 4, 5, 6 ⇒ Translation Phases

Q. Match the following according to input

	LIST I	LIST II
(P)	Syntax tree	(i) Code generator
(Q)	Character Stream	(ii) Syntax analyser
(R)	Intermediate Representation	(iii) Semantic analyser
(S)	Token Stream	(iv) Lexical analyser

a) P → (ii), Q → (iii), R → (iv), S → (i)
b) P → (ii), Q → (i), R → (iii), S → (iv)
✓ c) P → (iii), Q → (iv), R → (i), S → (ii)
d) P → (i), Q → (iv), R → (ii), S → (iii)

Q. Match the following

	LIST II	LIST II
(P)	Lexical analysis	(i) Leftmost Derivation
(Q)	Top down parsing	(ii) Type Checking
(R)	Semantic analysis	(iii) Regular Expressions
(S)	Runtime Environments	(iv) Activation records

a) P → (i), Q → (ii), R → (iv), S → (iii)
✓ b) P → (iii), Q → (i), R → (ii), S → (iv)
c) P → (ii), Q → (iii), R → (i), S → (iv)
d) P → (iv), Q → (i), R → (ii), S → (iii)

Q. TYPE Checking is normally done during
a) Lexical analysis
b) Syntax analysis
✓ c) Sytax directed translation
d) Code Optimization

Symbol Table -
(Repository)

It is a data structure that contains information about identifiers and constants present in the program.

Value	Token	Type
a	id	int
b	id	float
c	id	long
d	id	double

* Generally implemented using Hash Table. [Search time O(1)].

Error Handler -

If any phase of compiler detects errors then it is stored in error handler.

E.g

```
main()
{
    int 1xyz; - Lexical Error
    a=b+c - Syntax Error
    d=e+f; - Semantic Error
}
```

Page No.
Date : / /201

Q. Which datastructure in a compiler is used for managing information about variables and their attributes?

a) Abstract syntax tree
✓ b) Symbol table
c) semantic stack
d) Parse table

I/P Source code
↓
Lexical Analyser → Stream of Tokens
↓
Syntax Analyser → Abstract Syntax Tree
↓
Semantic Analyzer → Parse Tree
↓
Intermediate code Generator → Intermediate code
↓
Code Optimiser → Optimised Code
↓
Target Code Generator.
↓
Target Code O/P

ANALYSIS

SYNTHESIS

Symbol Table

Error Handler

Classification of Compiler Phases

Front End

Back End

Front End

Front end comprises of phases which are dependent on the input (source language) and independent on the target machine (target language).

Lexical Analysis
↓
Syntax Analysis
↓
Semantic Analysis
↓
Intermediate Code Generation

Back End

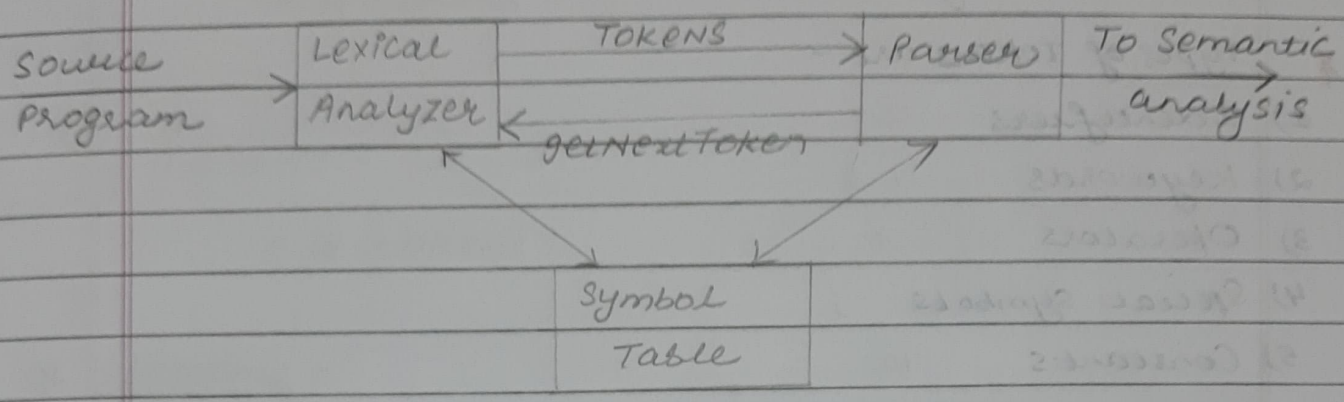
Back end comprises of those phases of the compiler that are dependent on the target machine and independent on the source language.

This includes :

- Machine dependent optimisation
- Code generation

LEXICAL ANALYSIS PHASE

(Regular Expressions and Finite Automata is used)



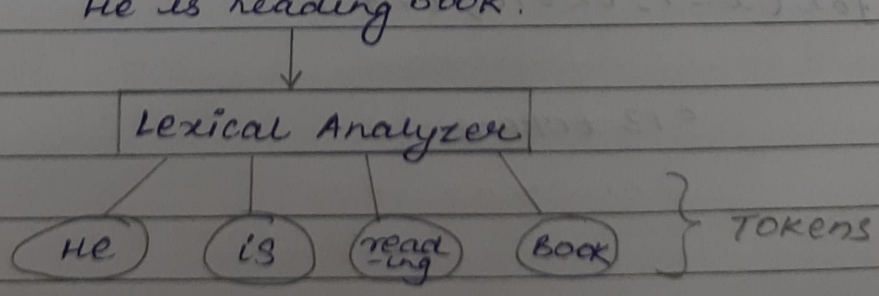
Role of Lexical Analyzer -

(i) Generating Tokens
(ii) Detecting Lexical errors
(iii) Removing all white spaces/ characters
(iv) Removing all comments
(v) Constructing Symbol table
(vi) Sending tokens to Parser.
(vii) Reads sequence of characters from source programs

Lexical Analyzer is called Scanner.

E.g

He is reading Book.



* **Token :-**

Token is a sequence of characters that can be treated as a single logical entity.

* Types of Tokens are -

1) Identifiers a
2) Keywords int
3) Operators *
4) Special symbols ;
5) Constants 10

Examples-

1) int a=10, b=20;
1 2 3 4 5 6 7 8 9

= 9 tokens

Keyword = int
Identifiers = a, b
Constants = 10, 20
Operators = '='
Special symbols = ; ,

2) for (i=1; i<=10; i++)
1 2 3 4 5 6 7 8 9 10 11 12 13

= 13 tokens

* **Pattern -**

Rules of token. A set of strings in the input for which the same token is produced as output.

3) if(a > b)
1 2 3 4 5 6

return a; /* Computer */
7 8 9

Removed by LA

= 9 tokens

4) a = b + c * d;
1 2 3 4 5 6 7 8

= 8 tokens

* **Lexeme** -

A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

* actual characters of program = lexeme
It's type = Token

E.g int a, b;

Lexeme

Keyword (Token)

int is lexeme of keyword type token.

; is a lexeme of token type special symbol

E.g ⇒ int 1xy3;

Not lexeme

invalid token

(Not matching with any token type)

Lexical error is produced

* If a group of characters is not lexeme ⇒ Lexical Error is produced

22-07-21

Lexical Analyzer Design

Two ways -

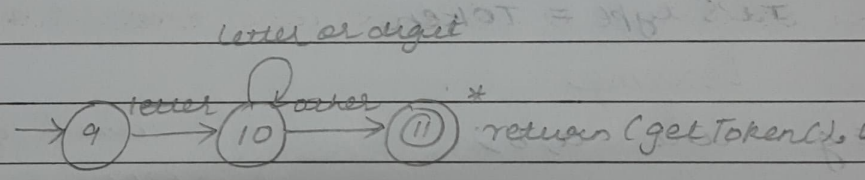
1. Handwritten code (Manual)
2. Tool based design

* Specification of token
Regular Expression

* Recognizing Tokens
Finite Automata

Example -

Finite Automata for keywords and identifiers



Regular Expression : letter(letter + digit)*

Tokens

I.D → RE → F.A

KW → R.E → F.A

OP → R.E → F.A

Constant → RE → FA

Special Symbols → RE → FA

Lexical Analyzer = Finite Automata

Language accepted by lexical Analyzer = Regular Expressions

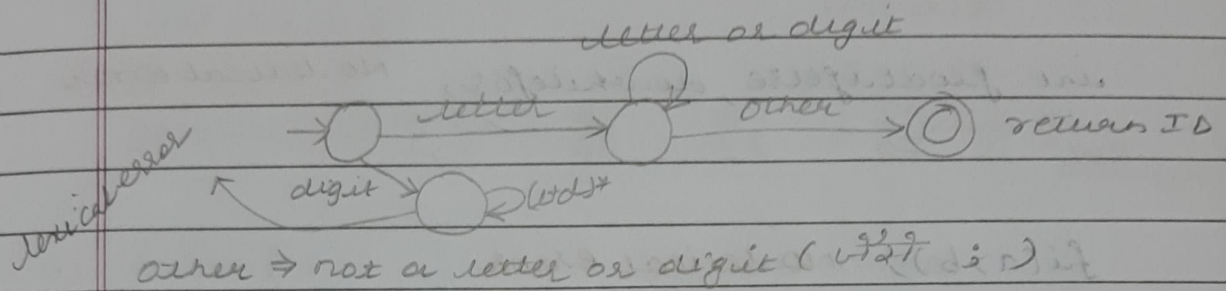
Lexical Analyzer ⇒ Regular Expressions

Page No.
Date : / /201

Example -

* Lexical Analyzer for Identifiers -

Regular Expression = $l(l+d)^*$



* Lexical Analyzer for Keywords -

Keyword = reserved identifiers

Same automata is used as that of identifiers. In addition to this, a keyword table is used.

Suppose int ab :
After reading 'int' the control goes to final state.
The compiler does not produce token immediately.
It first searches in the keyword table. If the search is success then keyword token otherwise identifier token.

Search operation differentiates between keywords & identifiers

E.g ⇒ int a,b ; NO Error is detected.
ID

a=b ; + *c ; NO Lexical Error

Lexical Analyzer $= l(l+d)^*$

Page No.
Date : / /201

int1 a, b; No lexical error.
However, there is a compile time error

int a = "Hello"; No lexical error
Keyword → Id

int floatifelse, dowhilefor; No lexical error
KW ID S.S ID S.S

fi(a>b); No lexical error
ID

fro(i=0; i<10; i++) No lexical error

GATE PYQ
2 MARKS

Q. Consider line number 3 of the following C-gram

```
int main () {          Line 1
  int i, n;            Line 2
  fro(i=0; i<n; ++i);  Line 3
}
```

Identify the compiler's response about this line while creating the object-module

a) No compilation error
b) Only a lexical error
✓ c) Only syntactic errors
d) Both lexical and syntactic errors

If on reading set of symbols you do not reach the final state then lexical errors are produced. E.g ⇒ 'A

Page No.
Date : / /201

Q. Which of the following C program having Lexical error ?

a)
```
main ()
{
int a,b,a,b;
}
```
Redeclaration of variables

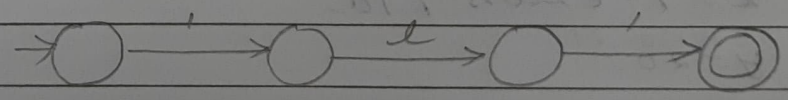
(b)
```
main()
{
int a,b;
c=a+b;
}
```
Using symbol without declaration

c)
```
main()
{
int a,b;
char c='a';
a=b+c;
}
```

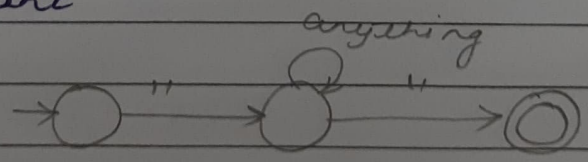
✓ (d) None

★ Lexical Analyzer for constants.

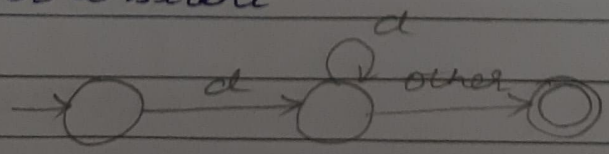
char constants



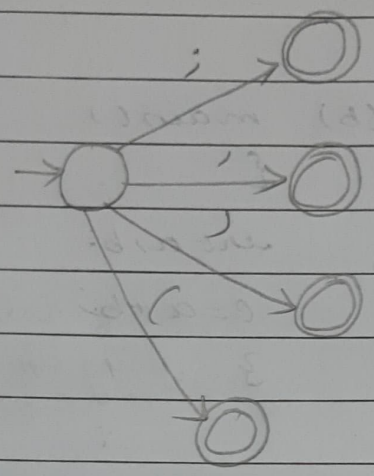
String Constant



~~Sp~~ integer constant



* Lexical Analyzer for Special Symbol



Q. How many tokens are generated by the lexical analyzer, if the following program has no lexical error?

```
main()
{
int x, y;
fl /*gate*/ oat Z;
x = /*exam*/ 10;
y = 20;
}
```

Total = 22

Q. Consider the following program segment:

```
main()
{
    int a,b;
    a = 5+8+;
    printf("%d", a);
    /* &b = 5; */
}
```

3
1
5
7
7
0
1

Total No. of tokens = 24

GATE PYQ

Q. The number of tokens in the following C statement
printf("i = %d, &i = %x", i, &i); is

a) 3
b) 26
✓c) 10
d) 21

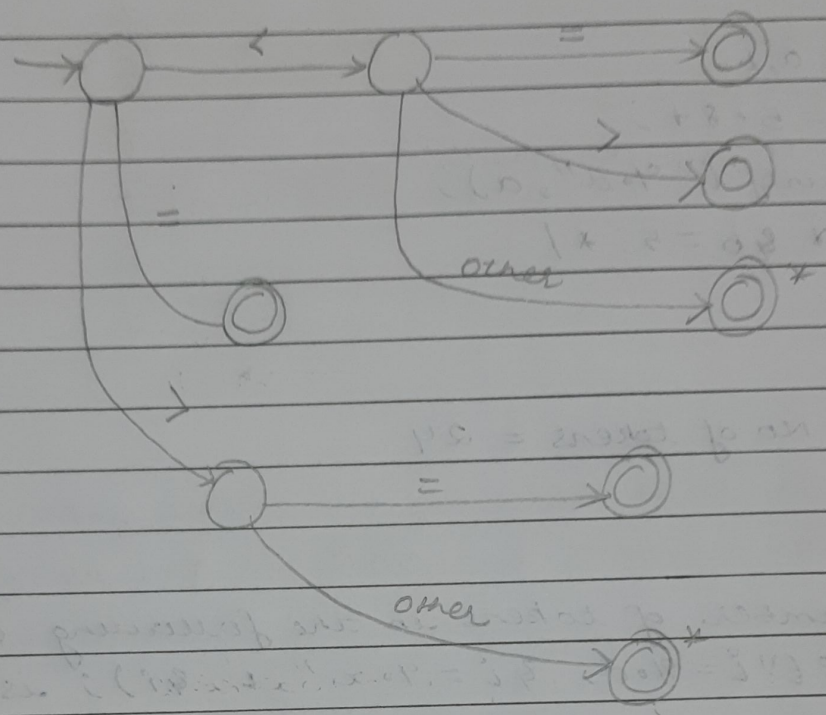
Q. Number of tokens in the following C program

```
main()
{   int x=10, *p;
    int y= x++;
    char *q;
    p= &x; q= 'A';
    if(*p>=10)
      {   *p= x+100;
      }
    else
      {   printf("%d", x);
          /* Comment */
    }
}
```

3
9
6
4
9
7
8
1
1
8
2

Tokens ⇒ 58

* Lexical Analyzer for Operator



	No. of Tokens
a=b;	4
a==b;	4
* a+b;	4
a++;	3
a<b;	4
a<=b;	4

Suppose

a	=	=	b	;
1	2	3	4	5

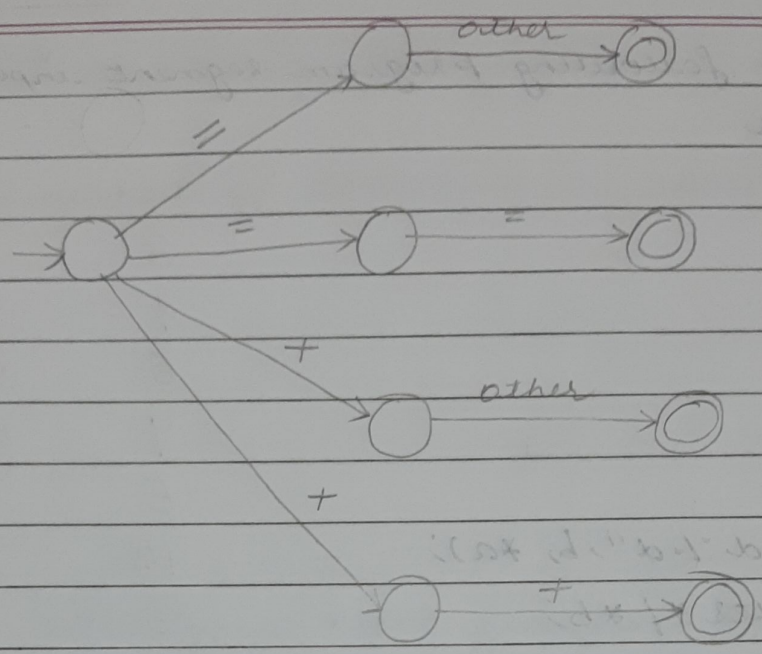
= 5 Tokens

+ +	- -	= =	*	+	;
1	2	3	4	5	6

= 6 Tokens

x	+ =	y	+ +	*	*	z	;
1	2	3	4	5	6	7	8

= 8 Tokens



for	(	i	=	0	;	i	<	=	10	;	++	i	)
1	2	3	4	5	6	7	8	9	10	11	12	13	

= 13 tokens

x = x+y ; 6 Tokens x += y ; 4 Tokens

int *p ; ⇒ 4 Tokens.
1 2 3 4

NOTE:-
pointers logic is not known at lexical Analysis phase.

Q. Consider the following program segment input to the compiler

```
main()
{
int *a,b;
a=10;
a= &b;
printf("%d %d",b, *a);
b = /*pointer*/ *b;
}
```

3
1
6
4
5
10
5
1

The number of tokens in the above C code are 35

GATE PYQ
2MARKS

Q. A lexical analyzer uses the following patterns to recognize three Tokens T_1, T_2, T_3 over the alphabet $\{a, b, c\}$

$T_1 : a?(b|c)^*a$

$T_2 : b?(a|c)^*b$

$T_3 : c?(b|a)^*c$

Note that 'x?' means 0 or 1 occurrence of the symbol x. Note also that the analyzer outputs the token that matches the longest possible prefix. If the string bbaacabc is produced/processed by the analyzer, which one of the following is sequence of tokens it outputs?

a) $T_1 T_2 T_3$

b) $T_1 T_1 T_3$

c) $T_2 T_1 T_3$

d) $T_3 T_3$

b b a a c a b c

T_3 T_3

GATE PYQ

Q. If(z= =a) function1 (10);

The number of tokens in the above statement are

a) 9
✓ b) 11
c) 10
d) 12

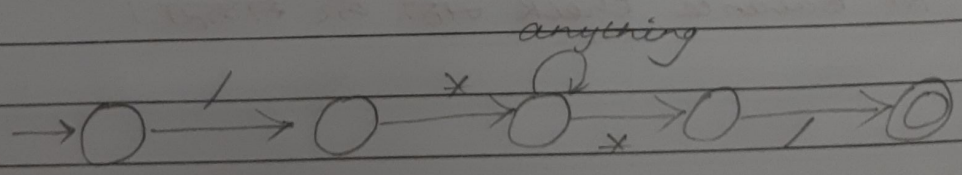
Q. Find the type of error produced by the following C code.

```
main()
{
    in /* comment t x;
    floa /* comment */ t gate;
}
```

Total Tokens = 9

a) Lexical error
✓ b) Syntax error
c) Both (a) and (b)
d) None

* Comment lexical Analyzer



After opening comment, closing comment is searched upto last.

/*_____ ⇒ Priority to Comment
_____ */ ⇒ Priority to operator

Page No.
Date : / /201

Q. int comment */;

5 Tokens

Q. /*comment;

Lexical error

Q. which of the following C Program statements have Lexical error.

(i) int a = "hello";

No lexical error

(ii) int a = "hello;

Lexical error

(iii) main ( ))
{
a = (b+c));
}

No lexical error.

FA does not have memory
तो Balance check नहीं कर सकता।

Q. Number of tokens in the following C program

abcd + (2 - 5 + x 6 / 2 - + ;

abcd	+	(	2	-	5	+	x	6	/	2	-	+	;
1	2	3	4	5	6	7	8	9	10	11	12	13	14

⇒ 14 Tokens

Q. Which of the following is not a token in C language?

a) Semicolon – Special Symbol
b) Identifier
c) Keyword
✓ d) Whitespace

Q. Number of Tokens in

int a, b, c; ⇒ 7 Tokens.

SYMBOL TABLE

Lexical value	Token	TYPE	SCOPE
a	id	–	–
c	id	–	–

Only partial information is filled at Lexical Analysis phase

* Partial Construction of Symbol Table is done. Full construction is done at semantic analyzer

Drawback of Lexical analyzer is that it cannot detect Syntax & semantic errors.

L-4

23-07-21

Q. Which of the following is not a token?

a) int
b) abc123
✓ c) #define → Executed by preprocessor
d) none

Q. Which of the following C statement has lexical error?

a) fi(z);
b) fro(a, b, c);
c) whil(a, b);
✓ d) none

Q. Which of the following is token of C language.

(a) #include
(b) #define
(c) 102.456
(d) None

Q. main () {
int x=16, y=12;
char *a;
x=1abc;
}

↓ ID should not start with digit

✓ a) Lexical error
b) Syntax error
c) no compile time error
d) none.

Page No.
Date : / /201

Most important

SYNTAX ANALYSIS (OR) PARSER

Syntax -
Structure of program statements known as syntax.

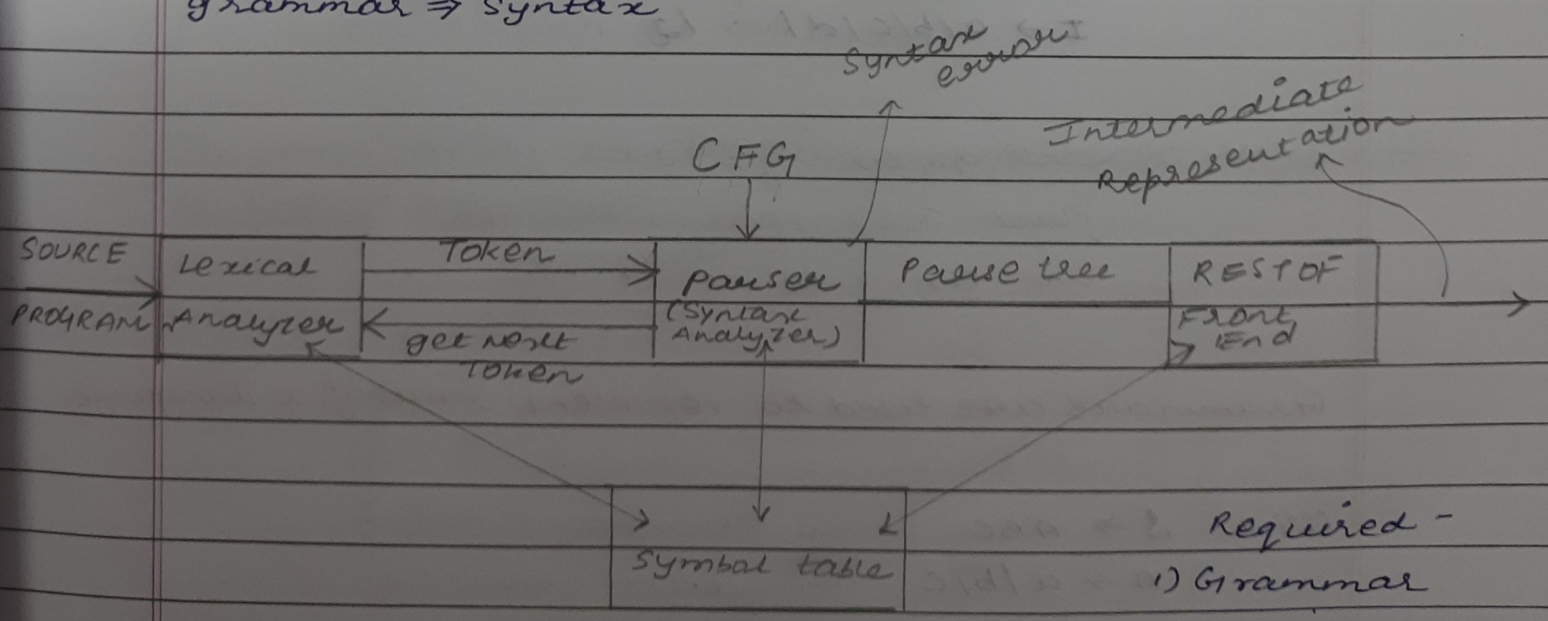
Parsing -
Checking given string is member of given grammar or not ?

The program that performs parsing is known as parser.

Parser -
It is a program that takes CFG and tokens as input and verifies tokens are members or not.

If tokens are member then parse tree is output otherwise syntax error generated.

words ⇒ Lexical
grammar ⇒ Syntax



Required -
1) Grammar
2) Parsing algorithm

** GRAMMAR

Set of rules used to describe strings of a language is known as grammar.

Formal Definition -

$$G = (N, T, P, S)$$

N = Non-Terminals or variables
T = Terminals
P = No. of production
S = Starting symbol

E.g

(i) $\alpha \xrightarrow{\text{derives}} \beta$

(ii)
$S \rightarrow TD;$
$T \rightarrow int / float / char$
$D \rightarrow I, D / I$
$I \rightarrow a/b/c/d/ \cdots /z$

int a,b,c;
float a,b;
int a;

} Derived from the above grammar

Grammars are used to represent rules of a language.

(iii)
$S \rightarrow ABC$
$A \rightarrow \overset{2}{a} / \overset{3}{b} / \overset{4}{c}$
$B \rightarrow \overset{5}{e} / \overset{6}{d}$
$C \rightarrow \overset{7}{d} / \overset{8}{e}$

Non-Terminals = {S, A, B, C}
Terminals = {a, b, c, d, e}
Productions = 8
Starting Symbol = S

(iv) $S \to I = E;$ (1)
$E \to E + E / E * E / I$ (2, 3, 4)
$I \to a / b$ (5, 6)

Starting Symbol = S
Non-Terminals = { S, I, E }
Terminals = { a, b, =, ;, *, + }
Productions = 6

- For a language many Grammars are possible
- For a grammar only one language is possible.

NOTE:-

1) For every language grammar exists & every grammar generates one language.
2) All grammars are of the form $\alpha \to \beta$, where β is replacement of α. $\alpha = \beta$

* Derivation -
The process of deriving strings from the given grammar is known as derivation.

The derivation can be either left most derivation or right most derivation

E.g - $S \to ABC$
$A \to a$
$B \to b$
$C \to d$

S
A B C
a b d → String derivation.

Derivation - Step wise manner
Parse Tree - Tree form

Page No.
Date : / /201

* Derivation Tree or Parse Tree

(i) Tree representation of the derivation is known as derivation tree.

(ii) All leaf nodes of the tree (parse tree) is known as yield of parse tree.

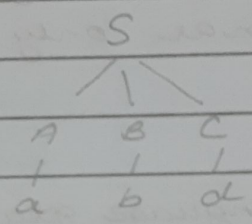
(iii) While reading yield from left to right, sentence of the grammar can be generated.

E.g $S \to ABC$

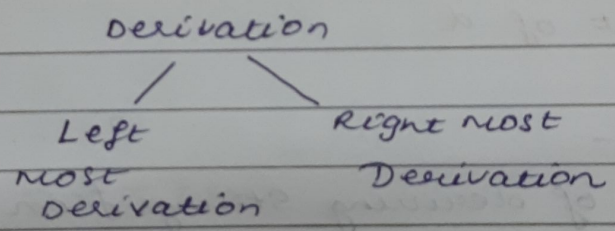
$A \to a$

$B \to b$

$C \to d$



Parse Tree



Left Most Derivation (LMD)

It is the derivation in which left most non terminal is replaced by its RHS part at every step.

Right Most Derivation (RMD)

It is a derivation in which right most non terminal is replaced by its RHS part at every step.

Example -

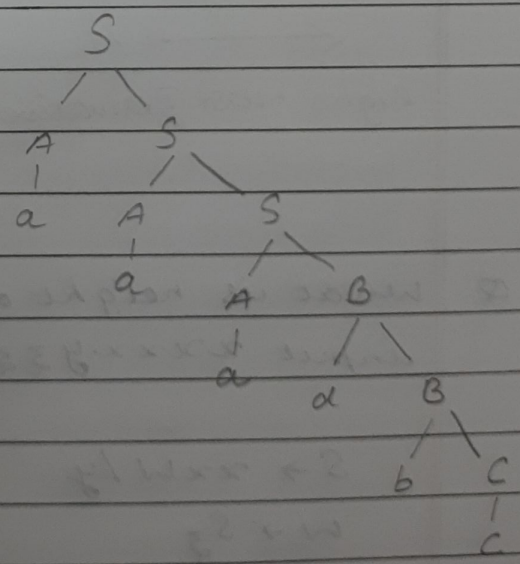
(i) $S \to ABC$
$A \to a$
$B \to b$
$C \to d$

Left most Derivation	Right most Derivation
S	S
A B C	A B C
a B C	A B d
a b C	A b d
a b d	a b d

(ii) $S \to AS / AB$
$A \to a$
$B \to bC / dB$
$C \to c$

Input = aaadbc

S
AS
aS
aAS
aaS
aaAB
aaaB
aaadB
aaadbC
aaadbc



Parse Tree

Left most Derivation

(iii) $S \to ABC$
$A \to a$
$B \to b$
$C \to d$

S
$A\ B\ C$
$A\ B\ d$
$A\ b\ d$
$a\ b\ d$

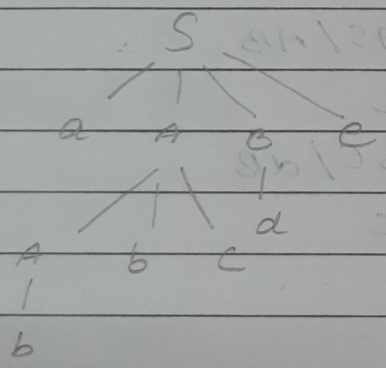
Right most Derivative.

(iv) $S \to aABe$
$A \to Abc / b$
$B \to d$

Input = abbcde

S
$a\ A\ B\ e$
$a\ A\ d\ e$
$a\ A\ b\ c\ d\ e$
$a\ b\ b\ c\ d\ e$
$\longleftarrow$

Right most Derivative

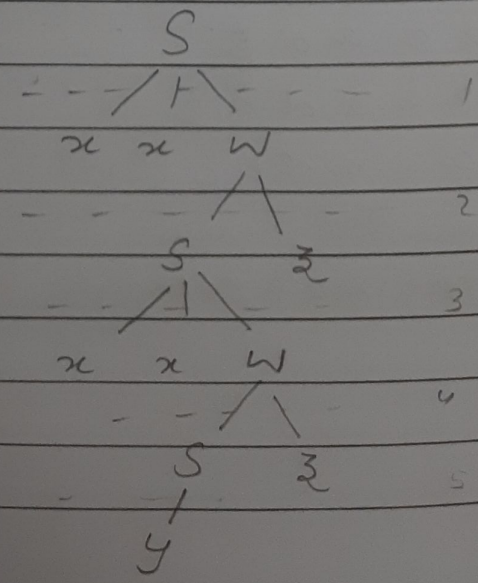


Parse Tree

Q. What is height of the parse tree for the input xxxxyzz using given grammar

$S \to xxW / y$
$W \to Sz$

Height = 5



Q. $L = \{a^n b^n \mid n \geq 1\}$

Grammar exists or not?

Yes, there exist a grammar for every language.

Q. $S \rightarrow AB / BC$

$A \rightarrow a$

$B \rightarrow CC / b$

$C \rightarrow AB / d$

(N, T, P, S) Analysis

Non-Terminals = $\{S, A, B, C\}$

Terminals = $\{a, b, d\}$

Productions = 7

Starting Symbol = S

Q. $S \rightarrow AS / AB$

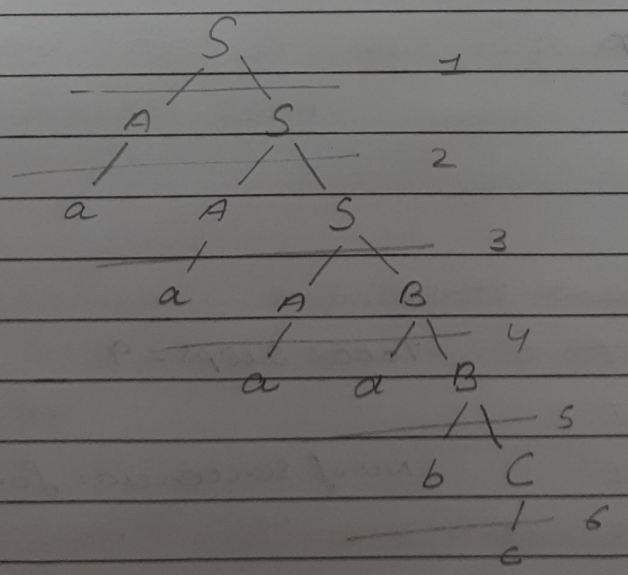
$A \rightarrow a$

$B \rightarrow dB / bC$

$C \rightarrow c$

Input = aaadbc

What is the height of the parse tree?



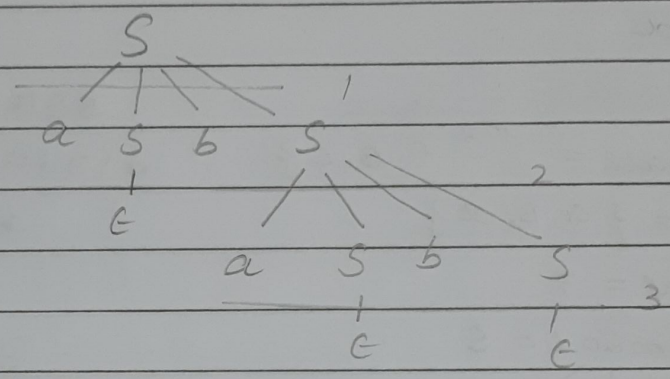
Height = 6

yeild length = String length = 6

Q. $S \to aSbS \,/\, bSaS \,/\, \epsilon$

Input = abab

Height of the parse tree ?



Height = 3

Q. How many steps in Left Most Derivation for the input $\underline{a+a*a}$.

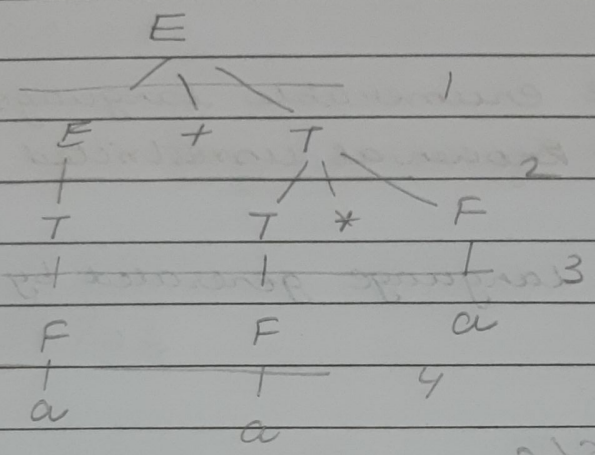
$E \to E+T \,/\, T$
$T \to T*F \,/\, F$
$F \to a$

1. E
2. E + T
3. T + T
4. F + T
5. a + T
6. a + T * F
7. a + F * F
8. a + a * F
9. a + a * a

Total steps = 9

No. of sentential forms = 9
Left sentential form.

parse tree for the previous question



Height of the parse tree = 4

* Sentential Form

(i) Each step in the derivation is one sentential form.
(ii) Hence sentential form is combination of terminals and non-terminals (sentence also can be included).
(iii) If the derivation is leftmost then sentential form is left sentential form.
(iv) If the derivation is right most then sentential is right sentential form.

NOTE:-
Every grammar represents only one language but for one language more than one grammar may exist.

For regular languages there exist a grammar known as regular grammar.

Context free language there exist a grammar known as Context free grammar.

Context Sensitive language there exist a grammar known as Context sensitive grammar.

For recursive enumerable language there exist a grammar known as unrestricted grammar.

Q. Identify the language generated by following grammar.

(i) $S \rightarrow aS / bS / \epsilon$

$L = \{ \epsilon, a, aa \ldots, b, bb, \ldots, ab \ldots, ba \ldots \}$

All strings of as & bs

$= (a+b)^*$

(ii) $S \rightarrow aS / bS / a / b$

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

$(a+b)^+$

$L = \{ a, b, aa, aaa \ldots, b, bb, bbb \ldots, ab \ldots, ba \ldots \}$

All strings of as & bs excluding ϵ

(iii) $S \to aS / a$

$L = \{ a, aa, aaa, \ldots \}$

$\Rightarrow a^+$ Any number of as excluding ϵ

(iv) $S \to aSb / ab$

$L = \{ ab, aabb, a^3b^3, \ldots \}$

$\Rightarrow \{ a^n b^n \mid n \geq 1 \}$

(v) $S \to aSb / \epsilon$

$L = \{ \epsilon, ab, a^2b^2, \ldots \}$

$\Rightarrow \{ a^n b^n \mid n \geq 0 \}$

Wherever there is middle recursion, Left side & Right side is equally generated.

(vi) $S \to AB$
$A \to aAb / ab$
$B \to CB / c$

$L = \{ abc, a^2b^2c, a^3b^3c, a^3b^3c^2 \ldots \}$

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

(vii) $S \to AB$

$A \to aAb / ab \to a^n b^n$

$B \to cBd / cd \to a^m b^m$

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

(viii) $S \to AB$

$A \to aAb / \epsilon \to a^n b^n \quad n \geq 0$

$B \to cBd / cd \to c^m d^m \quad m \geq 1$

$L = \{ a^n b^n c^m d^m \mid n \geq 0; m \geq 1 \}$

(ix) $S \to AB$

$A \to aAb / ab \to a^n b^n$

$B \to bBc / bc \to b^m c^m$

$L = \{ a^n b^{n+m} c^m \mid n \geq 1 ; m \geq 1 \}$

(x) $S \to aSd / aAd \to a^n \underbrace{\quad}_{b^m c^m} d^n$

$A \to bAc / bc \to$

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

(xi) $S \to aSbS / bSaS / \epsilon$

$L = \{ \epsilon, ab, ba, aabb, baab \ldots \}$

Equal number of as & bs

24-07-21

(xii) $S \to aSb / aAb$ $\quad a^n a^m b^n$

$A \to aA / a \to a^+$ or a^m

$L = \{ a^{n+m} b^n / n, m \geq 1 \}$ or $L = \{ a^n b^m / n > m \}$

(xiii) $S \to aSb / aAb / aBb$ $\quad a^n a^m b^n / a^n b^+ b^n$

$A \to aA / a \to a^+$ or a^m

$B \to bB / b \to b^+$ $\quad b^m$

$L = \{ a^n b^m / n \neq m \}$ or $\{ m > n$ or $m < n \}$

Q. Construct grammar for the following languages.

Grammar $= (N, T, P, S)$

One grammar $\to$ 1 language

(i) $L = \{ a^n / n \geq 0 \}$

$S \to \epsilon / aS$ or $S \to Sa / \epsilon$ or $S \to \epsilon / a / aS$

(ii) $L = \{ a^n b^m / n \geq 0 \quad m \geq 1 \}$

$S \to AB$

$A \to aA / \epsilon$

$B \to bB / b$

(iii) $L = \{ a^n b^m c^k \mid n, m \geq 0, k \geq 1 \}$

$S \rightarrow ABD$

$D \rightarrow CD / C$

$B \rightarrow bB / \epsilon$

$A \rightarrow aA / \epsilon$

(iv) $L = \{ a^n b^m \mid (n+m) \text{ is even} \}$

even a's even b's

odd a's odd b's

$S \rightarrow AB / aAbB$ odd as & odd bs

$B \rightarrow bbB / \epsilon$ even bs

$A \rightarrow aaA / \epsilon$ even as

Non-terminals = $\{ S, A, B \}$

Terminals = $\{ a, b \}$

Productions = 6

Starting symbol = $\{ S \}$

(v) $L = \{ a^n b^m \mid (n+m) \text{ is odd} \}$

even as odd bs

odd as even bs

$S \rightarrow AbB / aAB$

$A \rightarrow aaA / \epsilon$

$B \rightarrow bbA / \epsilon$

(vi) $L = \{ a^{n+m} b^m c^n \mid m, n \geq 1 \}$

$\Rightarrow a^n a^m b^m c^n$

$S \rightarrow aSc \mid aAc$
$A \rightarrow aAb \mid ab$

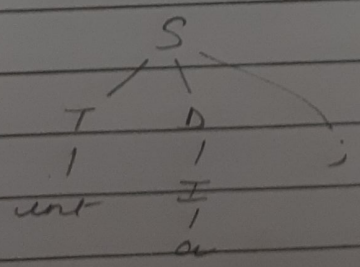
(vii) $L = \{ a^n b^m c^m d^n \mid n, m \geq 1 \}$

$S \rightarrow aSd \mid aAd$
$A \rightarrow bAc \mid bc$

* Declaration statement Grammar for C language

int a, b, c ;
TYPE Declaration;

$S \rightarrow TD;$
$T \rightarrow int \mid float \mid char$
$D \rightarrow I \mid I, D$
$I \rightarrow a \mid b \mid c \mid \cdots \mid z$

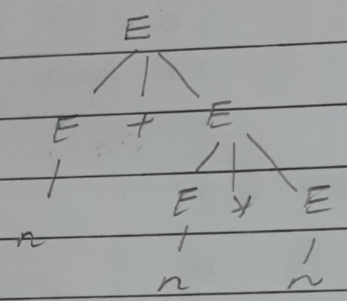


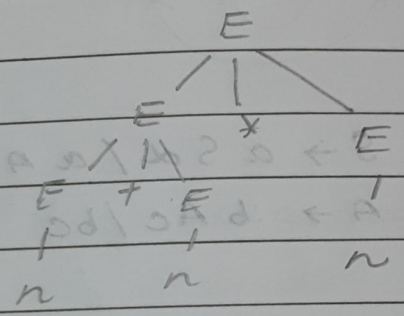
Ambiguous Grammar

A grammar is said to be ambiguous if for atleast one string more than 1 parse tree exists or more than 1 LMD/RMD exists.

Example -

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





Precedence of * is higher

Precedence of + is higher

Grammar

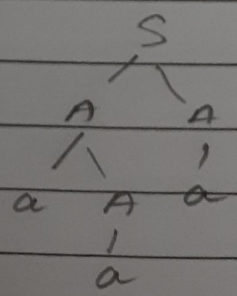
Ambiguous grammar
compiler cannot be constructed

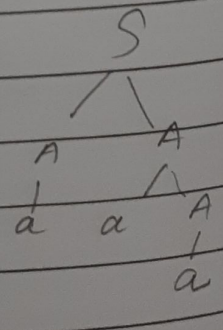
Unambiguous grammar
compiler can be constructed.

Example

$S \rightarrow AA$

$A \rightarrow aA \mid a$

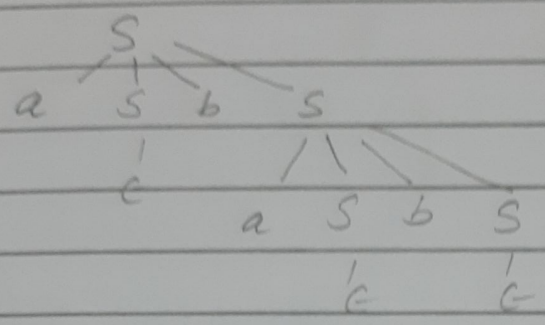




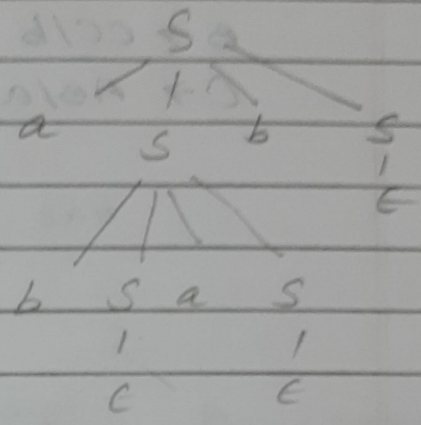
Ambiguous

Example -

$S \rightarrow aSbS \mid bSaS \mid \epsilon$



abab



abab

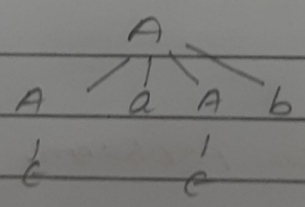
Ambiguous grammar

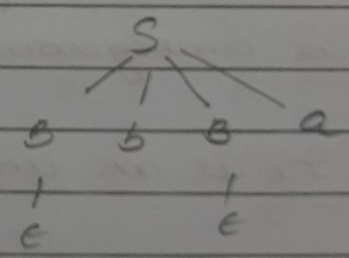
Example -

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$





Non-ambiguous grammar.

Example -

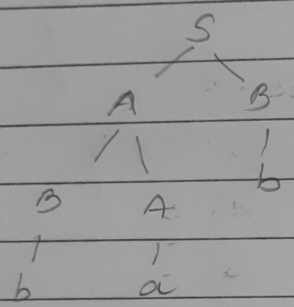
$S \rightarrow AB / BC$

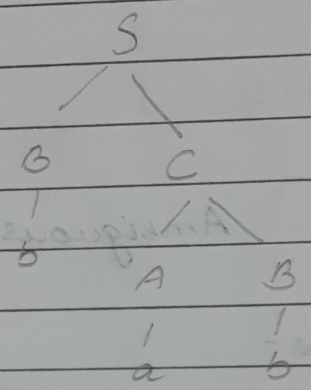
$A \rightarrow BA / a$

$B \rightarrow CC / b$

$C \rightarrow AB / a$

bab.





Ambiguous grammar

There is no algorithm to check whether a grammar is ambiguous or not.

It is an undecidable problem

Elimination of Ambiguity is Undecidable problem.
(No algorithm exists)

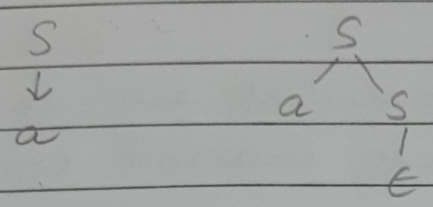
If LMD & RMD exists ⇒ Different parse trees.

L-8-9

26-07-2021

Q. verify if the grammar is ambiguous or not?

$S \to \epsilon / a / aS$



TYPES OF GRAMMARS

1. Regular grammar → Lexical Analysis
2. Context-free grammar → Syntax Analysis
3. Context sensitive grammar → Semantic Analysis

} compiler Designing

4. Unrestricted grammar → Natural languages (not in compiler)

Types of Grammar

Type 3 grammar or Regular grammar	Type 2 Grammar or CFG	Type 1 Grammar or CSG	Type 0 Grammar or unrestricted grammar
Right Linear Gramm $A \to xB / x$	$A \to \alpha$ $\alpha \in (V+T)^*$	$\alpha \to \beta$ $\lvert\alpha\rvert \leq \lvert\beta\rvert$ length equal or more $\alpha, \beta \in (V+T)^+$	$\alpha \to \beta$ exclude ε $\alpha \in (V+T)^+$ include ε $\beta \in (V+T)^*$
Left Linear G. $A \to Bx / x$	$A \to BC$ $B \to a$ $C \to b$	E.g → $S_a \to aB$ $S \to b$	E.g- $aBc \to a$

* Expression Grammar :-

$S \to I = E;$
$E \to E+E \mid E*E \mid E-E \mid E/E \mid I$
$I \to a/b/c$

CFG

Q. Identify the type of grammar

$S \to ABC$
$A \to a$
$B \to b$
$C \to \epsilon$

CFG

Regular Grammar

Linear Grammar { LHS = Only one Non-terminal
RHS = Atmost one Non-terminal

E.g $S \to aS / Sb / a$
$aS \to$ RL, $Sb \to$ LL

Combination of both is not allowed.
⇒ Non-Regular

E.g $S \to aS / a$
$aS \to$ RL
⇒ Regular

E.g $S \to Sa / b$
$Sa \to$ LL
⇒ Regular

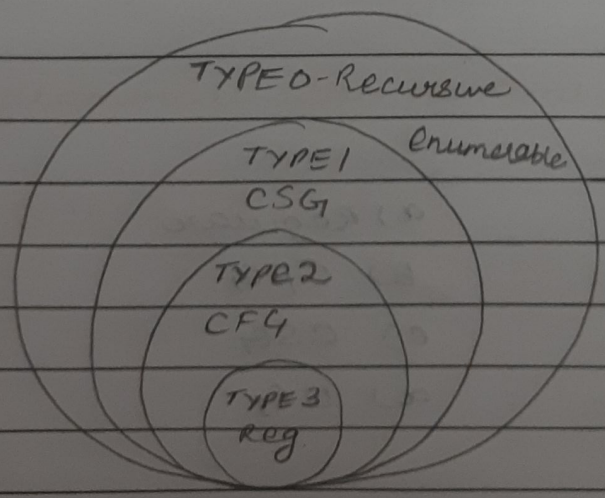
Q. $S \to aAbB$
$aA \to b$
$bB \to bb$.

a) Regular
b) Context Free grammar
c) Context Sensitive grammar
✓ d) Unrestricted grammar

Q. $S \to AaB$
$A \to ab$
$aB \to b$

Unrestricted grammar
Why ?
$\alpha \to \beta$
$\alpha \in (V+T)^+$
$\beta \in (V+T)^*$

Relation between types of grammar



Q which of the following is false?

a) Every Regular grammar is CFG
b) Every CFG is CSG
✓c) Every UG is CFG
d) Every CFG is U.G

Q. Identify type of following grammar?

(i) $S \rightarrow aS / bS / \epsilon$

✓a) Regular
b) CFG
c) CSG
d) U.G

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

a) Regular
✓b) CFG
c) CSG
d) U.G

(iii) $S \rightarrow aS$
$S \rightarrow Sb$
$S \rightarrow a$

a) Regular
✓b) CFG
c) CSG
d) UG

(iv) $S \to aSb / bSa / c$

a) Regular
b) CFG
c) CSG
d) UG

(v) $S \to aAbB$
$aA \to bb$
$bB \to a$

Unrestricted grammar

$|\epsilon| = 0$

(vi) $S \to aAbB / \epsilon$
$aA \to aa$
$bB \to bb$

Context sensitive grammar

$S \to \epsilon$ is added to CSG [only at the starting symbol]. So that the set becomes proper.

Regular grammars can be ambiguous L-10

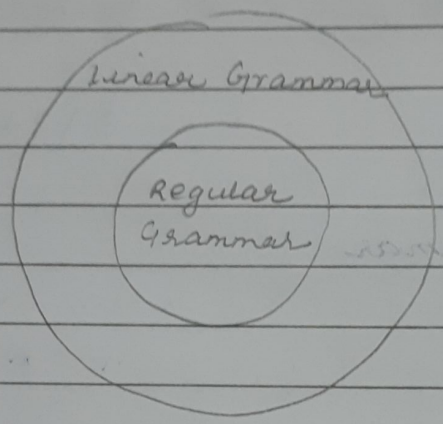
$S \to a / aS / \epsilon$

Linear Grammar

If any grammar LHS contains exactly one Non-terminal and RHS contains atmost 1 non-terminal.

E.g ⇒ $S \rightarrow aSb / ab$

Linear ; but not regular



Every regular grammar is linear but every linear grammar is not linear regular.

Every Linear is CFG.

Though CSG does not produce ε-production, yet to satisfy the chomsky hierarchy $S \rightarrow \epsilon$ is added (only to the starting symbol).

Q. (i) $S \rightarrow \epsilon / aS / bS$ Regular

(ii) $S \rightarrow aAbB / \epsilon$

$aA \rightarrow aa$ CSG.

$bB \rightarrow aa$

NOTE :-

In Context Sensitive Grammars if any non-terminal derives ϵ, that non-terminal should not be present in the RHS of the resultant grammar

E.g ⇒ $S \to aAbB / \epsilon$

$aA \to aaS$ → Not allowed

$bB \to aa$

Difference between Context Free and Context Sensitive grammar

	Context Free Grammar	Context Sensitive Grammar
#	$A \to \alpha$	$\alpha \to \beta$ $\lvert\alpha\rvert \leq \lvert\beta\rvert$
#	Only structure exists	Meaning + structure exists
#	Syntax analysis	Semantic Analysis

Example in English Grammar

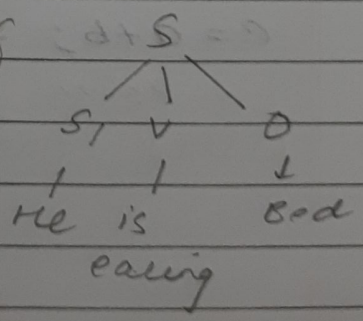
$S \to S_1 V O$

$S_1 \to$ He / She / They

$V \to$ reading / eating / sleeping

$O \to$ book / bed / food

Context Not maintained



Hence, CFGs are good for Syntax analysis

Q. C language is

a) Regular language
b) Context free language
c) Context Sensitive language
d) None

Any language compiler that contains semantic analysis will always be Context sensitive language.

C, C++, java, Python ⇒ CSL

Structure = Syntax
meaning also important = Semantic

⇒ int a, b. Syntax error

int a = b * C * d Syntax error

int a, a, b, b; Semantic error

C = a+b; Syntaxwise correct but semantically wrong.

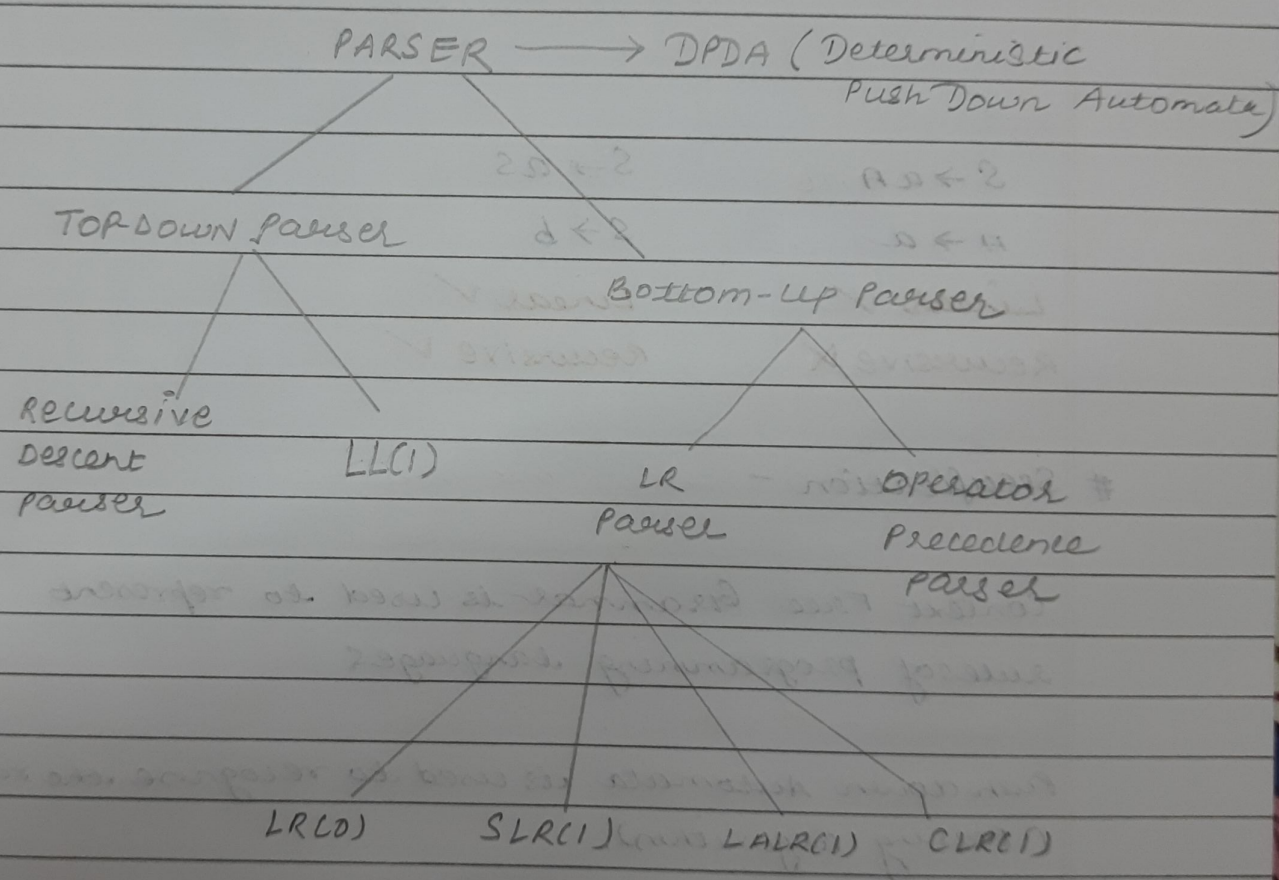
very important

PARSER

⇒ Parsing - Checking a given string is member of given grammar (or) not ?

The program that performs parsing is called parser

If tokens are members, parse tree is output otherwise syntax error is generated.



Note :-

1) Bottom-up parser is more powerful parser than top-down parser; Hence, no. of grammars suitable for B.U.P is more than T.D.P
2) Left Recursive grammars are not suitable for Top Down parser
3) Left Recursive grammars are suitable for B.U.P.

Recursive Grammar

$S \to aS/b$	$S \to Sa/b$	$S \to aSb/ab$
Right Recursive	Left Recursive	Middle Recursion

Same

Topdown parser ✗

Bottomup parser ✓

$S \to aA$	$S \to aS$
$A \to a$	$S \to b$
Linear ✓	Linear ✓
Recursive ✗	Recursive ✓

Specification -

Context Free Grammar is used to represent rules of programming languages

Pushdown Automata is used to recognise the rules (Parsing algorithm).

* Top Down Parsing
  (i) Recursive Descent Parsing
  ** (ii) LL(1)

* Bottom-UP Parsing
  ** (i) LR Parsing
  * a) LR(0)

b) SLR(1)

c) CLR(1)

d) LALR(1)

(ii) Operator Precedence Parsing

L-11

27-07-2021

Bottom-Up Parsing

Bottom-Up parser - Constructs a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top).

Bottom up parser uses right most derivation in reverse order

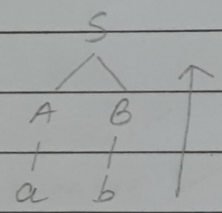
Bottomup parser performs parsing using handle detection.

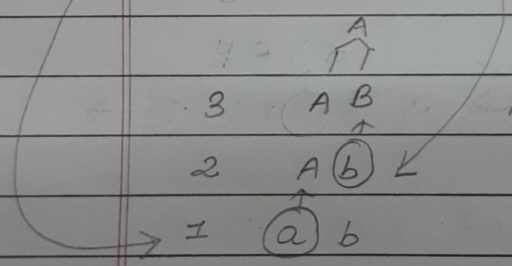
Example -

$S \to AB$

$A \to a$

$B \to b$





Right Most Derivation in Reverse Order

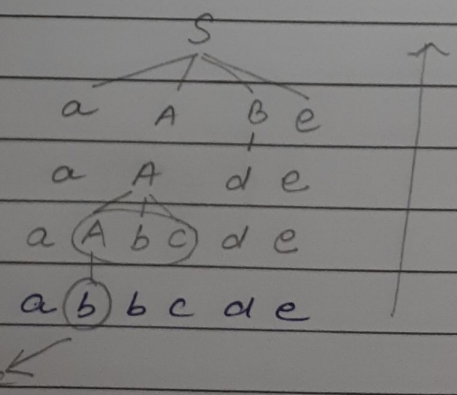
Example -

$S \to aABe$

$A \to Abc/b$

$B \to d$

Total 4 handles



Handle - Substring matching RHS of Grammar.

* Handle -

Informally, a "handle" of a string is a substring that matches the right side of the production, and whose reduction to nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation.

- But not every substring matches the right side of a production rule is handle.

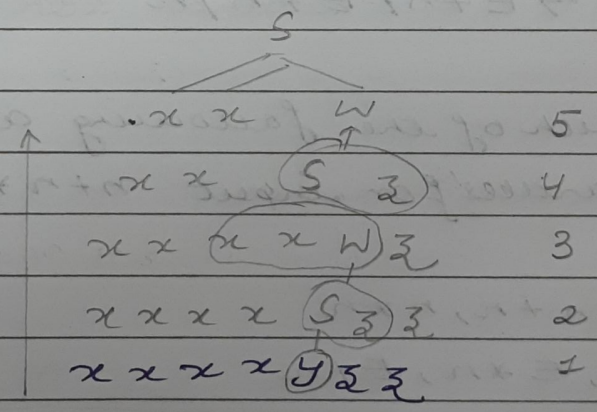
Substring matching RHS of grammar. Replacement of handle should lead to starting symbol.

Q. How many no. of handles detected by bottomup parser for the following grammar and string.

(i) $S \to xxW$

$S \to y$

$W \to Sz$



Total 5 handles

Sequence of handles detected by the parser

y, Sz, xxW, Sz, xxW

(ii) Q. $S \to AS / AB$

$A \to a$

$B \to bC / dB$

$C \to c$

S

A S

A A S

A A A B

A A A d B

A A A d b C

A A A d b c

A A a d b c

A a a d b c

a a a d b c

Always start from Left most

Total no. of handles = 9

GATE PYQ (iii) $E \to E+n / E*n / n$

2 MARKS

Which of the following are sequence of handles for input $n+n*n$

a) $n, n+n, n*n$

b) $n, E*n, E+n$

✓ c) $n, E+n, E*n$

d) None

E

E * n

E + n * n

n + n * n

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

E	
$E+T$	8
$E+T*F$	7
$E+T*a$	6
$E+F*a$	5
$E+a*a$	4
$T+a*a$	3
$F+a*a$	2
$a+a*a$	1

In the above example, Line : 6 ⇒ Had we taken E+T it wouldn't have lead to the Starting Symbol, hence not considered.

NOTE :-
Every matching is not handle.

(v) $S \to AB$
$A \to a$
$B \to a$

		S
A A	B B	A B
A a	B a	A a
a a	a a	a a
Invalid handles.		Valid handle

GATE 2016

(vi) $S \to aA$ What are the sequences of handles detected by the parser?

$S \to a$

$A \to Sb$

S

$\boxed{a\ A}$	3
$a\ \boxed{S\ b}$	2
$a\ a\ b$	1

Sequence $\Rightarrow$ a, Sb, aA

(vii) $E \to T+E \,/\, T$

$T \to a$

Sequence of handles

E

$\boxed{T+E}$
$T+T$
$T+a$
$a+a$

a, a, T, T+E

NOTE:-

Bottom-up parsing is also known as Shift-reduce parsing because its two main actions are shift & reduce

Shift Reduce Parser in Compiler

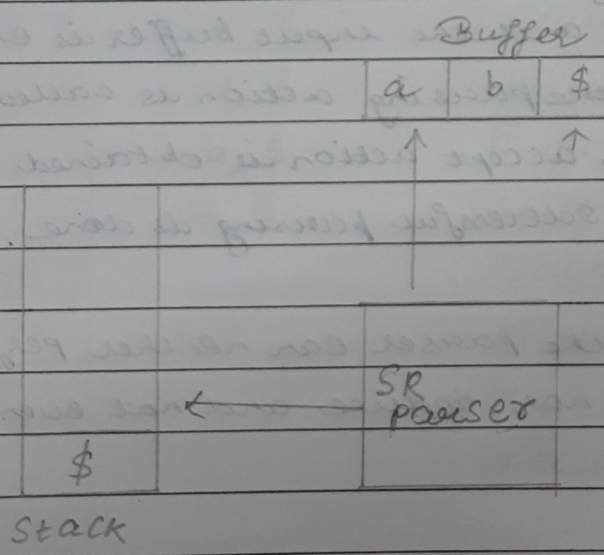
(i) Shift Reduce parser attempts for the construction of parse tree in a similar manner as done in bottom up parsing; i.e; the parse tree is constructed from leaves (bottom) to the root (up).

(ii) A more general form of shift reduce parser is LR parser.

(iii) This parser requires stack datastructure.

(iv) An input buffer for storing the input string + $ (end marker)

(v) A stack for storing and accessing the production rules.



Shift = Push

Reduce = Replace

STEPS:-

1) Shift (Push) symbols from input buffer to stack until there is is a handle.

2) whenever handle present in the stack is detected, reduce (Replace) it by the non-terminal.

Basic Operations :-

Shift - Move symbols from input buffer onto the stack.

Reduce - If the handle appears on the top of the stack then its reduction by using appropriate production rule is done i.e RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.

Accept - If only start symbol is present in the stack and the input buffer is empty then the parsing action is called accept. When accept action is obtained it means that successful parsing is done.

Error - When the parser can neither perfectly shift nor reduce and not even accept.

Example -

1 $E \rightarrow E + T$
2 $E \rightarrow T$
3 $T \rightarrow T * F$
4 $T \rightarrow F$
5 $F \rightarrow (E)$
6 $F \rightarrow id$

Input ⇒ id + id

	STACK	INPUT	ACTION
1	$	$id_1 + id_2$ $	Shift
2	$id	$+ id_2$ $	Reduce 6
3	$F	$+ id_2$ $	Reduce 4
4	$T	$+ id_2$ $	Reduce 2
5	$E	$+ id_2$ $	Shift
6	$E+	id_2 $	Shift
7	$E+id_2		Reduce 6
8	$E+F		Reduce 4
9	$E+T		Reduce 1
10	$E		Accept

Number of shift actions ⇒ 3
Number of Reduce actions ⇒ 6

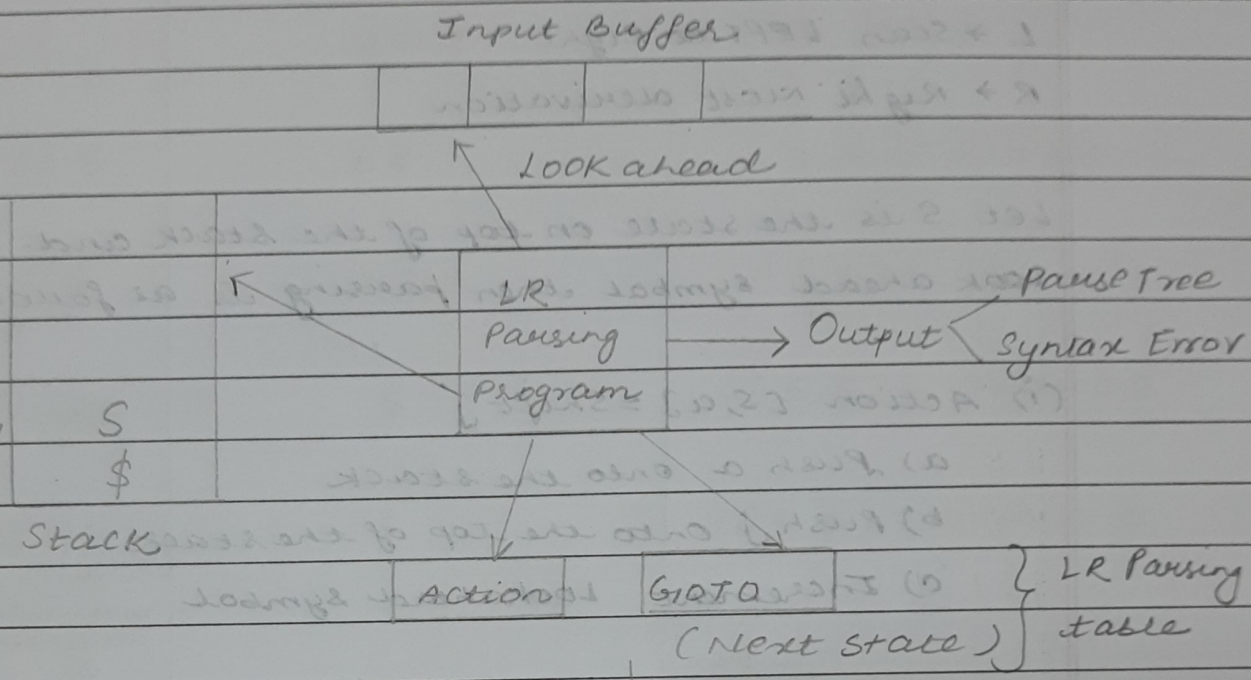
Example :-

1 $S \rightarrow S+S$ Input = id+id+id
2 $S \rightarrow S*S$
3 $S \rightarrow id$

Stack	Input Buffer	Parsing Action
$	id+id+id$	Shift
$ id	+id+id$	Reduce 3
$ S	+id+id$	Shift
$ S+	id+id$	Shift
$ S+id	+id$	Reduce 3
$ S+S	+id$	Shift
$ S+S+	id$	Shift
$ S+S+id	$	Reduce 3
$ S+S+S	$	Reduce 1
$ S+S	$	Reduce 1
$ S	$	Accept

No. of Shift actions = 5
No. of Reduce actions = 5

LR PARSING
(Shift Reduce Parser)



L ⇒ Left to Right I/P scan
R ⇒ Right Most Derivation in reverse

	ACTION	GOTO
S1	Shift/ ~~Reduce~~/ Accept/ Error	Next State
⋮		
Sn		

LR Parsing Table

DFA

Top of stack is state of the table.

28-07-21

Page No.
Date : / /201

LR Parsing algorithm

L → Scan LEFT to Right
R → Right most derivation

Let S is the state on top of the stack and a is Look ahead symbol then parsing is as follows

(i) Action [S, a] = Shift j
  a) Push a onto the stack
  b) Push j onto the top of the stack
  c) Increment Lookahead symbol

(ii) Action [S, a] = reduce A → B
  a) 2 * |B| Pop from stack.
  b) Push A into stack.
  c) Push GOTO(i, A) Onto top of the stack.
  Previous state on A

Example -

	Action			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			

(1) S → AB
(2) A → a
(3) B → b

Stack.	Input Buffer	Action.
$ 0	ab $	Shift 3 [0,a]
$ 0 a 3	b $	[3, b] = reduce 2
$ 0 A 2	b $	[2, b] = S5
$ 0 A 2 b 5	$	[5, $] = reduce 3
$ 0 A 2 B 4	$	[4, $] = reduce 1
$ 0 S 1	$	[1, $] = Accept

previous state on B

(iii) Action[S, a] = accept
Input is valid
No syntax error
Parsing successful

(iv) Action [S, a] = Blank
Syntax Error
Return to Error handler.

Time Complexity → O(n)

Example:-

< USE PREVIOUS TABLE >

$S \to AB$

$A \to a$

$B \to b$

Stack	Input Buffer	Action
$ | 0	a a $	[0, a] Shift 3
$ | 0 | a | 3	a $	[3, a] Black Error

Q. How many shift actions and reduce actions are taken by LR parser to parse input id + id * id by using following grammar and parsing table

	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		r_1	S_4	r_1	
I_6		r_2	r_2	r_2	

1 $E \to E + E$

2 $E \to E * E$

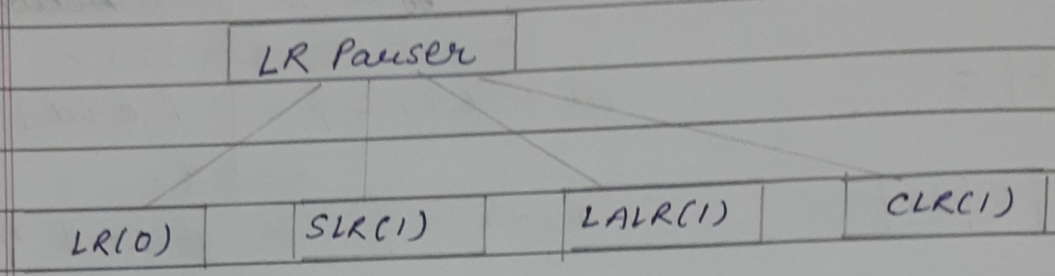
3 $E \to id$

Stack	Input Buffer	Action
$ 0	id + id * id $	[0, id] Shift 2
$ 0 id 2	+ id * id $	[2, +] reduce 3
$ 0 E 1	+ id * id $	[1, +] Shift 3
$ 0 E 1 + 3	id * id $	[3, id] Shift 2
$ 0 E 1 + 3 id 2	* id $	[2, *] Reduce r3
$ 0 E 1 + 3 E 5	* id $	[5, *] Shift 4
$ 0 E 1 + 3 E 5 * 4	id $	[4, id] Shift 2
$ 0 E 1 + 3 E 5 * 4 id 2	$	[2, $] Reduce 3
$ 0 E 1 + 3 E 5 * 4 E 6	$	[6, $] Reduce 2
$ 0 E 1 + 3 E 5	$	[5, $] Reduce 1
$ 0 E 1	$	[1, $] Accept

5 Shift actions.
5 reduce actions.

DFA cannot be constructed for CFGs
DFA can be constructed for LR(0) items

Page No.
Date : / /201



★★ LR parsing table Construction → DFA

(1) LR(0) table Construction

1. Construct Augmented Grammar
2. Construct LR(0) items DFA using closure() & goto().
3. Construct parsing table

E.g →

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

→ Augmented Grammar

LR(0) items – CFG Productions having dot on the RHS of grammar

E.g.

$S \to \cdot AB$

$S \to \cdot a$

$S \to \cdot b$

→ LR(0) items

$A \to \cdot XYZ$

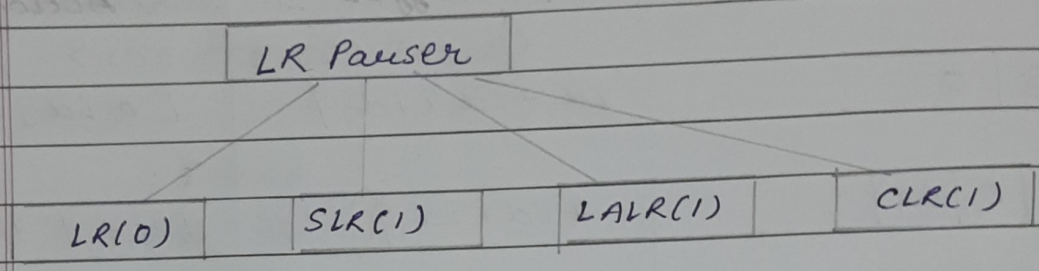
$A \to X \cdot YZ$

$A \to XY \cdot Z$

$A \to XYZ \cdot$

DFA cannot be constructed for CFGs
DFA can be constructed for LR(0) items

Page No.
Date : / /201



** LR parsing table Construction → DFA

(1) LR(0) table construction

1. Construct Augmented Grammar
2. Construct LR(0) items DFA using closure() & goto().
3. Construct parsing table

E.g ⇒

Augmented Grammar

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

LR(0) items – CFG Productions having dot on the RHS of grammar

E.g.

LR(0) items

$S \to \cdot AB$

$S \to \cdot a$

$S \to \cdot b$

$A \to \cdot XYZ$

$A \to X \cdot YZ$

$A \to XY \cdot Z$

$A \to XYZ \cdot$

Finite LR(0) items = Regular = DFA

closure() -

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

$S \to \cdot AB$
$A \to \cdot a$

$S \to A \cdot B$
$B \to \cdot b$

In any LR(0) items after dot non-terminal exist then closure () adds that non-terminal production and puts dot on LHS

goto() -

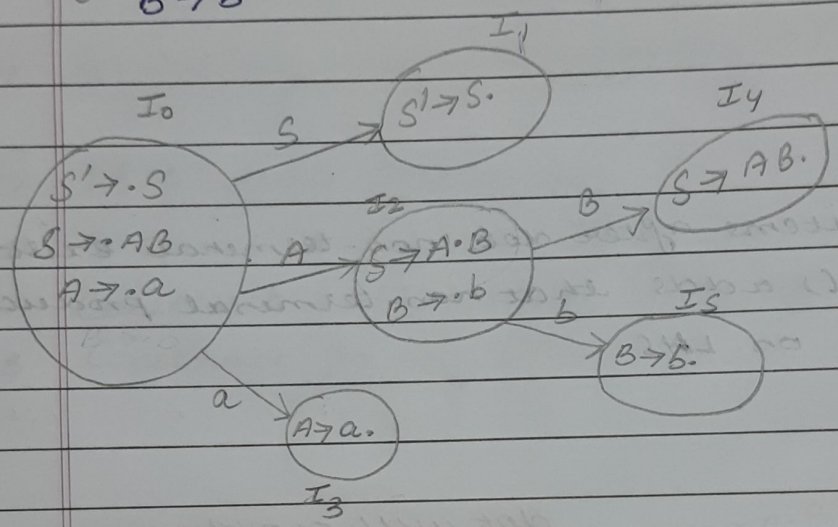
$A \to \cdot a \xrightarrow{goto()} A \to a \cdot$

dot will move one position towards RHS.

$S \to A \cdot B \xrightarrow{goto()} S \to AB \cdot$

Q. Construct LR(0) parsing table for the following grammar.

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



LR(0) Table

	Action			GOTO		
Lookahead	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			

Size of the table ⇒

No. of State * (NonTerminals + Terminals + 1)

Page No.
Date : / /201

Points to Remember

1) All terminals transition = Shift actions
2) Non-terminals transition = GOTO entries
3) LR(0) items dot at Rightmost hand = Reduce

Shift actions in the table = 2
GOTO actions in the table = 3

Based on transitions we can fill shift entries and GOTO entries

L-15

CFG has different names -

(i) Ambiguous grammar
(ii) Unambiguous grammar
(iii) LR(0) grammar
(iv) SLR(1) grammar
(v) CLR(1) grammar
(vi) LALR(1) grammar
(vii) Operator Precedence grammar
(viii) LL(1) grammar

29-07-2021

LR(0) grammar –

Any CFG LR(0) parsing table contains shift/Reduce conflict or Reduce/Reduce conflict exists then that grammar is NOT LR(0) grammar

Conflict States in LR(0) items

1) S/R conflict

$A \to \alpha . a X$ (Shift)

$B \to Y.$ (Reduce)

2) R/R conflict

$A \to X.$ (Reduce)

$B \to Y.$ (Reduce)

If LR(0) items contains any one of these states then the grammar is NOT LR(0) grammar

Q. Verify the following grammar is LR(0) grammar or not?

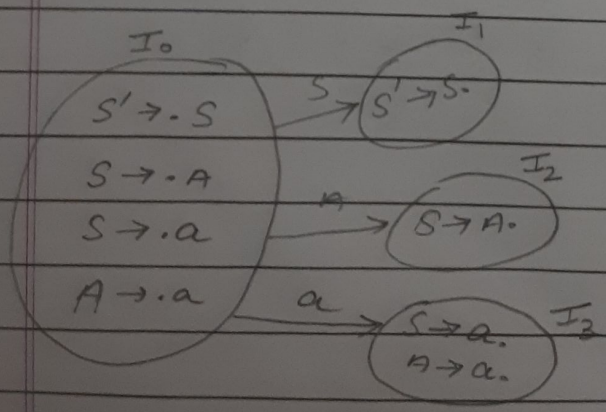
(i) $S \to A$

$S \to a$

$A \to a$

Reduce/Reduce (R/R) Conflict

Hence, NOT LR(0)

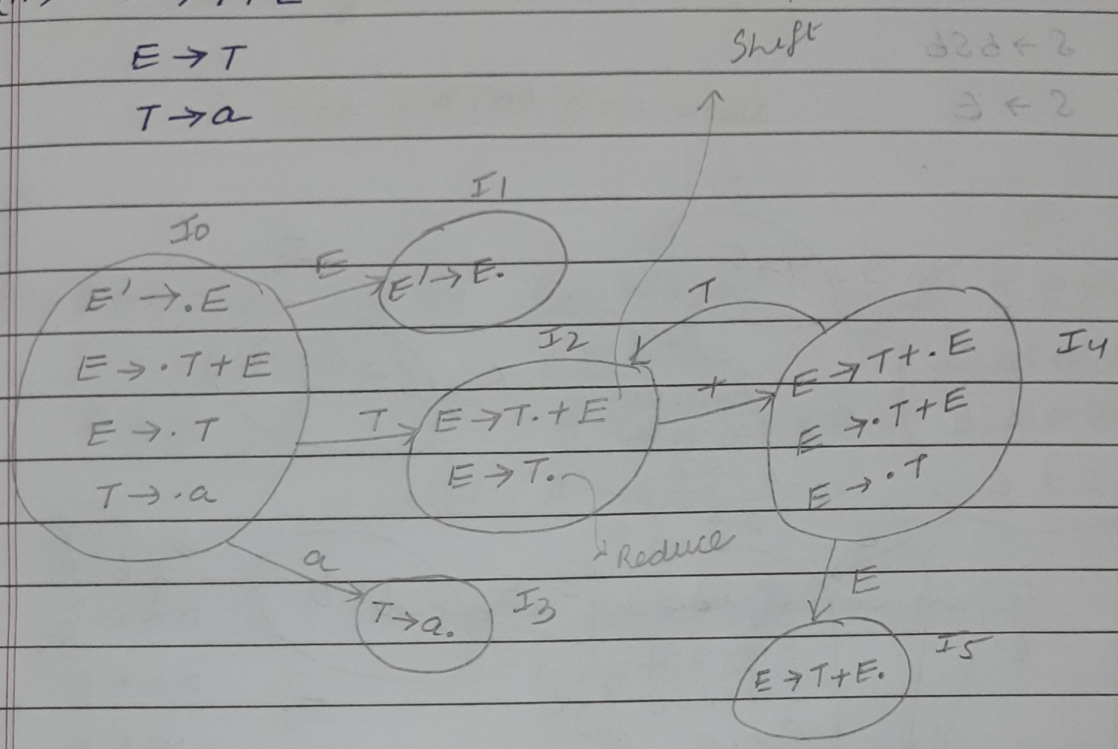


GOTO ⇒ 2

(ii) $E \to T+E$

$E \to T$

$T \to a$

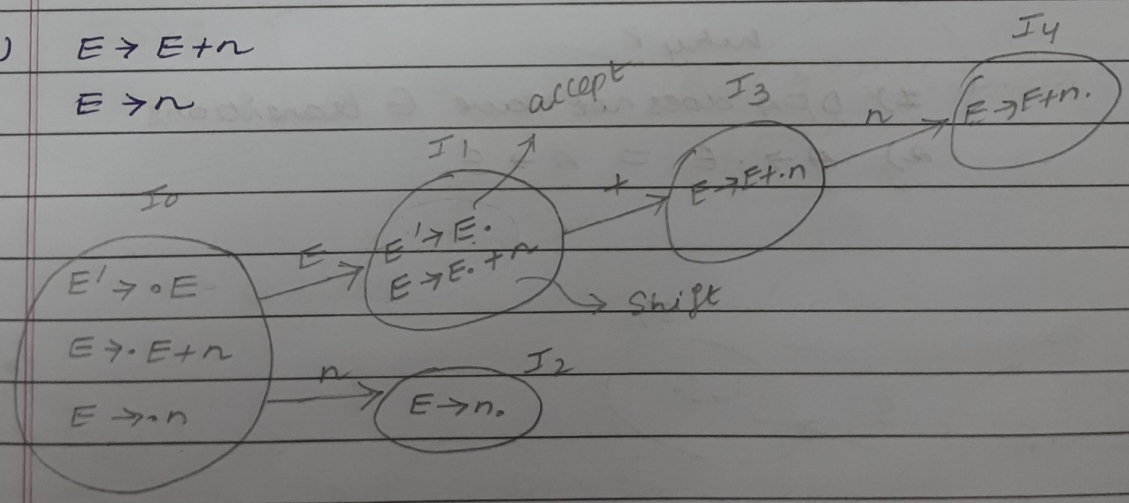


S/R conflict at state I_2.

∴ Not LR(0) grammar

(iii) $E \to E+n$

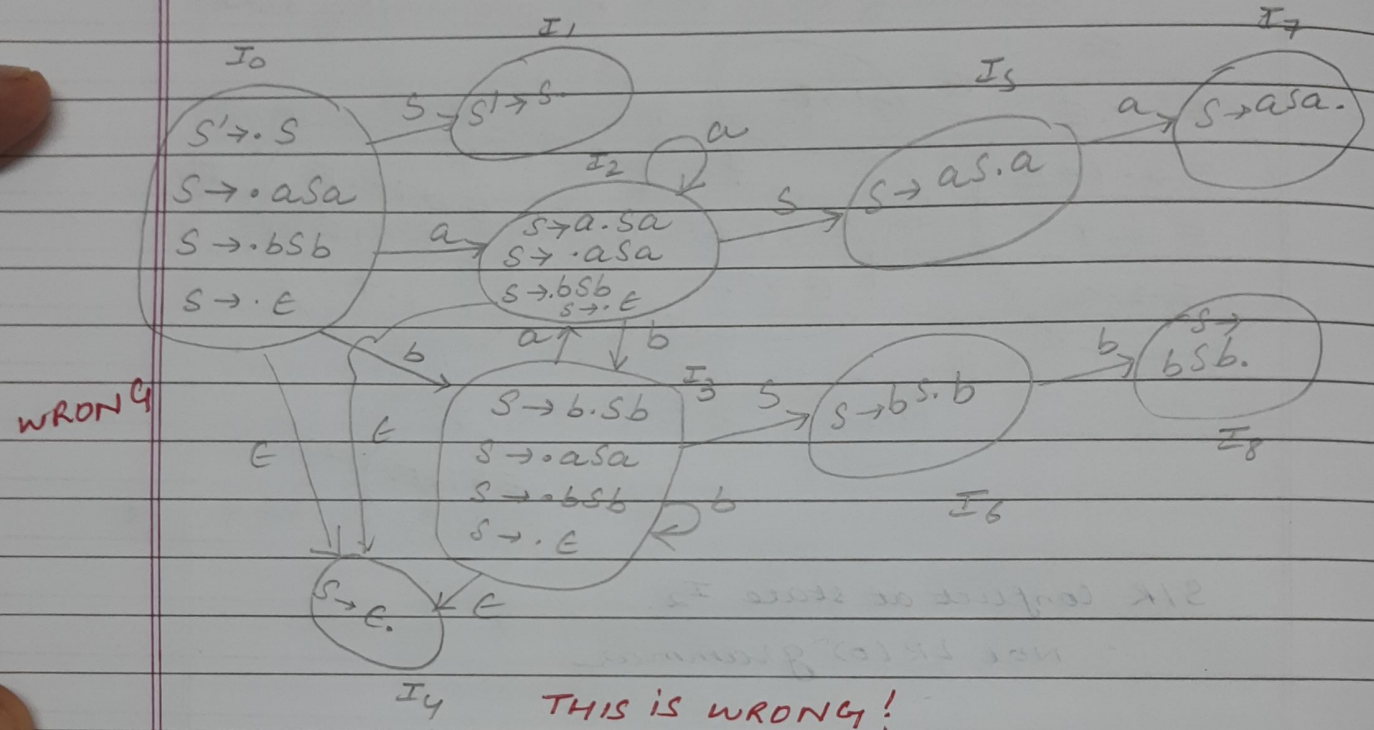
$E \to n$



Shift / accept is not a conflict

∴ Grammar is LR(0).

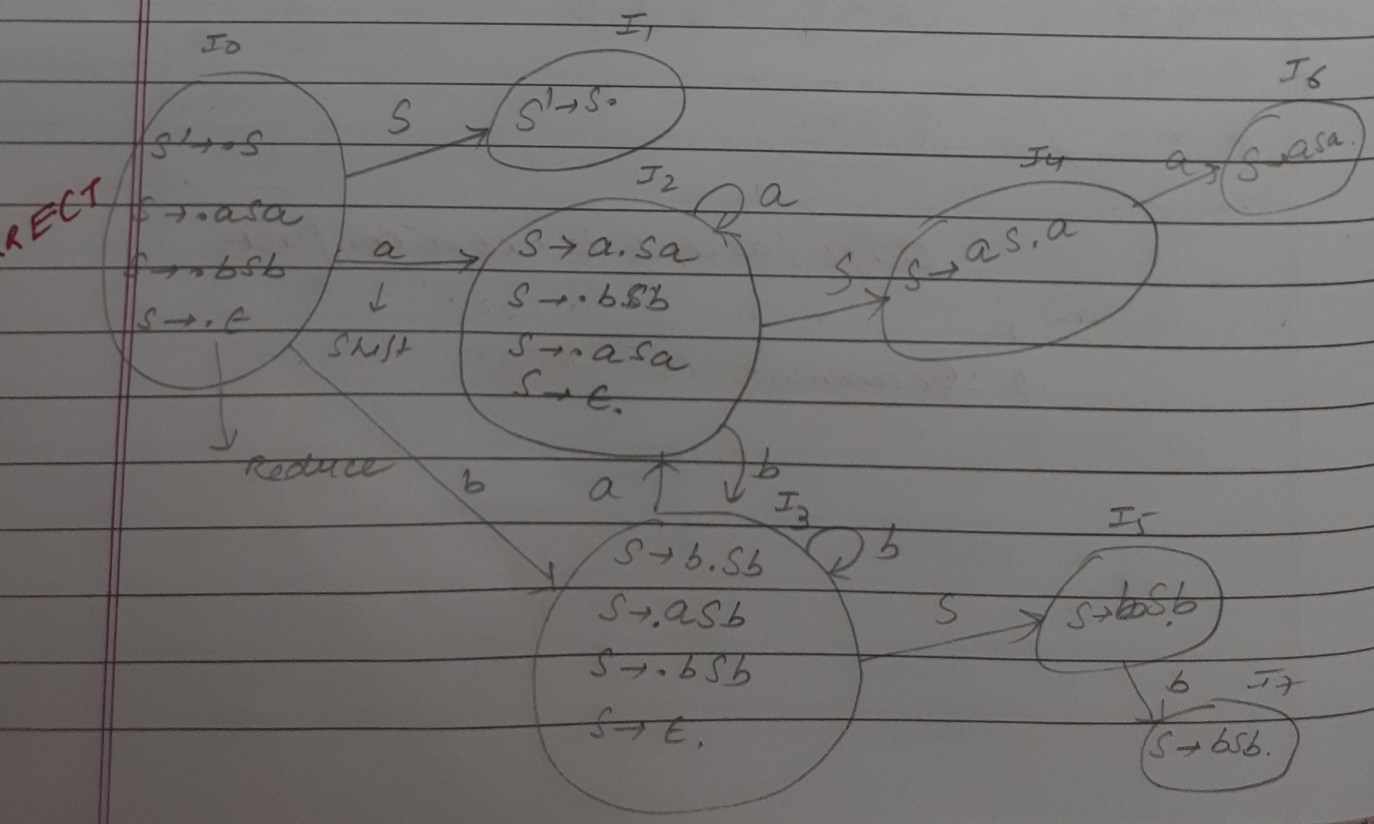
(iv) $S \to aSa$
$S \to bSb$
$S \to \epsilon$



THIS IS WRONG!

why?

1) DFA does not have ϵ transitions
2) $A \to \cdot \epsilon = A \to \epsilon \cdot$



S/R conflict at I_0 itself.

∴ Not LR(0) Grammar

(V) a) $S \to aSb$
$S \to bSa$
$S \to \epsilon$

b) $S \to AaAb$
$S \to BbBa$
$A \to \epsilon$
$B \to \epsilon$

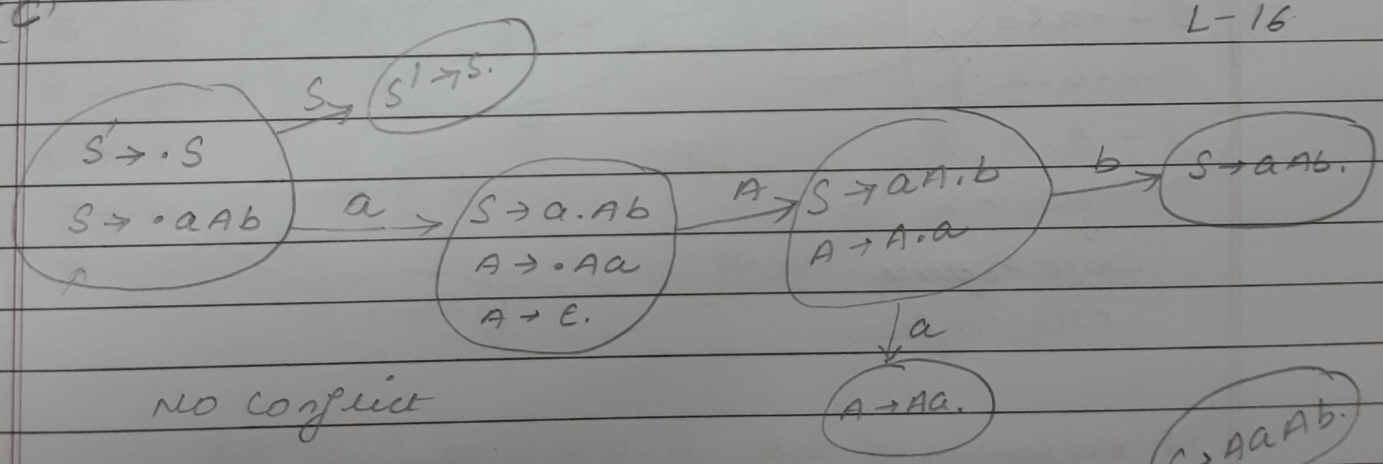
c) $S \to aAb$
$A \to Aa$
$A \to \epsilon$

d) $S \to AA$
$A \to aA$
$A \to a$

Which of the following is LR(0) grammar.

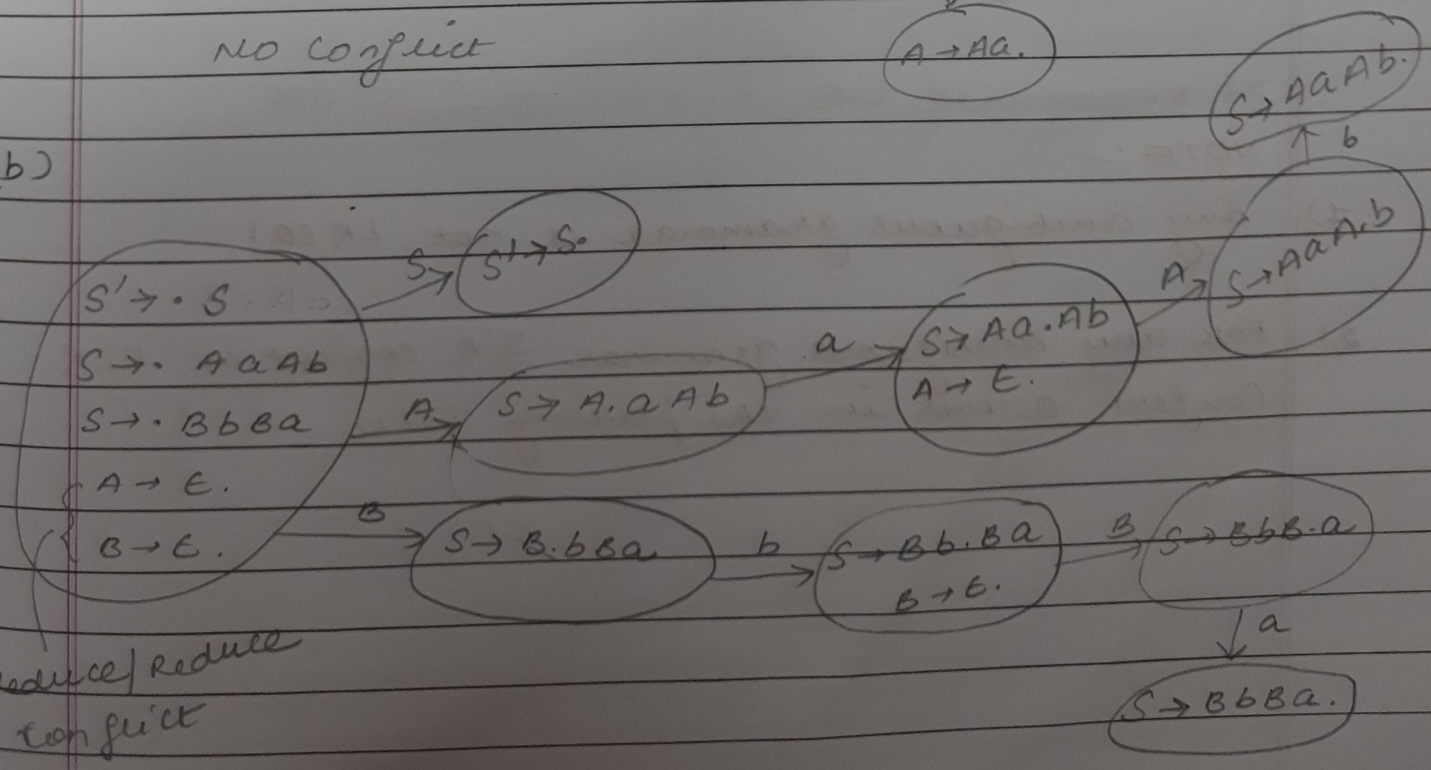
(c)

L-16



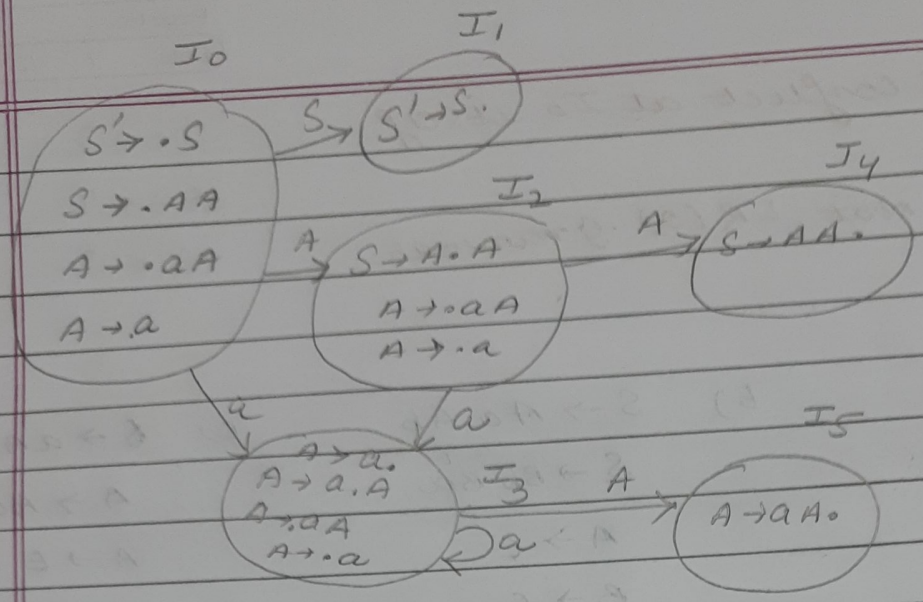
NO conflict

(b)



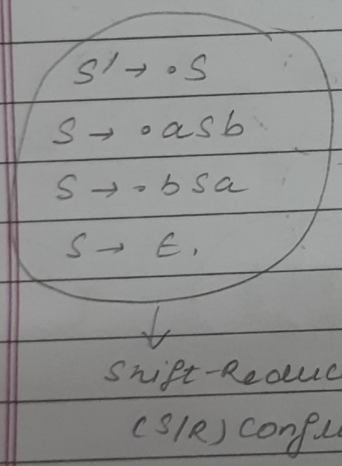
Reduce/Reduce conflict

d)



I_3 में S/R conflict है।

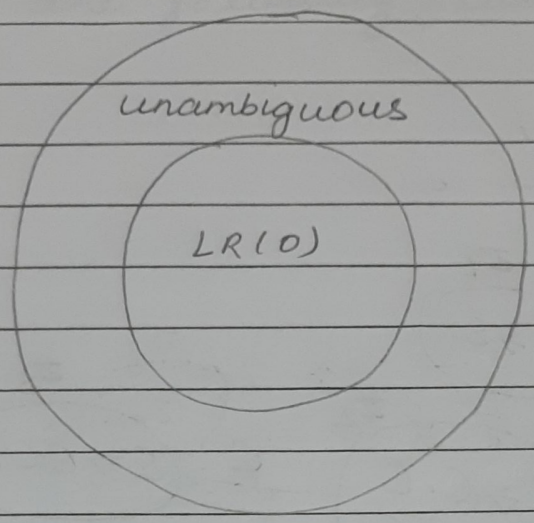
a)



NOTE :-

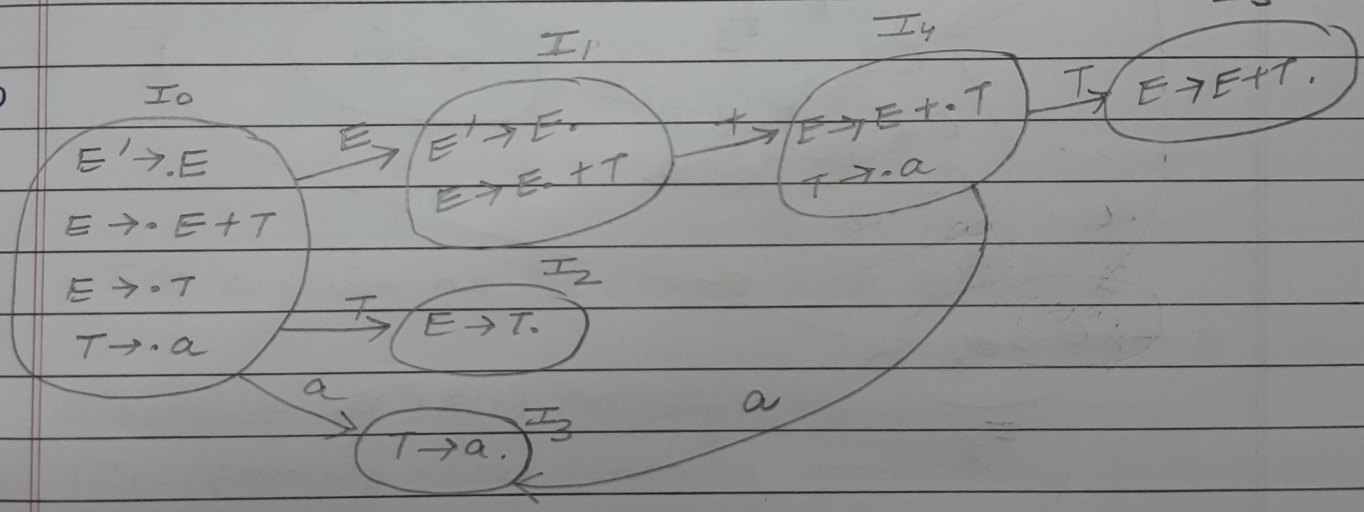
1) Any ambiguous grammar is not LR(0)

2) For any ambiguous grammar S/R conflict or R/R Conflict occurs in the parsing Table.



Q. verify the following grammar is LR(0) grammar or not ?

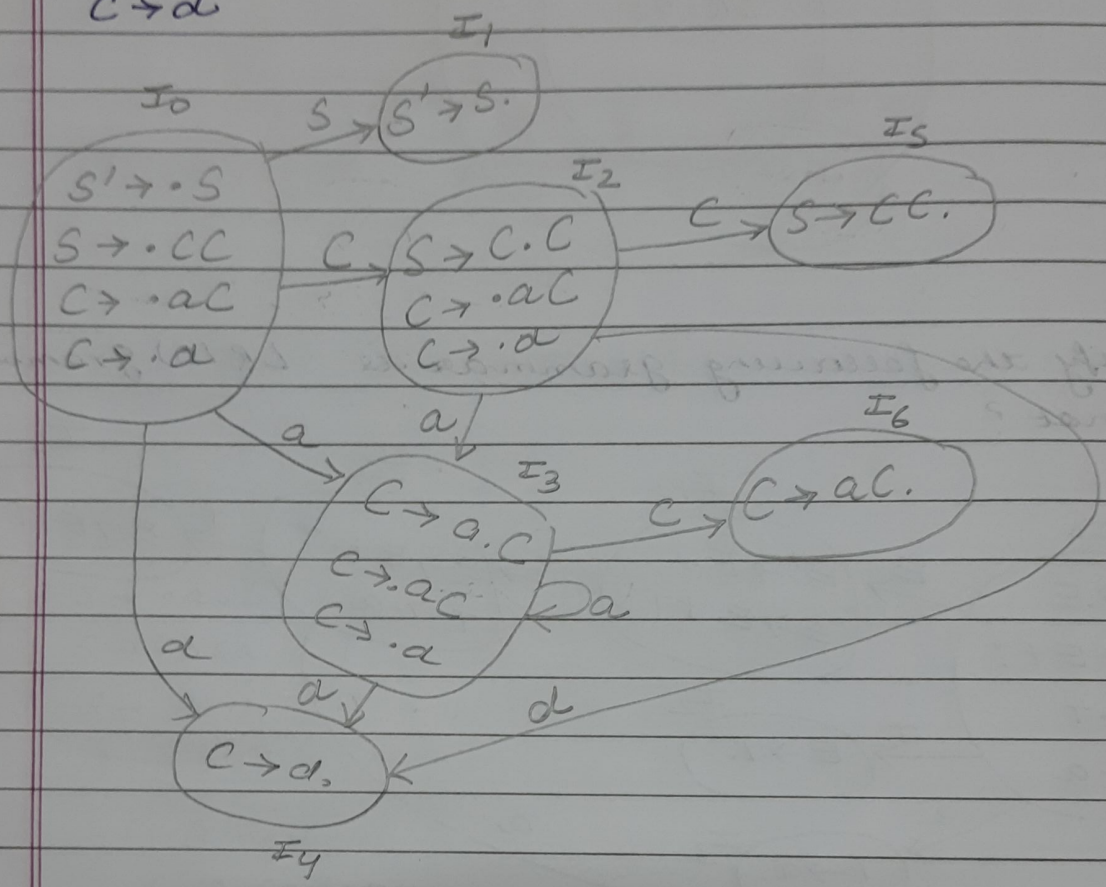
(i)



No conflict ; Here The grammar is LR(0)

* (11) GATE2003

$S \to CC$
$C \to aC$
$C \to d$



No conflict, hence LR(0) grammar

L-17

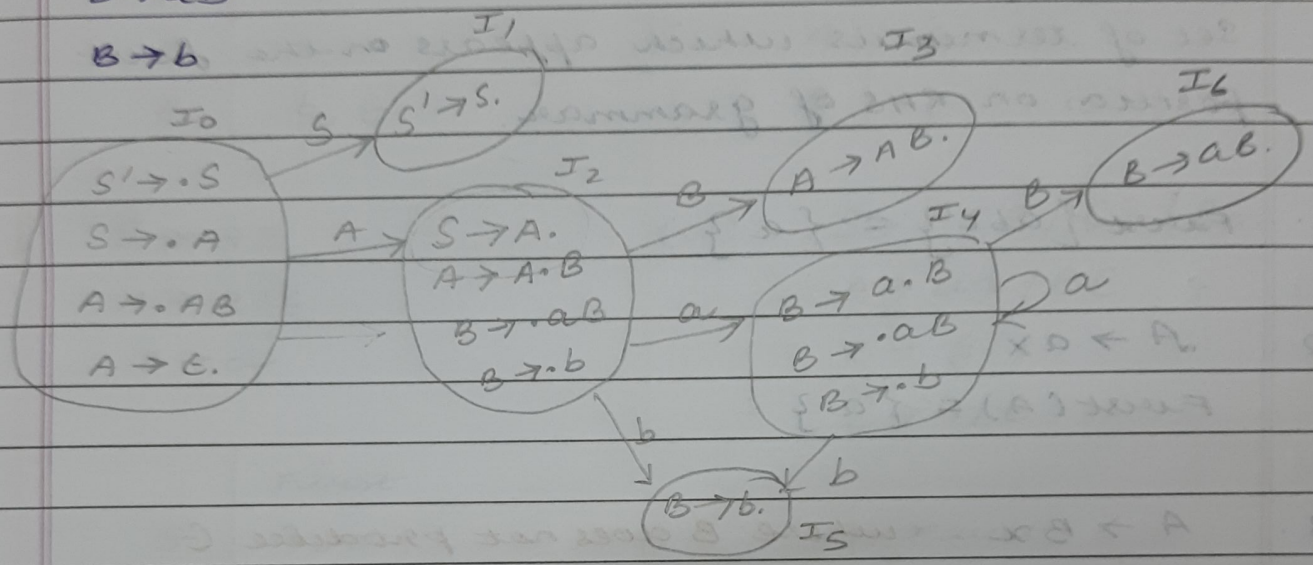
शुक्रवार

Page No.
Date : / /201

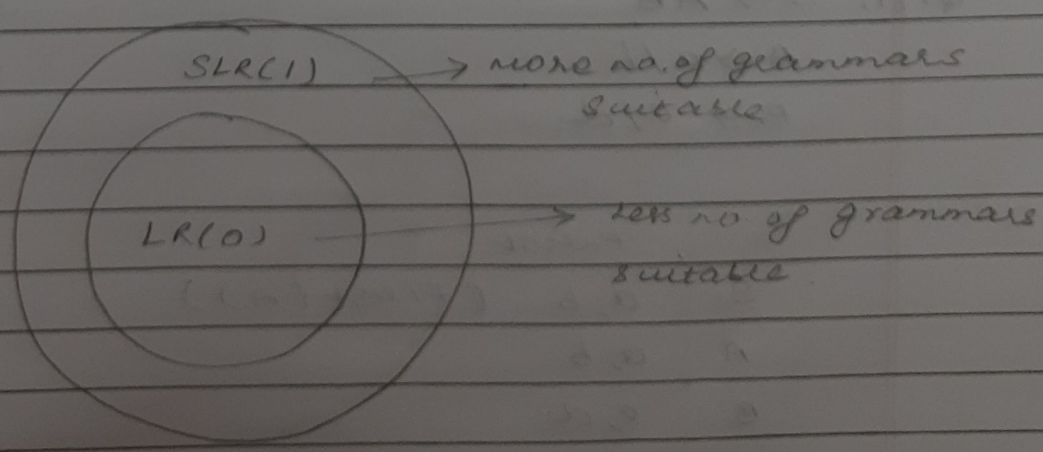
30-07-2021

Q. Verify if the following grammar is LR(0) or not?

$S \to A$
$A \to AB$
$A \to \epsilon$
$B \to aB$
$B \to b$



S/R conflict at I_2, hence not LR(0)

Drawback of LR(0) Parsing

1. LR(0) parsing table contains unnecessary reduce actions
2. In LR(0) table reduce actions are placed without considering lookahead symbols.