18-7-2024.

- A compiler is a s/w that translates a HLL to LLL. or converts program written in

LANGUAGE PROCESSING SYSTEM. (convert HLL→LLL)
I/P - Source code

1. PREPROCESSOR.
   - removes the preprocessed directories.
   - replaces lib / headers with pure HLL.
   - O/P - expanded source code.

2. COMPILER.
   - Lexical Analysis
   - Syntactical Analysis
   - Semantic Analysis
   - Intermediate code generator.
   - Code Optimizer.
   - Code Generator.

Language Processing System

HLL
↓
Preprocessor
↓ Pure HLL
Compiler
↓ Assembly Language
Assembler
↓ Relocatable Machine code
Loader / Linker
↓
Absolute Machine code.

19-07-2024

- Phases of compiler:-

1. LEXICAL ANALYSIS.
- reads source program character by character & converts corresponding character into some meaningful sequences called lexemes.
- Lexeme means sequence of characters
- each lexeme converted into form of token ie keyword, identifier, const, operator
- o/p → token.

2. SYNTAX ANALYSIS.
- generates parse tree
- checks whether syntax of source program is correct or not.
- if not then error handler will report to user.
- if correct then generates parse tree corresponding to the source program.
o/p → parse tree.

3. SEMANTIC ANALYSIS:
- checks meaning of parse tree is correct or not.
- whether it follows rules of language.
- keeps tracks of identifiers, their type & expressions.

4. INTERMEDIATE CODE GENERATOR:
- generates intermediate code which can be readily executed by the machine. ~~(incomplete check slides)~~. eg: 3 address codes.
Intermediate code is converted to M Language using last phases which are platform dependent.

5. CODE OPTIMIZER :
- optimizes the intermediate code.
- transforms s.t. it consumes fewer resources & produces more speed.
- code is not altered.

24-07-24

6. CODE GENERATOR:
- generates actual machine code.

25-07-24

Eg:

$a = b + c * 60$

Lexemes	Tokens	token name ↑
a	id.	<id, 1> (1 → "a")
=	Assignment operator	
b	id	
+	Addition (Arithmetic) operator	
c	id	
*	Multiplication operator.	
60	constant.	

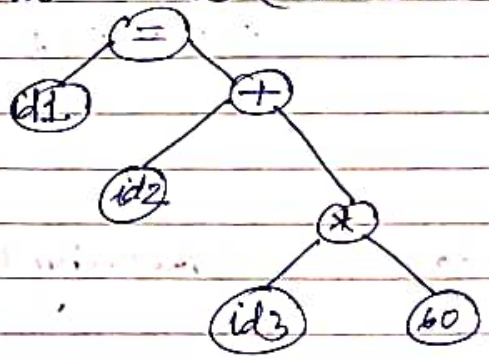
if token is identifier, it will have 2 components
- Token Name
- Attribute.

<id, 1>, <=>, <id, 2>, <+>, <id, 3>, <*>, <60>

(7 - tokens) id1 = id2 + id3 * 60

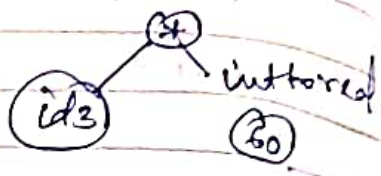
Information is stored in 'symbol table'. (info about identifiers, attr.)

→ For construction of parse tree:
operator is taken as node (internal & parent node).



25-07-24.

→ Semantic analysis phase eg: if b & c in floating point then type conversion will be performed in this phase.
Tool & s/w will be used.
Type checker will be used.



→ Intermediate code generator: (notation) eg: 3 address code.

→ i) 3 address code assignment inst^{n} should have at most 1 operator on RHS
ii) The compiler must generate a temporarily variable for storing the result.
iii) Some inst^{n} may contain fewer than 3 operand.

eg:

(Rule ii) t1 = int to real(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3 (Rule iii) ↑ (Rule i)

} Intermediate code.

4 LOC

→ Code optimizer

t1 = id3 * 60.0
id1 = id2 + t1

2 LOC
∴ optimised.

→ code generator. (Assembly language).

25-07-24

LD - Reg M
ST - M Reg.
ADD - R1 R2 R3
MUL - R1 R2 R3

```
LDF   R2, id3
MUL   R2, R2, #60.0
LDF   R1, id2
ADDF  R2, R2, R1
STF   id1, R2
```

26-07-24

→ LEXICAL ANALYSIS.

- Lexical analysis is the process of converting a sequence of characters from source program into a sequence of tokens.
- Program which performs this process is called lexical analyser (lexer), tokenizer or scanner.
- Sequence of tokens produced by lexical analyser helps the parser in analysing syntax of programming language.
- I/P → pure HLL
  identifies valid lexemes
  Returns tokens to parser (syntax analyser), corresponding to getNextToken command from parser.

26-07-24.

→ ROLE / USE / WORK of LEXICAL ANALYSER:-

i) generates tokens.
ii) Removes comments
iii) Removes whitespace like new line, linespace & wordspace.
iv) counts the line number of the source program
v) it generates the symbol table which stores the info. about identifier.
vi) provides error messages with corresponding line no. & column no.

→ 3 important terms,

→ TOKENS.

- predefined sequence of characters that cannot be broken down further.
- abstract symbol that represents a unit.
- can have optional attr value.

Eg: identifiers

31-07-24

Notation

+ → 1/more occurence
* → 0/more
? → 0/1
/ → only one.

Regular Expression:

id → (letter) (letter | digit)*

letter → A|B|....|Z|a|b|...|z
digit → 0|1|...|9

Identifier:

id → [A-Za-z][A-Za-z ≡ 0-9]*

~~[0-9]+[.0-9]*~~

~~[0-9]*~~

Number →
[0-9]+ [.]? [0-9]* ~~[0-9]+[.0-9]?~~

Number:

digit → 0|1|2|...|9
digits → digits+
optional_fraction → (.digits)?
optional_exponent → (E (+|-)? digits)?

num → digits optional_fraction optional_exponent.

01-08-24

Ch-2 Syntax Analysis (Parser).

- I/P – string of token
- generate a parse tree

- Lexical analyser can't check syntax of a sentence due to limitation of regular expression.

> Role of parser :-

1. It verifies the structure generated by the tokens based on the grammar.
2. reports error
3. constructs parse tree
4.

- 2 decision while parsing :-

1) Leftmost derivation
2) Rightmost derivation

(V)
(which Non T to replace)

01/8/24

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

$E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id.$

$w = id + id * id.$

Left

$E \rightarrow E+E$ — T
$E \rightarrow id + E$ $(E \rightarrow id.)$ — V
$E \rightarrow id + E*E$ $(E \rightarrow E*E).$ — S
$E \rightarrow id + id * E$ —
$E \rightarrow id + id * id.$

Right

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

— Ro.

Parse tree — pictorial representation of derivation

Rules :

Root — Start symbol
Leaf — terminal
Internal node — Non-terminal

01/08/24

$S \rightarrow S+S \mid S*S$

$S \rightarrow a \mid b \mid c.$

$a * b + c$

~~$S \rightarrow S$~~

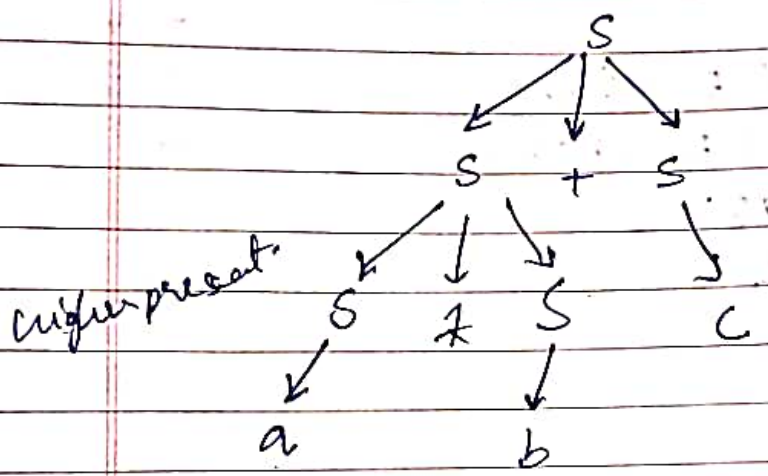
$S \rightarrow S+S$

$S \rightarrow S*S+S$

$S \rightarrow a*S+S$

$S \rightarrow a*b+S$

$S \rightarrow a*b+c$



9. $S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S.$

start symbol – S

Nonterminal – {L, S}

Terminal – {(,), ,}

Lm

Rm

Up –

a) (a, a)

b) (a, (a, a))

01/08/24

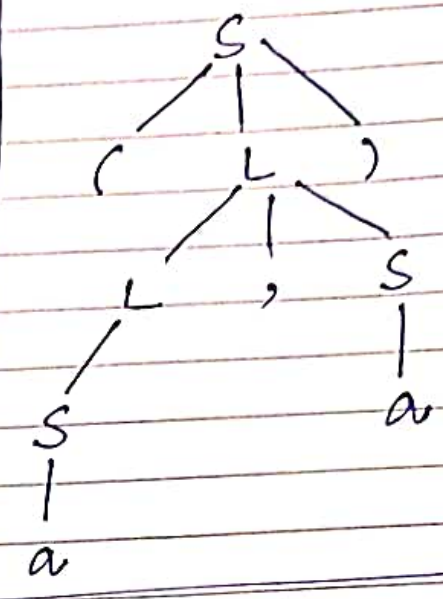
Lm

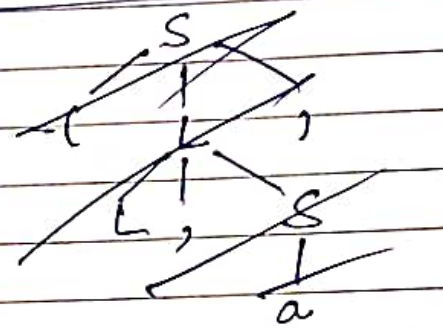
$S \xrightarrow{lm} (L)$
$S \xrightarrow{lm} (L,S)$ [L → L,S]
$S \xrightarrow{lm} (S,S)$ [L → S]
$S \xrightarrow{lm} (a,S)$ [S → a]
$S \longrightarrow (a,a)$ [S → a]

Rm

$S \xrightarrow{rm} (L)$
$S \longrightarrow (L,S)$
$S \longrightarrow (L,a)$
$S \longrightarrow (S,a)$
$S \longrightarrow (a,a)$

Parse tree





Lm

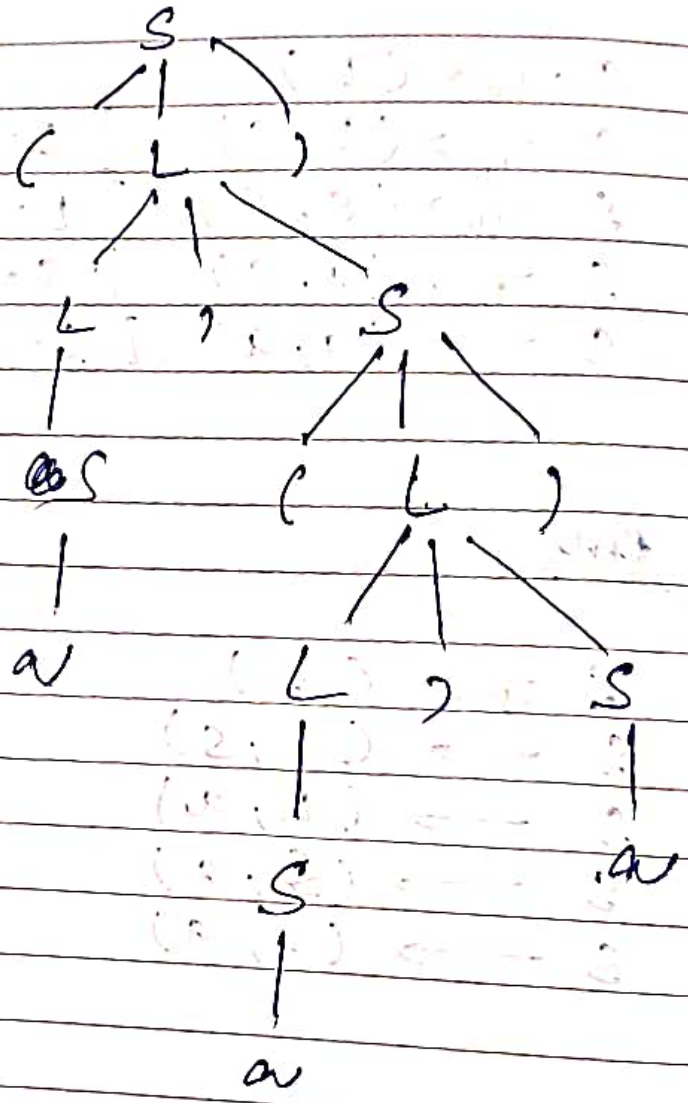
$S \longrightarrow (L)$
$S \longrightarrow (L,S)$ [L → L,S]
$S \longrightarrow (S,S)$ [L → S]
$S \longrightarrow (a,S)$ [S → a]
$S \longrightarrow (a,(L))$ [S → (L)]
$S \longrightarrow (a,(L,S))$ [L → L,S]
$S \longrightarrow (a,(S,S))$ L → S
$S \longrightarrow (a,(a,S))$ S → a
$S \longrightarrow (a,(a,a))$

Rm

$S \longrightarrow (L)$
$S \longrightarrow (L,S)$
$S \longrightarrow (L,(L))$
$S \longrightarrow (L,(L,S))$
$S \longrightarrow (L,(L,a))$
$S \longrightarrow (L,(S,a))$
$S \longrightarrow (L,(a,a))$
$S \longrightarrow (S,(a,a))$
$S \longrightarrow (a,(a,a))$

01/08/24

Parse tree



02/08/24

- Ambiguity in grammar.
- if we derive more than 1 lm/rm derivation or one/more than 1 parse tree
- No ambiguous.
- Instead of removing the ambiguity we can rewrite the whole grammar w/o ambiguity.

Q. . . Rm

$E \rightarrow E+E$

~~$E \rightarrow E-E+E$~~

~~$E \rightarrow id-E+E$~~

~~$E \rightarrow id-id$~~

$E \rightarrow E+E-E$

$E \rightarrow E+E-id$

$E \rightarrow E+id-id$

$E \rightarrow id+id-id$

$E \rightarrow E-E$

$E \rightarrow E+E-E$

$E \rightarrow id+E-E$

$E \rightarrow id+id-E$

$E \rightarrow id+id-id.$

$E \rightarrow E+E$

$E \rightarrow id+E$

$E \rightarrow id+E-E$

$E \rightarrow id+id-E$

$E \rightarrow id+id-$

$E \rightarrow E+E$

$E \rightarrow E-E$

$E \rightarrow id$

$w = id+id-id.$

02/08/24.

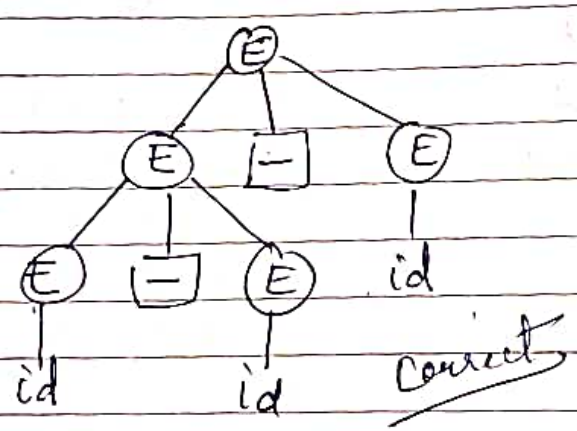
- Remove the ambiguity.
  i) Precedence
  ii) Associativity.

Q. $E \rightarrow E-E \mid id.$

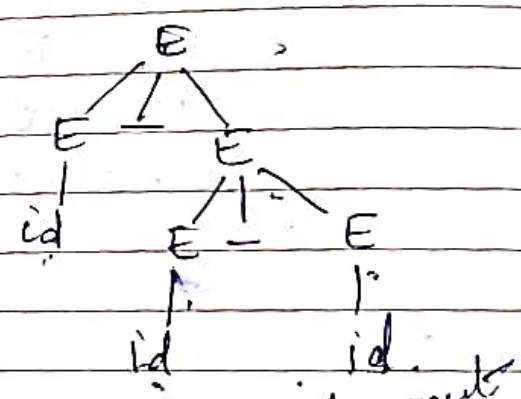
$w = id-id-id.$

$E \rightarrow E-E$
$E \rightarrow E-E-E$
$E \rightarrow id-id-id.$

$E \rightarrow E-E$
$E \rightarrow id-E$
$E \rightarrow id-E-E$
$E \rightarrow id-id-id.$



Correct



Incorrect

Left association.
∴ expand leftmost side.

02/08/24.

→

$E \rightarrow E - E \mid id$

$E \rightarrow E - P \mid P$
$P \rightarrow id$

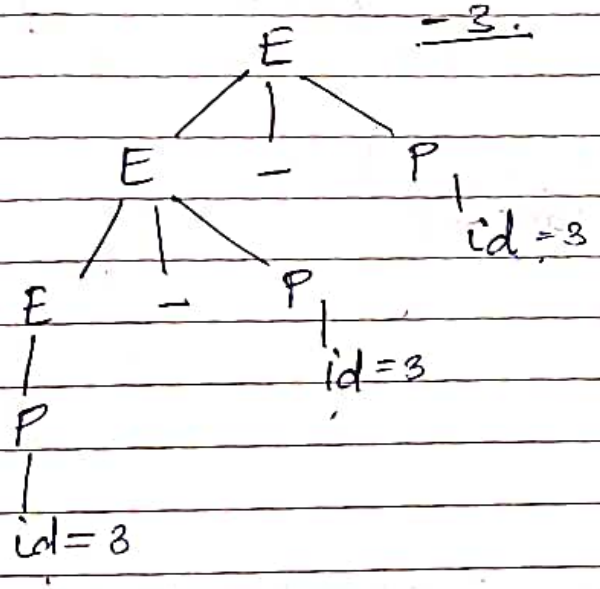
} unambiguous

P.

1) left recursive
2) Replace with random variable

$E \rightarrow E - P$
$E \rightarrow E - P - P$
$E \rightarrow P - P - P$
$E \rightarrow id - P - P$
$E \rightarrow id - id - P$
$E \rightarrow id - id - id.$

= 3.

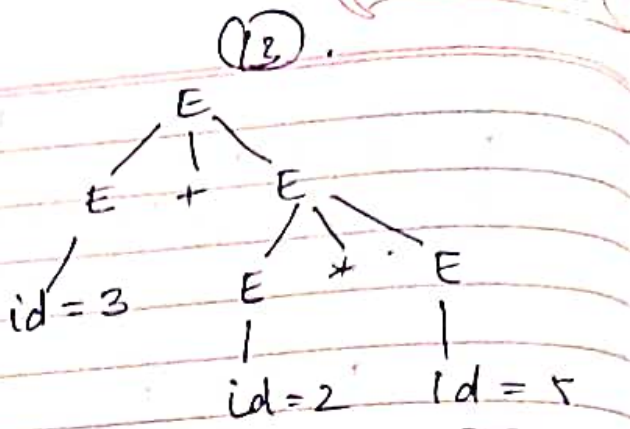


00/08/24.

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

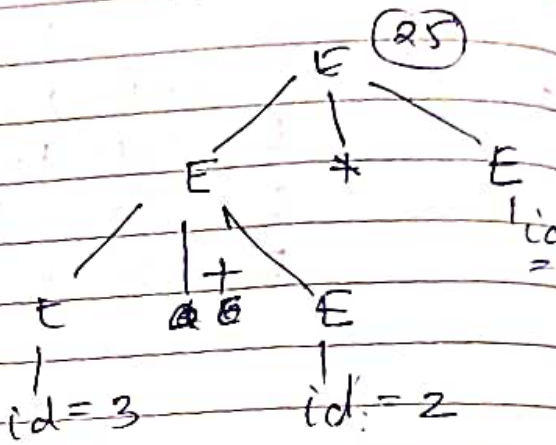
$id + id * id.$

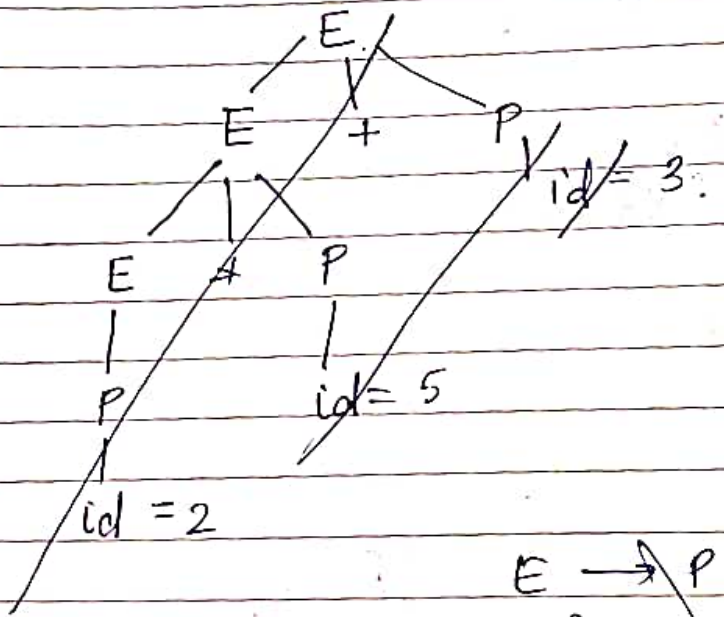
$3 + 2 * 5 = 13.$



$E \rightarrow E*P \mid E+P \mid P$

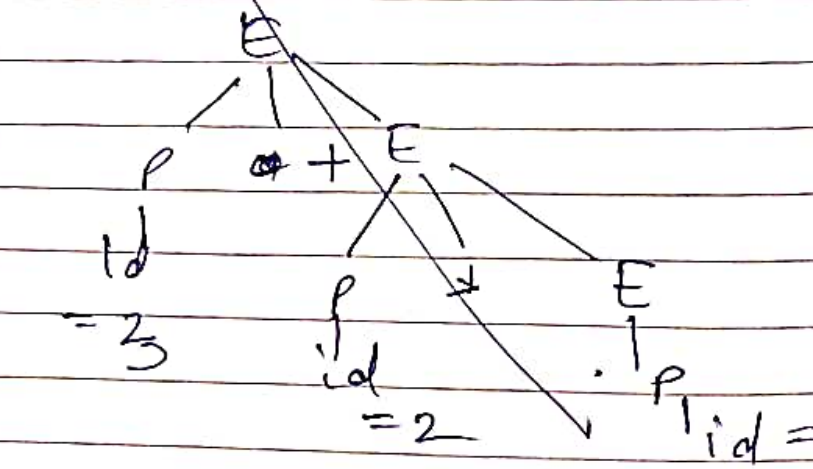
$P \rightarrow id$





$E \rightarrow P*E \mid P+E \mid P$

$P \rightarrow id.$



02/08/24.

$E \rightarrow E+P \mid P \mid A$.
$P \rightarrow A * A$
$A \rightarrow id$.

$E \rightarrow E+P \mid P \mid Q$.
$P \rightarrow P * Q \mid Q$
$Q \rightarrow id$.

→ Left Recursive

$A \rightarrow A\alpha \mid \beta$.

in syntax analysis phase, it will create an infinite loop.

Q. $A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \varepsilon$

Q1 $E \rightarrow E+T \mid T$
Q. $T \rightarrow T * F \mid F$
Q. $F \rightarrow (E) \mid id$.

A1 $E \rightarrow T+E'$
$E' \rightarrow T+E' \mid \varepsilon$

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

A2 $T \rightarrow F*T'$
$T' \rightarrow F*T' \mid \varepsilon$

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

A3 $F \rightarrow (E) \mid id$.

LLG XLG

02/09/24

- Rule Indirect Left Recur.

Eg $S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Sd \mid \varepsilon$

Q. $A \rightarrow \underset{\alpha_1}{Ad} \mid \underset{\alpha_2}{Ae} \mid \underset{\beta_1}{aB} \mid \underset{\beta_2}{aC}$

$B \rightarrow bBcf.$

$A \rightarrow aBA' \mid aCA'$

$A' \rightarrow dA' \mid eA' \mid \varepsilon$

Eg $S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Sd \mid \varepsilon$

till Ass 1 here

$S \rightarrow Aa \mid b$

$A \rightarrow SdA' \mid A'$

$A' \rightarrow cA' \mid \varepsilon$

03/09/24

ELG

Q. $S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Sd \mid \varepsilon$

$A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$

$A \rightarrow bdA' \mid bdA' \mid A'$

$A' \rightarrow cA' \mid adA' \mid \varepsilon$

Q. $X \rightarrow XSb \mid Sa \mid b$

$S \rightarrow Sb \mid Xa \mid a$

$X \rightarrow SaX' \mid bX'$

$X' \rightarrow SbX' \mid \varepsilon$

$S \rightarrow Sb \mid Xa \mid a$

$S \rightarrow Sb \mid SaX'a \mid bX'a \mid a$

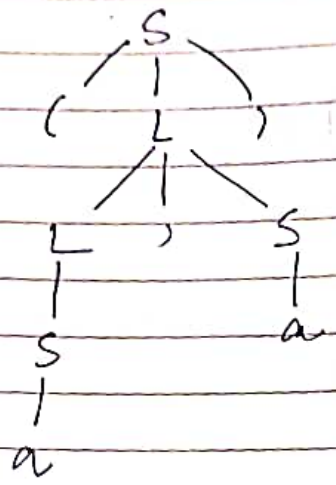
$S \rightarrow bX'aS' \mid aS'$

$S' \rightarrow bS' \mid aX'aS' \mid \varepsilon$

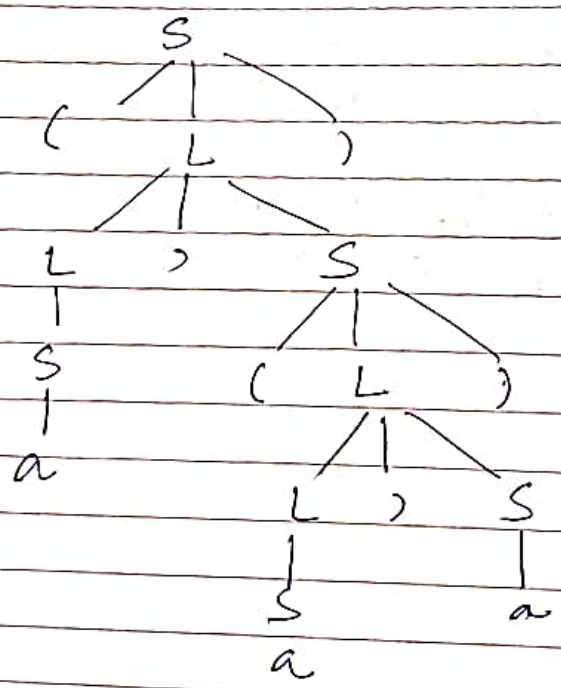
7

S → (L) | a
S → L,S | S

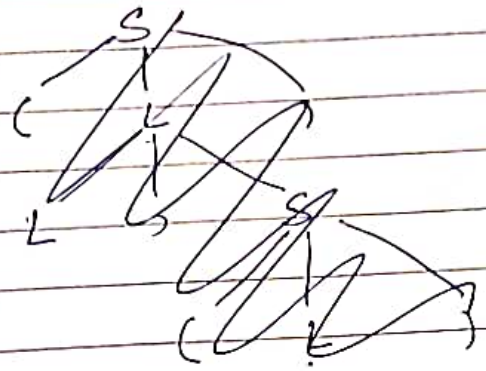
a) (a,a)

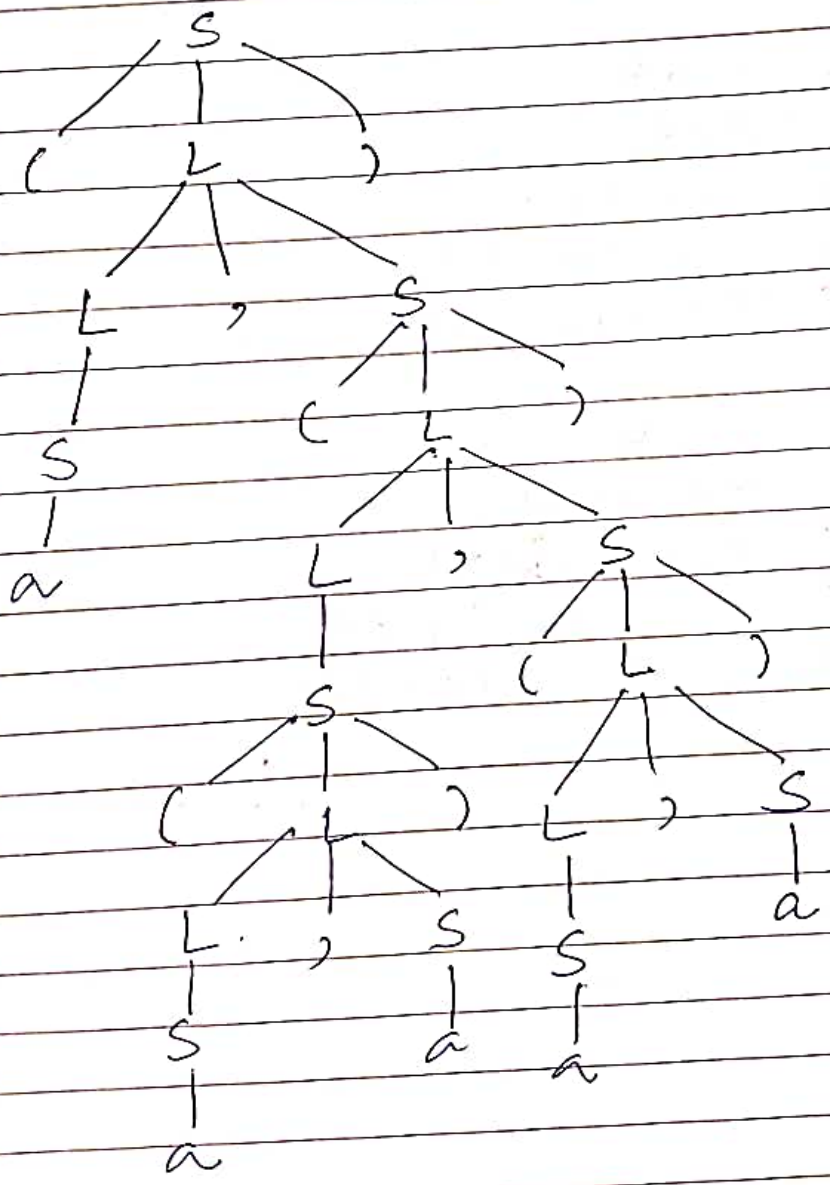


b) (a, (a,a))



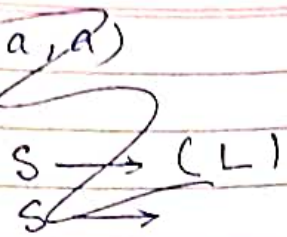
c) (a,((a,a),(a,a)))





b. (a, a)

7



9) $S \rightarrow AaAb \mid Bbba$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

8.

$S \rightarrow AB$

$S \rightarrow aAbB$

$S \rightarrow aabbB$

$S \rightarrow aabbcBd.$

$S \rightarrow aabbccdd.$

$S \rightarrow C$

$S \rightarrow aCd$

$S \rightarrow aaDdd$

$S \rightarrow aabDcdd$

$S \rightarrow aabbccdd.$

∴ ambiguous.

$E \longrightarrow T + E \mid T$

$E \longrightarrow int \mid int + T \mid (E)$

$A \longrightarrow ABd \mid Aa \mid a$

$B \longrightarrow Be \mid b.$

Step 1

$A \longrightarrow aA'$

$A' \longrightarrow Bd A' \mid aA' \mid \varepsilon$

Step 2

$B \longrightarrow bB'$

$B' \longrightarrow eB' \mid \varepsilon.$