Derivation of CFG

DATE: / /
PAGE:

Derivation is a sequence of production rules. It is used to get i/p strings. During parsing, we have to take two decisions.

→ we have to decide the non-terminal which is to be replaced.

→ we have to decide the production rule by which the non-terminal will be replaced.

* * We have two options to decide with non terminal to be placed with production rule.

① Left most derivation

② Right most derivation

e.g. $S \rightarrow aABb$ [non-terminal element should be replaced]

A has to be replaced first in Left most derivation

B has to be replaced in Rightmost derivation

DATE: / /
PAGE:

Left-most derivation Rule

In the left-most derivation the i/p is scanned and replaced with the production rule from left to right. So, we have to read i/p string from left to right.

eg. Production rules:

Read this string from left to right

Input: a - b + a

E = E + E
E = E - E
E = a/b a, b
a or b

The Left most derivation is

E = E + E
^
E - E + E
/
a - E + E
a - b + E
a - b + a

Right Most derivation

DATE: / /
PAGE:

Eg. $E = E - E$

$E - \hat{E} + E$

$E - E + a$

$E - b + a$

$a - b + a$

Examples:

Q. Derive the string "abb" for the left most derivation & right most derivation using CFG given by

$S \to AB / \varepsilon$ --①

$A \to aB$ --②

$B \to Sb$ --③

Sol^{n} Left-Most-Derivation (LMD)

start is S

S

↓

AB

/

aB B

a S b B

ab SB

abb

$S \to AB$

$A \to aB$

$B \to Sb$

$S \to \varepsilon$

$B \to SB$

$S \to \varepsilon$

Right Most Derivation

S

$\Rightarrow$ AB

$\Rightarrow$ A Sb

$\Rightarrow$ A ε b

$\Rightarrow$ A b

$\Rightarrow$ aB b $\Rightarrow$ a Sb b $\Rightarrow$ abb

$S \to AB$

$B \to Sb$

$S \to b$

$A \to aB$

$B \to Sb$ | $S \to \varepsilon$

DATE: / /
PAGE:

Q. Derive the string 00101 for left most derivation & right most derivation using CFG give

$S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

Sol: LMD:

S

A1B (S → A1B)

↓

0A1B (A → 0A)

↓

00ε1B (A → ε)

↓

001 0B (B → 0B)

↓

0010 1B (B → 1B)

00101ε (B → ε)

00101 Derived

RMD:

S

A1B (S → A1B)

↓

A10B (B → 0B)

A101B (B → 1B)

A101ε (B → ε)

A101

↑

0A101 (A → 0A)

→ 0A101

00A101 (A

00ε101 (

00101

Derived

DATE: / /
PAGE:

2. Derive the string "aabbabba" for ~~left~~ LMD & RMD using a CFG given as

$S \to aB / bA$
$A \to a / aS / bAA$
$B \to b / bS / aBB$

Sol^n LMD :

S
↓
aB (S → aB)
↓
aaBB (B → a/B.B)
↓
aabSB (B → bS)
~~aabbAABB (S →~~
↓
aabbAB (S → b/A)
↓
aabbaSB (A → aS)
↓
aabbabAB (S → bA)
↓
aabbabbAAB (S → bA)
aabbabbabb

a a b b a b b a
↑ ↑ ↑ ↑ ↑ ↑ ↑

Sol^n LMD

S
↓
aB (S → aB)
aaBB (B → aBB)
aabB (B → b)
aabbS (B → bS)
aabbaB (B → S → aB)
aabbabS (B → bS)
aabbabbA (S → bA)
aabbabba (A → a)
Derived.

DATE: / /
PAGE:

RMD

S → aB / bA
A → a / aS / bA
B → b / bS / aBB

"aabbabba"

S
aB (S → aB)
aaBB (B → aBB)
aabbS (B → bS)
~~aabbbA~~ (S → bA)
aaBbbA (S → A)
aaBbba (A → a)
aabSbba (B → bS)
aabbAbba (S → bA)
aabbabba (A → a)
Derived.

Derivation Tree

It is a graphical representation for the derivation of the given production rules for a given CFG. It is simple way to show the derivation can be done to obtain some string from a given set of production rules. It is also called as Parse Tree.

→ Parse tree follows the precedence of operators. The deepest subtree traversed first. So, the operation in the parent node has less precedence over the operation in the subtree.

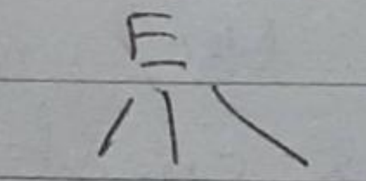
DATE: / /
PAGE:

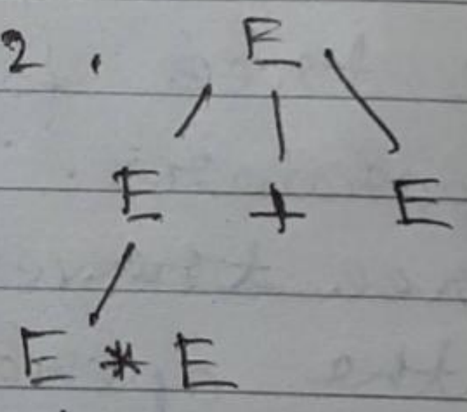
The Parse contains the following properties:

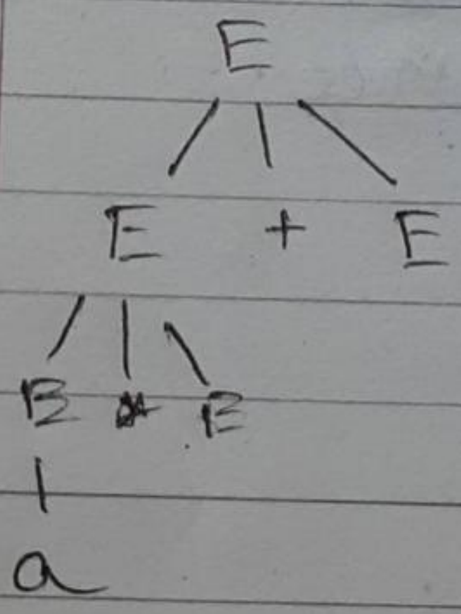
1. The root node is always a node indicating start symbols.
2. The derivation is read from left to right
3. The leaf node is always terminal nodes.
4. The intermidate nodes are always non terminal nodes.

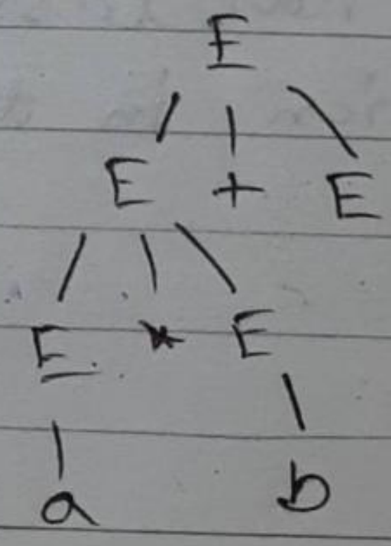
Ex. E = E + E
E = E * E
E = a/b/c

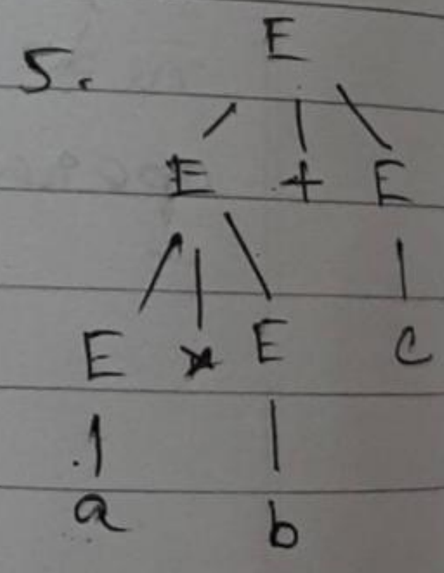
Input
a * b + c

1.


2.


3.


4.


5.


DATE: / /
PAGE:

Ex: Let Gr be the grammar

S → 0B / 1A

A → 0 / 0S / 1AA

B → 1 / 1S / 0BB

For the string 00110101.

Find a) Left most derivation

b) Right most "

& c) The derivation tree

Sol n=1

LMD

S

⇒ 0B | S → 0B

⇒ 00BB | B → 0BB

⇒ 001SB | B → 1S

⇒ 0011AB | S → 1A

⇒ 00110SB | A → 0S

⇒ 001101AB | S → 1A

⇒ 00110101B | A → 0

⇒ 00110101 | B → 1

Derived.

S

⇒ 0B | S → 0B

⇒ 00BB | B → 0BB

⇒ 001SB | B → 1S

⇒ 0011AB | S → 1A

⇒ 00110SB | A → 0S

⇒ 001101AB | S → 1A

⇒ 00110101B | A → 0

⇒ 00110101 | B → 1

Derived.

DATE:

PAGE:

R. M. S.

$S \to 0B \mid 1A$

$A \to 0 \mid 0S \mid 1AA$

$B \to 1 \mid 1S \mid 0BB$

00110101

S

$\Rightarrow$ 0B | S → 0B

$\Rightarrow$ 00BB | B → 0B

$\Rightarrow$ 00B1S | B → 1S

$\Rightarrow$ 00BB1 | S → 0B

$\Rightarrow$ 00BB10B | B → 1S

$\Rightarrow$ 00B1S10 | S → 0B

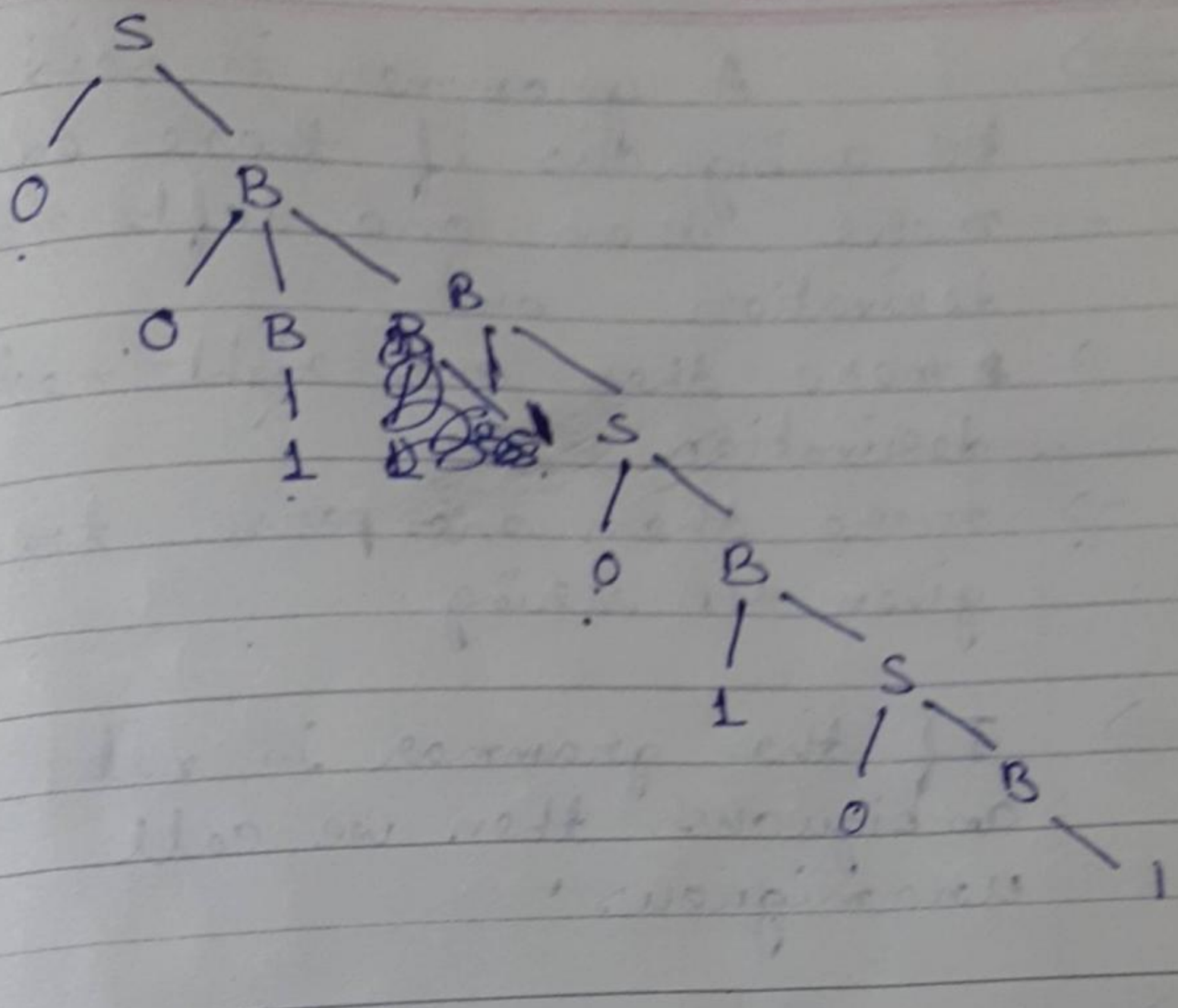
$\Rightarrow$ 00B1010

$\Rightarrow$ 00110101

Derived.

DATE: / /

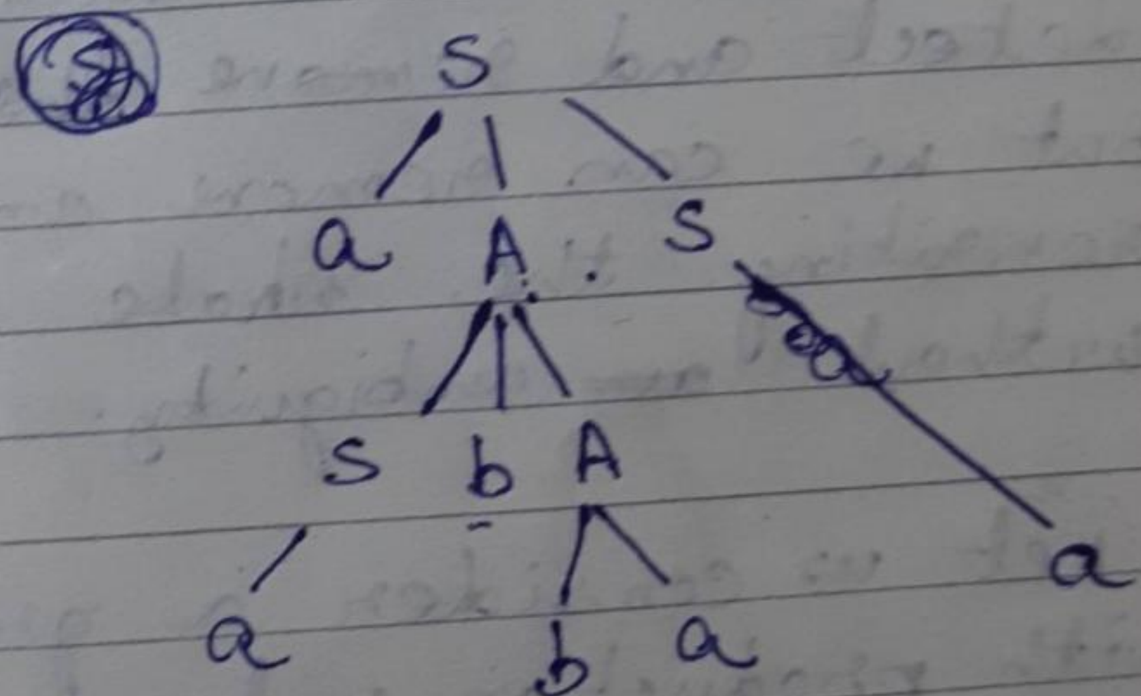
PAGE:



Ex. string aabbaa

$S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid SS \mid ba$



CFG

DATE: / /
PAGE:

Ambiguity in Grammar

⇒ A grammar is said to be ambiguous if there exists
→ more than one left most derivation or
→ more than one right-most derivation or
→ more than one parse tree for given i/p string.

⇒ If the grammar is not ambiguous, then we call unambiguous.

⇒ If the grammar has ambiguity, then it is not good for compiler construction.

⇒ No method can automatically detect and remove the ambiguity, but we can remove ambiguity by rewriting the whole grammar without ambiguity.

Ex. Let us consider a grammar with production rule.

E → I
E → E+E
E → E*E
E → (E)
I → ε / 0 / 1 / 2 / ... 9

DATE: / /
PAGE:

Sol^{n}

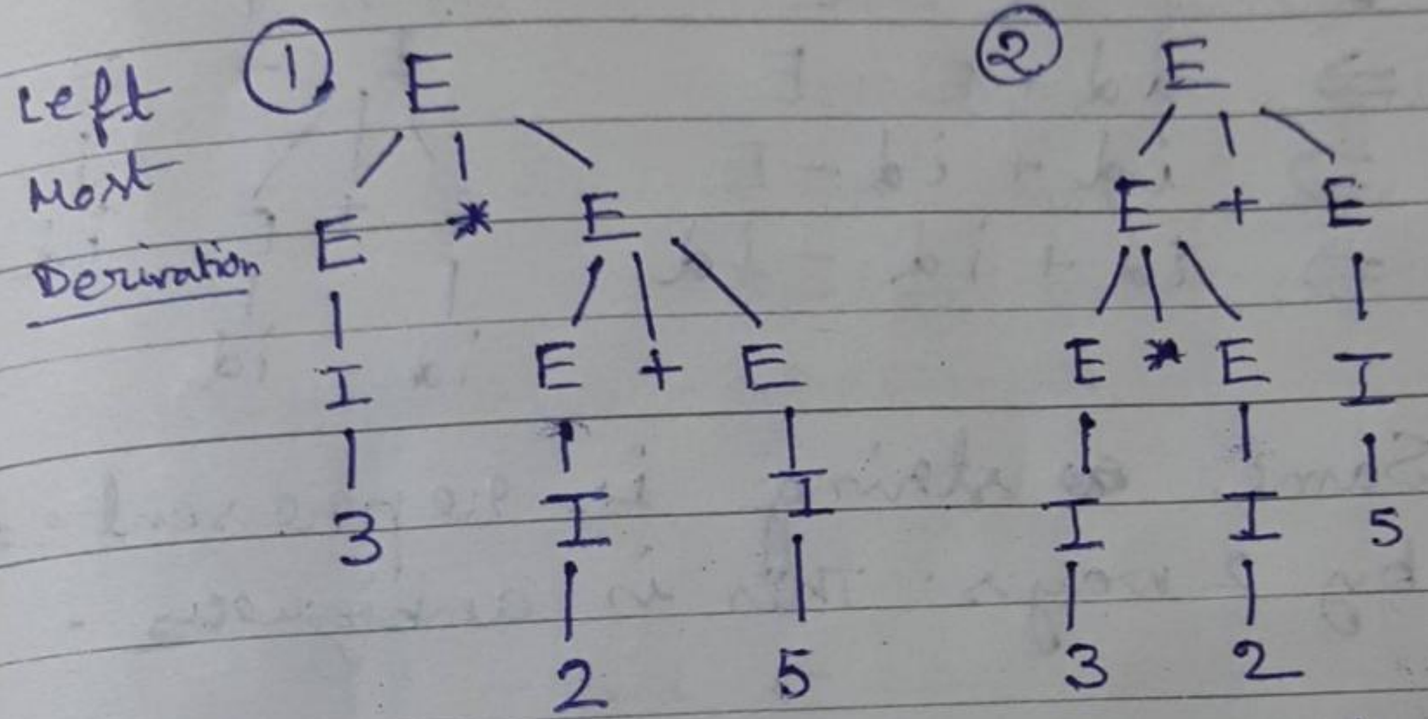
$G = \{V, T, P, S\}$

$V = \{I, E\}$

$T = \{+, *, (,), \varepsilon, 0, 1, 2, \cdots 9\}$

For the string "3*2+5", the above grammar can generate two phase tree by leftmost derivation.

Left Most Derivation



Since there are 2 parse tree for the single ~~tree~~ string.
So, this grammar is ambiguous.

Ex. check whether the given grammar is ambiguous or not-

$E \rightarrow E+E$

$E \rightarrow E-E$

$E \rightarrow id$

Sol^{n} "id + id - id"

DATE: / /
PAGE:

LMD

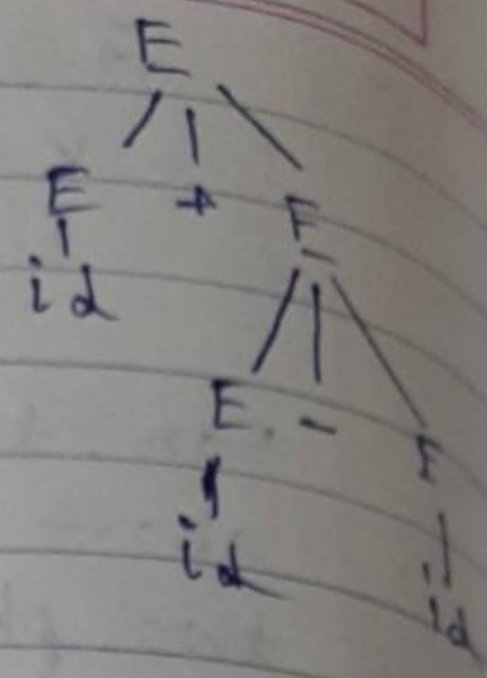
(1) $E \to E + E$

$\Rightarrow id + E$

$\Rightarrow id + E - E$

$\Rightarrow id + id - E$

$\Rightarrow id + id - id$



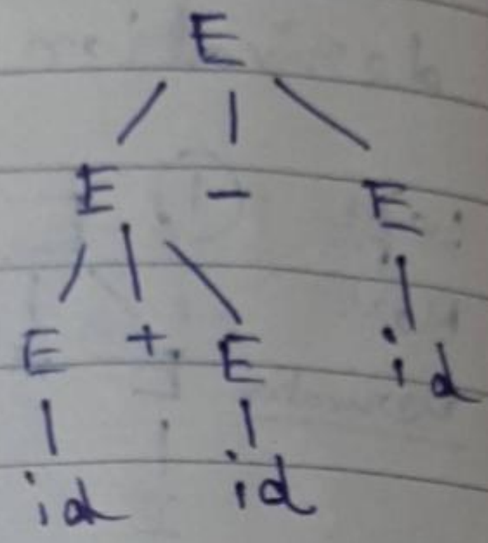
(2) $E \to E - E$

$\Rightarrow E + E - E$

$\Rightarrow id + E - E$

$\Rightarrow id + id - E$

$\Rightarrow id + id - id$



Same string is represented by 2 ways. This is ambiguous.

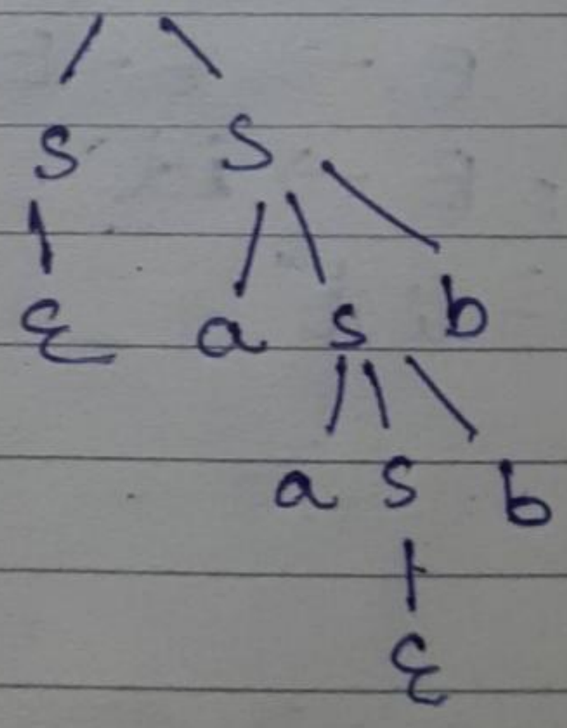
Ex. Check whether the given grammar is ambiguous or not

$S \to aSb / SS$

$S \to \varepsilon$

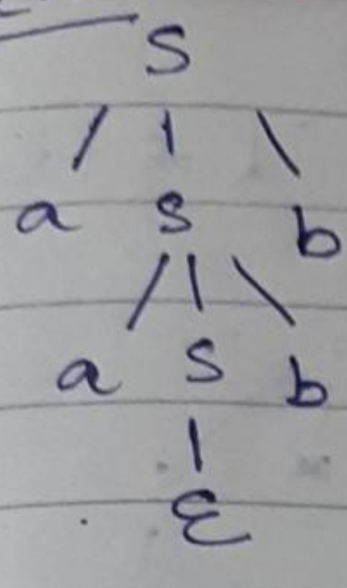
"aabb"

Soln. LMD



DATE: / /
PAGE:

② LMD



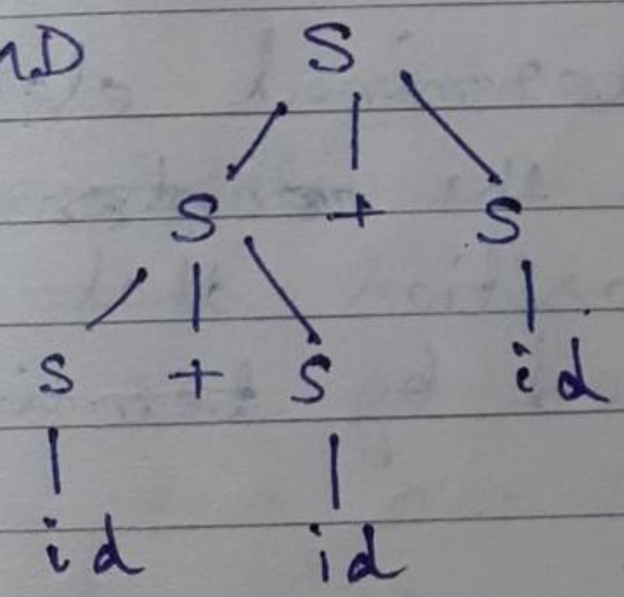
∴ The grammar is ambiguous.

Ex. $S \to S + id$
$S \to id$

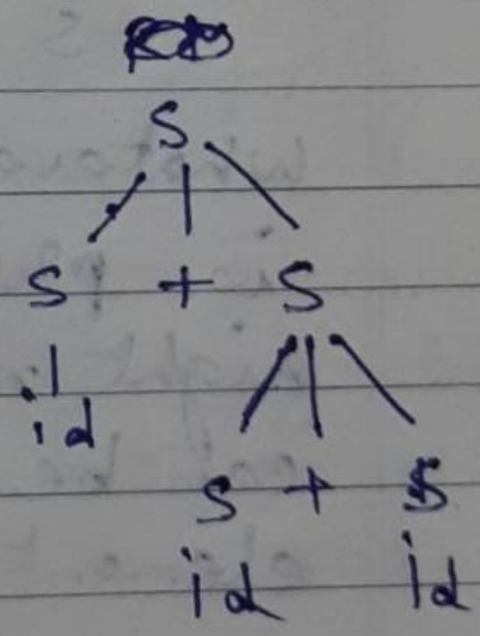
Determine whether the grammar is ambiguous grammar or not. If it is ambiguous construct unambiguous grammar.

Soln "id + id + id"

① LMD

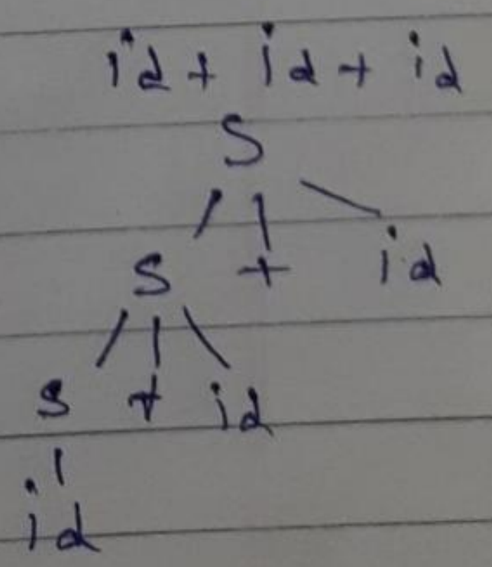


②



This is an ambiguous grammar.

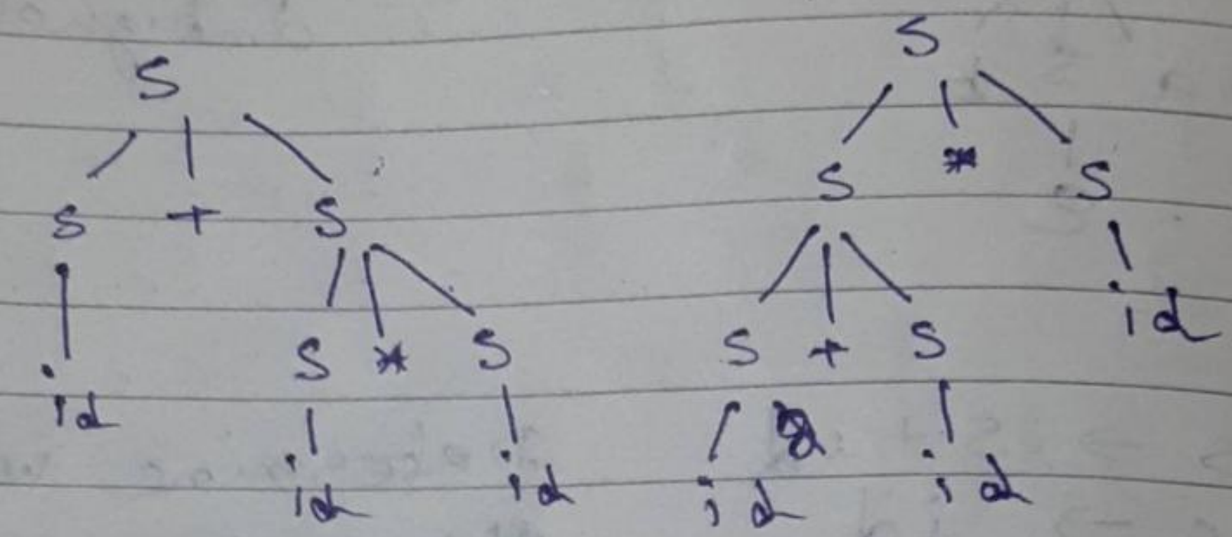
$S \to S + S$
$S \to id$
$S \to S + id$
$S \to id$



DATE: / /
PAGE:

Ex. $S \rightarrow S + S / S * S / id$

parse tree - id + id * id



$S \rightarrow S + S$

$S \rightarrow S * S$

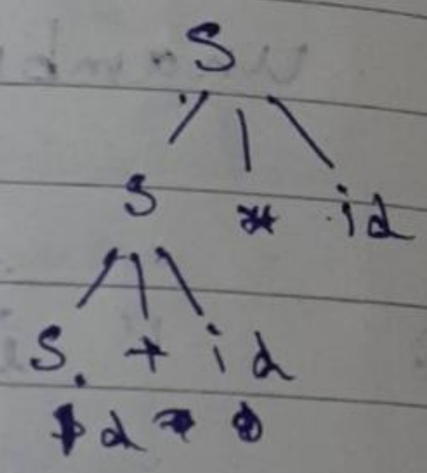
$S \rightarrow id$

$S \rightarrow S + id$

$S \rightarrow S * id$

$S \rightarrow id$

id + id * id



Whatever non terminal element is present in the ~~non terminal~~ right most position that can be replaced by terminal element.

DATE: / /
PAGE:

Unambiguous grammar

A grammar can be ambiguous if the grammar does not contain ambiguity that means if it does not contain more than one left most derivation or more than one right most derivation and more than one parse tree for the given i/p string.

To convert ambiguous grammar, we apply the following rules

① If the left associative operators (+, −, *, /) are used in the production rule, then apply the left recursion in the production rule. Right recursion means that right most symbol on the left side is the same as the non-terminal on the right side.

PAGE:

Ex. Consider a grammar Gr is

$S \rightarrow AB \mid aaB$

$A \rightarrow a \mid Aa$

$B \rightarrow b$

Determine whether the grammar Gr is ambiguous or not. If Gr is ambiguous, construct an unambiguous grammar equivalent to G.

Sol^{n}

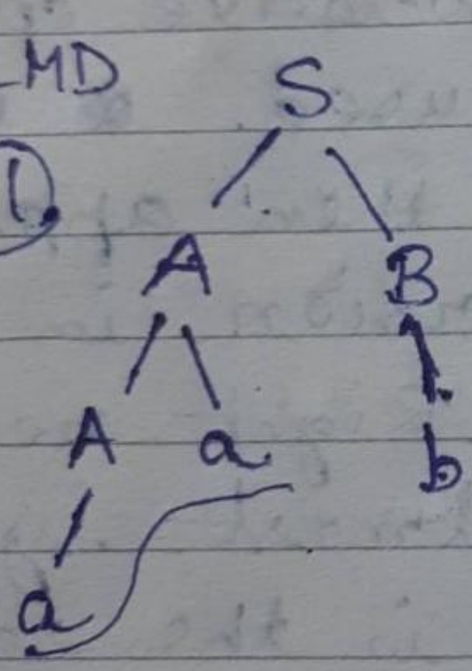
$S \rightarrow AB \mid aaB$

$A \rightarrow a \mid Aa$ ... "aab"

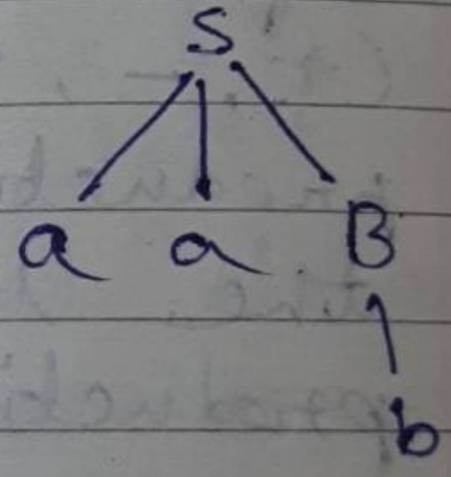
$B \rightarrow b$

LMD

(1)



(2)



The given grammar is ambiguous.

⇒ Unambiguous → convertion steps

$S \rightarrow AB$

$A \rightarrow Aa \mid a$

$B \rightarrow b$