TUTORIAL 8

U19CS012

1.) Consider the following language
The abstract language, $L1 = \{ wcw \mid w \text{ is in } (a|b)^* \}$
Check whether language is context free or not.

1.) The given language is not contexs free grammer because there is no midpoint or no PDA can be designed to accept it.

Ex:- $\underbrace{abaab}_{\omega}\,c\,\underbrace{abaab}_{\omega}$, there is no midpoint possible.

2.) What is Left recursive grammer? Remove Left recursion from the following:

1) $E \rightarrow E+T \mid T$
$T \rightarrow TXF \mid F$
$F \rightarrow id$

⇒

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow XFT' \mid \varepsilon$
$F \rightarrow id$

2) $S \rightarrow (L) \mid a$
$L \rightarrow L,S \mid S$

⇒

$S \rightarrow (L) \mid a$
$L \rightarrow SL'$
$L' \rightarrow ,SL' \mid \varepsilon$

Removing S from RHS in production,

$S \rightarrow (L) \mid a$
$L \rightarrow (L)L' \mid aL'$
$L' \rightarrow ,SL' \mid \varepsilon$

Next Page →

3) $S \rightarrow A$
$A \rightarrow Ad \mid Ae \mid aB \mid aC$
$B \rightarrow bBc \mid f$

$\Rightarrow S \rightarrow A$
$A \rightarrow aBA' \mid aCA'$
$A' \rightarrow dA' \mid eA' \mid \epsilon$
$B \rightarrow bBc \mid f$

4) $X \rightarrow XSb \mid Sa \mid b$
$S \rightarrow Sb \mid Xa \mid a$

$\Rightarrow X \rightarrow SaX' \mid bX'$
$X' \rightarrow SbX' \mid \epsilon$
$S \rightarrow XaS' \mid aS'$
$S' \rightarrow bS' \mid \epsilon$

5) $S \rightarrow aB \mid aC \mid Sd \mid Se$
$B \rightarrow bBc \mid f$
$C \rightarrow g$

$S \rightarrow aBS' \mid aCS'$
$S' \rightarrow dS' \mid eS' \mid \varepsilon$
$B \rightarrow bBc \mid f$
$C \rightarrow g$

Left Recursion :

A grammer is Left recursive, if it has non-terminal A such that there is ~~de~~ a derivation $A \rightarrow A\alpha$

→ Top down parser cant handle them as the left most symbol is never a non-terminal, it will be stuck in an infinite lorp.

→ A simple rule for direct left recursion elemination

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

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$
$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \quad \beta_n A'$
$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$

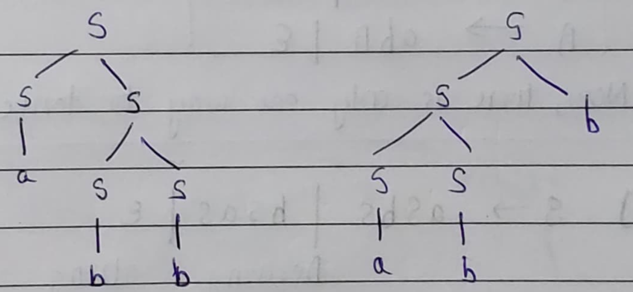
2.> Define ambiguity. Is the following grammer ambigous or not? Justify your answer and correct grammer if necessary.

2.>

Ambiguity : We call a grammer ambigous if a string of terminal symbols can be reached by two different derivation sequences i.e. it can have more than one parse tree.

② It makes design of parser difficult because we don't know which parse tree will be discovered.

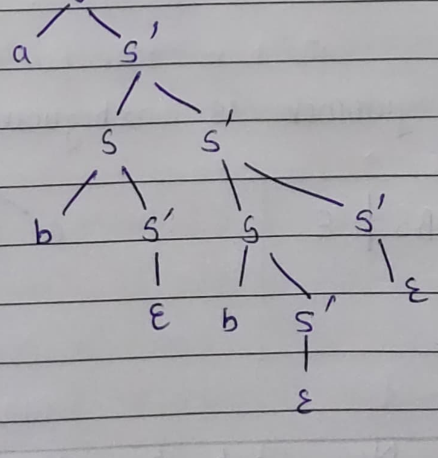
1) $S \rightarrow SS$ Deriving string abb
$S \rightarrow a$
$S \rightarrow b$



∴ The grammer is ambigoous

Remove Left recursion $S \rightarrow aS' \mid bS'$
$S' \rightarrow SS' \mid \varepsilon$
grammer, is unambigous

Deriving abb



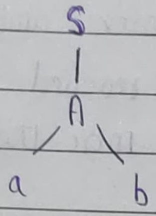
is only parse tree now.

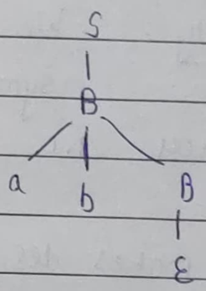
2) $S \rightarrow A \mid B$

$A \rightarrow aAb \mid ab$

$B \rightarrow abB \mid \varepsilon$

String 'ab'



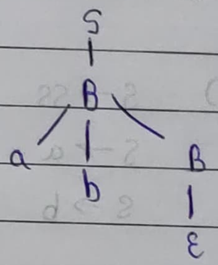


Hence, the grammer is ambigous.
(one place ambiguity removed)

$S \rightarrow aAb \mid B$

$A \rightarrow aAb \mid ab$

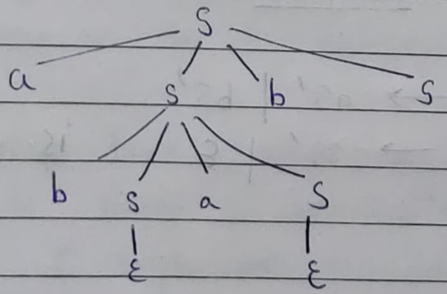
$B \rightarrow abB \mid \varepsilon$

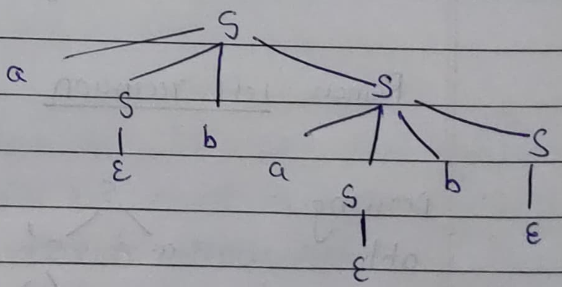


Now, there is only one way to derive ab.

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

Deriving string "abab"



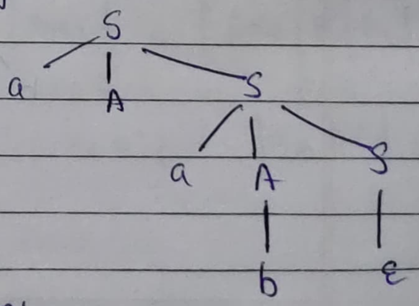


Hence the grammer is ambiguous.

$S \rightarrow aAS \mid bBS \mid \varepsilon$

$A \rightarrow aAb \mid b$

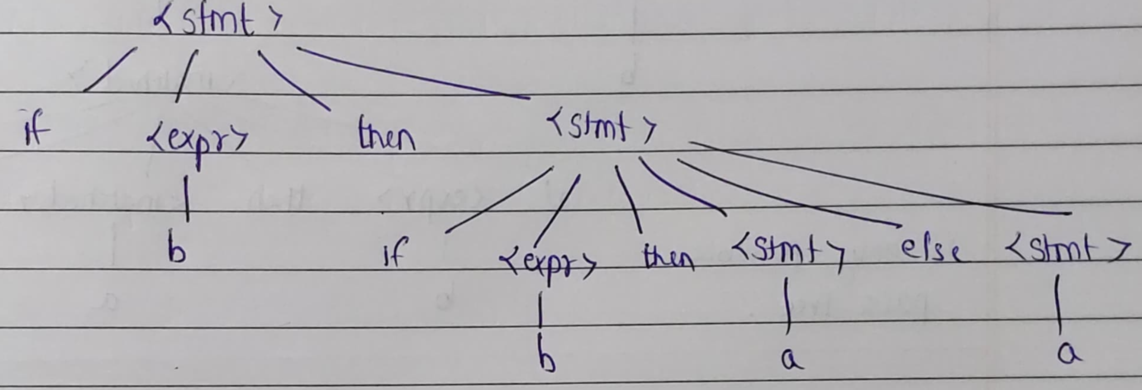
$B \rightarrow bBb \mid a$

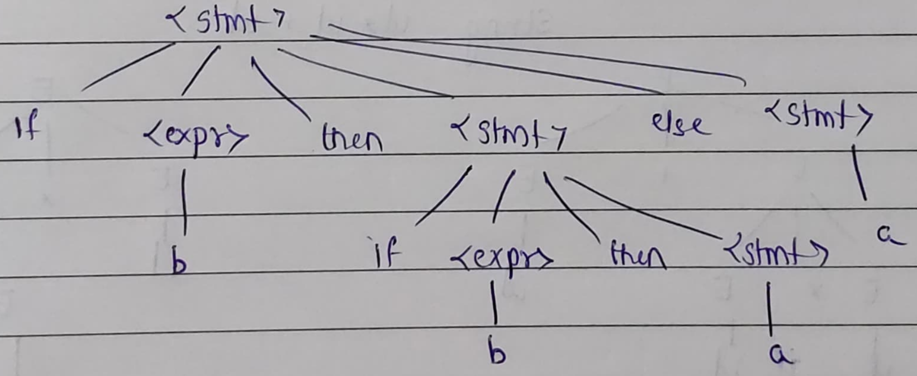


Now abab has only one parse tree ↑

4.) <stmt> → if <expr> then <stmt>
| if <expr> then <stmt> else <stmt>
| a

<expr> → b



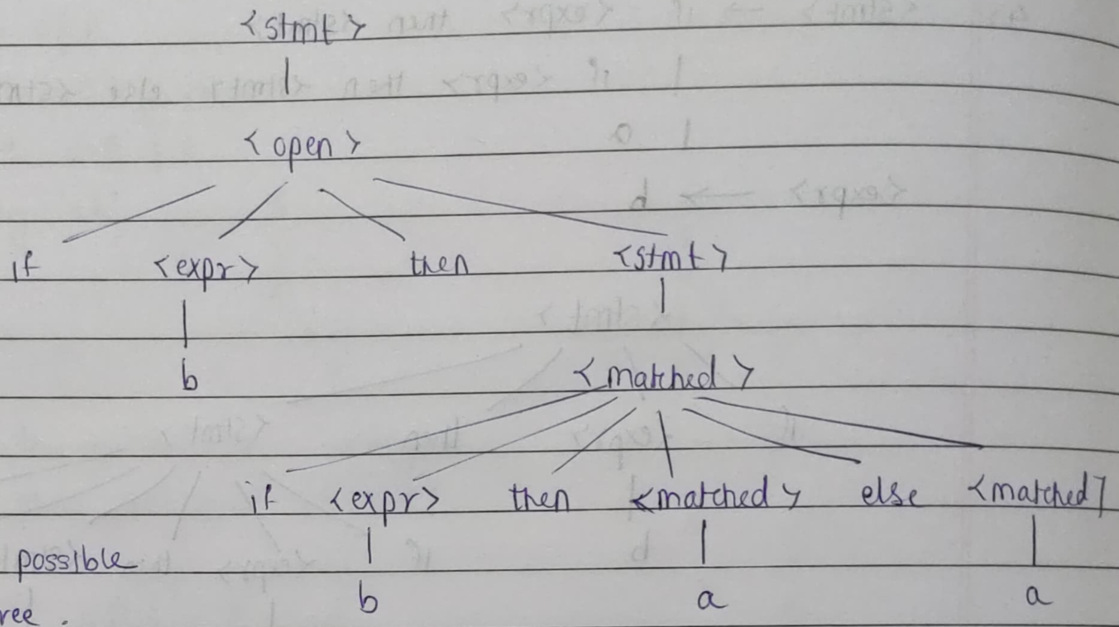


Hence, the grammer is ambigous for the string.

Here, a statement appearing between then and else must be matched (otherwise it is difficult to differentiate which if following else is for)

<stmt> → <matched> | <open>
<matched> → if <expr> then <matched> else <matched> | a
<open> → if <expr> then <stmt>
| if <expr> then <matched> else <open>
<expr> → b

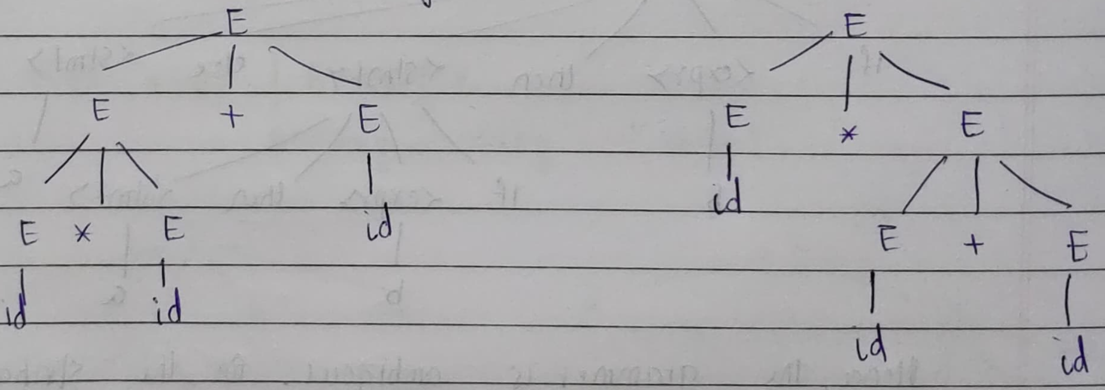
U19CS012



is only possible parse tree.

5> E → E + E | E*E | (E) | id

String id* id + id

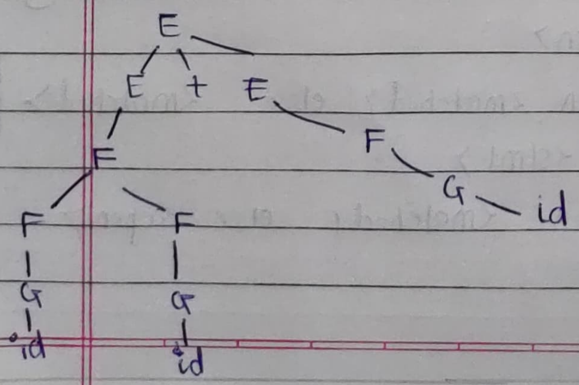


E → E+E | F
F → E*F | G
G → id | (E)

Here, the production which is on left, will have less precedence (+ < *)

→① Start with + which has lowest precedence

→ Use different non-terminal for precedence level.



Submitted By –
U19CS012
BHAGYA RANA