Tutorial-08

U19CS009

Page No.:
Date : / /

Q-1 $L_1 = \{wcw \mid w \text{ is in } (a|b)^*\}$

→ Given language is not CFGs because there is no midpoint ∴ no PDA can be designed to accept it

eg-s $\underbrace{abaab}_{w} c \underbrace{abaab}_{w}$ → no midpaint possible

Q-2 What is left recursive grammar? Remove left recursion from following.

a) $E \to E+T \mid T$
$T \to T*F \mid F$
$F \to id$

⇒

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to id$

b) $S \to (L) \mid a$
$L \to L,S \mid S$

→

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

↳

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

c) $S \to A$
$A \to Ad \mid Ae \mid aB \mid ac$
$B \to bBc \mid f$

→

$S \to A$
$A \to aBA' \mid aCA'$
$A' \to dA' \mid eA' \mid \varepsilon$
$B \to bBc \mid f$

d) $X \to XSb \mid Sa \mid b$
$S \to Sb \mid Xa \mid a$

→

$X \to SaX' \mid bX'$
$X' \to SbX' \mid \varepsilon$
$S \to XaS' \mid aS'$
$S' \to bS' \mid \varepsilon$

e) $S \to aB \mid aC \mid Sd \mid Se$ → $S \to aBS' \mid aCS'$
$B \to bBc \mid f$ $S' \to dS' \mid eS' \mid \varepsilon$
$C \to g$ $B \to bBc \mid f$
$C \to g$

Left recursion → a grammar is left recursion if it has a non-terminal a such that there is a derivation $A \to A\alpha$.

→ Top down parser can't handle them as left most symbol is never a non-terminal, it will be stuck in an infinite loop.

→ A simple rule for direct left recursion elimination

$A \to A\alpha \mid B$
↳ $A \to BA'$
$A' \to \alpha A' \mid \varepsilon$

or $A \to A\alpha_1 \mid A\alpha_2 \mid \dots A\alpha_n \mid \beta_1 \mid \dots \mid \beta_n$
↳ $A \to \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$
$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$

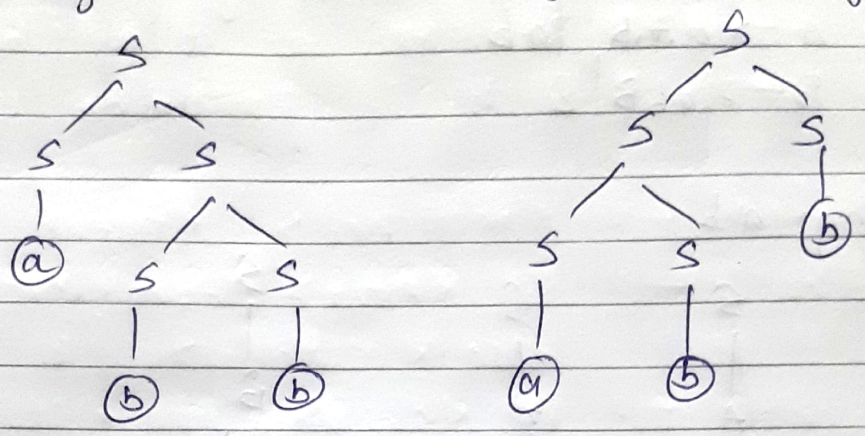
Q-2 Define ambiguity. Is following grammar ambiguous or not? Justify your answer and correct grammar if necessary.

→ Ambiguity → We call a grammar ambiguous if a string of terminal symbols can be reached by two different derivation sequences i.e, it can have more than one parse tree.

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

1) $S \to SS$
$S \to a$
$S \to b$

By deriving abb we can say it is ambiguous



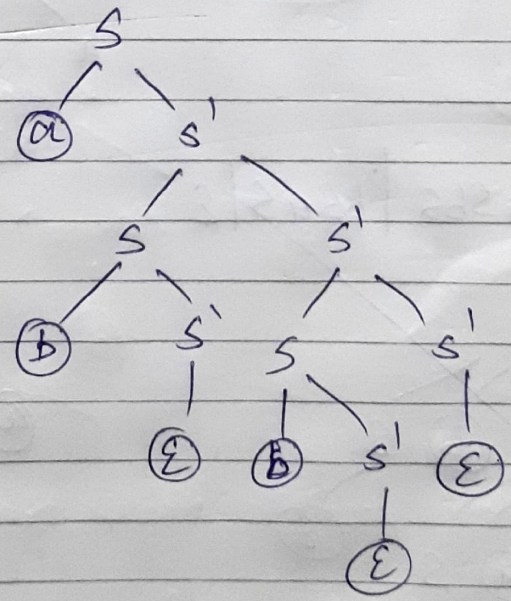
Remove left recursion

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

$S' \rightarrow SS' \mid \varepsilon$

now the above grammar is un-ambiguous

Deriving abb

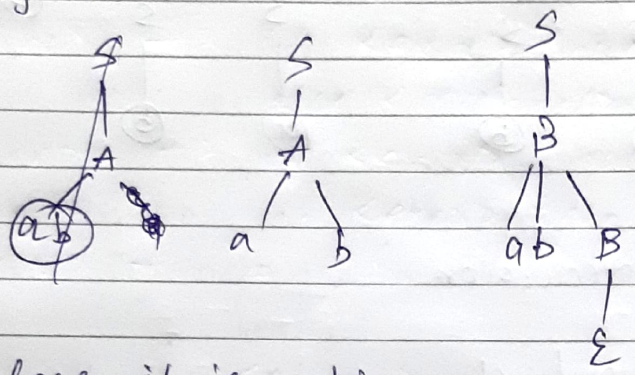


is the only possible parse tree possible

Page No.:
Date : / /

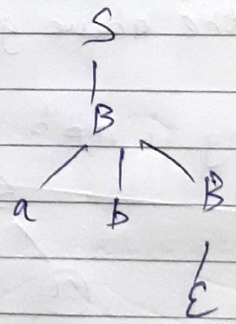
2) $S \rightarrow A \mid B$
$A \rightarrow aAb \mid ab$
$B \rightarrow abB \mid \varepsilon$

String ab

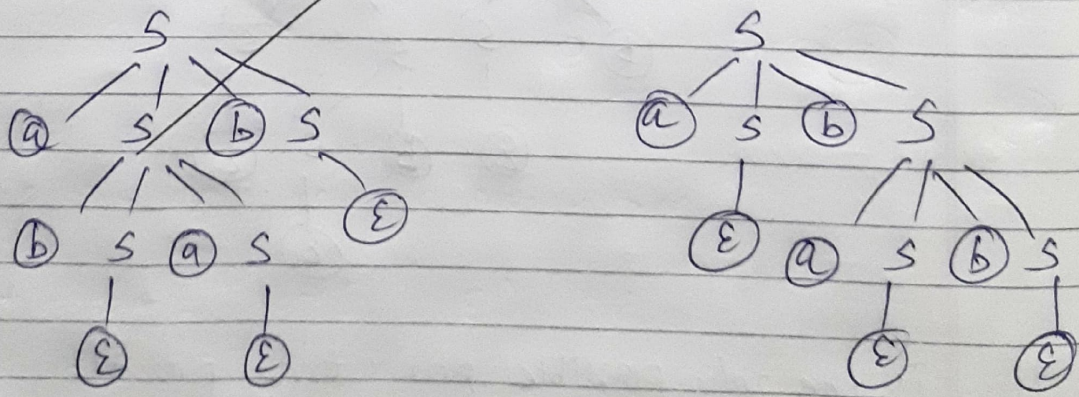


Therefore it is ambiguous.

$S \rightarrow aAb \mid B$
$A \rightarrow aAb \mid ab$
$B \rightarrow abB \mid \varepsilon$



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

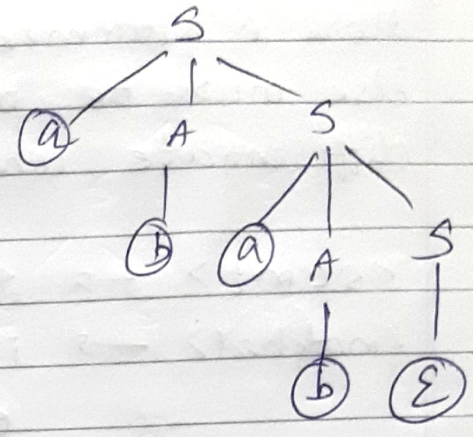


Hence grammar is ambiguous

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

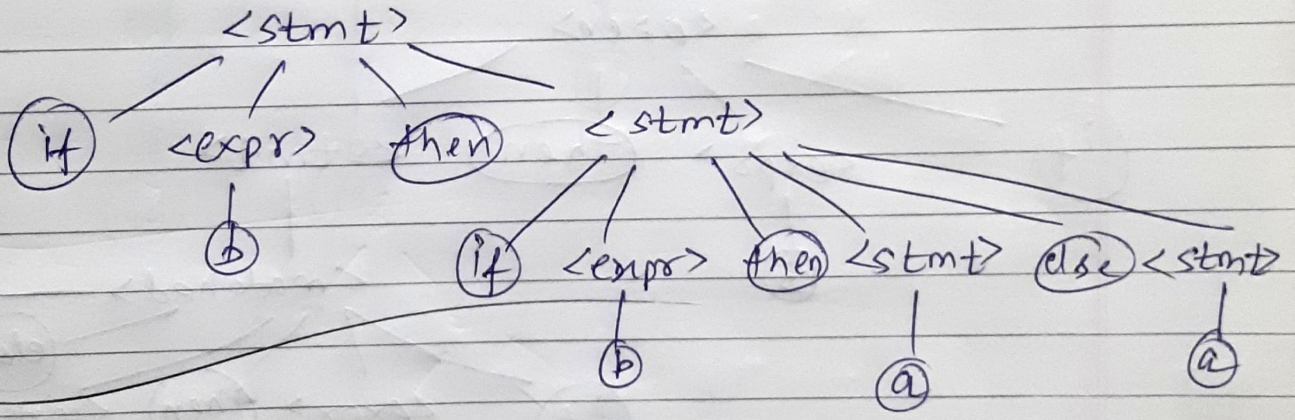
$A \rightarrow aAb \mid b$

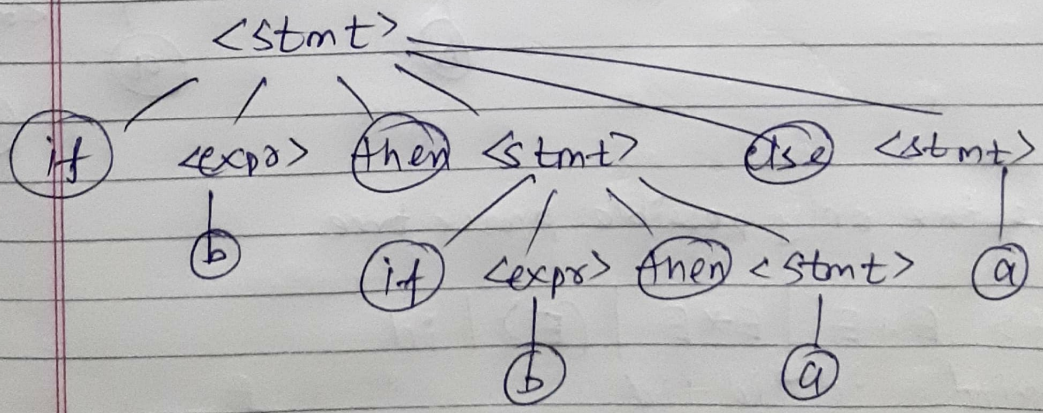
$B \rightarrow bBb \mid a$



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

<expr> → b



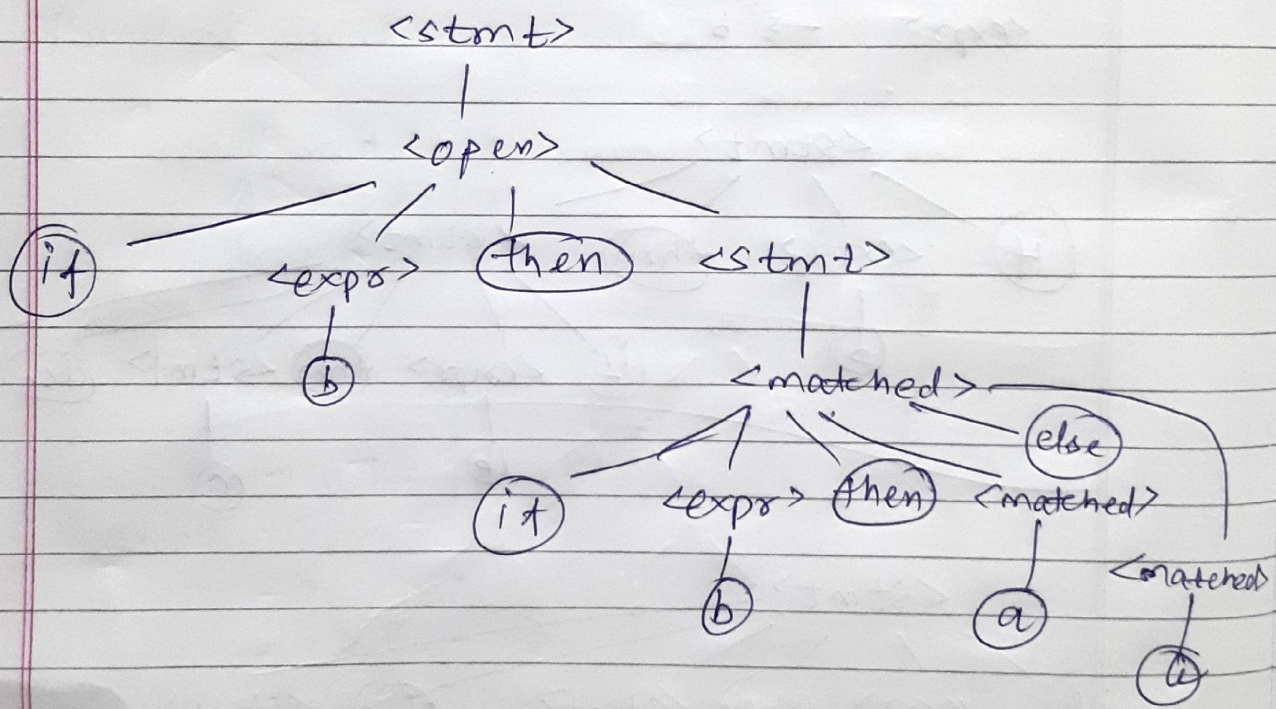


∴ Grammar is ambiguous. for string

if b then if b than a else a

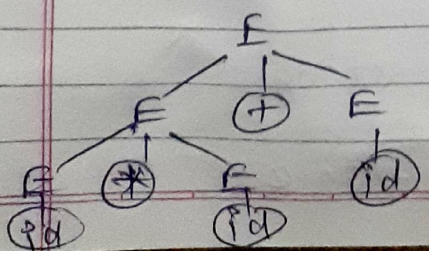
Here a statement appearing between then and else must be matched or it will be difficult to differentiate which 'if' following 'else' is for.

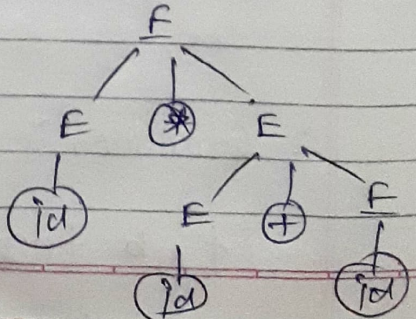
<stmt> → <matched> | <open>
<matched> → if <expr> then <matched>
else <matched> | a
<open> → if <expr> then <stmt>
| if <expr> then <matched> else <open>
<expr> → b



It is the only possible parse tree

(5) E → E+E | E*E | (E) | id
→ string id*id+id





$E \rightarrow E + E \mid F$

$F \rightarrow F * F \mid G$

$G \rightarrow id \mid (E)$

Here (+) will have less precedence than *