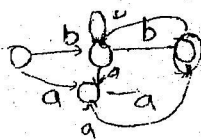


30-Apr



Page No.

Date: / /

∴ We can say that $1, k, k^2, \dots, k^n$ will be regular in base k alphabet set.

Ques - if a is divisible by 6 & a is divisible by 8

$$\therefore a \div \text{LCM}(6, 8)$$

$$a \div 24$$

24 states

$$\cdot a \div 2 \& a \div 8$$

$$\therefore a \div 8$$

$$\cdot a \div 2 \text{ or } a \div 8$$

$$\therefore a \div 2$$

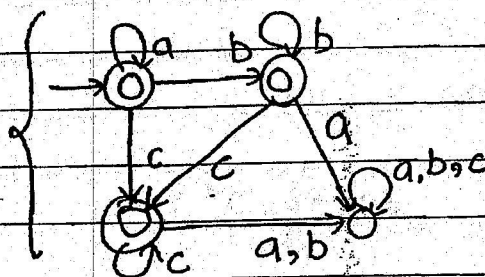
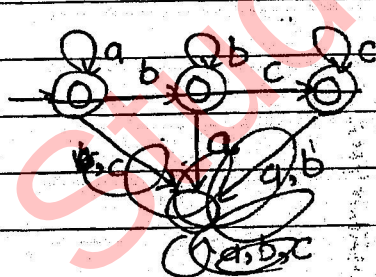
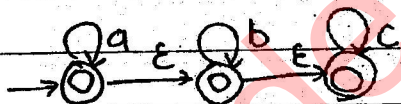
$$\left\{ \begin{array}{l} \text{No of final states} \\ = 8 + 6 - 1 \\ = 13 \end{array} \right\}$$

149 $((0+1)(0+1)(0+1))^* ((b^*ab^*ab^*ab^*)^* + b^*)$ - No of a divisible by 3

150 - both c & d

$$(c) \quad 0^{m+n} + 1^{m+n} \Rightarrow 0^* + 1^*$$

$$a^*b^*c^*$$



$a^*b^*c^*$
No of states = 4

$$\{ww^Rw^R \mid w \in (0,1)^*\} - \text{CSL}$$

$$w_1 \# w_1^R \# w_2 \# w_2^R$$

$$w_1w_1^Rw_2w_2^R - \text{CFL but not DCFL}$$

DCFL

$$ww^R \# w_2w_2^R - \text{DCFL}$$

$$w_1 \# w_2^R \# w_1^R - \text{CSL}$$

$$w_1 \# w_1 - \text{CSL}$$

EXCELLENT

$w_1 \# w_2 \# w_2^R \# w_1^R \rightarrow \text{CFL}$

$w_1 \# w_2 \# w_1^R \# w_2^R \rightarrow \text{CSL}$

$\{w_1 w_1^R w_2 w_2^R \dots\}$ infinite matching - Not a CFL

{NPDA can take finite no of copies ~~but~~ only}

because

$\{a^n b^{2n}\} \cup \{a^n b^{3n}\}$

↓ it requires infinite no of copies be maintained.

$\delta(q_0, a) = \{ (q_0, a) (q_0, aa) (q_0, aaa) \dots \}$

X it is invalid as it must be finite.

$\{a^m b^n \mid m = kn, k \in \mathbb{Z}^+\}$ - it is not a CFL as

$m = n, kn = 2n, \dots$ so on.

$\{a^n b^n\} \cup \{a^n b^{2n}\} \cup \{a^n b^{3n}\} \dots$

FA can do finite storage only.

$\{L \cap I\}$ is always regular
 $\{L \cup I\}$ is always Regular

$$(xyz)^R = z^R y^R x^R$$

164 - (c) as $f(c) = \epsilon$ it can be placed anywhere
 $a^n b^n$ - won't allow c

(d) $a^n c^m b^n$ - it does not allow c to appear before a.

e.g. none of these.

q⁴

Page No.

Date : / /

169 -

$$L = \{a^m\}$$

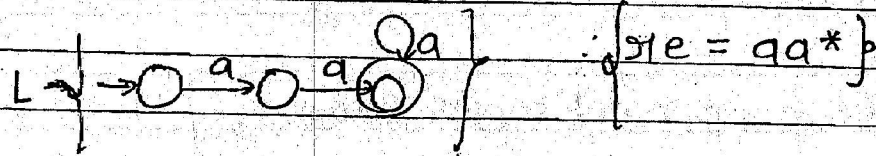
$$\Sigma = \{a\}$$

→ try to check if it covers all the possibility

$$m > n, n \geq 1$$

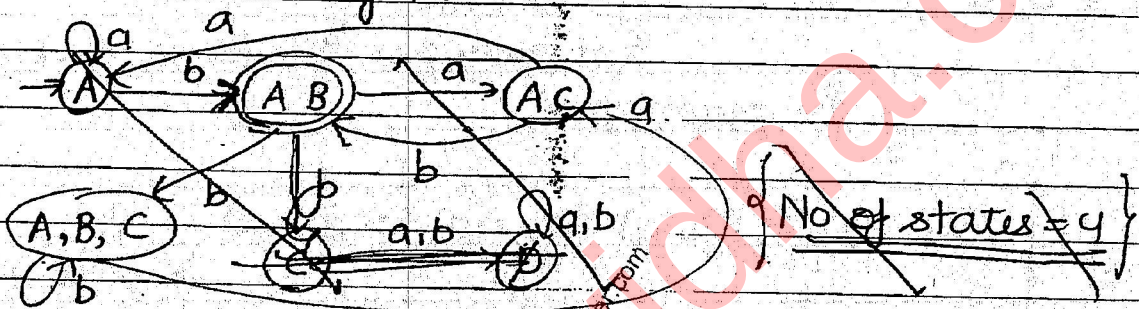
$$L = \{a^2, a^3, a^4, \dots\}$$

$$= \Sigma^* - \{1, a\}$$



∴ No of states = 3

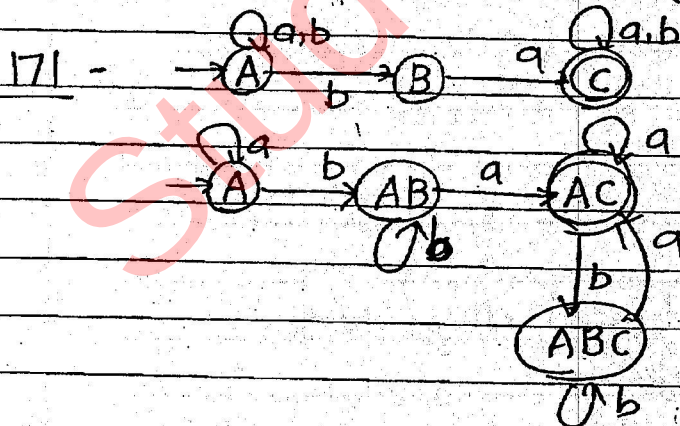
170



$\{a^m\} \mid m < n, n \geq 1$ - Not a regular language as every no can not be written in this form

(170) language ending with b.

∴ No of state = 2



final
No of state = 2

equivalent DFA - by subset construction algorithm

if designed using minimal
 $\left\{ \begin{array}{l} \text{No of state} = 3 \\ \text{No of final state} = 1 \end{array} \right\}$

EXCELLENT

173

$$L = \{ (a^p)^* \mid p \text{ is prime no} \} \quad \Sigma = \{a\}$$

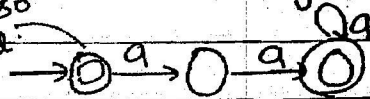
prime no are atom & each no is broken down in term of prime no

$$L = \{a^2, a^3, a^5, \dots\}^*$$

$$\rightarrow (e + a^2 a^3 a^5)^*$$

Minimal string is a^2

as it also accept null



$\therefore$ O/P DFA has No of states = 3

174

$$(a^* + a^* b^*) = a^* (e + b^*) = a^* b^*$$

175 - No of states = 2

as $\{A, C\} \neq \{B, D\}$

176

No of string is ϕ

$$L_1 = 0^* 1^*$$

$$L_2 = 1^* 0^*$$

$$L_3 = (0+1)^*$$

$$L_4 = 0^* 1^* 0^*$$

$$(a^2 + a^3)^*$$

$$= a^2 (e + a)^*$$

$$= (e + a^2 a^3 a^5)^*$$

$$L_1 \cap L_2 = \{1, 0, 1, 00, \dots, 111\} = 0^* + 1^*$$

$$L_3 \cap L_4 = 0^* 1^* 0^*$$

$$L = L_1 \cap L_2 - L_3 \cap L_4$$

$$= A - B$$

$$= A \cap \bar{B}$$

$$= A - (A \cap B)$$

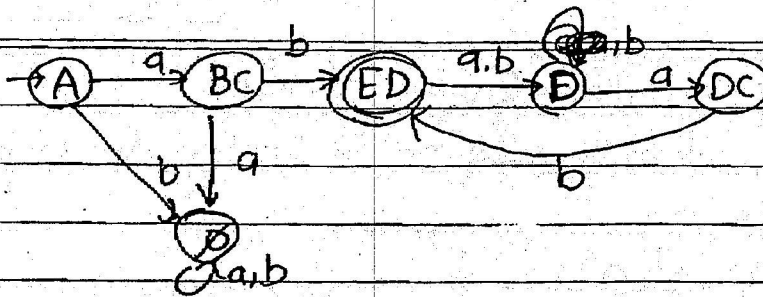
$$A \cap B = 0^* + 1^*$$

$$L = A - A \cap B$$

$$= \phi$$

EXCELLENT

177 -



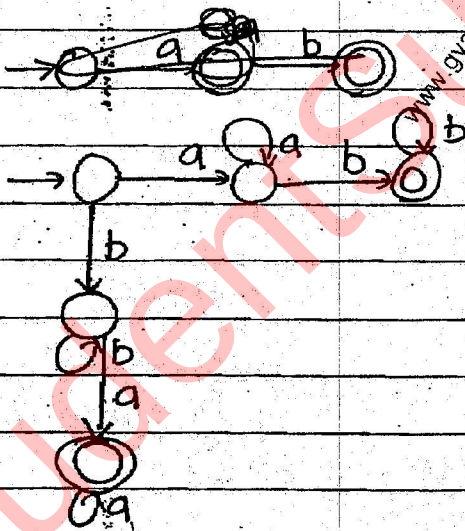
No of states = 6

 $\{A, BC, \phi, D, DC\} \rightarrow ED$

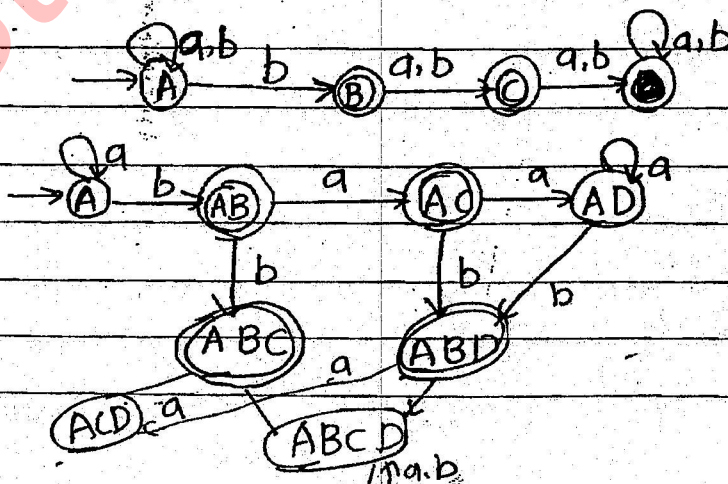
178

 $L = \{abab\}$ $\text{prefix}(L) = \{\epsilon, a, ab, abab\}$ $\text{suffix}(L) = \{\epsilon, a, ba, abab\}$ $X = \{\epsilon, a, abab\} / L$ $X = \{\epsilon\}$

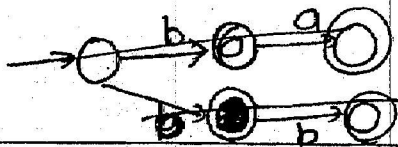
179 -



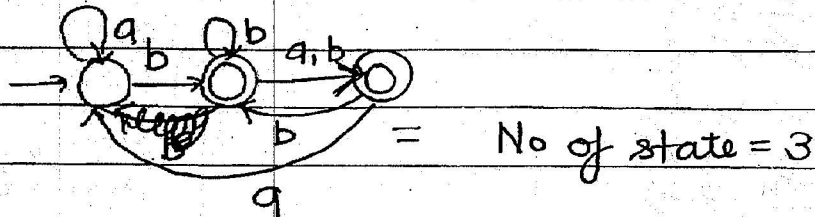
180 -



EXCELLENT



180 -



181 - $L_1 = (0+1)^*$ $L_2 = (01^*0+1^*)$ $L_3 = (01+0+1++\epsilon)$

$$L = L_1 \cap L_2 \cap L_3$$

$$= L_2 \cap L_3$$

$$= 01^*0+1^*$$

Not containing $\epsilon = \{\epsilon, 1, 010\}$

$\rightarrow$ No of string = 3

182 -

No of states = 3

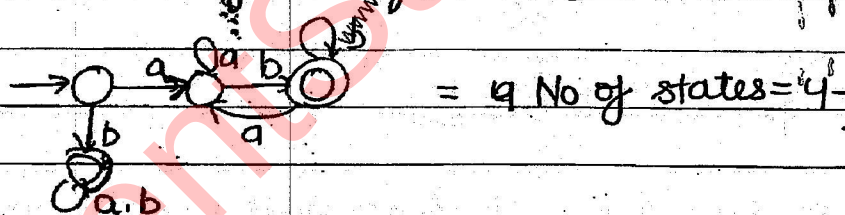
183 -

$L = \text{[made easy]}$

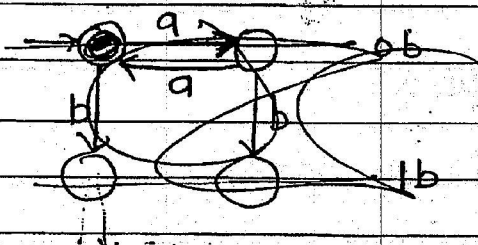
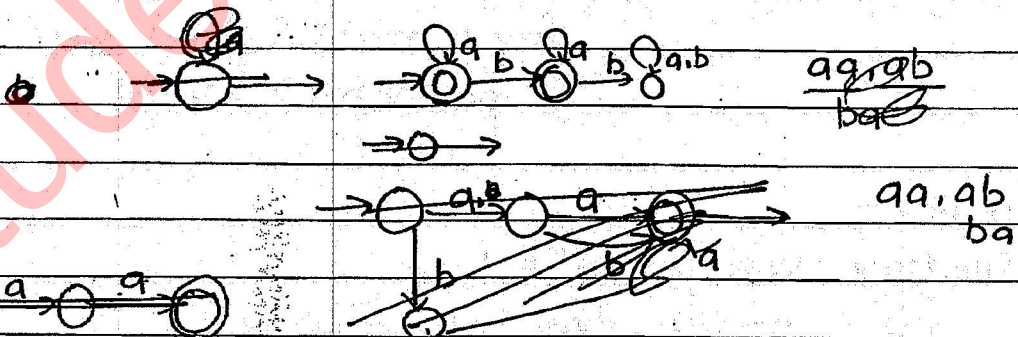
$L_2 =$

No of string in $L_2 = 8$

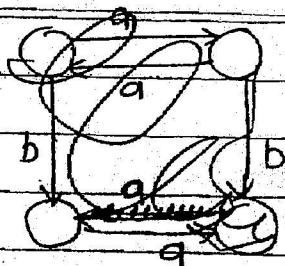
184 -



185 -



EXCELLENT

183

wx

0110

1010

0101

1001

yz

184 - (w,x) can be placed two ways $\rightarrow$ 2 ways (w,x can be same)
 (y,z) can be placed two ways $\rightarrow$ 2 ways (y,z can be same)

 $\Downarrow$ ~~4~~

same)

4 ways

186 - 5 states $\rightarrow$ two for exactly
 two for even

1 for trap

through product automata

[Any state combined with trap is also a trap in product automata.]

187 - $e(s) = \{s, T, A, E\}$ 4

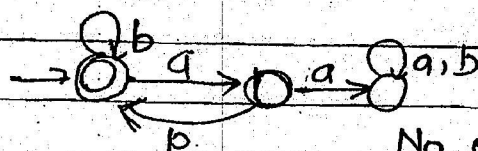
188 - No of states = 1 - If from null move I can reach final state & final state is permanent accept.

189 - $R = (ab+ab)^*$

equivalence class for $\Sigma^* = 1$

$\{b, \epsilon, ab, ab, abab, bab, babab, bb\}$

end with b -

 $\downarrow$ ab+ab

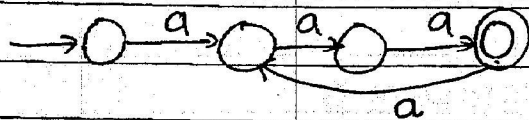
No of states = 3

190 - $L_1 = a^*b^*$

$L_2 = ab$

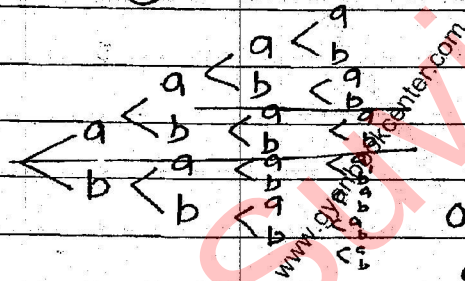
$L_3 = \text{prefix}(L_1 * L_2)$
 $= \{\epsilon, a, ab\} = 3$

191 - $L = \{a^{3(n+1)} \mid n \geq 0\}$



No of state = 4

192 -



out of 16:

aaaa	baaa
aaab	baab
aaba	bbba
aabb	baba
abaa	
abab	
abba	

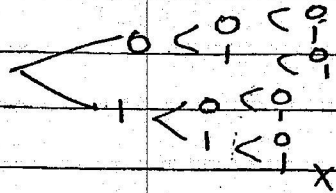
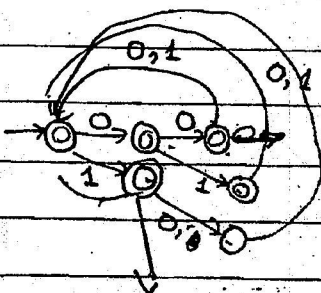
16 - 11 accepted

5 rejected

Draw for reject-

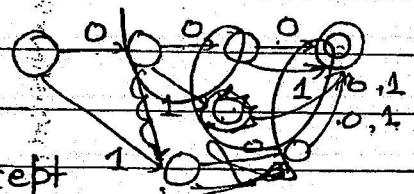
We will need 11 state for 11 accepted but 1 for showing reject.

Ques - 3 block - atleast 1 0



7 accept
 + 1 reject

∴ state = 8



EXCELLENT

192. will go on with every block of 4 string ababab

Page No.

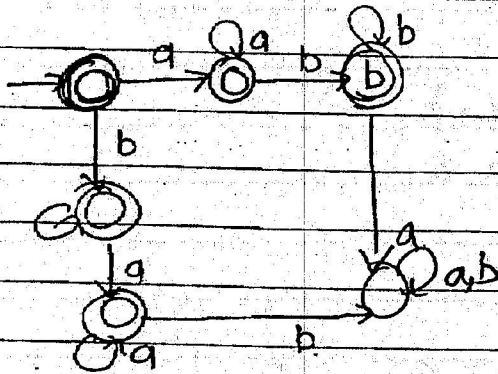
Date: / /

product automata is used when separate cond

is given

193.

$$R = a^*b^* + b^*a^*$$



No of final state = 5

194 -

No of equivalence classes = 6

Top Down - LMD
Bottom Up - RMD

Page No.

Date: / /

CFG

1. $S \rightarrow ISI | OSO \rightarrow$ palindrome

2. $S \rightarrow OSI | ISO \rightarrow$ counting comparison.

Derivations of Tree

Theorem - $w \in L(G)$ if and only if $\exists$ a derivation of w using the productions of G .

An ambiguous string is also a member of a language.

Each Even an unambiguous grammar have more than one derivation for a string. i.e LMD or RMD. so from here, concept of LMD & RMD arises. you have to specify LMD or RMD so the compiler don't get confused.

for each grammar, LMD & RMD are unique, then grammar is unambiguous.

Ex - Let $S \rightarrow AB$

$A \rightarrow aA | \epsilon$

$L(G) = a^*(bb)^*$

$B \rightarrow bB | \epsilon$

if $w = abb$ to check whether it belongs to $L(G)$, check for derivation?

$S \rightarrow AB \rightarrow aAB \rightarrow aAbbb \rightarrow abbB | abb$

if there is a choice for substitution of variable, if left most variable is always substituted is LMD otherwise RMD (if Right variable is substituted)

if a string has derivation, it has both RMD as well as LMD

{A string has exactly 1D $\rightarrow$ false as it has two^P LMD & RMD

EXCELLENT

Ques - $S \rightarrow S+S \mid S*S \mid a \mid b \mid c$

$$w = a + b * c$$

I $S \rightarrow S+S \rightarrow a+S \rightarrow a+S*S \rightarrow a+b*S \rightarrow a+b*c$
 II $S \rightarrow S*S \rightarrow S+S*S \rightarrow a+S*S \rightarrow a+b*S \rightarrow a+b*c$

for this grammar two LMD possible then it is ambiguous.

RMD-

for a Grammar, No of LMD = No of RMD for a given string w because each LMD corresponds to one Derivation tree & each derivation tree has one corresponding RMD.

RMD- $S \rightarrow S+S \rightarrow S+c \rightarrow S*S+$

III $S \rightarrow S+S \rightarrow S+S*S \rightarrow S+S*c \rightarrow S+b*c \rightarrow a+b*c$

IV $S \rightarrow S*S \rightarrow S*c \rightarrow S+S*c \rightarrow S+b*c \rightarrow a+b*c$

Neither LMD nor RMD - $S \rightarrow S*S \rightarrow S*c \rightarrow S+S*c \rightarrow a+S*c$
 $\rightarrow a+b*c$

mixture of LMD & RMD

LMD is also can be same as RMD if

$S \rightarrow a \mid b$

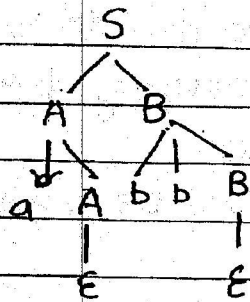
$S \rightarrow aas \rightarrow aab$

$S \rightarrow aab$

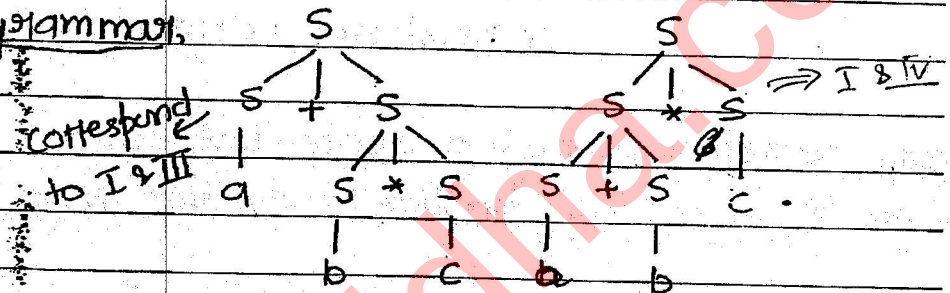
LMD is always similar to RMD for regular Grammars.

for every linear grammar, LMD is similar to RMD i.e. it has only one derivation. (beoz it has single variable to be replaced)

for an unambiguous grammar, only one DT is possible for a particular string.



for ambiguous grammar,



- No of LMD = No of RMD = No of derivation tree.
- A derivation tree corresponds to one LMD & one RMD.

Problem caused by Multiple DT { Syntax - Derivation
Semantics - Derivation tree }

- Multiple Meaning of a single stmt & meaning of ~~each~~ statement is confused & then compiler got confused, then it will create run time error.

Yield of tree - the leaf nodes from left to right
abb in above.

if a string have two DT's then the yield of both tree must be same.

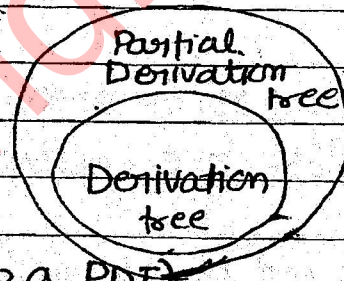
if a string has two or more DT's with same yield, then the grammar is ambiguous.

Derivation Tree - A tree is said to be DT if

- I) it must start with Start symbol i.e. root
- (II) Every expansion of tree must corresponds to a production in Grammar.
- III the yield must be a sentential form belonging to $L(G)$.
- IV Yield must not be a sentential form i.e. it must not variable.

- Every sentence is also a sentential form.
- Every DT is also a partial derivation tree.

Partial Derivation tree -



1. Yield may be a sentential form. (it may or may not be sentential form as each DT is also a PDT)
2. Every expansion must correspond to grammar in G .
3. the root of PDT need not be a start symbol.

• Derivation Tree is also known as Parse Tree.

For ambiguous Grammar - if atleast one string in G which have atleast 2 Derivation trees

In Ambiguous - $\forall w \in L(G)$ has exactly 1 derivation tree.

What DT - Not a Member

→ 1 DT - UAG & a member

2 DT - Member with ambiguity

} for $\forall w \in L(G)$

A Grammar is ambiguous if it has atleast 2 LMD or atleast 2 RMD's $\exists w \in L(G)$

A Grammar is unambiguous if it has exactly 1 LMD for $\forall w \in L(G)$

- CFG ~~un~~ ambiguity problem is ~~decidable~~ not decidable.

Reasons for Ambiguity-

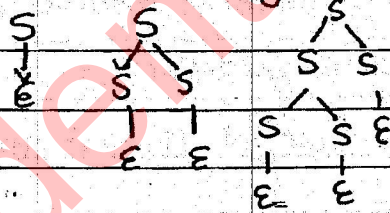
(1) Left Recursion: $S \rightarrow S\alpha$ $A \rightarrow A\alpha \mid \beta$

(2) Left factoring: $A \rightarrow \alpha\beta \mid \alpha\gamma$

not always cause ambiguity but sometime it can cause ambiguity.

If a Grammar has left factoring & left recursion then Grammar may or may not be ambiguous.

(3) $S \rightarrow SS \mid \epsilon$ - it always causes ambiguity



(4) $S \rightarrow aA \mid bA \mid ab \mid \epsilon$ $\rightarrow$ Never causes ambiguity

(5) $S \rightarrow S_1 \mid S_2$ because it is possible if $L(S_1) \cap L(S_2) = \phi$

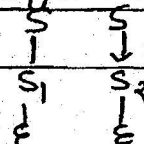
then S_1 & S_2 has common string then it causes ambiguity

Ex- $S \rightarrow S_1 \mid S_2$

$AS_1 \rightarrow aA \mid \epsilon$ $\rightarrow$ have ϵ as common

$S_2 \rightarrow bB \mid \epsilon$

$\therefore$ ambiguous



EXCELLENT

Ex - $S \rightarrow S_1 | S_2$

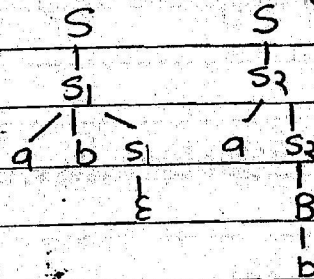
$S_1 \rightarrow abS_1 | \epsilon$

$S_2 \rightarrow aS_2 | B$

$B \rightarrow bB | b$

$$L(S_1) \cap L(S_2) = ab \cdot ab$$

$\therefore$ it is ambiguous



Ex - $S \rightarrow AB | BA$

$A \rightarrow aA | \epsilon \Rightarrow$ ambiguous

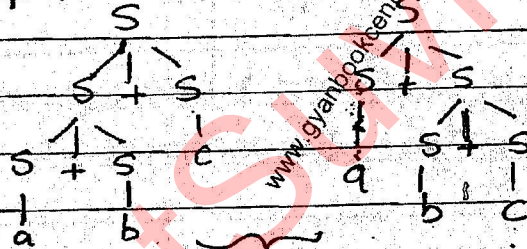
$B \rightarrow bB | \epsilon$

$S \rightarrow AB | BA$

$A \rightarrow aA | a \Rightarrow$ Not ambiguous

$B \rightarrow bB | b$

Ex - $S \rightarrow S+S | a | b | c$



$\hookrightarrow$ two different derivation tree

Hence it is ambiguous

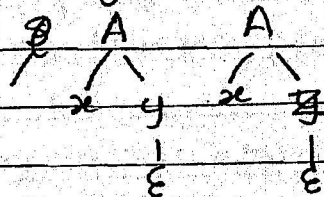
Ex - $S \rightarrow Sa | \epsilon \Rightarrow$ Left recursion but it is not ambiguous

Left factoring - $A \rightarrow xy | xz$

it may or may not cause ambiguity

Left factoring is ambiguous if

$$L(y) \cap L(z) \neq \emptyset$$

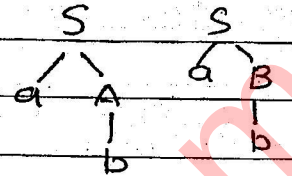


Ex - $S \rightarrow aA|aB$

$A \rightarrow a$ $\rightarrow$ Not ambiguous
 $B \rightarrow b$

$S \rightarrow aA|aB$

$A \rightarrow b$ ambiguous
 $B \rightarrow b$



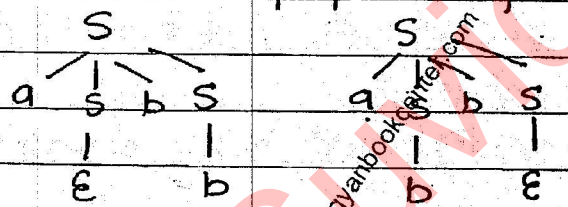
for $S \rightarrow aA|aB$

$A \rightarrow aA|\epsilon$ — Unambiguous as $L(A) \cap L(B) = \epsilon$
 $B \rightarrow bB|\epsilon$

(6) - $S \rightarrow S a S$ it can cause ambiguity

Ex -

$S \rightarrow a S b S | b | \epsilon$ for abb



$\Rightarrow$ it is ambiguous.
because of presence
of null.

Ambiguity of language:

A language is ambiguous if every Grammar to generate L is ambiguous

Such a language is also known as inherently

ambiguous language. this problem is known as Non-removable
ambiguity.

A language is unambiguous if $\exists$ atleast a grammar which is unambiguous.

Ex - $S \rightarrow S + S | S * S | a | b | c$

(a) G is ambiguous

(b) $L(G)$ is ambiguous

- Every ambiguous grammar can't be converted to unambiguous grammar.

Example of Ambiguous language-

Language which are ambiguous contain 3 features -

Ex - $\{a^m b^m c^n\} \cup \{a^m b^n c^n\}$ - Ambiguous language
- CFL but Not DCFL

• if has no unambiguous grammar.

Condition for a language to be ambiguous.

$S \rightarrow aAB|cD$

$S \rightarrow S_1 | S_2$

$S_1 \rightarrow AB$

$A \rightarrow aAB|e$

$B \rightarrow cB|e$

$S_2 \rightarrow cD$

$C \rightarrow ac|e \quad D \rightarrow bDc|e$

1. two condition are in Union
 $A \cup B$

2. two condⁿ must not overlap
i.e. union is not Removable
 $A \not\subseteq B$ or $B \not\subseteq A$

3. $A \cap B \neq \phi$, the both condⁿ must have something common to them.

Ex - $\{a^n b^n c^m d^m\} \cup \{a^n b^m c^m d^n\}$ - ambiguous

Ex - $S \rightarrow asb|bsa|ss|e \rightarrow$ Not a ambiguous language as its ambiguity can be removed.

Membership Algorithm-

A procedure P is Membership algorithm for a language, if it can find $w \in L$ in finite amount of time.

$\forall w \in \Sigma^*$, if $w \in L \rightarrow$ say yes & Halt
 if $w \notin L \rightarrow$ No & Halt.
 (it must not go in hang)

Membership is decidable for each language except RE.

Algorithm-

1. Brute Force Parsing - time complexity - $O(k^n)$ • NP problem
 (for CFL) where n is size of string

2. CYK - FC - $O(n^3)$ • it is faster
 (for CFL) • P problem
 • optimum time Membership Algorithm

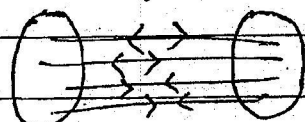
"P & NP is defined for problem but Not for Algorithm."

3. Require - $O(n)$ time for DCFL.

- It is used now a days
- Run of LL(k) or LR(k) compilers.

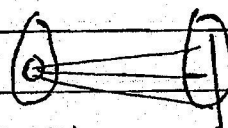
- Grammar corresponding to DCFL - LR(k) or LL(k)
 (most suited or best suited)

for every DCFL, there exist a LR(k) Grammar
 but LL(k) can create DCFL but for each DCFL, it may
 there may or may not exist ~~DCFL~~ LL(k)



DCFL

LR(k)



DCFL

LL(k)

for each DCFL, LL(k) does not
 exist. EXCELLENT

for a DCFL, LL(k) is not guaranteed.

Page No.

Date : / /

LL(k) uses ~~for~~ only one step of d

LR(k) uses 1 production (only) for one step of derivation.
n size string $\rightarrow$ n step $\rightarrow$ n production - $O(n)$

LR(k) grammars are always unambiguous.

bcz for each step only one derivation is possible so a single DT is possible.

Programming language compiler
require G to be

Existence

fast

Unambiguous

[LR(k) satisfy all 3 condition]

LL(k) satisfy

fast

Unambiguous

Only one production is processed at a single step but LL(k) do not guarantee the existence of grammar.

• both LL(k) & LR(k) can only create DCFL

• LL(k) is based on LMD but LR(k) grammar are based on RMD.

• LL(k) create top-down compilers & LR(k) create bottom-up compilers.

EXCELLENT

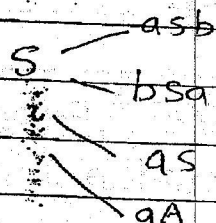
Properties of LL(k) & LR(k)

(Decidable)

Brute Force Parsing

In this Method, all the possibilities are covered from start symbol

each. No intelligence.



If no of production P

$$\text{Time} = |P| + |P| \cdot |P| + |P| \cdot |P| \cdot |P| + \dots + |P|^{2n}$$

$$= |P| (|P|^{2n} - 1)$$

$$(|P| - 1)$$

$$= O(k^n)$$

Input-It will not CFG with ϵ production & unit production.Stopping condⁿ - length of stringWe remove ϵ production & unit production as we have to satisfy non contracting condition.2n - n time to replace by production like

$$S \rightarrow ABCDE$$

n time to replace

$$S \rightarrow abcde$$

$$n + n = 2n$$

once n size is obtained it can't be reduced as ϵ & unit production are removed.

$$\therefore O(P^{2n})$$

$$= O(k^n) \quad P^2 = k$$

CYK Algorithm

Input - Grammar is in CNF (Chomsky Normal form)

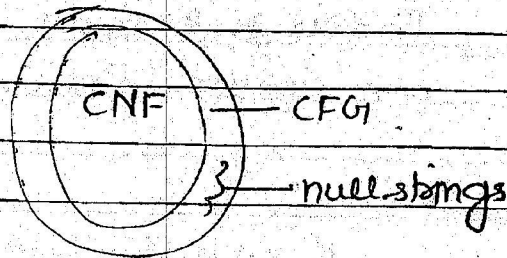
• Each CFG can be converted to CFG CNF.

• CNF & CFG both have same power.

$$\text{CNF allows - } \begin{cases} V \rightarrow VV \\ V \rightarrow T \end{cases}$$

EXCELLENT
EXCELLENT WITH
PRACTICE

- ① CNF has same power as CFG practically but theoretically it has less power i.e. CNF cannot generate Null string.



- ② for each CFG, is it possible to write CNF? False
- ③ for each CFL (null free CFL) or CFG without null string, is it possible to write CNF? True.
- ④ language generated by CNF is proper subset of language generated by CFG.

$$\text{CNF} = \text{Null free CFL's} = \text{GNF}$$

- Greibach Normal Form - it also has same power as CFG

$$\left\{ \begin{array}{l} V \rightarrow TV^* \\ V \rightarrow T \end{array} \right.$$

CNF & GNF has exactly same power

$\forall \text{ CFG} \rightarrow \exists \text{ CNF or GNF}$ (false as it cannot generate null CFL)

$$\text{for } \left\{ \begin{array}{l} \forall \text{ CNF} \rightarrow \exists \text{ GNF} \\ \forall \text{ GNF} \rightarrow \exists \text{ CNF} \end{array} \right.$$

GNF is used to convert Grammar to machine i.e. PDA.

- To convert CFG to PDA, Input Grammar must be in GNF Form.