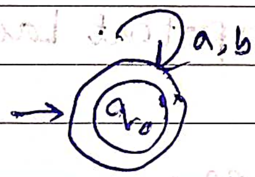
Properties of Regular Expression:-

Algebric Laws for Regular Expr-

(a) Commutative — $xy = yx$

L & M regular expression



$L = a$
$M = b$

$L + M = M + L$

$a + b = b + a$

but $ab \neq ba$

(b) Associativity — a) $(L+M)+N = L+(M+N)$

b) $(LM)N = L(MN)$

(c) Distributive — $L(M+N) = LM+LN$ (Left Distributive Law)

$(M+N)L = ML+NL$ (Right Distributive Law)

Theorem (Based on Distributive Law):-

If L, M, N are regular language then $L(M+N) = LM + LN$

$\underbrace{LM}_{xy} + \underbrace{MN}_{xy}$

$w = x \cdot y$

So, $x \in L$ and $y \in (M+N)$

$\Rightarrow y \in M$ or $y \in N$

$w = L \cdot (M+N)$

$w = xy$

$w = LM + LN, w = xy$

x contains L

y contains M or either N

$xy \in LM$ or $xy \in LN$

$\Rightarrow x \in L$ and $y \in M$ or $x \in L$ and $y \in N$

(d) Identity:-

$0 + 5 = 5$ $\quad \phi + L = L + \phi = L$

$\epsilon \cdot L = L \cdot \epsilon = L$ $\quad \phi$ is identity for union

(e) Annihilators:- $\cancel{L\phi = \phi L = \phi}$ $\quad L\phi = \phi L = \phi$

(f) Idempotent Law:- $L + L = L$

(g) Closure:- $(L^*)^* = L^*$ $\quad \phi^* = \epsilon$ $\quad \epsilon^* = \epsilon$

$L^+ = LL^*$ $\quad L? = \epsilon + L$

Closure Properties of Regular ~~Expression~~ Language:-

a) union - The union of 2 Regular Language is regular.

b) Intersection
c) Complement
d) difference
e) Reversal
f) Closure
g) Concatenation
h) Homomorphism – (substitution of strings for a symbol of a regular Language)

⊛ Infinite Union is not a closure property of R.L

0101 → aabaab

$h(0) = aa$
$h(1) = b$

i) Inverse Homomorphism → $h^{-1}(aa) = 0$

Th. If L & M are regular Languages then so is LUM.

LUM

$L = L(R)$
$M = M(S)$

$LUM = L(R) \cup L(S)$
$= L(R \cup S)$

Th - If L is ~~Reg~~ Regular Language then the compliment $\bar{L}$ is also a Regular Language.

$\bar{L} = \Sigma^* - L$

$A' = (Q, \Sigma, q_0, \delta, Q-F)$

Th L & M is Regular Language then $L \cap M$ is R.L.

Proof :- $L \cap M = \overline{\bar{L} \cup \bar{M}}$

$A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$

$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$

The finite automata for $L \cap M$ is A

p, q are 2 states where A gives $p \in A_M$ and $q \in A_L$

$\delta_L(p, a) = s$

$\delta_M(q, a) = t$ $s \in A_L$ and $t \in A_M$

$\delta((p,q), a) = (s,t) = (\delta_L(p,a), \delta_M(q,a))$.

$A = (Q_L \times Q_M, \Sigma, (q_L, q_M), \delta, F_L \times F_M)$

Th $L - M = L \cap \bar{M}$ [Closure Under Difference]

Th If L is R.L then the reversal of L (L^R) is also R.L.

Th If L is R.L then Homomorphism of L is also R.L.

Th If L is R.L then inverse-Homomorphism of L is also R.L.

How to proove a Languange is not Regular?

Pumping Lemma for Regular Language :-

$L = \{0^n 1^n ; n \geq 0\}$ not a regular Language

Lemma →

Let L here R.L then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into 3 strings $w = xyz$ such that – (i) $y \neq \epsilon$ / $|y| > 0$ (ii) $|xy| \leq n$ (iii) for all $k \geq 0$, the string xy^kz is also in L

$L = L(A)$

L is a regular Language for which there is a finite automata A

$w = a_1 a_2 \cdots a_m$ (where $m \geq n$) and each a_i is the input symbol

For $i = (0, 1, \cdots, n)$ we can define the state P_i as

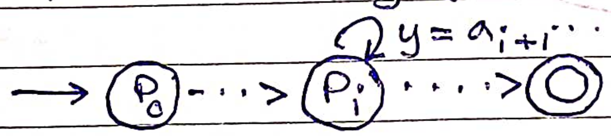
$$\hat{\delta}(q_0, (a_0 a_1 a_2 \cdots a_i)) = P_i$$

and q_0 is the start state of A

From i, j, we have $\hat{\delta}(P_i, (a_i \cdots a_j)) = P_i$

such that $0 \leq i \leq j \leq n$

$y = a_{i+1} \cdots a_j$



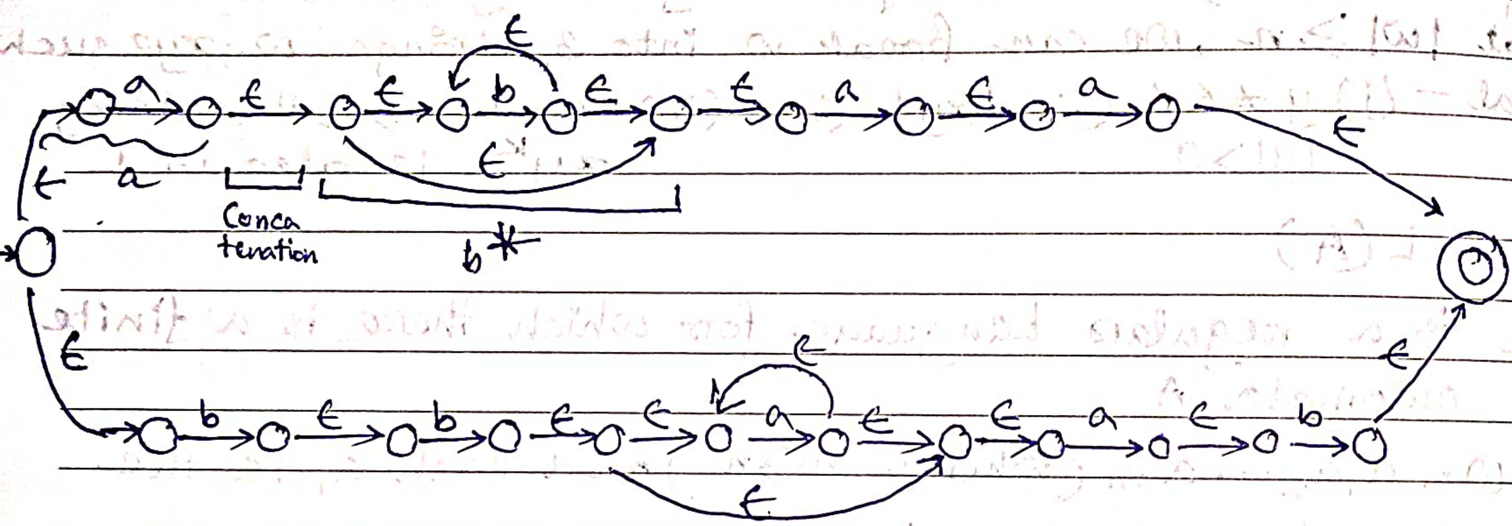
$x = a_1 \cdots a_i$ $z = a_{j+1} \cdots a_m$

(*) Automata Exam Solution

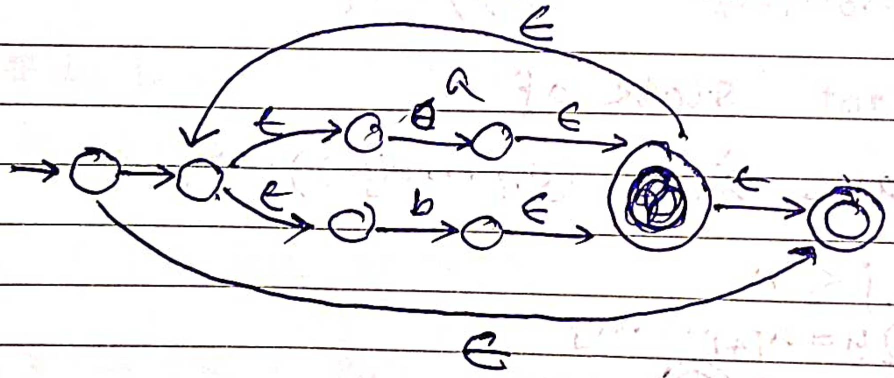
~~5.b) L=(a+b)*b(a+bb)*~~

Thomson Construction :-

$L = (ab^*aa) + (bba^*ab)$



$\boxed{(a+b)^*}$



Q/ Prove $0^n 1^n$ regular Language or not. $[a^n b^n]$

Ans:- not regular

$w = xy z$ $y \neq \epsilon$: $A = \{a^n b^n \mid n \geq 0\}$

Assume A is regular

Pumping Length = p

$|xy| \leq n$

$S = a^p b^p$

$xy^k z$ where $k \geq 0$

$x = a^{p-1}$ ~~$y = b^p z$~~ .

$y = a$ $z = b^p$

$xy^2 z = a^{p-1} a^2 b^p$

$= a^{p+1} b^p \notin A$

∴ not regular

Q/ $L = 0^{2n}, n \geq 0$

$w = xyz$ $y \neq \epsilon$

$x = \epsilon$ $y = 00$

$z = \epsilon$ $xy^k z$

$k = 0$ ϵ

$k = 1$ 00

$k = 2$ 0000

Regular Grammar: – (Type-3)

$G = (V, T, S, P)$

$V = \{A, B\}$ $T = \{a\}$ $S = \{A\}$

$P \Rightarrow$
$A \rightarrow aB$
$B \rightarrow a$
$A \rightarrow a$
$A \rightarrow \epsilon$

$A \rightarrow AB \mid B$
Left Linear Grammar

$A \rightarrow BA \mid B$
Right Linear Grammar

Context Free Grammar (Type-2): –

→ It's have some natural recursive nature

Parsing
DTD → LATEX
Document type Definition

1) ~~There is a finite set of symbols that form the strings of the language being~~ defined, We ~~can~~ call this Alphabet the Terminals / Terminal Symbols.

2) ~~There is a finite set of variable also called sometimes not-terminals or syntactic categories.~~ ① Each variable represent a Language ~~symbol~~ that is a set of strings.

3) One of the variables represents the Language being defined. It is called the start symbols. Other variable represents auxilary classes of strings that are used to help define the Language of ~~sta~~ start symbol.

(4) There is finite set of productions or production rules that represent the recursive def^n of a Language. Each production consists of –

(a) A variable that is being (partially) defined by the production is called the head of the production.

(b) The production symbol arrow (→)

(c) A string of 0 ~~and~~ or more terminals and variables. This string is called the body of the production.

From The body of the production there is one way to form strings in the Language of the variable head. In doing so we leave the terminals ~~we~~ unchanged and substitute for each variable of the body of any string that is known to be in the language of that variable

Recursive Inference

Leftmost derivation →

$$S \overset{lm}{\Rightarrow} AB \overset{lm}{\Rightarrow} a\underline{A}B \overset{lm}{\Rightarrow} aa\underline{B} \overset{lm}{\Rightarrow} aab$$

$S \rightarrow AB$
$A \rightarrow aA|a$
$B \rightarrow aB|b$

$S \overset{*}{\Rightarrow} aab$
aab can be derived from S in 0 or more steps.

Rightmost derivation →

$$S \Rightarrow A\underline{B} \Rightarrow \underline{A}b \Rightarrow aAb \Rightarrow aab$$

~~If~~ Context-Free Language (CFL):–

If $G(V,T,S,P)$ is a CFG the Language of G, $L(G)$

is the set of terminal strings that have derivation from the start symbols.

$$L(G) = \{ w \text{ in } T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w \}$$

If a Language L of some context free grammar then L is called CFL.

~~Parse Tree~~:-

$E \rightarrow I$

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow (E)$

$I \rightarrow a$

$I \rightarrow b$

$I \rightarrow Ia$

$I \rightarrow Ib$

$I \rightarrow I0$

$I \rightarrow I1$

$(a101+b1)^*(a1+b)$

Left most Derivation

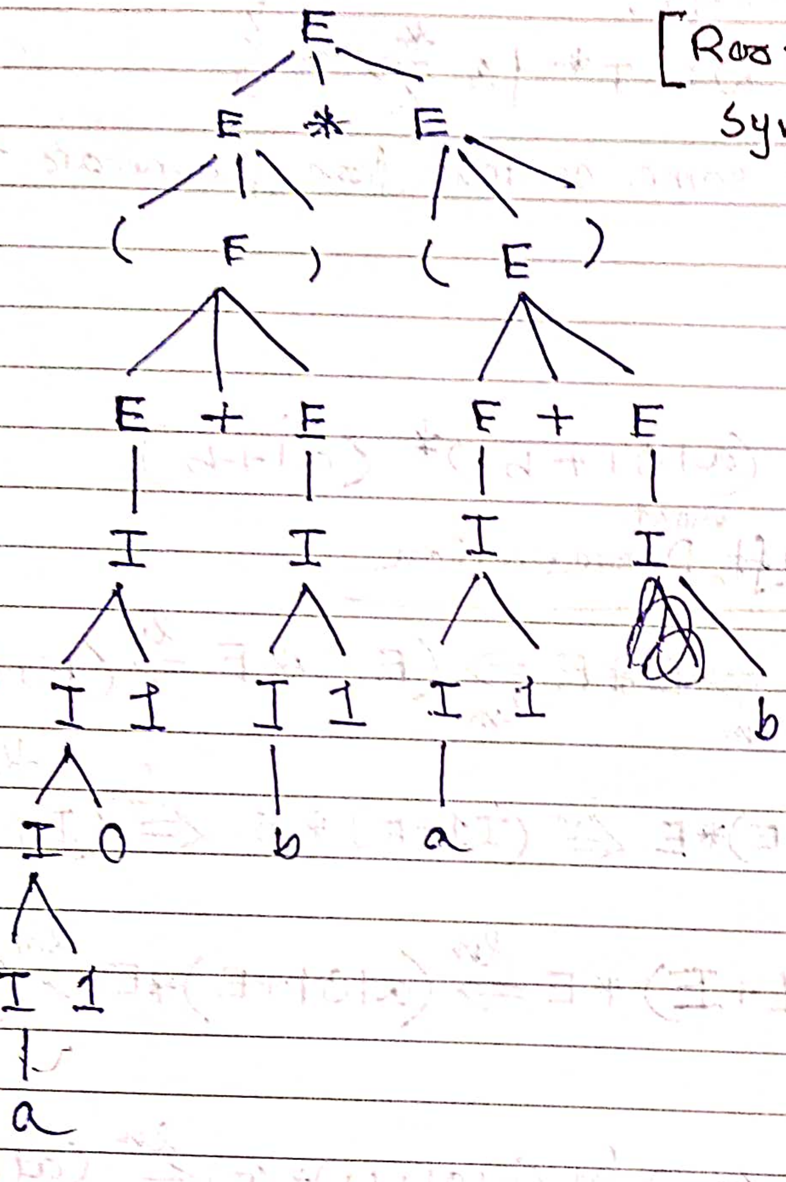
$E \underset{lm}{\Rightarrow} E*E \underset{lm}{\Rightarrow} (E)*E \overset{lm}{\Rightarrow} (E+E)*E \overset{lm}{\Rightarrow} (I+I)*E \overset{lm}{\Rightarrow} (I1+E)*E \overset{lm}{\Rightarrow} (I01+E)*E \overset{lm}{\Rightarrow} (I101+E)*E \overset{lm}{\Rightarrow} (a101+E)*E \overset{lm}{\Rightarrow} (a101+I1)*E \overset{lm}{\Rightarrow} (a101+b1)*E \overset{lm}{\Rightarrow} (a101+b1)*(E) \overset{lm}{\Rightarrow} (a101+b1)*(E+E) \overset{lm}{\Rightarrow} (a101+b1)*(I1+E) \overset{lm}{\Rightarrow} (a101+b1)*(a1+E) \overset{lm}{\Rightarrow} (a101+b1)*(a1+I) \Rightarrow (a101+b1)^*(a1+b)$

H.W → Right most Derivation

Parse Tree

Parse tree for (a101+b1) * (~~ab~~ a1+b)

[Root is the start symbol of the grammar



17 Sunday

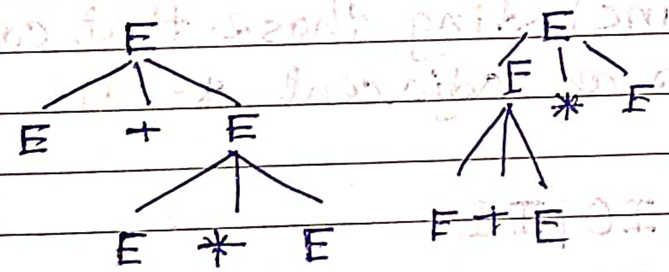
Grammars for which distinct parse tree cannot be obtained are called ambiguous grammar.

From Inferences to Trees:-

Th:- Let $G=(V,T,S,P)$ be a CFG if the recursive inference procedure tells us that terminal string w is in the language of variable A, then there is a parse tree with root A and yield w.

Ambigious Grammar:-

$E \to E+E \mid E*E$
$E \to a$
$a+a*a$



As there are two diff. parse trees having same yield, so this grammar is ambiguous.

Rules to remove ambiguity from the grammar:-

1) A factor is an expression that cannot be broken apart by any adjacent operator, either $a*$ or a^+. The only factor in our expression language are —

(a) Identifiers — It is not possible to seperate the letters of an identifier by attaching an operator.

(b) Any parenthasized expression, no matter what appears inside the ~~p~~ ( ) to present what is inside from becoming the operand of any of the operator outside the ( ).

2) A term is an expression that cannot be broken by + operator.

Eg. $a+a*a \Rightarrow a+(a*a)$

$a*a*a \Rightarrow (a*a)*a$

3) An expression will henceforth refer to any possible expression including those that can't be broken by either an adjacent * or adjacent +.

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$F \rightarrow I \mid (E)$

$T \rightarrow F \mid T*F$

$E \rightarrow (T) \mid E+I$

Th. For a grammar $G=(V,T,S,P)$ and string w in T^*, w has two distinct parse tree iff w has two distinct left most derivations from S.

Inherent Ambiguity :-

$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$

$W = aabbccdd$

$S \rightarrow AB \mid C$

$A \rightarrow aAb \mid ab$

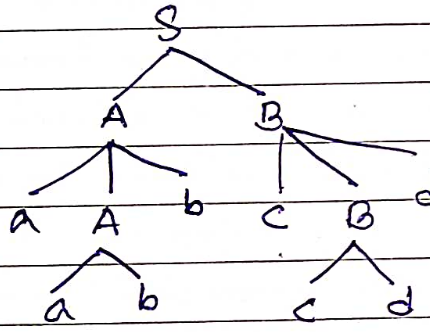
$B \rightarrow cBd \mid cd$

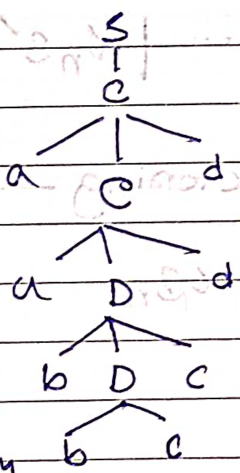
$C \rightarrow aCd \mid aDd$

$D \rightarrow bDc \mid bc$

$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB$
$\Downarrow$
$aabbcBd$
$\Downarrow$
$aabbacd$

$S \Rightarrow C \Rightarrow aCd \Rightarrow aaDdd$
$\Downarrow$
$aabDcdd \Rightarrow aabbccdd$





As there are two distinct parse trees and this ambiguity cann't be removed, so this grammar has inherent ambiguity.

Left-recursive Grammar :-

$S \rightarrow S\alpha \mid \beta$

$\alpha \longrightarrow$ a comb. of terminals and non-terminals

We can rewrite this as

$$\boxed{\begin{array}{l} S \rightarrow \beta S' \\ S' \rightarrow \alpha S' \mid \epsilon \end{array}}$$

Eg. $E \rightarrow E \underbrace{+T}_{\alpha} \mid \underbrace{T}_{\beta}$

$E \rightarrow TE'$

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

$S \rightarrow A$
$A \rightarrow Sd \mid b$ } Indirectly LR

$S \xRightarrow{*} Sd$

$S \rightarrow A$

$S \rightarrow Sd \mid b$ $\quad b = \beta$

$S \rightarrow bS'$

$S' \rightarrow dS' \mid \epsilon$

$S \rightarrow S\alpha_1 \mid S\alpha_2 \mid \cdots \mid S\alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$

$S \rightarrow \beta_1 S' \mid \beta_2 S' \mid \cdots \mid \beta_n S'$

$S' \rightarrow \alpha_1 S' \mid \alpha_2 S' \mid \cdots \mid \alpha_n S' \mid \epsilon$

Eliminating Left Factoring →

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n$

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \cdots \beta_n$

Linear Grammar :-

$A \rightarrow B$

Left Linear grammar (when non terminals are at the extreme left)

$S \rightarrow Sa$

$S \rightarrow Ab$

$A \rightarrow a$

Right Linear Grammar:-

$S \rightarrow aS$

$S \rightarrow bA$

$A \rightarrow a$

Simplification of Grammar :-

1. Eleminate null production
2. Eleminate unit production
3. " useless production

CNF GNF

1.

$S \to ABaC$
$A \to BC$
$B \to b|\epsilon$
$C \to D|\epsilon$
$D \to d$

$B \to \epsilon$ Nullable
$C \to \epsilon$

$S \to AB\overset{\to\epsilon}{}aC$

$S \to BCaC$ with $B \to \epsilon$, $C \to \epsilon$, $C \to \epsilon$

$S \to a$

$\cancel{S \to ABaC|B}$

(*)

$A \to BC \Rightarrow A \to \epsilon$ with $B \to \epsilon$, $C \to \epsilon$

$A \Rightarrow BC \Rightarrow C \Rightarrow \epsilon$

$A \to \epsilon$

24 Sunday

(*) $S \to ABaC | BaC | AaC | ABa | aC | Aa | Ba | a$

$A \to BC | C | B$

$B \to b$

$C \to D$

$D \to d$

2. $S \rightarrow Aa \mid B$

$B \rightarrow A \mid bb$

$A \rightarrow a \mid bc \mid B$

$S \rightarrow B \rightarrow A$

$$\boxed{\begin{array}{l} S \rightarrow Aa \\ A \rightarrow a \mid bc \\ B \rightarrow bb \end{array}}$$

$S \rightarrow B \rightarrow A \rightarrow a$, $\rightarrow bc$, $\rightarrow B \rightarrow bb$

$$\boxed{\begin{array}{l} S \rightarrow B \\ S \rightarrow A \\ B \rightarrow A \\ A \rightarrow B \end{array}}$$

$B \rightarrow A \rightarrow a$, $\rightarrow bc$; $B \rightarrow bb$

$A \rightarrow a \mid bc \mid bb$

$B \rightarrow a \mid bc \mid bb$

$S \rightarrow Aa \mid a \mid bc \mid bb$
$A \rightarrow a \mid bc \mid bb$
$B \rightarrow a \mid bc \mid bb$
$\rightarrow$ final result

3. $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C$ ×

$B \rightarrow b$

$C \rightarrow D$ ×

$D \rightarrow E$ ×

~~usef~~ useless

GNF → Greibach Normal Formal

~~Cho~~ Chomsky Normal Form (CNF)

A context free grammar is in CNF if all production are of the forms $A \to BC$ or $A \to a$ where $A, B, C \in V$ and $a \in T$

Ex – $S \to AS \mid a$
$A \to SA \mid b$

Th → Any context free grammar $G = (V, T, S, P)$ with $\epsilon \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, \hat{S}, \hat{P})$ is in CNF.

$S \to ABa$
$A \to aab$
$B \to Ac$

$B_a \to a$
$S \to ABB_a$
$D_1 \to BB_a$
$S \to AD_1$

$A \to aab$
$B_b \to b$
$A \to B_a B_a B_b$
$D_2 \to B_a B_b$
$A \to B_a D_2$

$B \to Ac$
$B_c \to c$
$B \to AB_c$

$G = (V, T, S, P)$

$V = \{S, A, B, B_a, B_b, B_c, D_1, D_2\}$

$T = \{a, b, c\}$ $S = \{S\}$

Arpan Basu

$\boxed{\text{GNF}}$ (Greibach Normal Form)

A context free grammar is said to be GNF if all productions have the form

$A \rightarrow a \cancel{\alpha} X$

where $a \in T$

$\alpha\ \cancel{X} \in V^*$ and $A \in V$

Simple Grammar or S-grammar:-

A CFG, $G = (V, T, S, P)$ is said to be a simple grammar or S-grammar if all its productions are in the form

$A \rightarrow a\cancel{x}$ where $a \in T$ and $\cancel{X} \in V^*$ and $A \in V$ and any pair (A, a) occurs at most once in P.

Eg. $S \rightarrow aS \mid bSS \mid c$] this is in s-grammar

Eg. $S \rightarrow aS \mid bSS \mid aSS \mid c$

This pair $S \rightarrow a$ occurs more than once. So this is not in s-grammar. But it is GNF.

Q. $S \to AB$
$A \to aA \mid bB \mid b$
$B \to b$

$A \to a x$
$a \in T$
$A \in V$
$x \in V^*$

$S \to AB$
$S \to aAB \mid bBB \mid bB$

be the grammar after continuing to GNF is

$S \to aAB \mid bBB \mid bB$
$A \to aA \mid bB \mid b$
$B \to b$

(S, b) and (A, b) occurs twice so it is not s-grammar

Q. $S \to abSb \mid aa$

$A \to a$
$B \to b$

$$\boxed{\begin{array}{l} S \to aBSB \mid aA \\ A \to a \\ B \to b \end{array}}$$

This is in GNF form.

Q. $S \to AB$
$A \to aAA \mid \epsilon$
$B \to bBB \mid \epsilon$

Eliminate ϵ-production from this.

$A \to aAA \to aA \to aaAA$
$\searrow \epsilon \quad \searrow aA \to a$

$A \to aAA \mid aA \mid a \}$ $\qquad S \to B$

$B \rightarrow bBB \mid bB \mid b$
$S \rightarrow AB \mid A \mid B$ } Ans.

$S \rightarrow A$

Removing non-terminals

$S \rightarrow AB \mid aAA \mid aA \mid a \mid bBB \mid bB \mid b$

$A \rightarrow aAA \mid aA \mid a$

$B \rightarrow bBB \mid bB \mid b$

Removing useless productions,

$S \rightarrow AB \mid aA \mid a \mid bB \mid b$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Eg. Convert the following grammars in CNF.

a) $S \rightarrow ABa$
$A \rightarrow aab$
$B \rightarrow Ac$

b) $S \rightarrow aSa \mid SSa \mid a$

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

Convert the following grammars in GNF.

a) $S \rightarrow aSb \mid ab$

b) $S \rightarrow ABb \mid a$
$A \rightarrow aaA \mid B$
$B \rightarrow bAb$

a) $S \to ABa$
$A \to aab$
$B \to Ac$

$S \to abAB$
$A \to bAB \mid \epsilon$
$B \to BAa \mid A \mid \epsilon$

$A \to \epsilon$
$S \to abAB \mid abB$
$A \to \cancel{BB}\, bAB \mid bB$
$B \to BAa \mid A \mid Ba \mid \epsilon$

$B \to \epsilon$
$S \to abAB \mid abB \mid ab \mid abA$
$A \to bAB \mid bB \mid bA \mid b$
$B \to BAa \mid A \mid Ba \mid Aa \mid a$

.vl

```
module name (input a, input b, output c);
begin
    assign c = a&b;
end
end module
```

Test

```
module testabc
    a = .x
    b = .y
    c = .z
initial
    x=1;
    y=0;
    #100;
end
end module
module name a1 (a(.x); b(.y); c(.z));
```