MOD-3

Derivation: It's process of replacing a variable (either left most or right most variable) at each and every step

Usually derivation can implement at RHS of the production group.

Types of Derivations:

1. Left most derivation (LMD)
   - The process of replacing a variable of left most at each & every step
2. Right most derivation (RMD)
   - The process of replacing a right most variable at each & every step.

Derivation Tree / Parse / Syntax tree:

Derivation tree: Visual or pictorial representation of derivation process.

Types of Derivation Tree:

1. Left most Derivation Tree (LMDT) : Visual representation of left most derivation process
2. Right most derivation Tree (RMDT) : Visual representation of right most derivation process.

Q) Find LMD, RMD, LMDT, RMDT for following CFG.

$E \rightarrow E+E / E*E / E-E / (E) / i$ & process the I/P strings are i) i+i ii) i−i iii) i*i iv) (i+i) v) (i−i) vi) (i*i) vii) (i)+(i) viii) (i)*(i) ix) (i)−(i)

→ Formal def of CFG.

i.e. $G = \langle V, T, P, S \rangle$

$V = \{ E \}$ ⇒ uppercase

$T = \{ +, -, *, (,), i \}$ ⇒ other than upper case letter

$P = \begin{cases} E \rightarrow E+E & \rightarrow ① \\ E \rightarrow E*E & \rightarrow ② \\ E \rightarrow E-E & \rightarrow ③ \\ E \rightarrow (E) & \rightarrow ④ \\ E \rightarrow i & \rightarrow ⑤ \end{cases}$ Production rules $\alpha \rightarrow \beta$

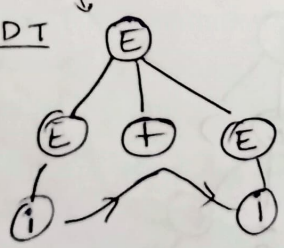
$S = \{ E \}$ ⇒ start symbol of Grammar

i) i+i

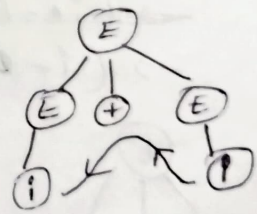
LMD: $E \rightarrow E+E \rightarrow ①$
$\rightarrow i+E \quad (E \rightarrow i) \rightarrow ⑤$
$\rightarrow i+i \quad (E \rightarrow i)$

RMD: $E \rightarrow E+E$
$\rightarrow E+i \quad (E \rightarrow i)$
$\rightarrow i+i \quad (E \rightarrow i)$

LMDT



RMDT

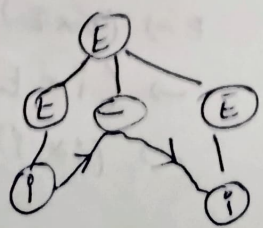


ii) i−i

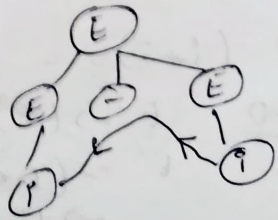
LMD: $E \rightarrow E-E \rightarrow ③$
$\rightarrow i-E \quad (E \rightarrow i)$
$\rightarrow i-i \quad (E \rightarrow i)$

RMD: $E \rightarrow E-E$
$\rightarrow E-i$
$\rightarrow i-i$

LMDT:



RMDT:

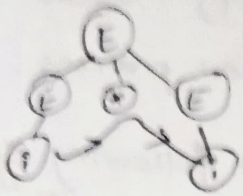


iii) i * i

LMD

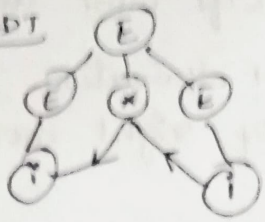
E → E * E -①
→ i * E (E → i)
→ i * E (E → i)

LMDT



RMD: E → E * E
→ E * i
→ i * i

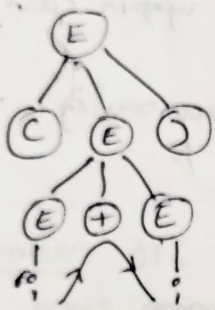
RMDT



iv) (i+i)

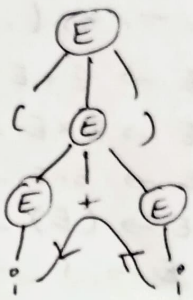
LMD E → (E)
E → (E + E)
→ (i + E)
→ (i + i)

LMDT



RMD E → (E)
E → (E + E)
→ (E + i)
→ (i + i)

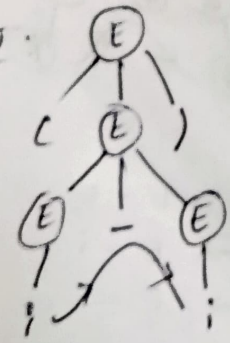
RMDT



v) (i - i)

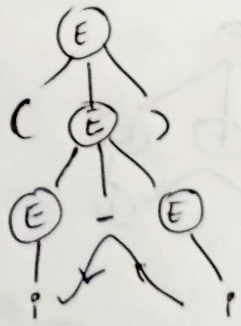
LMD E → (E)
E → (E - E)
→ (i - E)
~~→ (i - E)~~
→ (i - i)

LMDT



RMD: E → (E)
E → (E - E)
→ (E - i)
→ (i - i)

RMDT:



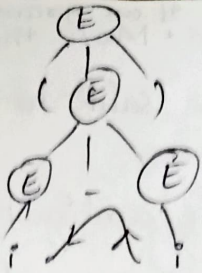
vi) (i * i)

LMD E → (E)
E → (E * E)
→ (i * E)
→ (i * i)

RMD: E → (E)
E → (E * E)
→ (i * E)
→ (i * i)

LMDT

RMDT

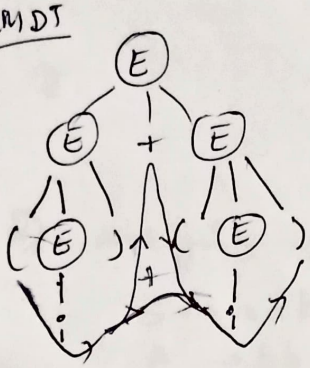


vii) (i) + (i)

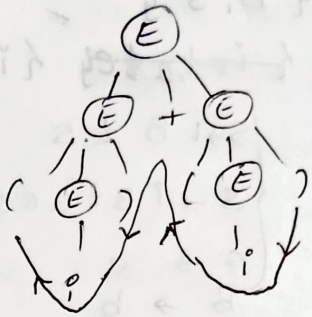
LMD $E \rightarrow E + E$
$\rightarrow (E) + E$
$\rightarrow (i) + E$
$\rightarrow (i) + (E)$
$\rightarrow (i) + (i)$

RMD $E \rightarrow E + E$
$\rightarrow E + (E)$
$\rightarrow E + (i)$
$\rightarrow (E) + (i)$
$\rightarrow (i) + (i)$

LMDT



RMDT

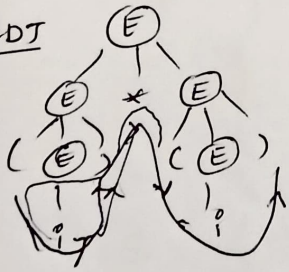


viii) (i) * (i)

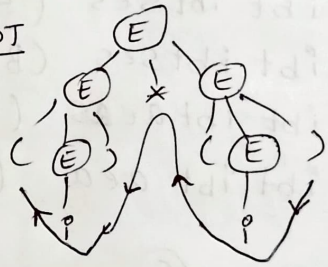
LMD: $E \rightarrow E * E$
$\rightarrow (E) * E$
$\rightarrow (i) * E$
$\rightarrow (i) * (E)$
$\rightarrow (i) * (i)$

RMD: $E \rightarrow E * E$
$\rightarrow E * (E)$
$\rightarrow E * (i)$
$\rightarrow (E) * (i)$
$\rightarrow (i) * (i)$

LMDT



RMDT

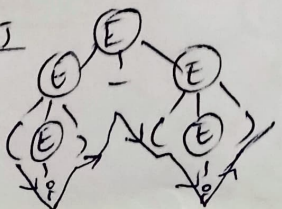


ix) (i) − (i)

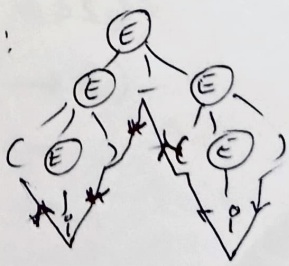
LMD: $E \rightarrow E - E$
$\rightarrow (E) - E$
$\rightarrow (i) - E$
$\rightarrow (i) - (E)$
$\rightarrow (i) - (i)$

RMD: $E \rightarrow E - E$
$\rightarrow E - (E)$
$\rightarrow E - (i)$
$\rightarrow (E) - (i)$
$\rightarrow (i) - (i)$

LMDT



RMDT:



Ambiguity Grammar: If an grammar produces More than 1 derivation Trees (LMD) or (RMD) then it is said to be Ambiguous Grammar.

a) Eliminate the ambiguity for following CFG.

i) $S \rightarrow iBtS \mid iBtSeS \mid a$

$B \rightarrow b$ and process the i/p string "ibt ibt aea".

→ Formal def of CFG

$G = < V, T, P, S >$

$V = \{ B, S \}$

$T = \{ i, t, e, a, b \}$

$P = \begin{cases} S \rightarrow iBtS & ① \\ S \rightarrow iBtSeS & ② \\ S \rightarrow a & ③ \\ B \rightarrow b & ④ \end{cases}$

$S = \{ S \}$

LMD 1:

$S \rightarrow iBtS$

$\rightarrow ibtS \quad (B \rightarrow b)$

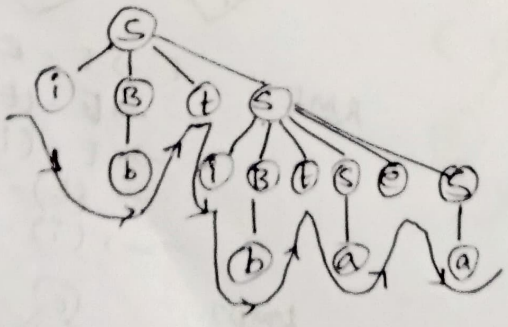
$\rightarrow ibt\,iBtSeS \quad (S \rightarrow iBtSeS)$

$\rightarrow ibt\,ibtSeS \quad (B \rightarrow b)$

$\rightarrow ibt\,ibtaeaS \quad (S \rightarrow a)$

$\rightarrow ibt\,ibt\,aea \quad (S \rightarrow a)$

LMDT



LMD2 : $S \rightarrow iBtSeS$.

$\rightarrow ibtSeS$ $(B \rightarrow b)$

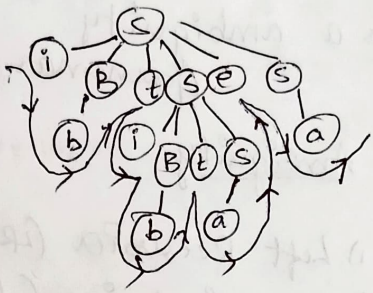
$\rightarrow ibtiBtSeS$ $(S \rightarrow iBtS)$

$\rightarrow ibtibtSeS$ $(B \rightarrow b)$

$\rightarrow ibtibtaeS$ $(S \rightarrow a)$

$\rightarrow ibtibtaea$ $(S \rightarrow a)$

LMDT2 :



∴ The given CFG generates more than 1 derivation trees so, it's said to be ambiguous Grammar.

2) Ambiguity Grammar: string : "abab"

$S \rightarrow aAb \mid abSb \mid a$

$A \rightarrow bS$

→ formal def of CFG.

$G = \langle V, T, P, S \rangle$

$V = \{A, S\}$

$T = \{a, b\}$

$$P = \begin{cases} S \rightarrow aAb & ① \\ S \rightarrow abSb & ② \\ S \rightarrow a & ③ \\ A \rightarrow bS & ④ \end{cases}$$

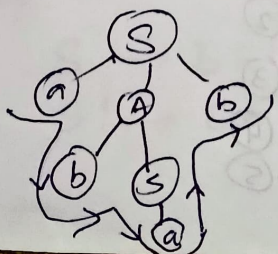
$S = \{S, \#\}$

LMD1 : $S \rightarrow aAb$

$S \rightarrow abSb$ $(A \rightarrow bS)$

$\rightarrow abab$ $(S \rightarrow a)$

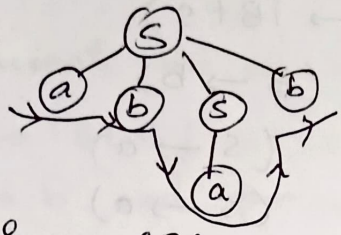
LMDT1 :



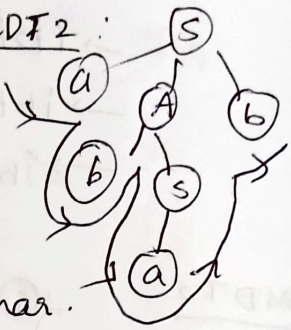
LMD2 : $S \to abSb$
$\to abab \quad (S \to a)$

$S \to aAb$
$\to abSb \quad (A \to Sb)$
$\to abab \quad (S \to a)$

LMDT2 :



LMDT2 :



∴ The given CFG generates more than 1 derivation trees, so, it's a ambiguity grammar.

Elimination or removing Ambiguity:

It contain 2 methods: 1) Left Recursion (LR)
2) Left factoring (LF)

Left Recursion (LR) : If any grammar follows $A \to A\alpha / \beta$. then, it's said to LR.

To remove the LR we will use following formula:

$$A \to \beta A'$$
$$A' \to \alpha A'$$
$$A' \to \epsilon$$

1) Eliminate the LR for following CFG.

$E \to E+T / T$
$T \to T*F / F$
$F \to (E) / id$

→ Formal def of CFG

$G = \langle V, T, P, S \rangle$

$V = \{ E, T, F \}$

$T = \{ +, *, i, d \}, (,))$

$P :$
$E \to E+T$ ①
$E \to T$ ②
$T \to T*F$ ③
$F \to (E)$ ④
$F \to id$ ⑤

$S = \{ E, T, F \}$

- $E \to E+T \mid T$ (*) $A \to A\alpha \mid \beta$

$A = E$; $\alpha = +T$; $\beta = T$; $A' = E'$

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

$A \to \beta A'$
$A' \to \alpha A'$
$A' \to \epsilon$

$\Rightarrow E \to TE'$
$E' \to +TE' \mid \epsilon$

~~$\Rightarrow E' \to +TE' \mid \epsilon$~~

- $T \to T * F \mid F$

$A \to A\alpha \mid \beta$

$A = T$; $\alpha = *F$; $\beta = F$

$A' = E'$

$T \to FT'$
$T' \to *FT'$
$T' \to \epsilon$

$\Rightarrow T \to FT'$
$T' \Rightarrow *FT' \mid \epsilon$.

$\therefore$ $E \to TE'$
$E' \to +TE' \mid \epsilon$
$T \to FT'$
$T' \to *FT' \mid \epsilon$.
$F \to (E) \mid id$.

2) $S \to Aa \mid b$
$A \to Sc \mid d$.

$P = \{ S \to Aa,\ S \to b,\ A \to Sc,\ A \to d \}$

$\Rightarrow$ ~~$S \to Sca \mid da \mid b$~~

~~$S \to$~~

$A \to A\alpha \mid \beta$

$A = S$; $\alpha = ca$

$\beta = b$; $A' = E'$

$S \to Sca \mid b$

$S \to bE'$
$S' \to cE'$
$S' \to \epsilon$

$A \to \beta A'$
$A' \to \alpha A'$
$A' \to \epsilon$

$\to$ $S \to bE'$
$S' \to cE'$

$S \to Sca$
$S \to da$
$S \to b$
$A \to Sc$
$A \to d$

$S \to Sca \mid da \mid b$: $A \to Sc \mid d$

↓ coz, β shd be single term.

$S \to Sca \mid b.$

$A \to A\alpha \mid \beta$

$A = S$
$\alpha = ca$
$\beta = b$

$S = bS'$
$S' \to cas' \mid \epsilon$

$\{ S \to Aa \mid b. \quad A \to A\alpha \mid \beta \}$ ⇒

$S \to bS'$
$S' \to cas' \mid \epsilon$
$A \to Sc \mid d$
$S \to da \mid b$

LBF :

If an Grammar follows $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \ldots \mid \alpha\beta_n$. where α = common prefix.

where $\beta_1, \beta_2, \beta_3 \ldots \beta_n$ are different β values should be depends on prblm requirements

$A \to \alpha A'$
$A' \to \beta_1 \mid \beta_2 \mid \beta_3 \ldots \mid \beta_n$

1) Eliminate the LF for following Grammar.

$S \to iBtS \mid iBtSeS.$

⇒ $\alpha = iBtS$
$\beta_1 = \epsilon$
$\beta_2 = eS.$

$A \to \alpha A' \Rightarrow$ $S \to iBtSS'$
$A' \to \beta_1 \mid \beta_2 \mid \beta_3 \ldots \mid \beta_n \Rightarrow$ $S' \to \epsilon \mid eS$

d) $N \rightarrow NCET / Npuc / NCMS.$

$\Rightarrow A = N$

$\beta_1 = CET$

$\beta_2 = PUC$

$\beta_3 = CMS$

$A \rightarrow \alpha A' \Rightarrow N \rightarrow NN'$

$A' \rightarrow \beta_1 / \beta_2 / \beta_3 .. / \beta_n \Rightarrow N' \rightarrow CET / PUC / CMS.$

Closure Properties of Context Free lang: (CFL):

~~1. Union: The clause~~ clause of

1. Union: The closure property of CFL closed under union if L_1 & L_2 are 2 lang then their union is denoted by $L_1 \cup L_2$.
2. Concatenation: $L_1 L_2$
3. Kleen closure : L^*
4. positive closure : L^+
5. Reverse : L^R
6. Compliment : L' (or) $\bar{L}$
7. Intersection : $L_1 \cap L_2$
8. Difference : $L_1 - L_2$
9. Substitution : (replacement).
10. Homomorphism : $h(L)$
11. Inverse hom : $h^{-1}(L)$

Simplification of CFG (or) Minimization of CFG

Simplifying Context free Grammar is crucial in automata theory.

As we have seen as effectively represented by CFG. All the Grammars are not always optimize that means Grammar may consist of unnecessarily symbols. Having extra symbols unnecessarily increasing the length of CFG. To avoid this we use Simplification of CFG.

Simplification of CFG can be done in 3 ways:

1. Elimination of useless production / (or) Variable
2. Elimination of unit production.
3. Elimination of ϵ - production / null production.

① Elimination of useless productions/variable:

It depends on 2 cases.

Case 1: ~~Here~~ A variable is useless if it can't find it's terminal symbol.

Case 2: If any production rule can't reachable from it's starting symbol then it's said to be useless production.

1) Eliminate the production for following symbols. CFG

$S \to AC \mid BA$
$C \to cB \mid Ac$
$A \to a$
$B \to aC \mid b$

⇒ Formal def of CFG:

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

$V = \{S, A, C, B\}$

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

P:
$S \to AC$ ①
$S \to BA$ ②
$C \to cB$ ③
$C \to Ac$ ④
$A \to a$ ⑤
$B \to aC$ ⑥
$B \to b$ ⑦

- $S \to AC$
  $\to acB$
  $\to acb$ (valid)
- $S \to BA$
  $\to ba$ (V)
- $C \to cB$
  $\to cb$ (V)
- $C \to Ac$
  $\to ac$ (V)
- $A \to a$ (V)
- $B \to aC$
  $\to acB$
  $\to acb$ (V)
- $B \to b$ (V)

Optimized Grammar is: $S \to AC \mid BA$; $C \to cB \mid Ac$

$B \to aC \mid b$ $A \to a$.

2) $S \to AB \mid bX$
$A \to BAd \mid bSX \mid a$
$B \to aSB \mid bBX$
$X \to SBD \mid aBX \mid ad$

→ Formal def of CFG: $G = \{V, T, P, S\}$

$V = \{S, A, B, X, D\}$; $T = \{b, d, a\}$

$S = \{S\}$.

P: $S \to AB$ (1)
$S \to bX$ (2)
$A \to BAd$ (3)
$A \to bSX$ (4)
$A \to a$ (5)
$B \to aSB$ (6)
$B \to bBX$ (7)
$X \to SBD$ (8)
$X \to aBX$ (9)
$X \to ad$ (10)

- $S \to AB$
  $\to a\,aSB$
  $\to aaabXbBX$
  $\to aaabadb$ (Invalid)
- $S \to bX$
  $\to bad$ (valid)
- $A \to BAd$
  $\to aSBad$ (Invalid)
- $A \to bSX$
  $\to bbXX$
  $\to bbadad$ (V)
- $A \to a$ (V)
- $B \to aSB$ (IV)
- $B \to bBX$ (IV)

'B' can't find it's terminal symbol. So it's useless

- $X \to SBD$ (IV)
- $X \to aBX$ (IV)
- $X \to ad$ (V)

$S \to AB$, $A \to BAd$, $B \to aSB/bBX$, $X \to SBD/aBX$

The Optimized Grammar is: $S \to bx$; $A \to bSX/a$

~~$A \to a$~~ ; $X \to ad$.

3) $S \to aC/SB$

Formal def of CFG: $G = \{V, T, P, S\}$

$V = \{S, C, B, A\}$; $T = \{a, b, d\}$; $S = \{S\}$

P:
$S \to aC$
$S \to SB$
$A \to bSCa$
$A \to ad$
$B \to aSB$
$B \to bBC$
$C \to aBC$
$C \to ad$

- $S \to aC$
  $\to aad$ (V)
- $S \to SB$
  $\to aC(aSB)$ (Inv)
- $A \to bSCa$
  $\to baadada$ (V)
- $A \to ad$ (V)
- $B \to aSB$ (Inv)
- $B \to bBC$ (IV)
- $C \to aBC$ (IV)
- $C \to ad$ (V)

Optimized Grammar is

$S \to aC$
$A \to bSCa/ad$
$C \to ad$.

Elimination of Unit production:

A production rule is said to unit production, if it contain production rule of the form $\boxed{V \rightarrow V}$

Q) Eliminate unit production for following CFG.

$S \rightarrow aX \mid Yb \mid Y \; ; \; X \rightarrow S \; ; \; Y \rightarrow Yb \mid b$

→ Formal def of CFG: $G = \langle V, T, P, S \rangle$

$V = \{S, X, Y\} : T \{a, b\} \; ; \; S = \{S\}$

$P = \begin{cases} S \rightarrow aX & ① \\ S \rightarrow Yb & ② \\ S \rightarrow Y & ③ \\ X \rightarrow S & ④ \\ Y \rightarrow Yb & ⑤ \\ Y \rightarrow b & ⑥ \end{cases}$

From the given CFG the unit production are: $(V \rightarrow V)$ Only capital letters on b:s.

$S \rightarrow Y$; $X \rightarrow S$; ~~$Y \rightarrow Y$~~

($X \rightarrow X$) Before eliminating unit production we need to replace 'Y'.

$Y \rightarrow Yb$ $\quad S \rightarrow aX$

$Y \rightarrow b$ $\quad S \rightarrow Yb$.

~~$S \rightarrow Y$~~ ; $S \rightarrow Yb \mid b$

Y can be replaced by Yb. and b

→ $S \rightarrow Yb \mid b$.

$X \rightarrow aX \mid Yb \mid b$.

Given: $S \rightarrow aX \mid Yb \mid Y$

$X \rightarrow aX \mid Yb \mid Y$.

$Y \rightarrow Yb \mid b$.

It's optimized CFG because it's does not have unit & useless production.

Q) $S \rightarrow AA \; ; \; A \rightarrow B \mid BB \; ; \; B \rightarrow abB \mid b \mid bb$.

→ Formal def: $G = \langle V, T, P, S \rangle$

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

$P = \begin{cases} S \rightarrow AA & B \rightarrow b \\ A \rightarrow B & B \rightarrow bb \\ A \rightarrow BB & \\ B \rightarrow abB & \end{cases}$

Unit production

$A \to B$.

replace $B \to abB : B \to b ; B \to bb$

$A \to abB \mid b \mid bb$.

Given : $S \to AA$
$A \to B \mid BB$
$B \to abB \mid b \mid bb$.

$\Rightarrow$

$S \to AA$
$A \to abB \mid b \mid bb \mid BB$
$B \to abB \mid b \mid bb$

It's optimized CFG.

Elimination of NULL / ϵ - production

Q)

$S \to ax \mid bx$
$X \to a \mid b \mid \epsilon$

→ Formal def : $G = \langle V, T, P, S \rangle$

$V = \{S, X\}$, $T \{a, b, \epsilon\}$, $S = \{S\}$

$P =$
$S \to ax$
$S \to bx$
$X \to a$
$X \to b$
$X \to \epsilon$

ϵ - production
$X \to \epsilon$

$S \to aX \mid bX \mid a \cdot \epsilon \mid b\epsilon$

$S \to aX \mid bX \mid a \mid b$

$X \to a \mid b$

2) $S \to ABaC$; $A \to BC$; $B \to b \mid \epsilon$; $C \to D \mid \epsilon$;
$D \to d$

→ Formal def : $G \prec V, T, S, P \succ$

$V = \{ S, A, B, C, D \}$

$T = \{ a, b, d, \epsilon \}$; $S = \{ S \}$

$P = \begin{cases} S \to ABaC \\ A \to BC \\ B \to b \\ B \to \epsilon \\ C \to D \\ C \to \epsilon \\ D \to d \end{cases}$

From a Given CFG,
ϵ - production => $B \to \epsilon$; $C \to \epsilon$.

~~$S \to AB\epsilon a\epsilon \Rightarrow Aa$; $A \to B\epsilon \Rightarrow \epsilon\epsilon$~~

$S \to ABaC \mid A\epsilon aC \mid ABa\epsilon$.

$S \to ABaC \mid AaC \mid ABa$.

$A \to BC \mid \epsilon C \mid$ ~~$B\epsilon \Rightarrow B\epsilon$~~

$A \to BC \mid C \mid B$; ~~$A \Rightarrow B$.~~

$B \to b$ ~~$\mid \epsilon$~~

$C \to D$ ~~$\mid \epsilon$~~.

$D \to d$.

From a Given CFG, unit productions are:

$A \to C$

$A \to B$

$C \to D$.

replace 'A' as 'C' 'B' and 'c' as 'D'

$S \to ABaC \mid AaC \mid ABa \mid CBaC \mid BBaC \mid ABaD \mid CaD \mid BdaD$

~~$S \to CBaC \mid B$~~ $BaC \mid AaD \mid CBa \mid BBa$.

$A \to BC \mid BD$

Optimized : ~~$A \to BC \mid BD$~~

~~$D \to d$~~.

~~$S \to ABa \mid$~~ $S \to Bad \mid BBC$

$B \to b$.

$D \to d$.

3) $S \to AB \mid aB$; $A \to aA \mid \epsilon$; $B \to bB \mid C$; $C \to c \mid \epsilon$.

⇒ Formal def : $G = \langle V, T, S, P \rangle$

$V = \{S, A, B, C\}$; $T = \{a, b, c, \epsilon\}$; $S = \{S\}$

$P = \begin{cases} S \to AB \\ S \to aB \\ A \to aA \\ A \to \epsilon \\ B \to bB \\ B \to C \end{cases} \quad \begin{cases} C \to c \\ C \to \epsilon \end{cases}$

NULL/ϵ- production ⇒ $A \to \epsilon$; $C \to \epsilon$; $B \to \epsilon$

$S \to AB \mid aB \mid \epsilon B \mid AB$; $S \to AB \mid aB \mid B \mid A \mid a \mid \epsilon$.

$S \to AB \mid aB \mid B \mid AB$. ; $S \to \epsilon$

$A \to aA \mid a\epsilon$

$A \to aA \mid a$.

$B \to bB \mid \epsilon$

$C \to c$

It's unable to remove because in a given CFG we don't have 'S' in RHS of the production rules.

unit

∴ $S \to AB \mid aB \mid B \mid A \mid a$

$A \to aA \mid a$

$B \to b$

$C \to c$

Unit production: $S \to B$; $S \to A$

no replacement because we don't have S variable

Question

1. LMD, RMD, LMDT, RMDT short note on this (theory)
2. Ambiguity problm (define)? & eliminate ambiguity for following CFG
3. closure properties of CFG (Theory).
4. Construct parse Trees for following CFG. (LMD, RMD, LMDT, RMDT)
5. construct CNF for following CFG.
6. Construct GNF. " " "
7. what is pumping lemma of CFL? check whether the following lang is CFL or NOT

Pumping Lemma Stmt of CFL :

$W = UVXYZ$. where, $Y \neq \epsilon$ then, unique stmt is UV^iXY^iZ ; where $i \geq 0$.

1) check whether the following lang is CFL or not

$L = \{ a^n b^n \mid n > 0 \}$.

→ $n = 1, 2, 3 \ldots$

$L = \{ a^1 b^1, a^2 b^2, a^3 b^3, \ldots \}$

$L = \{ ab, aabb, aaabbb \ldots \}$

choose any string 'aabb'

$w = aabb$

$w = UVXYZ$ but $Y \neq \epsilon$

case(i) $U = a$; $V = \epsilon$; $X = a$; $Y = b$; $Z = b$

ii) $U = \epsilon$; $V = \epsilon$; $X = aa$; $Y = bb$; $Z = \epsilon$

iii) $U = a$; $V = a$; $X = b$; $Y = b$; $Z = \epsilon$

choose any case :

case (i) $u = a$; $v = \epsilon$; $x = a$; $y = b$; $z = b$

$uv^i x y^i z$ where $i \geq 0$

$i = 0 \Rightarrow a\epsilon^0 ab^0 b = aab \notin L$

$i = 1 \Rightarrow a\epsilon^1 ab^1 b = aabb \in L$

$i = 2 \Rightarrow a\epsilon^2 ab^2 b = aabbb \notin L$

$i = 3 \Rightarrow a\epsilon^3 ab^3 b = aabbbb \notin L$

The given language is not a CFL

2) $L = \{ a^p \mid$ where p is prime $\}$

→ $P = 2, 3, 5, 7, 11, 13, 17$

$L = \{ a^2, a^3, a^5, a^7 - - \}$

$L = \{ aa, aaa, aaaaa, aaaaaaa - - - \}$

choose any string 'aaaaaa'.

$w = aaaaaa$

$w = uvxyz$ but $\forall \epsilon$

Cases i) $u = a$; $v = a$; $x = a$; $y = a$; $z = a$

ii) $u = \epsilon$; $v = \epsilon$; $x = aa$; $y = aa$; $z = a$

iii) $u = aa$; $v = \epsilon$; $x = \epsilon$; $y = aaa$; $z = \epsilon$

choose any case :

Case (iii) $u = aa$; $v = \epsilon$; $x = \epsilon$; $y = aaa$; $z = \epsilon$

$uv^i x y^i z$ where $i \geq 0$

$i = 0 \Rightarrow aa\epsilon^0 \epsilon (aaa)^0 \epsilon = aa \in L$

$i = 1 \Rightarrow aa\epsilon^1 \epsilon (aaa)^1 \epsilon = aaaaa \in L$

$i = 2 \Rightarrow aa\epsilon^2 \epsilon (aaa)^2 \epsilon = aaaaaaaaa \notin L$

$i = 3 \Rightarrow aa\epsilon^3 \epsilon (aaa)^3 \epsilon = aaaaaaaaaaaaa \in L$

$i = 4 \Rightarrow aa\epsilon^4 \epsilon (aaa)^4 \epsilon = aaaaaaaaaaaaaaaaa \notin L$

The given Lang is not CFL.

Normal Forms

To reduce the RHS of production symbols in a fixed format then we use normal forms.

Types:

1. Chomsky's normal form (CNF)
2. Griebach normal form (GNF).

1) CNF Stmt: $V \to VV$, $V \to T$

NOTE: Before going to construct CNF we have to make sure that, if any CFG contains useless productions, unit production, NULL / ϵ production then, we need to eliminate them.

1) Construct CNF for following CFG

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

→ Formal def of CFG

$G = \langle V, T, S, P \rangle$

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

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

$P = \{ S \to bA,\ S \to aB,\ A \to bAA,\ A \to aS,\ A \to a,\ B \to aBB,\ B \to bS,\ B \to b \}$

From a given CFG, there's no ϵ, unit, useless production.

- $S \to bA$ ✗ It's not CNF Stmt

Introduction of new variable $C_b \to b$

replace, $S \to C_bA$

- $S \to aB$ $\quad C_a \to a$

  $S \to C_aB$

- $A \to bAA$ $\quad C_b \to b$

  $A \to C_bAA$ $\quad D \to AA$

  $A \to C_bD$

- $A \to aS$ $\quad C_a \to a$

  $A \to C_aS$

- $A \to a$

- $B \to aBB$ $\quad C_a \to a$

  $B \to C_aBB$ $\quad E \to BB$

  $B \to C_aE$

- $B \to bS$ $\quad C_b \to b$

  $B \to C_bS$

- $B \to b$

Optimized CNF state:

$C_b \to b$; $S \to C_b a$

$C_a \to a$; $S \to C_a B$

$D \to AA$; $A \to C_b D$

$A \to C_a S$

$A \to a$

$E \to BB$; $b \to C_a E$

$b \to C_b S$

$B \to b$.

ii) $S \to ax / bx$; $x \to a / b / \epsilon$

→ $x \to \epsilon$

$S \to ax / bx / a\epsilon / b\epsilon$

$S \to ax / bx / a / b$

$x \to a / b$.

CNF → $V \to VV$, $V \to T$

- $S \to ax$. $S \to bx$. $S \to a$. $S \to b$

$S \to C_a x$ $S \to C_b x$ $S \to C_a$ $S \to C_b$

Optimised CNF: $S \to C_a x / C_b x / a / b$

$x \to a / b$.

iii) $S \to aA / bB$; $A \to Baa / ba$; $B \to bAA / ab$.

→ $S \to aA$. $S \to bB$. $A \to Baa$. $B \to bAA$

$S \to C_a A$ $S \to C_b B$ $A \to BD$ $B \to C_b E$

$B \to ab$ $A \to ba$

$B \to C_a C_b$ $A \to C_b C_a$

Optimised CNF:

$S \to C_a A / C_b B$.

$A \to C_b E / C_b C_a$

$B \to C_b E / C_a C_b$.

iv) $S \to AB / aB$; $A \to a / \epsilon$; $B \to b$.

vi) $S \to AB \mid Cd \mid ET$; $A \to aB \mid d$; $B \to Ba \mid b$; $T \to t$

⇒ $S \to AB$ $S \to ET$ $A \to d$ $B \to b$

$S \to Cd$ $A \to aB$ $B \to Ba$ $T \to t$

$S \to CCd$ $A \to CaB$ $B \to BCa$

→ $S \to Cd$. $S \to ET$ (useless productions).

vii) $S \to ABaC \mid Aa \mid ABa \mid Aa \mid BaC \mid aC \mid Ba \mid a$

$A \to BC \mid b \mid d$

$B \to b$: $C \to d$.

$V \to VT$

→ From a given CFG there's no ϵ, unit & useless productions.

CNF : $V \to VV$; $V \to T$

$S \to ABaC$ $S \to Aa$ $S \to Aa$ $S \to AC$

$S \to ABCaC$ $S \to ACa$ $S \to ACa$ $S \to CaCc$

$D \to CaC$ $S \to ABa$ $S \to BaC$

$S \to ABD$ $S \to ABCa$ $S \to BCaC$

$E \to BD$ $F \to BCa$ $D \to CaC$

$S \to AE$ $S \to AF$ $S \to BD$

$S \to Ba$ $A \to BC$ $A \to d$ $C \to d$

$S \to BCa$ ~~$A \to Cd$~~ ~~$A \to Cd$~~

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

~~$A \to Cb$~~ ~~$B \to Cb$~~

CNF steps : $C_a \to a$ | $S \to ACa$ | $S \to CaCc$ | $A \to b$

$D \to CaC$ $F \to BCa$ $S \to BCa$ $A \to d$

$E \to BD$ $S \to AF$ $S \to a$ $B \to b$

$S \to AE$ $S \to BD$ $A \to BC$ $C \to d$

viii) $S \to ABCD\{\}$;

$A \to \epsilon \mid a$; $B \to bB \mid b$; $C \to cad$.

⇒ $A \to \epsilon$

$S \to ABCD \mid \epsilon BCD$.

$S \to ABCD \mid BCD$; $A \to a$; $B \to bB \mid b$; $C \to cad$

useless production : $S \to ABCD$; $S \to BCD$

useful production : $A \to a$; $B \to bB \mid b$; $C \to cad$

CNF : $V \to VV$; $V \to T$

. $A \to a$; $B \to bB$: $C \to cad$

$B \to C_bB$ $C \to C_cC_aC_d$

. $B \to b$ $E \to C_aC_d$

$C \to C_cE$

CNF Stmts : $A \to a$; $b \to C_b$; $B \to C_bB$

$C \to C_c$; $a \to C_a$; $d \to C_d$; $E \to C_aC_d$

$C \to C_cE$ ✗

CNF Stmt : $A \to a$; $C_b \to b$; $B \to C_bB$; $C_c \to C$

$C_a \to a$; $C_d \to d$; $E \to C_aC_d$; $C \to C_cE$; $B \to b$

② Greibach Normal form (GNF)

Statement : $V \to TV^*$ $V^* \to \bigcup_{i=\infty}^{\infty} V^i$

$V \to T(V^0 \cup V^1 \cup V^2 \cup V^3 \dots)$

$V \to T(\epsilon, V, VV, VVV \dots)$

$V \to T, TV, TVV, TVVV \dots$

Lemma 1 : Statemt : Replacement

Let 'G' = $\langle V, T, P, S \rangle$ bound context Grammar

Lemma 1 problems:

① Convert the following CFG into GNF

$S \to Aa$; $A \to aA/bA/aAS$

⇒ Formal def of CFG: $G < V, T, S, P >$

$V = \{S, A\}$; $T = \{a, b\}$; $S = \{S\}$

$P = \{ S \to Aa \ (1),\ A \to aA \ (2),\ A \to bA \ (3),\ A \to aAS \ (4) \}$ no unit : $V \to V$

There's no ϵ-production.
All the production are useless
∴ Then we can't convert Enter CFG into GNF.

② $S \to CA$; $A \to a$; $C \to aB/a$; $B \to b$

⇒ $G < V, T, S, P >$

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

$P = \{ S \to CA,\ A \to a,\ C \to aB,\ C \to a \}$ no ϵ-production, unit and useless productions.

$S \to CA$ ✗ (where $C \to aB/a$)

$S \to aBA/aA$ ✓

$A \to a$ ✓

$C \to aB/a$

$C \to a$

∴ $V \to T, TV, TVV, TVVV \ldots$

GNF Stmt:

$S \to aBA/aA$

$A \to a$

$C \to aB$

③ $S \to XA/BB$; $B \to b/SB$; $X \to b$; $A \to a$

⇒ there's no ϵ-unit & useless production

$S \to XA$, $S \to BB$, $B \to b$

$S \to bA$, $S \to bB$, $B \to SB$

$B \to XAB / BBB / bB$

• $X \xrightarrow{V} b$
• $A \xrightarrow{V} a$

GNF stmt : $S \to bA \mid bB$

$B \to bAB \mid bBB \mid b$; $X \to b$; $A \to a$

Lemma 2 stmt : Let $G = \langle V, T, P, S \rangle$ be a CFG if 'P' contains of $A \to Aa_1 \mid Aa_2 \mid Aa_3 \mid \dots \mid Aa_n \mid \beta_1 \mid \beta_2 \mid \beta_2 \mid \dots \mid \beta_k$ such that 'β' don't start with 'A' in such cases equivalent GNF statment according to Lemma 2

1. $A \to \beta_1 \mid \beta_2 \mid \beta_3 \dots \mid \beta_k$
2. $A \to \beta_1 Z \mid \beta_2 Z \mid \beta_3 Z \dots \mid \beta_k Z$.
3. $Z \to A$ (a_1, a_2, a_3 values).
4. $Z \to a_1 Z \mid a_2 Z \mid a_3 Z \dots \mid a_n Z$.

} Formulas

① Convert following CFG into it's equivalent GNF stmt

$A \to Aa \mid Ab \mid a \mid b$.

Lemma 2 : $A \to Aa_1 \mid Aa_2 \mid Aa_3 \dots \mid Aa_n \mid \beta_1 \mid \beta_2 \dots \mid \beta_k$

→ The given CFG follows lemma 2 stmt in this $a_1 = a$; $a_2 = b$ & acc to lemma 2 stmt

$\beta_1 = a$; $\beta_2 = b$.

• $A \to \beta_1 \mid \beta_2 \mid \beta_3 \dots \mid \beta_k$

$A \to a \mid b$.
$V \quad T \quad T$

• $A \to \beta_1 Z \mid \beta_2 Z \mid \beta_3 Z \dots \mid \beta_k Z$

$A \to aZ \mid bZ$.
$V \quad TV \quad TV$

• $Z \to A$ (a_1, a_2 values)

$Z \to a \mid b$.
$V \quad T \quad T$

• $Z \to a_1 Z \mid a_2 Z \mid a_3 Z \dots \mid a_n Z$

$Z \to aZ \mid bZ$.
$V \quad TV \quad TV$

(2) convert following CFG into its equivalent GNF stmt

$A_1 \to A_2 A_3$; $A_2 \to A_3 A_1 \mid b$; $A_3 \to A_1 A_2 \mid a$

→ • GNF Stmt : T, TV, TVV, TVVV . . .

$A_1 \to A_2 A_3$ (indirectly follows GNF Stmt) $A_1^{IV} \to b^{T} A_2^{V}$ ~

$A_2 \to A_3 A_1 \mid b$ (indirectly) $A_2^{V} \to b^{T} A_1^{IV} A_1^{V} \mid b^{T}$ ~

$A_3 \to A_1 A_2 \mid a$

$A_2 A_3 A_2 A_1 \mid ba$ (x)

$A_3 \to \underline{A_1} A_2 \mid a$ (apply lemma 1)

$A_3 \to A_2 A_3 A_2 \mid a$.

again apply lemma 1 (?)

$A_3 \to \underline{A_2} A_3 A_2 \mid a$

$A_3 \to A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a$

• Lemma 2 : $A \to Aa_1 \mid Aa_2 \mid Aa_3 \cdots \mid Aa_n \mid \beta_1 \mid \beta_2 \cdots \mid \beta_k$.

$a_1 = A_1 A_3 A_2$; $\beta_i = bA_3A_2 \mid \beta_2 = a$; $A = A_3$

As per the lemma 2 stmt

$A_3 \to bA_3A_2 \mid a$

$A_3 \to bA_3A_2Z \mid aZ$

$Z \to bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$

$Z \to A_1A_3A_2Z$

- $A \to \beta_1 \mid \beta_2 \mid \beta_3 \cdots \mid \beta_k$
- $A \to \beta_1 Z \mid \beta_2 Z \mid \beta_3 Z \cdots \mid \beta_k Z$
- ~~Z~~ → A (a_1, a_2 values)
- $Z \to a_1 Z \mid a_2 Z \mid a_3 Z \cdots \mid a_n Z$

• $A_2 \to A_3 A_1 \mid b$

$A_2 \to bA_3A_2A_1 \mid aA_1 \mid bA_3A_2ZA_1 \mid aZA_1 \mid b$

• $A_1 \to A_2 A_3$

$A_1 \to bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_3A_2ZA_1A_3 \mid aZA_1A_3 \mid bA_3$

• ~~A~~ $Z \to bA_3A_2A_1A_3A_2Z \mid aA_1A_3A_3A_2Z \mid bA_3A_1A_3A_2ZA_1A_3$

$A_3A_2Z \mid aZA_1A_3A_3A_2Z \mid bA_3A_3A_2Z$.

MODULE-04

Push-down automata (PDA)

Automata of a system but its divided in seven tuple

PDA = FA + stack

$= \{Q, \Sigma, q_0, F, \delta\} + \{Z_0, \Upsilon\}$

$PDA = \{Q, \Sigma, q_0, F, \delta, Z_0, \Upsilon\}$.

where Q = no of states
Σ = no of input symbols
q_0 = inital state.
F = final state
δ = Transition fun
Z_0 = stack alphabets (by default top most element)
Υ = stack alphabets apart from Z_0.

Operations of stack : (primitive).

1. PUSH - to insert/store elements into stack
2. POP - To remove/delete elements from the stack.

① → Types of PDA :

1. Deterministic push-down automata (DPDA)
A State must contain only 1 transition on same i/p Symbol.

2. NPDA - A state must be more than 1 transition from a state on same i/p Symbol.

① Construct/design PDA for following languages

$L = \{a^n b^{2n} / n \geq 1\}$

→ $n = 1, 2, 3, 4, 5 \ldots n$.

$L = \{a^1b^2, a^2b^4, a^3b^6, a^4b^8 \ldots\}$

$L = \{abb, aabbbb, aaabbbbbb, aaaabbbbbbbb \ldots\}$

Minimum string : abb

when ever minimum string is appear in stack then we need perform pop operations.

Any string : aabbbb

$\delta(q_0, a, z_0) \Rightarrow \delta(q_0, a z_0)$

$\delta(q_0, a, a) \Rightarrow \delta(q_0, aa)$

$\delta(q_0, b, a) \Rightarrow (q_0, ba)$

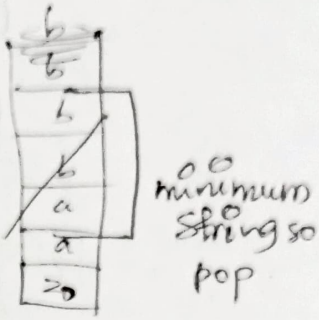
$\delta(q_0, b, b)$

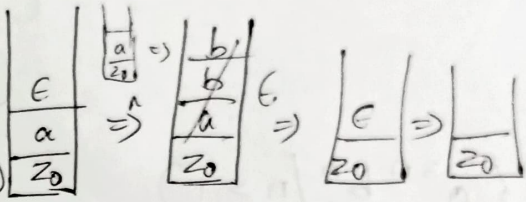
$\delta(q_1, a, z_0)$

$\delta(q_1, b, a) = (q_1, ba)$

$\delta(q_1, b, b) = (q_1, bb)$

$\delta(q_2, z_0) \cdot (q_2, z_0)$



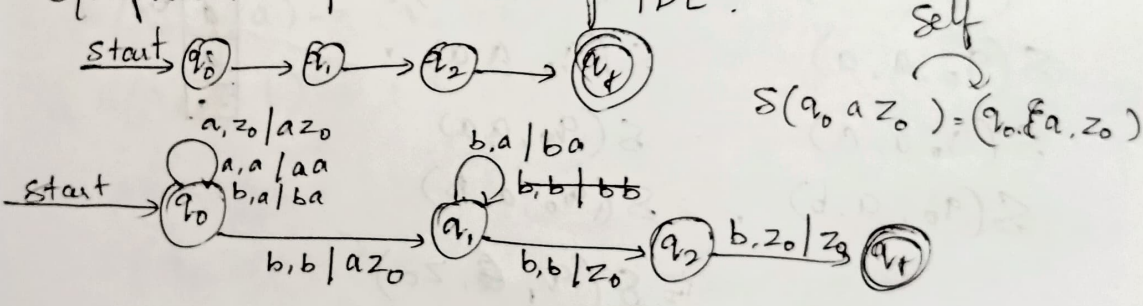


- aabbbb . ϵ

  $(q_2, \underline{\epsilon}, z_0)$ when we reach ϵ there's no

  (q_f, z_0)

Graphical representation of PDE :



Acceptance process aabbbb

$\delta(q_0, aabbbb, z_0) \Rightarrow (q_0, abbbb, a z_0)$

$\vdash (q_0, bbbb, aa z_0)$

$(q_0, bbb, baa z_0)$

$(q_0, bb, bbaa z_0)$

$(q_1, bb, a z_0)$

$\delta(q_1, b, b a z_0)$

$(q_1, \epsilon, bba z_0)$

(q_2, ϵ, z_0)

(q_f, z_0) . (accepted)

aabbb

$\delta(q_0, aabbb, z_0) \vdash (q_0, abbb, az_0)$

$(q_0, abbb, aaz_0)$

$(q_0, bbb, baaz_0)$

$(q_0, bb, bbaaz_0)$ (ϵ)

$(q_0, b, bbbaaz_0)$

~~$(q_0, \epsilon, bbbaaz_0$~~

(q_0, b, az_0)

(q_0, ϵ, baz_0) (rejected)

(2) $L = \{a^{3n} b^n / n \geq 1\}$

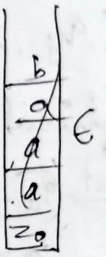
$\Rightarrow$ $n = 1, 2, 3, 4, 5 \ldots$

$L = \{a^3b, a^6b^2, a^9b^3, a^{12}b^4 \ldots\}$

$L = \{aaab, aaaaaabb, aaaaaaaaabbb \ldots\}$

minimum string : aaab.

any string : ~~aaaaaabb~~. aaab



$\delta(q_0\ a\ z_0) = \delta(q_0\ a\ z_0)$

$\delta(q_0, a, a) = \delta(q_0, a\ a)$

$\delta(q_0\ a, a) = \delta(q_0, aa)$

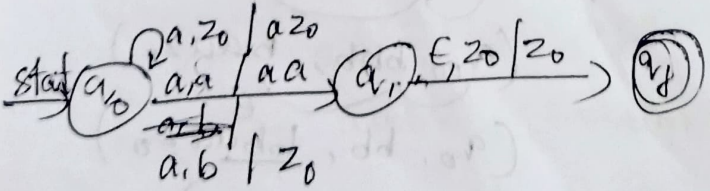
$\delta(q_0, a, b) = \delta(q_0, a, b)$

$\vdash \delta(q_1, \epsilon, z_0)$

aaab.ϵ

$(q_1, \epsilon\ z_0) \Rightarrow (q_f\ z_0)$

Graphical representation :



Acceptance process

$\delta(q_0, aaab, z_0)$

$\delta(q_0, aab, az_0)$

$\delta(q_0, ab, aaz_0)$

$\delta(q_0, b, aaaz_0)$

$\delta(q_0, baaaz_0)$

$\delta(q_0, \epsilon, baaaz_0) \xrightarrow{\epsilon} (q_1, \epsilon, z_0)$

$= (q_f, z_0)$

① Design PDA for even length palindrome

② Design PDA for the following $ww^R / wz(a+b)^+$

③ ~~Design PDA for equal no. of a's & b's.~~

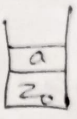
⇒ $L = \{a, b, aa, ab, ba, bb, aaa, aba \ldots bbb \ldots\}$

$w = aba, \quad w^R = aba$

$ww^R = abaaba$

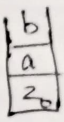
aba aba . ϵ

$\delta(q_0, a, z_0) \xrightarrow{w\,w^R} (q_0, a, z_0)$



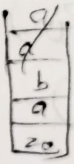
aba . aba . ϵ

$\delta(q_0, b, a) \rightarrow (q_0, ba)$



$\delta(q_0, a, b) \rightarrow (q_0, ab)$

aba . aba



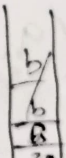
ϵ equal no of a's

$\delta(q_0, a, a) \rightarrow (q_0, aa)$

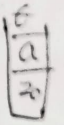
$\rightarrow (q_1, ba)$

$\delta(q_1, b, b) \rightarrow (q_1, bb)$

$\rightarrow (q_2, az_0)$



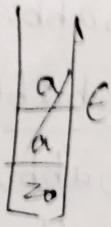
$\epsilon \rightarrow$



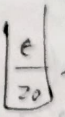
$\rightarrow$

$\delta(q_2, a, a) \rightarrow (q_2, aa)$

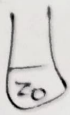
$\rightarrow (q_3, z_0)$



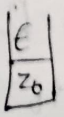
$\epsilon \rightarrow$



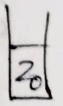
$\rightarrow$



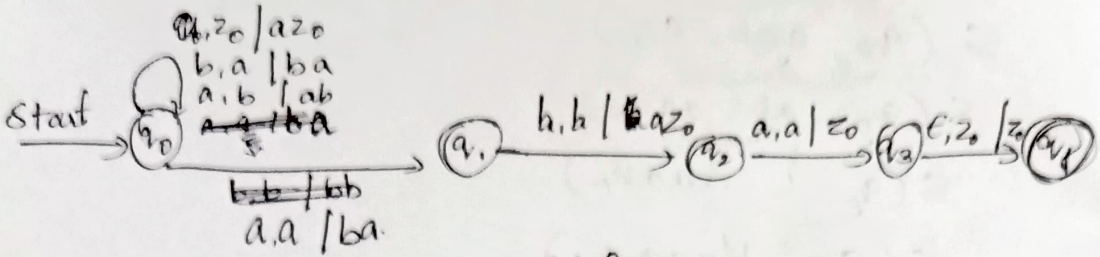
$\delta(q_3, \epsilon, z_0) \rightarrow (q_f, z_0) \rightarrow$



$\rightarrow$



Graphical Representation:



Acceptance process: String: abba.

$\delta(q_0, abba, z_0) \vdash \delta(q_0, bba, az_0)$

$\delta(q_0, ba, baz_0)$

$\delta(q_0, a, bbaz_0)$

$\delta(q_1, a, abbaz_0)$

$\delta(q_1, a, az_0)$

$\delta(q_1, \epsilon, z_0)$

$\delta(q_2, \epsilon, z_0) \rightarrow \delta(q_f, z_0)$

(accepted).

String: aababbabb

$\delta(q_0, aababbabb, z_0) \vdash \delta(q_0, ababbabb, az_0)$

$\delta(q_0, babbabb, aaz_0)$

$\delta(q_1, babbabb, z_0)$

$\delta(q_1, abbabb, bz_0)$

$\delta(q_1, bbabb, baz_0)$

$\delta(q_1, babb, bbaz_0)$

$\delta(q_2, abb, az_0)$

$\delta(q_2, bb, aaz_0)$

$\delta(q_3, b, bz_0)$

$\delta(q_3, \epsilon, bbz_0)$

$\delta(q_4, \epsilon, z_0)$

$\Rightarrow \delta(q_f, z_0)$

(accepted)

String: ababbabb.

$\delta(q_0, ababbabb, z_0)$

$\delta(q_0, babbabb, az_0)$

$\delta(q_0, abbabb, baz_0)$

$\delta(q_0, babb, abaz_0)$

$\delta(q_0, abb, babaz_0)$

$\delta(q_0, bb, ababaz_0)$

$\delta(q_0, b, bababaz_0)$

$\delta(q_0, \epsilon, bbababaz_0)$

$\delta(q_1, \epsilon, ababaz_0)$

(rejected).

(62) Design PDA for odd length palindrome.

(63) Design PDA for the following lang

$$L = \{ wCw^R / w \in (a+b)^+ \}.$$

→ $L = \{ a, b, aa, ab, ba, bb, aaa, aba \ldots bbb \ldots \}$

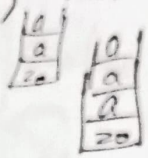
Any string: $w = aaa$, $w^R = aaa$

$wCw^R = aaaCaaa \cdot \epsilon$; $aaaCaaa$

$\delta(q_0, a, Z_0) \rightarrow (q_0, aZ_0)$

$\delta(q_0, a, a) \rightarrow (q_0, aa)$

$\delta(q_0, a, a) \rightarrow (q_0, aa)$

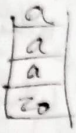


NOTE: ~~$\delta(q_0$~~ Skipping rule of C

$aaaCaaa \cdot \epsilon$ same

↑ Here no need PUSH the C into stack contains then we will use skipping rule formula. and switch to new state

$\delta(q_0, C, a) \rightarrow (q_1, aa)$

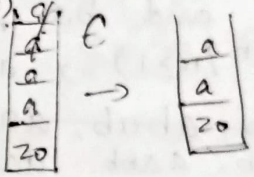


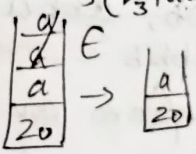
$aaaCaaa$ $\delta(q_1, a, a) \rightarrow (q_1, aa)$

$\rightarrow (q_2, aa)$

$\delta(q_2, a, a) \rightarrow (q_3, aa)$

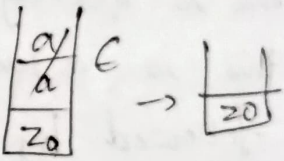
$\rightarrow (q_3, aa)$





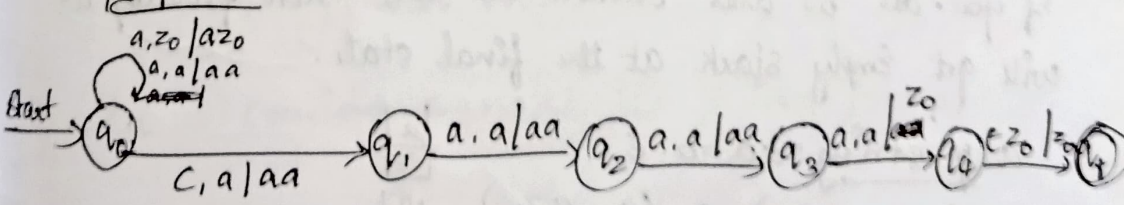
$\delta(q_3, a, a) \rightarrow (q_3, aa)$

$\rightarrow (q_4, Z_0)$



$\delta(q_4, \epsilon, Z_0) \rightarrow (q_f, Z_0)$.

Graphical



Acceptance process : abacaba

$\delta(q_0, abacaba, z_0) \rightarrow \delta(q_0, bacaba, az_0)$

$\delta(q_0, acaba, baz_0)$

$\delta(q_0, caba, abaz_0)$

$\delta(q_1, ba, abaz_0)$

$\delta(q_2, ba, baz_0)$

$\delta(q_2, a, baz_0)$

$\delta(q_3, a, az_0)$

$\delta(q_3, \epsilon, aaz_0)$

$\delta(q_4, \epsilon, z_0) \rightarrow \delta(q_f, z_0)$

accepted.

(4) (a) Design PDA for following lang $L = \{a^n b^n / n \geq 1\}$

Design PDA which accepts all strings of a's and b's in which every string contains equal no's a's and b's. $n = 1, 2, 3 \ldots$

$L = \{a^n b^n / n \geq 1\} \Rightarrow a^1 b^1 = ab ; a^2 b^2 = aabb \ldots$

$L = \{ab, abab, aabb, abba, baab, ababab\}$

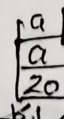
$L = \{ab, aabb, aaabbb \ldots\}$

This is quitely different from an ealier method. this is a lang in which equal no of a's are followed by equal no of b's.

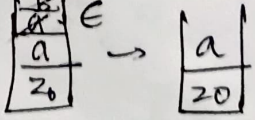
The logic for this PDA can be applied as first we will push all a's into the stack then on reading every single 'b' each is poped from the stack, if you read all b's and remove all a's then finally we will get Empty stack at the final state.

any string : aabb. ϵ

$\delta(q_0, a, z_0) \rightarrow (q_0, az_0)$



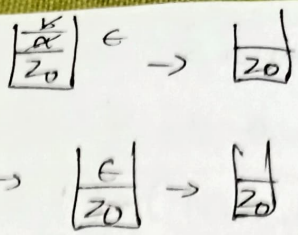
$\delta(q_0, a, a) \rightarrow (q_0, aa)$



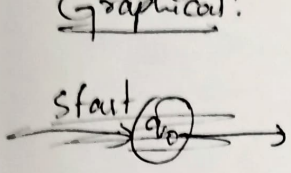
$\delta(q_0, b, a) \rightarrow (q_0, \epsilon a)$

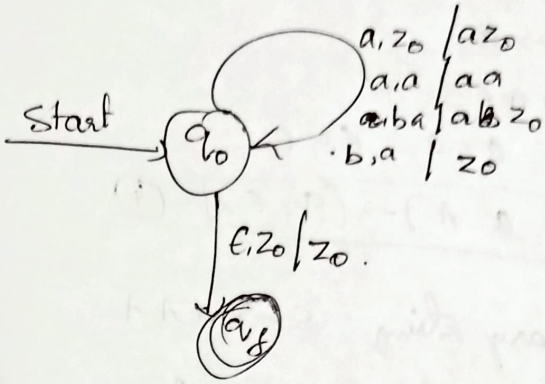
$\rightarrow (q_0, az_0)$

$\delta(q_0, b, a) \to (q_0, ba)$
(q_0, z_0)

$\delta(q_0, \epsilon, z_0) \to (q_f, z_0)$ → 

Graphical:





Acceptance : aabb

$\delta(q_0, aabb, z_0) \vdash \delta(q_0, abb, az_0)$
$\delta(q_0, bb, aaz_0)$
$\delta(q_0, b, baaz_0)$
$\delta(q_0, b, az_0)$
$\delta(q_0, \epsilon, baz_0)$
$\delta(q_f, \epsilon, z_0) \to \delta(q_f, z_0)$

Construct PDA from Given CFG:

CFG must be in GNF format [T, TV, TVV, TVVV---]

Formula : $A \to a\alpha$ (or) $A \to a\epsilon$ $(\alpha = \epsilon)$.

$\boxed{\delta(q, a, A) \to (q, \alpha)}$

① $S \to aAA$; $A \to aS \mid bS \mid a$

→ $S \to aAA$
$A \to aS$
$A \to bS$
$A \to a$
} are in GNF format (T, TV, TVV--)

Let us consider $S \to aAA$ it's in GNF format
(V T V V)

$A \to a\alpha$

So, $A = S$, $a = a$, $\alpha = AA$.

$\boxed{\delta(q, a, S) \to \delta(q, AA)}$ — ①

• $A \to aS$

$A = A,\ a = a,\ S = S$

$\delta(q, a, A) \to (q, S)$ — ②

• $A \to bS$

$A = A,\ a = b,\ S = S$

$\delta(q, b, A) \to (q, S)$ ↳ ③

• $A \to a$

$A \to a\epsilon$

$A = A,\ a = a,\ x = \epsilon$

$\delta(q, a, A) \to (q, \epsilon)$ — ④

Primary string $S \to aAA$

• $S \to aAA$
$\to aaA$ $(A \to a)$
$\to aaa$ $(A \to a)$ } ①

• $S \to aAA$
$\to aSA$ $(A \to aS)$
$\to aaAAA$ $(S \to aAA)$
$\to aaaAA$ $(A \to a)$
$\to aaaaA$ $(A \to a)$
$\to aaaaa$ } ②

• $S \to aAA$
$S \to abSA$ $(A \to bS)$
$\to abaAAA$ $(S \to aAA)$
$\to abaaAA$
$\to abaaaA$
$\to abaaaa$ } ③

Any string. (Top/starting symbol)

① $\Rightarrow \delta(q, aaa, S)$ — stack: S

$\to \delta(q, a, S) \to (q, AA)$

$\vdash (q, aa, AA)$ — stack: A, A

② $\to \delta(q, a, A) \to (q, S)$

$\vdash (q, a, SA)$ — stack: S, A

① $\to \delta(q, a, S) \to (q, AA)$

$\vdash (q, \epsilon, AAA)$ — stack: A, A, A

$\vdash (q_f, AAA) \to$ Accepted.

→ Explain about instantaneous discription:

- Instantaneous discription is nothing but acceptance process
- ID consist of triple format i.e state (q), i/p string (w), by default stack top most symbol (Z_0)
  i.e (q, w, Z_0).

MODULE - 05 TURING MACHINE

Turing Machine: (TM)

- It's also known as universal turing machine (UTM)
- It's a simple mathematical model of a modern digital computer.
- TM is a abstract computing machine because, it's has ability to improve all the features that ~~RE~~ RL, CFL, recursive ~~R~~enumerable lang, PDA and FA due to this reason TM is also known as universal TM
- ~~Alert~~ Alan turing is a father of TM, which has computing capability of general purpose computer.

→ Features of TM:

- It has unlimited memory capability when compare to previous machine
- The model has a capacity which the input left-right or right-left depending on the problm. requirement
- TM can produce certain output based on the certain input.

→ Formal definition of TM:

- It can be mathematically defined by 7 Tuples

{ Q, Σ, q_0, F, δ } { B, Γ }
(blank) ↑ ↑ i/p tape symbol

: {Q, Σ, q_0, F, δ, B, Γ}.

(12m)

** → Explain variants of TM: (Types)

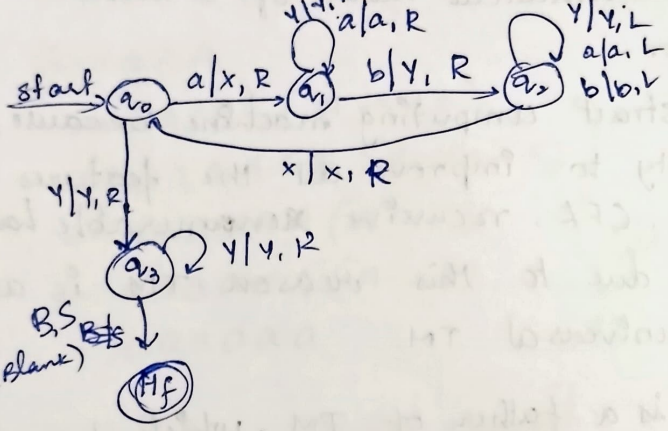
aaabbb

*** → Construct TM for following Lang: $L = \{a^n b^n / n \geq 1\}$

⇒ $L = \{a^n b^n / n \geq 1\}$ n = 1, 2, 3, 4 ....

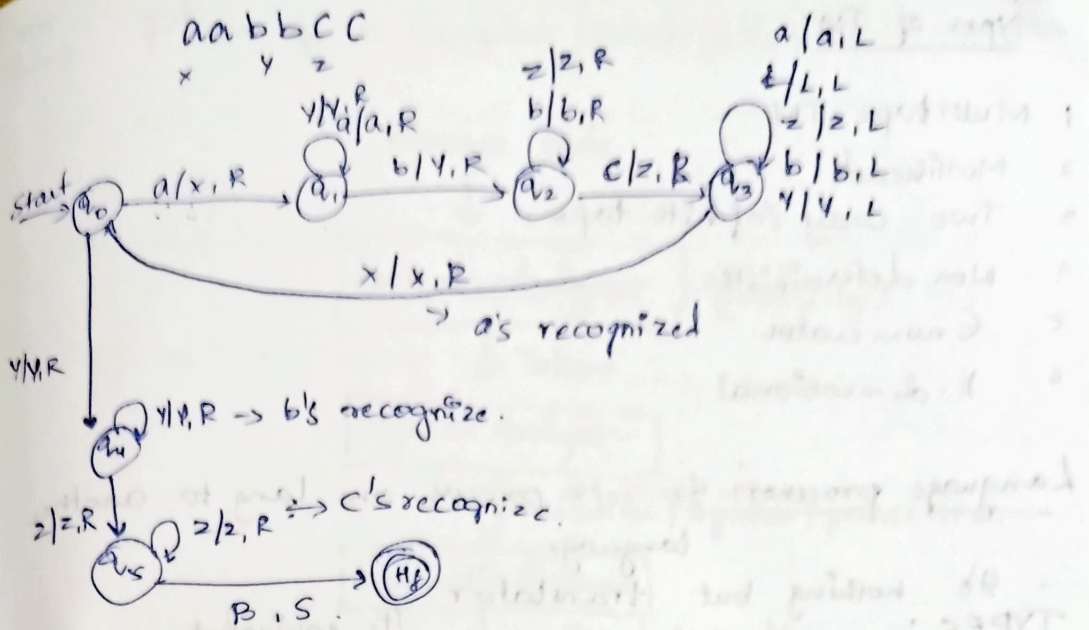
$L = \{ab, aabb, aaabbb ....\}$

x y
aaabbb



STT of TM

Q/Σ	a	b	X	Y	B
q_0	(q_1, X, R)	—	-	(q_3, Y, R)	
q_1	(q_1, a, R)	(q_2, Y, R)	-	(q_1, Y, R)	
q_2	(q_2, a, L)	(q_3, b, L)	(q_0, X, R)	(q_2, Y, L)	
q_3	-	-	-	(q_3, Y, L)	(H_f, S)
H_f	-	-	-	-	-



STT

Q \ Σ	a	b	C	x	Y	z	B
q_0	(q_1, X, R)	–	–	–	(q_4, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, R)	–	–	(q_1, Y, R)	–	–
q_2	–	(q_2, b, R)	(q_3, z, R)	–	–	(q_2, z, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	(q_3, z, L)	(q_0, x, R)	(q_3, Y, L)	(q_3, z, L)	–
q_4	–	–	–	–	(q_4, Y, R)	(q_5, z, R)	–
q_5	–	–	–	–	–	(q_5, z, R)	(H_f, S) → accepted
H_f							

Types of TM:

1. Multitape TM
2. Multihead
3. Two-way infinite tape.
4. Non deterministic
5. Enumerator
6. k-dimensional

Language processor: It will process one lang to another language.

- It's nothing but translator.

TYPES:

1. **Complier:** used to convert high level lang into low level lang; It's equivalent

```
main(){        to    011
- - -}               101
                     111
```

- It's a translator.

2. **Interpreter:** It's a translator which is used to convert HLL interpreter to LL

when compared to complier it will go line by line of the source program.

Eg: java, python.

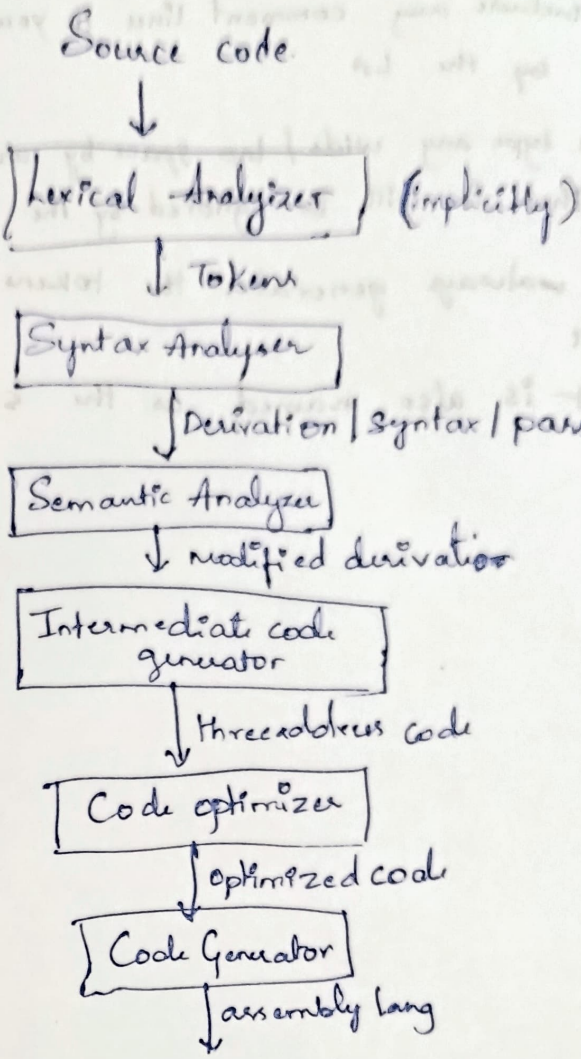
3. **Assembler:** Translator used to convert HLL to LLL

Eg: ADD, MUL, DIV, ST, LD.

4. **Hybrid complier:** Translator used to convert HLL to LLL

Eg: java bcoz, half features belongs to complier & half to interpreter features.

Phases of a Compiler / Construction of a compiler.



Operators
ARLBASIC

Token: It's valid sequence of character which are given by lexeme in a programming language
Eg: Keywords (32), Constants, identifiers, numbers, Operators., punctation symbols all these are possible tokens to identified.

Lexemes: It's a sequence of characters in the source prgm that matches the pattern for a token & it's identified by the lexical analyser as an instance of that token.

$C = a + b * 5$

Lexemes	Tokens
c	identifiers
=	assignment op
a	identifier
+	arithmentic
b	identifier
*	arithmitic
5	Constant

Pattern: It describes a rule that must be matched by sequence of characters (lexemes) to form a token it can be defined by RE or grammar rules.
In the case a keyword as a token the pattern is just the sequence of character that form the keyword.
Eg: $c = a + b * 5$

Responsibilities of Lexical Analyser:

1. If you include any comment lines in your code that will be ignored by the LA
2. If you type any wide / lap space by unknowing in your code then it will be ignored by the LA
3. LA always generates the tokens based on source code
4. LA is also named as the scanner.