Closure property of regular languages.

When we perform certain operation on regular languages, then produced lang is also regular.

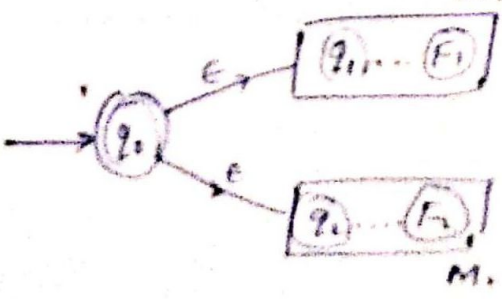
Theorem:-

If L_1 and L_2 are regular lang, then so are $L_1 \cup L_2$, $L_1 . L_2$, $\overline{L_1}$, $L_1 \cap L_2$ & L_1^*

Then we say that family of regular lang is closed under union, concatenation, complement, intersection & KLEENE closure

Note:- these operation work on strings of a lang., not on RE of lang.

Union operation



$M = \{Q, q_0, \Sigma, \delta, F\}$

$M_1 = \{Q_1, q_1, \Sigma_1, \delta_1, F_1\}$

$M_2 = \{Q_2, q_2, \Sigma_2, \delta_2, F_2\}$

$M = \{Q_1 \cup Q_2 \cup \{q_0\}, q_0, \Sigma_1 \cup \Sigma_2, \delta', F_1 \cup F_2\}$

let $L = L_1 \cup L_2$

& x be a string : $x \in L$.

T.P $L(M) = L_1 \cup L_2$

then $\delta^*(q_0, x) \in F_1 \cup F_2$ (on i/p x, M stop @ F_1 or F_2)

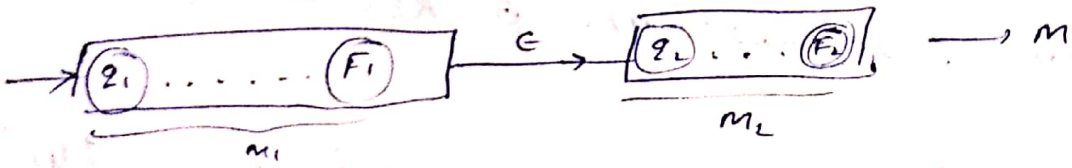
$\Rightarrow \delta^*(q_0, x) \in F_1$ OR $\delta^*(q_1, x) \in F_2$

$\Rightarrow \delta^*(q_1, x) \in F_1$ OR $\delta^*(q_2, x) \in F_2$ ($\because q_1 \rightarrow q_1$ & $q_0 \rightarrow q_2$ on null moves $\because q_0 \equiv q_1$)

$\Rightarrow x \in L_1$ OR $x \in L_2$

$\Rightarrow x \in L_1 \cup L_2$

Concatenation



T.P $L(M) = L_1 \cdot L_2$?

Compliment

If L is a regular lang.

$M(L) = \{Q, q_0, \Sigma, \delta, F\}$

$M(\bar{L}) = \{Q, q_0, \Sigma, \delta, Q - F\}$

Intersection

$L_1 \cap L_2 \equiv \overline{(\bar{L_1} \cup \bar{L_2})}$ since $\cup$ & compl. are closed, $\cap$ is also closed.

Ex: $L_1 = \{w \mid w$ starts with a over $\{a,b\}\}$

$L_2 = \{w \mid n_a(w) = n_b(w)$ over $\{a,b\}\}$ → not regular as we can't count.

$L_1 \cap L_2 = \{w \mid w$ starts with a and $n_a(w) = n_b(w)\}$

$= L_2$

Reversal

L^R

Homomorphism

Suppose Σ & Γ are alphabets, then a fn $\boxed{h: \Sigma \to \Gamma^*}$

is called homomorphism.

i.e., a homomorphism is a substitution in which a single letter is replaced with a string of

i.e., if $\omega = a_1 a_2 \ldots a_n$, then ($a_i$ is a symbol)

$$h(\omega) = h(a_1).h(a_2)\ldots h(a_n)$$

$$\therefore h(L) = \{h(\omega) \mid \omega \in L\} \Rightarrow \text{homomorphic image of } L$$

eg: $\Sigma = \{a, b\}$ $\Gamma = \{0, 1, 2\}$

let $h(a) = 01$

$h(b) = 112$

eg: let $RE(L_1) = a^* bb$

$\therefore$ homomorphism of L_1 $\rightarrow$ $(01)^*.112.112$

i.e. $R \in (h(L_1))$ $\rightarrow$ $h(a^*).h(b).h(b)$

i.e. homomorphic image of a $R \in RL$ a Regular language

Finite automata to R.E

→ many methods.

→ Arden method not possible in ε NFA is there

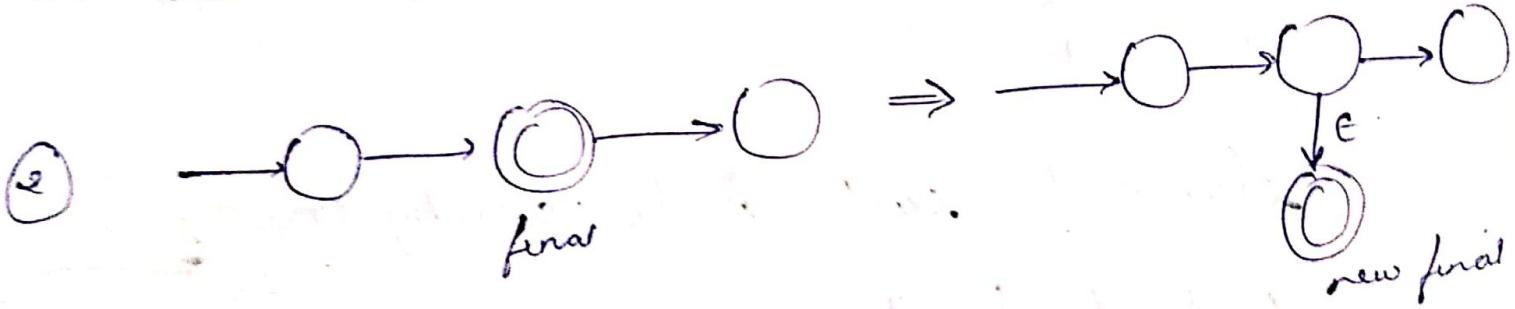
→ ∴ generally use state elimination method

Rules

(1)

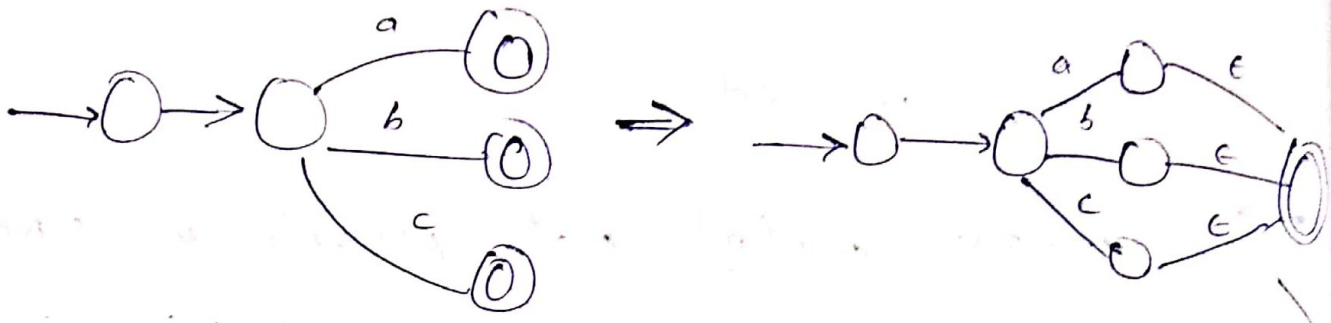
if initial state has incoming edge, then make an new initial state and give ε-transition to old initial state

(2)



if final state has outgoing transition :

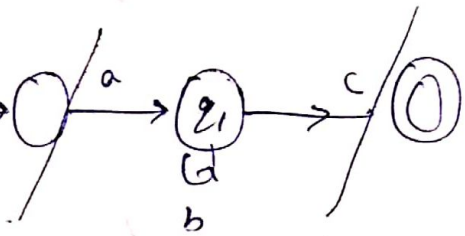
(3) if there are multiple finite state



make it - ϵ transition to common final state.

We can't use ardest theorem here as it has ϵ transition

Method.

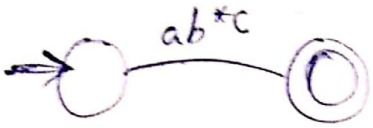
eg: 

→ all rules applied

eliminate each node one by one except initial & final state.

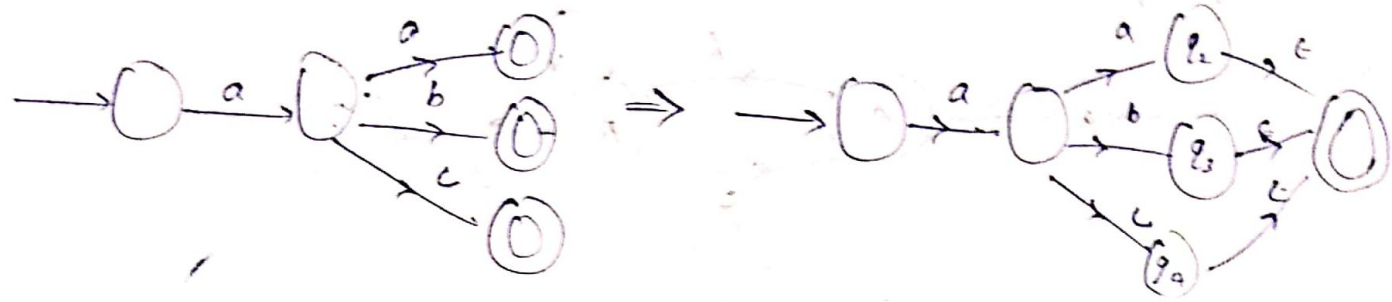
⊙ what RE will you miss if you eliminate a state from F.A

eg: remove q_1 ⇒ missing $a \cdot b^* c$

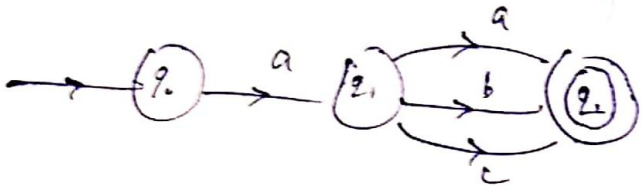


$RE = ab^*c$

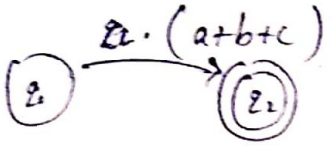
eg:



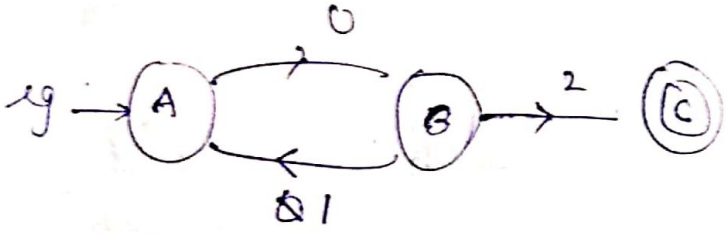
eliminate one by 1 (q_2 ⇒ miss a
q_3 ⇒ b
q_4 ⇒ c)



→ eliminate q_1 ⇒

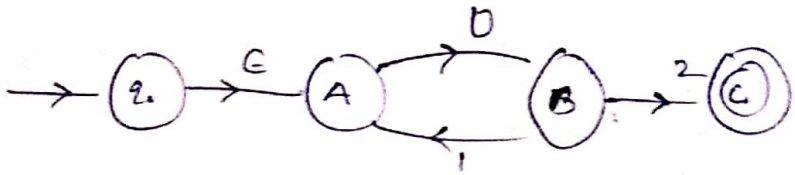


eg:



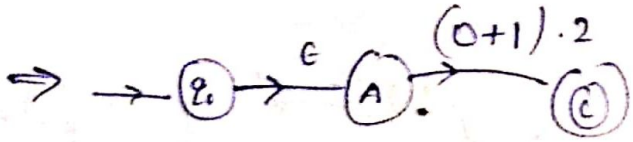
multiple transition ⇒ +
single ⇒ '.'

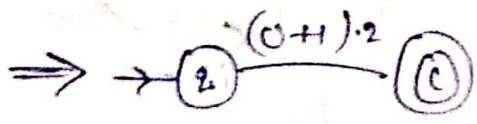
⇒



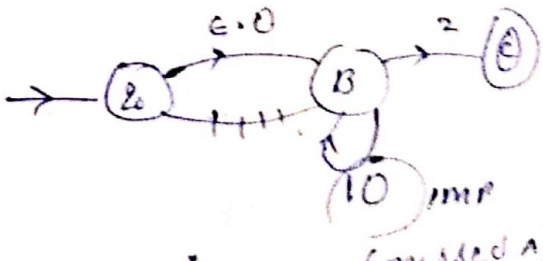
eliminate A;

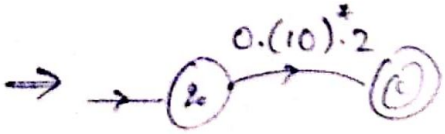
wrong!



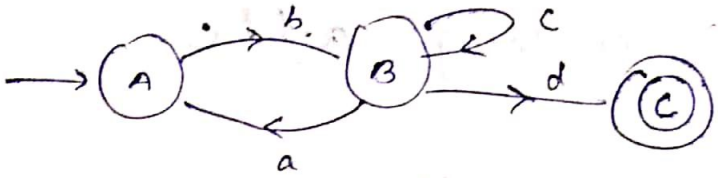


≡

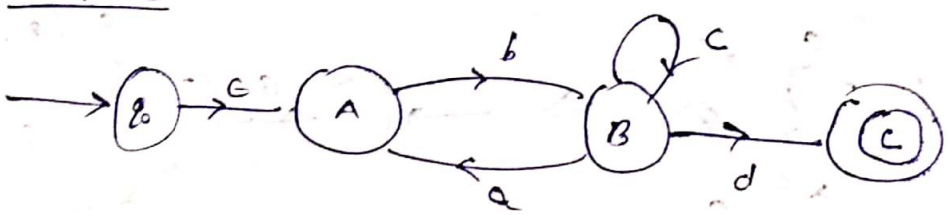




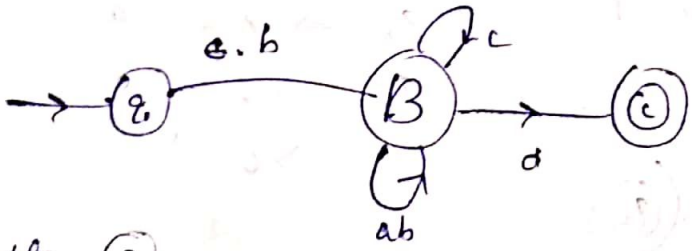
eg



Step (1)

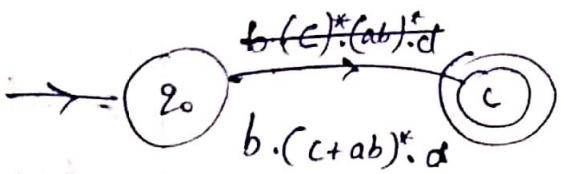


Step (2) Eliminate A



Step (3)

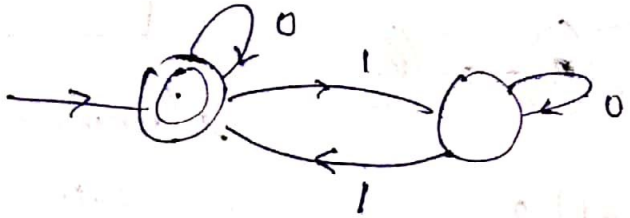
Eliminate B



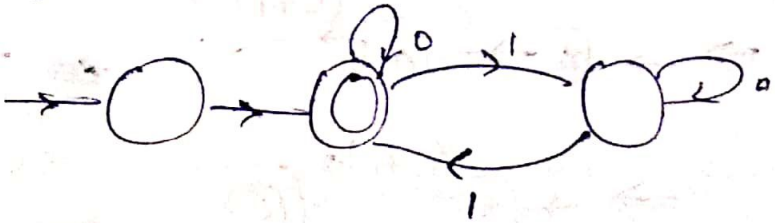
not $c^*.ab^*$ or they may differ

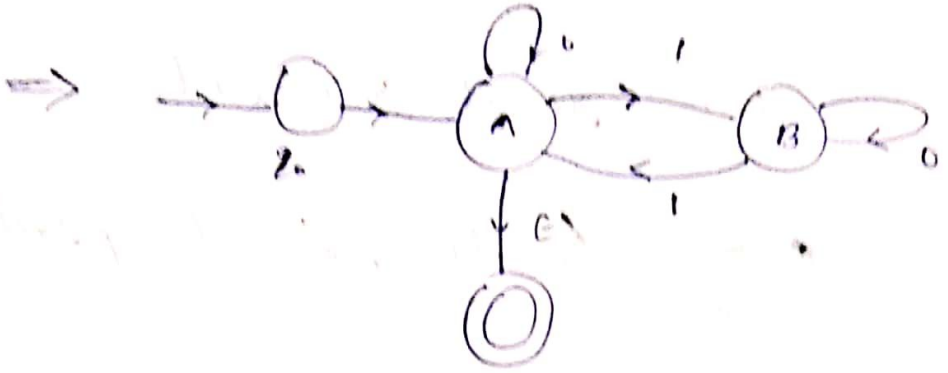
$a^*.b^* = (a+b)^*$ wrong

eg:

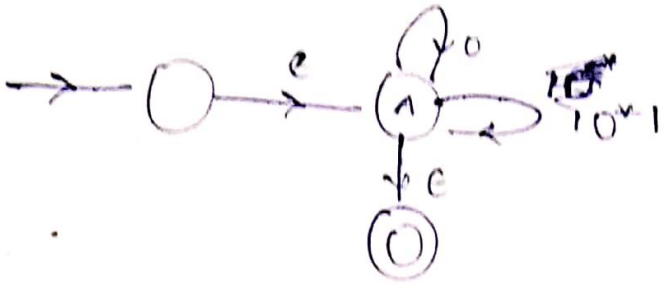


Step (1) →





eliminate B



loop → 0*
multiple → *

eliminate A

RE - $(0 + 10^*1)^*$

Regular Grammars (RG)

⊙ RG are associated with RL & ∀ RL ∃ RG

RG can be used to represent FA

∴ RG ⟷ RE ⟷ FA ⟷ RL

⊙ A Grammar is defined by quadruple

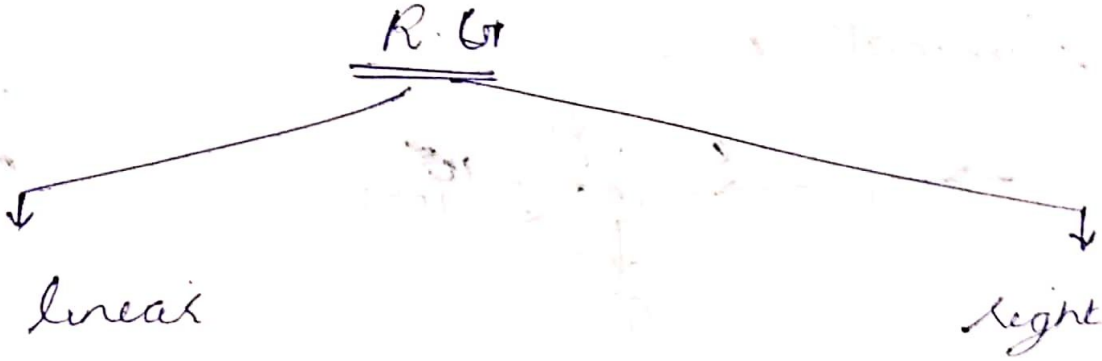
G - (V, T, S, P)

V - finite set of variable (non-terminal)

T - finite set of terminal (Σ)

S = start variable

P = finite set of production



$A \longrightarrow Ba$

$A \longrightarrow b$

$A \longrightarrow aB$

$A \longrightarrow b$

$A, B \in V$

$a, b \in T$

Note: If a RG is in left linear then every production must be in left linear form.

→ LHS only one var. must appear

→ RHS atmost one var. should appear

↳ LHS → RHS
must side

Production as operation

for klun closure → left recursive or right recursive

self loop

→ A → aA or A → Aa ↔ a*

for concatination → ab (no need for '.') ↔ a.b

for union (or) → a/b ↔ (a+b)

eg
S → aA
A → Bb
B → c

} linear but not RG

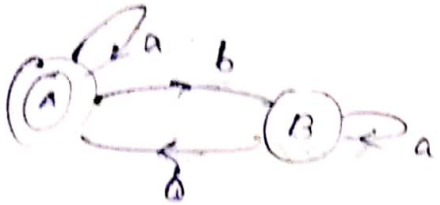
as S in Left linear
A in Right linear

Transition for production rule.

$\delta(q_1, a) \longrightarrow q_1$ ⇒ $q_1 \longrightarrow a q_1$ or $q_1 \longrightarrow q_1 a$

$\delta(q_1, b) \longrightarrow q_2 \rightarrow q_1 \longrightarrow bq_2$ or $q_1 \longrightarrow q_2 b$

q_f ∈ final state $\longrightarrow$ $q_f \leftrightarrow \in$

eg 

note: all transitio
u need
production

Let RG be left linear

$\delta(A, a) \rightarrow A \longrightarrow A \longrightarrow Aa$

$\delta(A, b) \rightarrow B \rightarrow A \longrightarrow Bb$

$\delta(B, a) \rightarrow B \rightarrow B \longrightarrow Ba$

$\delta(B, b) \longrightarrow A \rightarrow B \longrightarrow Bb$

$A \in q_f \Rightarrow A \longrightarrow \in$

Context Free Grammar (CFG)

A grammar $G = (V, T, S, P)$ is said to be CFG if all productions in P have the form $A \rightarrow \alpha$

where $A \in V$ & $\alpha \in (V \cup T)^*$

→ A lang. L is context free iff ∃ a CFG G_1 such that $L = L(G_1)$

- Reg ∈ CFG not vice versa.

Q. Write CFG for $L = \{a^n \mid n \geq 0\}$

$\Rightarrow$ $S \to \epsilon \mid aS$

(S: start)

$S \to aS$

$S \to a(aS)$

$S \to aaaS$

$S \to aaa\epsilon \Rightarrow aaa \in L$

Q. $L = \{a^n \mid n \geq 1\}$

$S \to a \mid aS$

if $S \to aS$ it goes on

Q. $L = \{w \mid |w| = 2\}$

$S \to aa \mid bb \mid ab \mid ba \to (a+b).(a+b)$

$A \to a \mid b$

$S \to AA$

$\to S \to a \mid b$

Q) $L = \{ w \mid |w| \geq 2 \}$

$S \longrightarrow \underbrace{(a+b)}_{A} \cdot \underbrace{(a+b)}_{A} \cdot \underbrace{(a+b)^*}_{B}$

$S \longrightarrow AAB$

$A \longrightarrow a|b.$

$B \longrightarrow \epsilon | aB | bB$ (klean closure)

→ CFG not RG

CFG → RG.

$S \longrightarrow a|b$

$S \longrightarrow \underbrace{(a|b)}_{(a+b)} \underbrace{(a|b)}_{(a+b)} \underbrace{(a|b)}_{(a+b)^*} B \longrightarrow (aa|ab|ba|bb)B \longrightarrow (aaB|abB|baB|bbB)$

RE

RG

linear

right linear

RG must be linear.

Q) $L = \{ w \underbrace{c}_{\text{constant}} w^R \mid w \in (a,b)^* \}$ ⟶ not reg.

But what if we have a stack.

∴ it is CFL (push down automata)

$S \longrightarrow c \mid asa \mid bsb$

$S \longrightarrow asa$

$\longrightarrow aasaa$

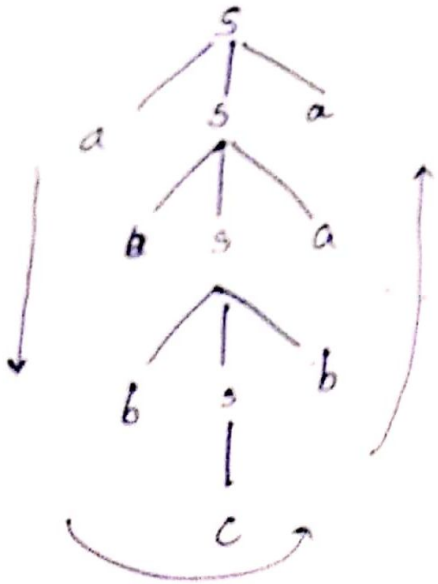
$\longrightarrow aabsbaa$

$\longrightarrow aabcbaa$

} derivation

derivation Tree

internal nodes as states, leaf as terminals



CFG must have der. tree.

Q. $L_1 = \{a^n b^n c^m \mid n, m \geq 1\}$

Q) $L_2 = \{ab^n \mid n \geq 1\} \cup \{a^m b^m \mid m \geq 0\}$

Q $L_3 = \{a^n b^{2n} \mid n \geq 1\}$

Q $L_4 = \{a^n b^m \mid n = m\}$

Q1. $L_1 \Rightarrow aAbcB$ $\quad L_1 \longrightarrow S$

$A \longrightarrow \epsilon / aAb$

$B \longrightarrow c / c$

Q2. $L_2 \longrightarrow abB \mid A$ $\quad L_2 \longrightarrow S$

$B \longrightarrow \epsilon / bB$

$A \longrightarrow aAb / \epsilon$

Q3. $L_3 \longrightarrow aAbb$ $\quad L_3 \longrightarrow S$

$A \longrightarrow aAbb / \epsilon$

Q4. $L_4 \longrightarrow aAb / \epsilon$ $\quad L_4 \longrightarrow S$

$A \longrightarrow aAb / \epsilon$

Push down Automata (PDA) = (FA + stack)

- deterministic PDA
  - lang: det.. cont. free (DCFL)
- non deterministic PDA
  - lang: non det. cont. free (NDCFL)

Its defined by:

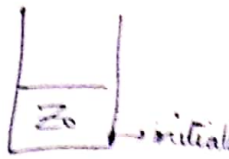
$$M = (Q, \Sigma, \delta, Q_0, Q_f, Z_0, \Gamma)$$

- Q = Set of states
- Σ = Input symbol
- δ = transition fn.
- Q_f = final states
- Q_0 = initial state
- Z_0 = bottom of stack
- Γ = stack alphabet

DPDA

not included in DFA

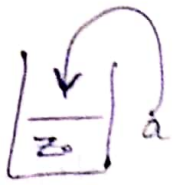
$\delta: Q \times \{\Sigma \cup \epsilon\} \times \Gamma \longrightarrow Q \times \Gamma^*$ (string)

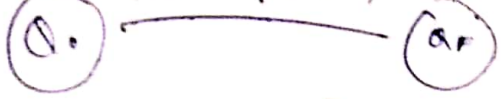


$\therefore \delta$: (curr. state, i/p symbol, symbol on stack top)

$\longrightarrow$ next state, resultant symbol on stack

eg: $(q_0, a, Z_0) \longrightarrow (q, \underbrace{aZ_0}_{\text{push } a})$.



$\therefore$ 

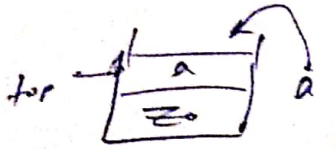
NPDA

$\delta: Q \times (\Sigma \cup \epsilon) \times T \longrightarrow 2^{(Q \times \Gamma^*)}$

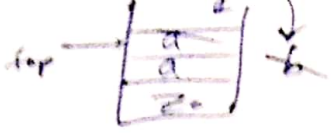
eg: $L = \{a^n b^n \mid n \geq 1\}$

aabb

$(q_1, Z_0 / \underbrace{aZ_0}_{\text{push a}})$



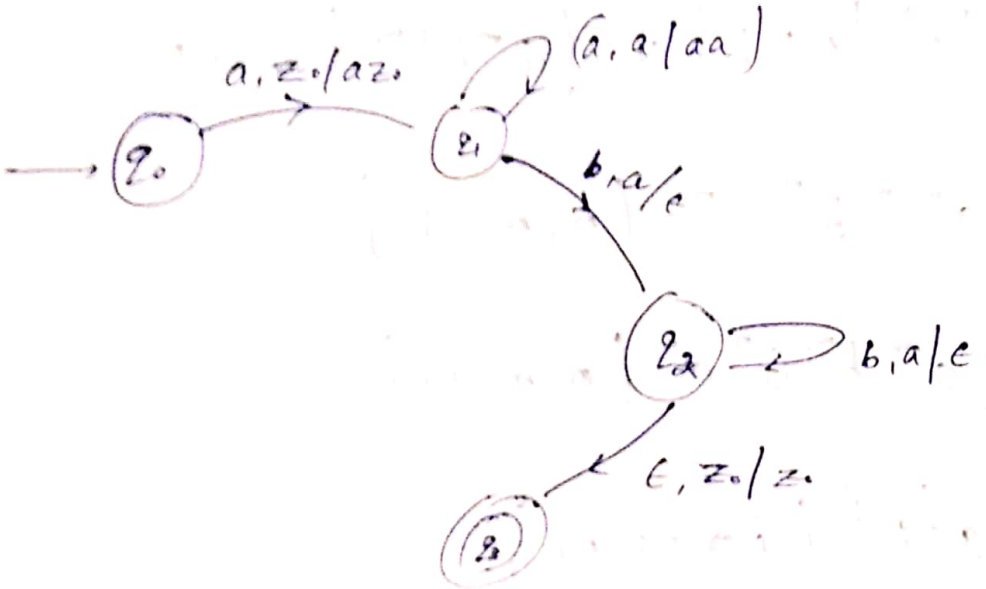
$(q_1, a / \underbrace{aa}_{\text{push a}})$



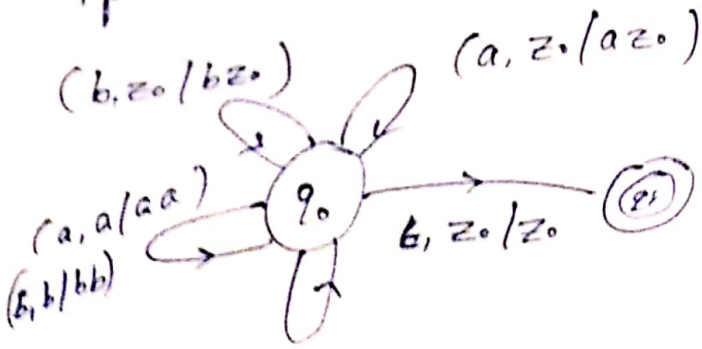
$\longrightarrow$ $(b, a \mid \underline{\epsilon})$

pop a (top)

$\longrightarrow$ $(b, a \mid \epsilon)$



Now if it was like



it accepts ~~$(a+b)^*$~~ $n_a(w) = n_b(w)$

$L_1 \leq_m^P L_2$

$L_1 \leq_m L_2$

(2) $L_2 = \{ a^n b^m \mid n \neq m \}$

(3) $L_3 = \{ w c w^R \mid w \in (a,b)^* \}$

(4) $L_4 = \{ a^n b^n \mid n \geq 1 \} \cup \{ a^n b^{2m} \mid m \geq 1 \}$

5) $L_5 = \{ a^{n+m} b^n c^m \mid n, m \geq 1 \}$

6) $L_6 = \{ a^m b^m c^n d^n \mid n, m \geq 1 \}$

7) $L_7 = \{ a^i b^j \mid i \neq 2j+1 \}$

try to write in right linear form

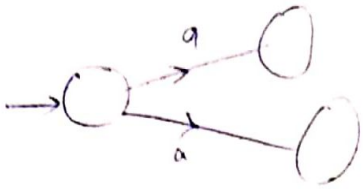
CFG

- deterministic
- non-det.

CFG

- unambigous

- Ambigous



CFG (may be)

- left recursive

  $A \rightarrow Aa$

- right recursive

  $A \rightarrow aA$

left most & right most derivations

→ If a derivation is said to be left most if in each step the leftmost variable in sentential form is replaced

In case of right most variable is replaced, it is called right most derivation

i) $S \rightarrow \underbrace{aAb}_{\text{sentential form}}$ ($A \rightarrow$ variable)

eg: consider CFG, $\rightarrow$ $w = ab^{2n}$

$S \rightarrow aAB$

$A \rightarrow A \mid \lambda$. $(\lambda \equiv \epsilon)$

$B \rightarrow bBb \mid \lambda$

TP abbbb $\in$ CFG,

left most derivation $\Rightarrow$ $S \rightarrow a\underset{\downarrow}{A}B$.

$S \rightarrow a\lambda\underset{\downarrow}{B}$

$S \rightarrow ab\underset{\downarrow}{B}b$

$S \rightarrow abbBbb$

$S \rightarrow abb\lambda bb \equiv abbbb$

right most derivation $\Rightarrow$ $S \rightarrow aA\underset{\downarrow}{B}$

$S \rightarrow aAbB\underset{\downarrow}{b}$

$S \rightarrow aAbbBbb$

$S \rightarrow aA\underset{\downarrow}{b}b\lambda bb$

$S \rightarrow a\lambda bbbb$

Derivation tree or parse tree

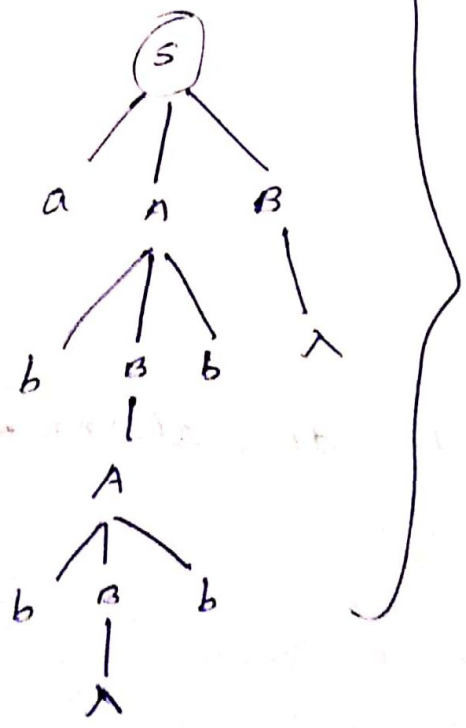
It is a ordered tree in which nodes are labelled with left sides of production and children of a node represent its right side

eg: CFG : $S \rightarrow aAB$

$B \rightarrow A \mid \lambda$

$A \rightarrow bBb$

for abbbb.



leftmost derivation's tree

inorder traversal gives the string

But it can also be done using
✓ right most derivation giving a different
tree structure. Hence its ambiguous

Ambiguous grammer

A CFG, G is said to be ambiguous
if ∃ some $w \in L(G)$ that has atleast
2 different parse trees / existence of 2 or more
(left most or right most derivation) ⇒ ?

∴ given CFG is ambiguous for w = abbbb

⇒ CFG is ambiguous

⇒ Assignment: Removal of ambiguity

Simplification of CFGs & Normal form.

steps to use CYK algorithm & make normal form

1) remove null productions ($A \to \lambda$)
2) remove unit productions
$A \to B$, $B \to C$, $C \to D$ $\equiv A \to$
3) remove left recursions $\longrightarrow A \to Aa$
4) remove unreachable productions & useless prodn

eg: $S \to aAB$
$A \to ab$
$B \to A/a$
$C \to aA$ → not needed

Null production

→ productions of the form $A \to \lambda$

→ A is called nullable variable

Elimination.

① find a set of all nullable variables

Write all remaining productions with/ without all nullable variable

eg let $S \longrightarrow aSb \mid aAb$.

$A \longrightarrow \epsilon$

$\rightarrow$ $S \longrightarrow aSb$

$S \longrightarrow aAb$

$A \longrightarrow \epsilon$ } only 1 production for A.

∴ no problem

∴ replace A with ϵ

$\rightarrow$ $S \longrightarrow aSb$

$S \longrightarrow ab$

$\Rightarrow$ $S \longrightarrow aSb \mid ab$.

Rules:

① make a set of nullable variables

eg: $S \longrightarrow ABaC$.

$A \longrightarrow BC$

$B \longrightarrow b \mid \lambda$

$C \longrightarrow D \mid \lambda$

$D \longrightarrow d$

$A \rightarrow BC \rightarrow bC \rightarrow aY$.

then $\{B, C\}$ — set of null. variable

Next $A \longrightarrow BC$

$\{A, B, C\}$

Rule ②: Write remaining productions with & without all nullable variable

a) ~~Without nullable var.~~

$S \longrightarrow ABaC \mid \lambda BaC \mid A\lambda aC \equiv AaC \mid ABa\lambda \mid \lambda aC \mid Aa \mid Ba \mid a$

ABaC: with {A, B, C}

λBaC: with A as ∈, B as C, C as C

Aλ aC ≡ AaC

ABaλ: A & B NULL

λaC

Aa

Ba

a: A, B, C as λ

[Substitute 1 var as null @ a time, then 2 and then 3.]

$A \longrightarrow BC \mid C \mid B$ ~~∈~~

BC: with null var

C | B: without null var.

$B \longrightarrow b$ (remove λ) ba

$C \longrightarrow D$

$D \longrightarrow d$.

Q]. $S \longrightarrow AB$

$A \longrightarrow aAA/\epsilon$

$B \longrightarrow bBB/\epsilon$.

null vars = $\{A, B\}$

$S \longrightarrow A/B/AB$.

$A \longrightarrow aAA/aA/a$.

$B \longrightarrow bBB/bB/b$

Unit production

eg: $S \longrightarrow A/bc$

$A \longrightarrow B/a$.

$B \longrightarrow C$

$\equiv$

$S \longrightarrow A/bc$

$A \longrightarrow c/a$.

eg: $S \longrightarrow Aa/B$

$B \longrightarrow A/bb$

$A \longrightarrow a/bc/B$.

(1) here unit productions are

$S \longrightarrow B$

$B \longrightarrow A$

$A \longrightarrow B$

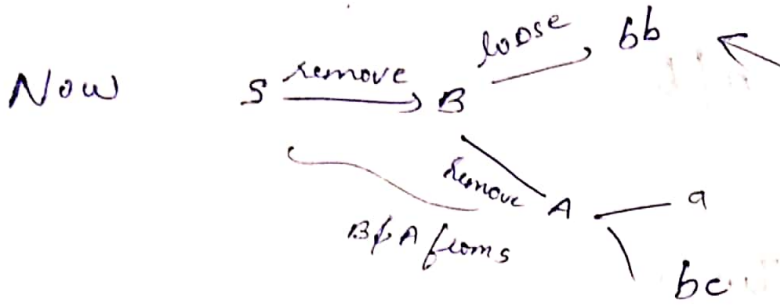
∴ dependency graph

(2) write CFG with non-unit production

$S \longrightarrow Aa$.

$B \longrightarrow bb$

$A \longrightarrow a/bc$

Now 

$\Rightarrow S \longrightarrow Aa/bb/a/bc$.

$B \longrightarrow bb/a/bc$

on removing A.

$A \longrightarrow a/bc/bb$

on removing B from A we loose bb!

useless $\Rightarrow$ doesn't produce string at end (goes on...)

unreachable $\Rightarrow$ can't reach. eg $(C \rightarrow a/b)$

(3) Removal of useless & unreachable variable

A prodn is useless if it

involves any useless variable

eg: $B \rightarrow AaD$ — D is useless → B is useless

but $B \rightarrow AaD \mid C \Rightarrow ?$

Remove useless symbol

eg: $S \rightarrow aS \mid A \mid C$

$B \rightarrow aa$

$A \rightarrow a$

$C \rightarrow aCb$

Rule: (1) : Identify the set of variable that can lead to terminal string

$S \rightarrow A \rightarrow (a)$

$\therefore \{B, A, S\}$

not C coz $C \rightarrow aCb$ / aCb / $\underline{ab}$ — never terminates

here

$S \longrightarrow aS/A$
$A \longrightarrow a$
$B \longrightarrow aa$

} Remove rest of symbol

(2) Remove unreachable symbol

i.e B

$\therefore S \longrightarrow aS/A$

$A \longrightarrow a$

reduce DFA pumping lemma

Normal form of CFG

CNF

[Chomsky Normal -form]

GNF

Greibach normal form

CNF: A CNG is in CNF if all production are of the form

$A \longrightarrow BC$ or

$A \longrightarrow a$

RHS 2 → var.
RHS 1 → terminal
only possible

where $A, B, C \in V$ (non-terminal/var)

& $a \in T$ (terminals)

Eg: CFG in CNF

$S \longrightarrow AS/a$

$A \longrightarrow SA/b$

Theorem: Any CFG $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, T, S, \hat{P})$ in CNF or GNF

Advantages

→ length of each production is restricted

→ parse tree / derivation tree obtained from CNF is always binary

→ # steps required to derive a string of length $|w|$ is $2^{|w|} - 1$

eg: $S \to AB$, $A \to a$, $B \to b$

ab

$S \to AB$

$\to aB$

$\to ab$

→ It is easy to apply to CYK membership algo.

Procedure to convert CFG to CNF

① Eliminate λ-productions, unit productions, useless productions

(4) Eliminate terminals from RHS if they exists with other terminals or non terminals

eg: $X \longrightarrow aY$ (a → terminal with other elements)

$\Downarrow$

$X \longrightarrow AY$

$A \longrightarrow a$.

(5) Eliminate RHS with more than 2 non terminals

eg $S \longrightarrow XYZP$.

$\Downarrow$

$S \longrightarrow AB$

$A \longrightarrow XY$

$B \longrightarrow ZP$

what if $S \longrightarrow XYZ$?

Q]. convert the following to CNF

$S \longrightarrow bA/aB$ — no unit used

$A \longrightarrow bAA/as/a$ — no useless or unreachable

$B \longrightarrow aBB/bs/b$ — no λ-used

(2) $S \to bA \mid aB$

⇓

(a) $S \to XA \mid YB$

$X \to b$.

$Y \to a$.

(b) $A \to bAA \mid aS \mid a$

$A \to ZA \mid YS \mid a$

$Z \to XA$.

(c) $B \to aBB \mid bS \mid b$

$B \to PB \mid XS \mid b$

$P \to YB$

~~$S \to XZ$.~~
~~$S \to XB$~~
~~$B \to YZ$~~

$S \to XA \mid YB$

$A \to ZA \mid YS \mid a$

$B \to PB \mid XS \mid b$

$Z \to XA$

$P \to YB$

$X \to b$

$Y \to a$.

(3) $S \to ABa$.

$A \to aab$

$B \to Ac$

$P \to \epsilon$. $\longrightarrow$ useless./unreachable.

$S \to ABa$.

~~$S \to AX$~~.

~~$X \to BY$~~

~~$Y \to a$~~

$S \to XY$

$X \to AB$

$Y \to b$.

$A \rightarrow aab$

$\Downarrow$

$A \rightarrow ZY$

$Z \rightarrow PP$

$P \rightarrow a.$

$B \rightarrow AC$

$\Downarrow$

$B \rightarrow AQ$

$Q \rightarrow c$

$S \rightarrow XY$

$X \rightarrow AB$

$A \rightarrow ZY$

$Z \rightarrow PP$

$Y \rightarrow b$

$P \rightarrow a$

$B \rightarrow AQ$

$Q \rightarrow c$

(4) $S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \lambda$

$P \rightarrow abc \longrightarrow$ unreachable.

GNF

→ A CFG is said to be in GNF if all productions have the form

$A \longrightarrow aX$

$a \in T$ ✓

$X \in V^+$ ✓ $\quad (V^* - \{\lambda\})$

eg: $S \longrightarrow a$? $\quad S \longrightarrow bAB$ ✓ $\quad S \longrightarrow bABCB$ ✓

$S \longrightarrow Aa$ ✗

Convert CNF to GNF?

$S \longrightarrow aB/BC$ → replace B

$A \longrightarrow aB/bA/a$

$B \longrightarrow bB/eC/b$

$C \longrightarrow c$

ii) $S \longrightarrow aD/bBC/eCC/bC$

$A \longrightarrow$
$B \longrightarrow$ } same
$C \longrightarrow$

Membership algo. for CFG

(if $w \in$ CFG or not)

CYK:

→ more ~~fast~~ complexity than parse tree ($O(n^3)$)

→ Its a universal ~~long~~ algo. to check whether a string is a member of $L(G)$ or not.

→ CYK is only applied if CFG is in CNF

$$V_{ij} = \bigcup_{k \in (i, i+1 \ldots j-1)} \{A : A \to BC \mid B \in V_{ik}, C \in V_{k+1,j}\}$$

---

eg. Check if 'baaba' is a member of

$S \to AB \mid BC$

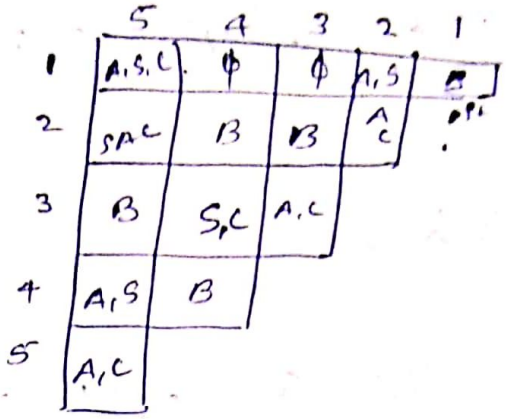
$A \to BA \mid a$

$B \to CC \mid b$

$C \to AB \mid a$

1 2 3 4 5

let b a a b a



(1) 11

non-terminals that produce 1-1 string directly

here 1-1 ⇒ b ⇒ {B}

Now 22 ⇒ a

which is produced directly from {A, C}

similarly 33 $\xrightarrow{a}$ {A, C}

44 $\xrightarrow{b}$ B

55 $\xrightarrow{a}$ {A, C}

(2) 12 ⇒ non-terminals the produce string from 1 to 2

12 ≡ ba ≡ (11)(22)

= (B)(A, C)

= BA, BC

≡ {A, S}

B produce 11

A & C produce 2 2

∴ BA & BC produce ba

look RHS of production to find a non-terminal the produce this

23 $\equiv$ (22)(33)

$=$ {A, C} {A, C} combination A,C & A,C give $\frac{23}{bb}$

$=$ {AA, AC, CA, CC}

↓ ↓ ↓ ↓

α α α B

No production is there which produce these non-terminals. (∵ they can't be made)

$=$ B

34 $\equiv$ (33)(44)

$\equiv$ (A, C)(B)

$=$ AB, CB ↳ no production

↓

S, C

9 produces AB which gives 34

$\equiv$ S, C $\longrightarrow$ 2 non terminals can produce AB.

43 $\equiv$ (44)(55)

$=$ (B)(A, C)

$=$ BA, BC

↓ ↓

A S

$=$ {A, S}

$$13 \equiv \begin{cases} (11)(23) \\ (12)(33) \\ (11)(22)(33) \longrightarrow \text{no need.} \end{cases}$$

$\underbrace{(11)(22)}_{12}\underbrace{(33)}_{23}$

we already found val. that make $(11)(22)$ & $(22)(33)$

$(11)(23) = (B)(B)$

$= BB$

$\downarrow$

$\alpha.$

$(12)(33) = (A,S)(A,C)$

$= AA, AC, SA, SC$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$\alpha \quad \alpha \quad \alpha \quad \alpha.$

$\Rightarrow 13 = \phi$ (3) can't be produced from any terminal dirctly or indirectly

---

$$(24) \equiv \begin{cases} (22)(34) = B \\ (23) \overset{\text{or}}{(44)} = B \end{cases}$$

$(22)(34) = (A,C)(S,C)$

$= AS, AC, CS, CC$

$\downarrow_{\alpha} \quad \downarrow_{\alpha} \quad \downarrow_{\alpha} \quad \downarrow_{B}$

$(23)(44) = (B)(B) = BA$

$\therefore (24) = B.$

$55 = \begin{cases} (33)(45) \equiv (A,C)(A,S) = \phi \\ (34)(55) \equiv (S,C)(A,C) = B \end{cases}$ B.

$14 = \begin{cases} (11)(24) \equiv (B)(B) = \phi \\ \text{or} \\ (12)(34) = (A,S)(S,C) = \phi \\ \text{or} \\ (13)(44) = \phi \cdot (44) = \phi \end{cases}$ (since 13 can't be made (13)(44) also can't be made.)

$= \phi$

$25 \equiv \begin{cases} (22)(35) \equiv (A,C)(B) = S, C \\ \text{or} \\ (23)(45) = (B)(A,S) = \phi \\ \text{or} \\ (24)(55) = (B)(A,C) = (A,S) \end{cases}$ or made from S or C

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

$$
15 = \begin{cases} (11)(25) = (B)(SAC) = A, S \\ (12)(35) = (A,S)(B) = S, C, \\ (13)(45) = \phi \cdot 45 = \phi \\ (14)(55) = \phi \cdot 55 = \phi \end{cases}
$$

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

⇒ string "baaba" can be made from

A, S, C

since CFG start form S production

we check if 15 has S or not

if yes then string ∈ CFG else not

here baaba ∈ CFG.

Now to find # substrings of 'baaba'

check strings that has S in it

⇒ It can be made from CFG

(Complexity :- $\frac{n(n+1)}{2} \times (n-1) = O(n^3)$)

Pumping Lemma.

5 theorem
5 - obj with sudeep

how to show a language $\in L_{regular}$.

$L_{reg} \Rightarrow \{L \mid \exists$ DFA that recognises it$\}$

If a lang L is regular, then $\exists\ p \in \mathbb{Z}^+$ such that for any string $w \in L$ of length atleast p, we can split w as $w = xyz$ such that

$|w| \geq p$,

(1) $|y| > 0$ (y is non-empty, $x \& z$ can be so)

(2) $xy^i z \in L$ for $i = 0, 1, 2 \ldots$ add/remove y

$\Rightarrow y$ can be pumped

(3) $|xy| \leq p$

---

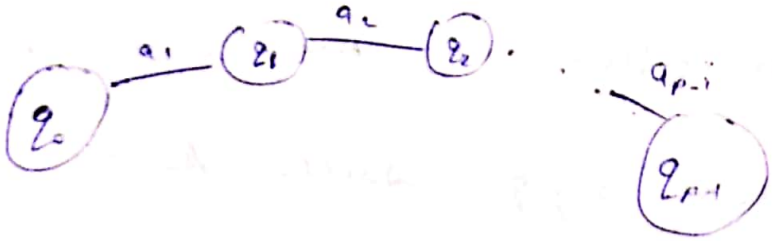
eg: L = string with even no of 1's

$1\overset{y}{0}01$ y can be 0

$w = 1\square\square1$ $|y| \neq 1$ $|y| = 2$. add or remove y. $\therefore w \in L$ still

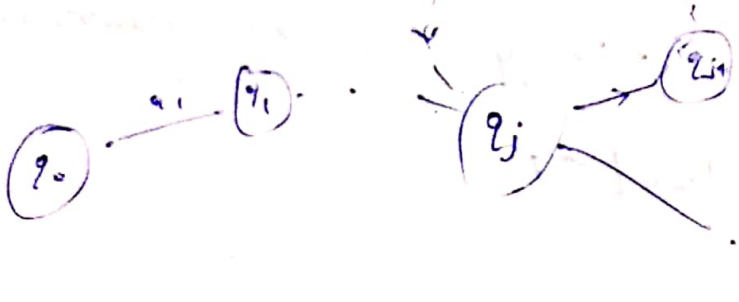
(0) $w = 11$
$x = \phi$
$y = 11$
$z = \phi$
$\Big\}$ $xy^i z \in L$

⇒ let p be the # states in DFA corresponding L (need not be min. DFA as we only need to prove ∃ a p)



if $|w| = p$, (no loops)

if pumping is performed, it means



⇒ y = # loop's states which even if removed or pumped reaches q_{p-1} eventually.

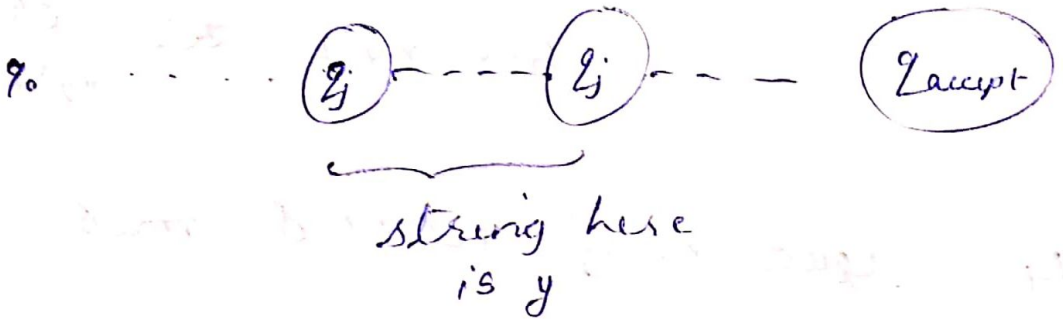
Since $|w| \geq p$ & # states = p (p states can give only p+1 transition)

there are atleast p+1 states in sequence

$q_0 - a_1 - a_2 \ldots a_n - q_{accept}$

so atleast one states appears twice in this sequence

Now, let a_j be the first sequence that repeats



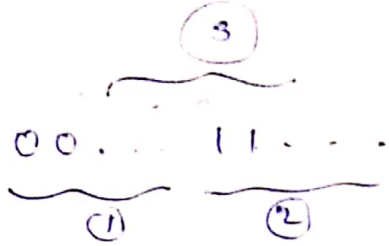
1. (1) $|y| > 0$ (y is non empty)

(2) $xy^iz \in L$ (obvious from DFA

(3) $|xy| \leq p$ (bcoz within 1st p symbol (or 1st p+1 states) there will be one state repeated as $|w| \geq p$ & # states = p (i. why we said a_j is 1st seq

eg: 0^n1^n is non regular.

Proof: let $\exists P$,



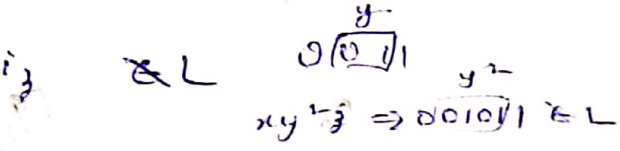
if $y \in$ 1st part or 2nd part

then pumping y $xy^i z \notin L$

if $y \in$ 3rd part,

removing $y \Rightarrow xz \in L$

but $xy^i z \notin L$



---

eg: L_1 = equal no. of zeroes & ones

let $w = 0^P1^P$ if y consists of only 0's or only 1's or diff no. of 0's & 1's

then $xy^2z \notin L$

$y = 01$

it can be pumped

but $|xy| > P$ violates 3rd condn

(within 1st P symbols, y must occur in loop must occur.

eg: ① $w \in$ palindrome

② $w\ w^R$

③ $w\ w$

④ $w \# w$

① let $w = 1^p 0 1^p$

here y cannot be found within 1st p symbols $\therefore |xy| > p$ $\therefore$ not regular

---

Time complexity of TM

$0^n 1^n \longrightarrow O(n^2)$ TM (single tape)

$\longrightarrow O(n \log n)$ "

$\longrightarrow O(n)$ 2-tape.

DBMS