Closure property of regular languages.

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

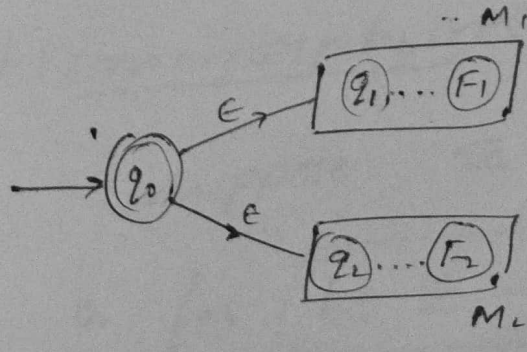
Theorem:

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, concatination, compliment, intersection & KLEENE closure

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

Union operation



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

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

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

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)

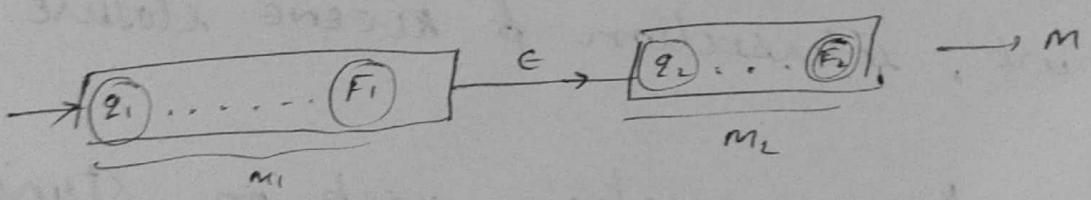
$\Rightarrow \delta^*(q_0, x) \in F_1$ OR $\delta^*(q_0, x) \in F_2$

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

$\Rightarrow x \in L_1$ 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 $w = a_1 a_2 \ldots a_n$, a_i is a symbol, then

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

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

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

$\therefore$ homomorphic image of a RE is a Regular language

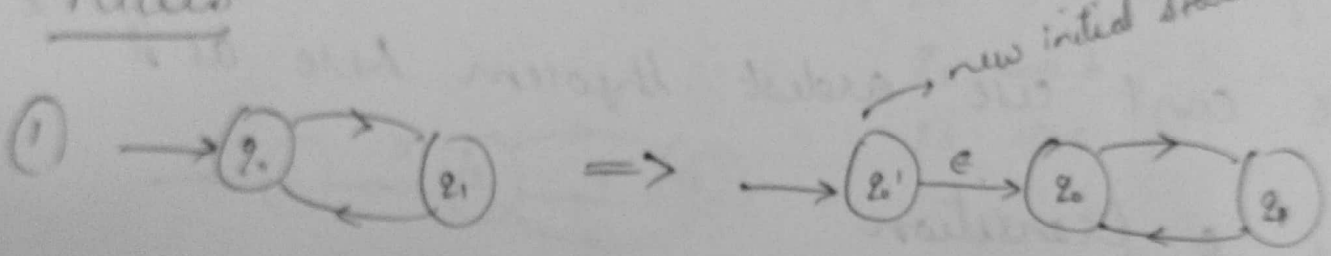
Finite automata to R.E

→ many methods.

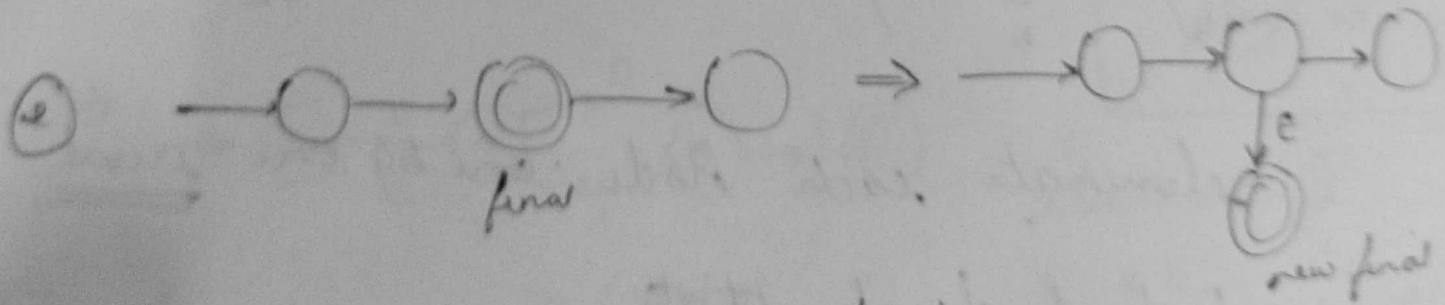
→ Arden method not possible in ε NFA is the

→ ∴ generally use state elimination method

Rules

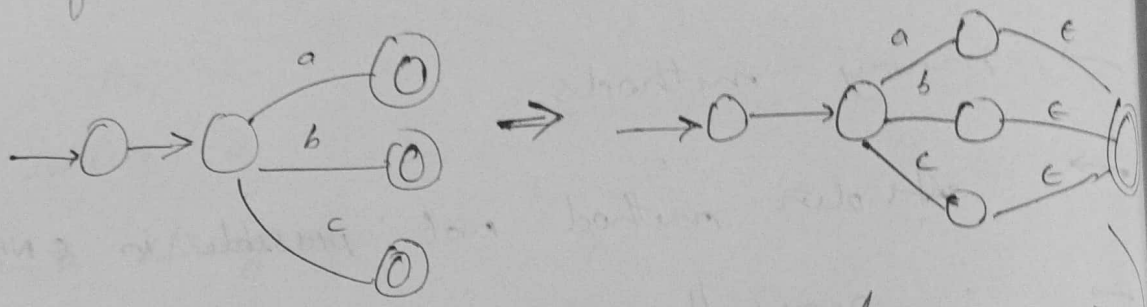


if initial state has incoming edge, the make an new initial state and give ε-transition to old initial state



If final state has outgoing transition :

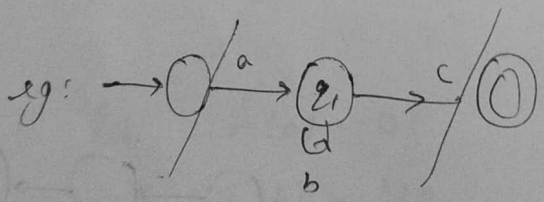
(3) if there are multiple finite state



make it ε transition to common final state.

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

Method.

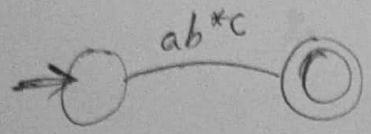
eg: 

→ all rules applied

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

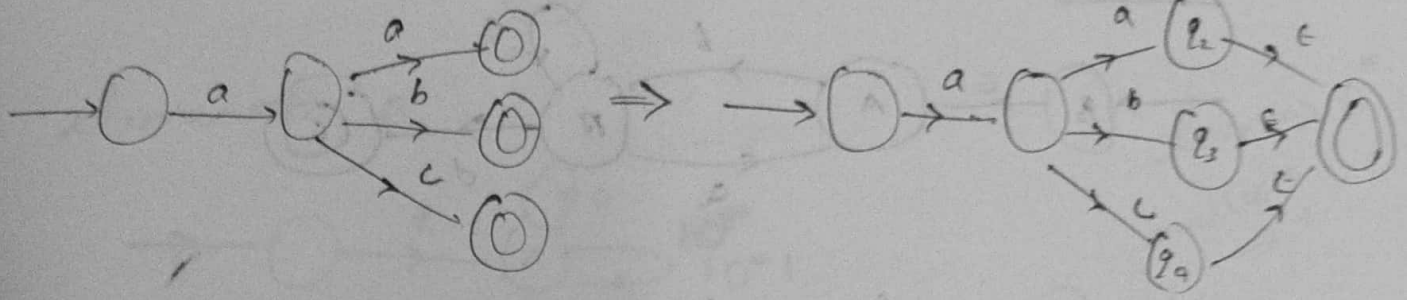
⊙ What RE will you miss if you eliminate a state from F.A

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

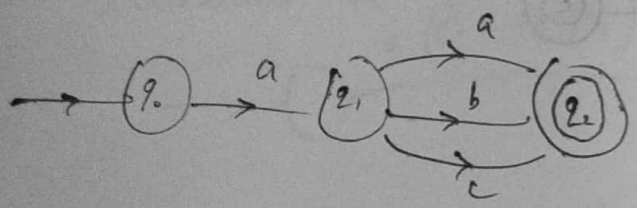


$RE = ab^*c$

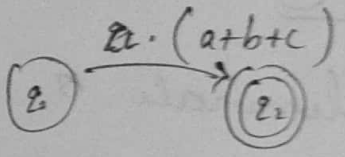
eg:

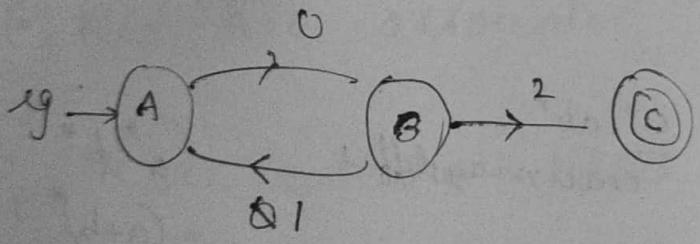


eliminate one by 1 (q_2 ⇒ miss a
q_3 ⇒ b
q_4 ⇒ c)



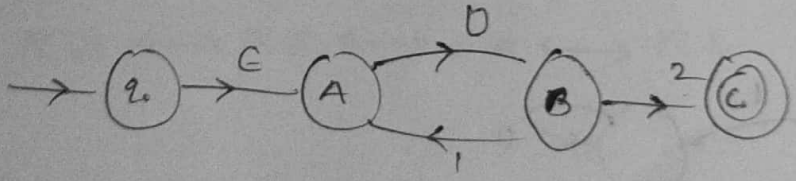
→ eliminate q_1 ⇒





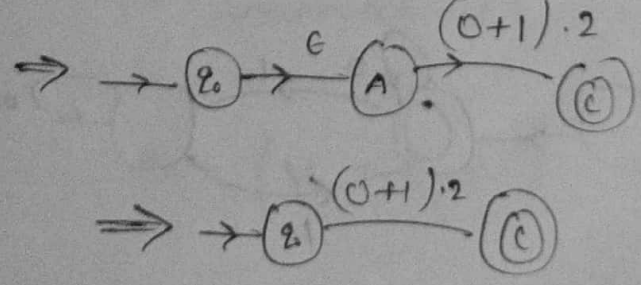
multiple transition ⇒ '+'
single ⇒ '.'

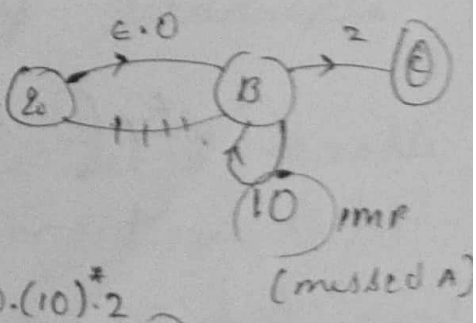
⇒

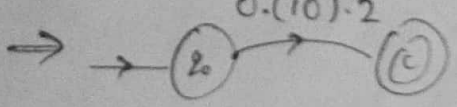


eliminate A;

wrong!

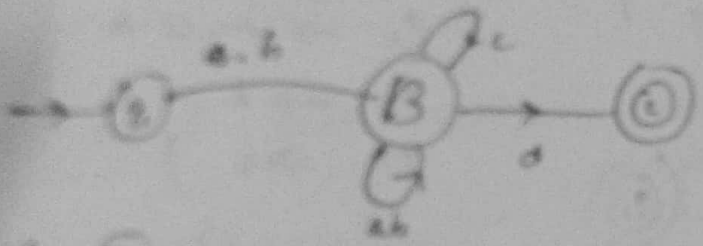






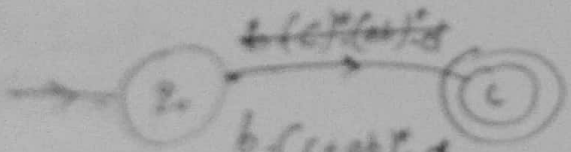
Step (1)

Step (2) eliminate A



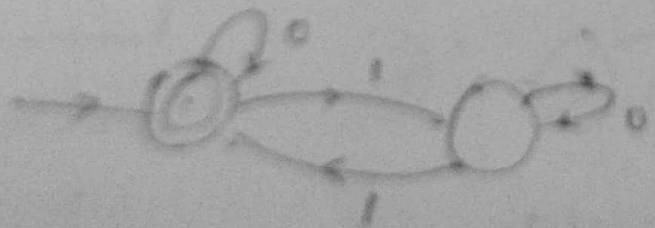
Step (3)

eliminate B.

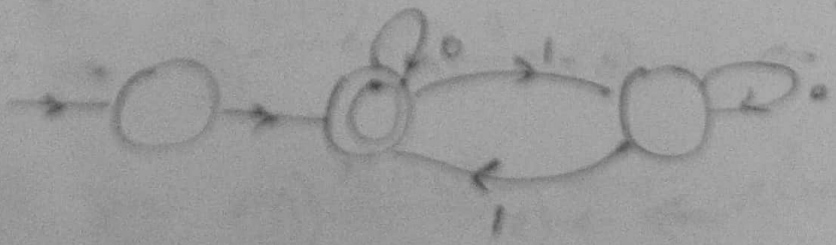


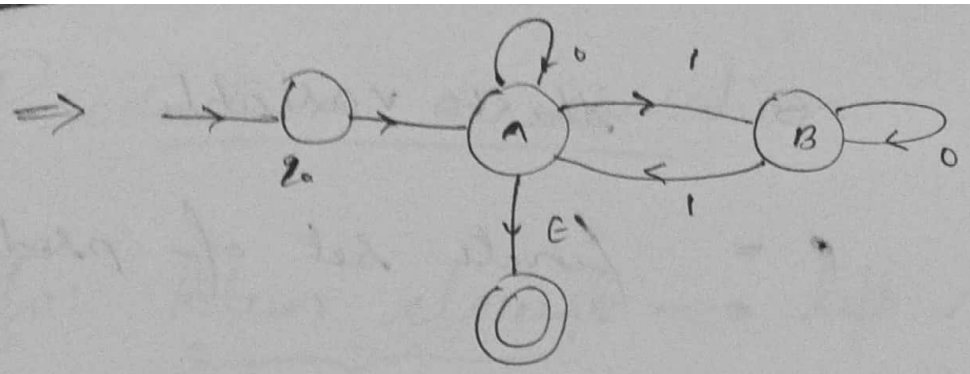
not $c^*.ab^*$
order may differ

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



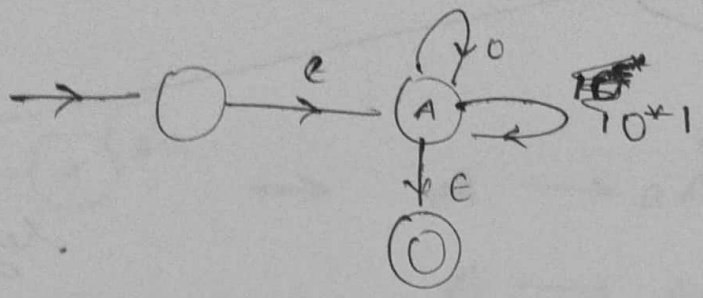
Step (1) →





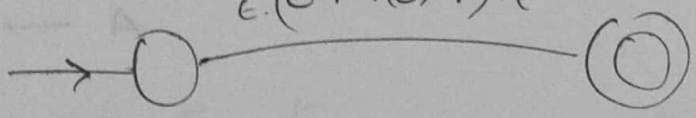
eliminate B

loop ⇒ 0^*
multiple ⇒ +



eliminate A

$\epsilon.(0+1(0)^*1)^*\epsilon$



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

Regular Grammar (RG)

⊙ RG are associated with RL ϕ ∀ 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

RG

Left linear

$A \rightarrow Ba$

$A \rightarrow b$

Right linear

$A \rightarrow aB$

$A \rightarrow b$

$A, B \in V$

$a, b \in T$

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

→ LHS only one var. must appear

→ RHS almost one var. should appear

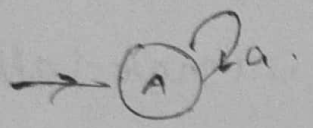
↳ LHS → RHS

RHS 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

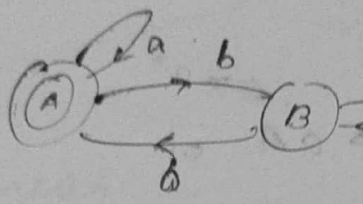
as S in Left linear
A in right linear

transition fn: production rule.

$\delta(q_1, a) \longrightarrow q_1$ ⇒ $q_1 \longrightarrow aq_1$ or $q_1 \longrightarrow q_1 a$

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

$q_f \in$ finite state $\longrightarrow$ $q_f \leftrightarrow \in$

eg: 

note: all transiti
u need
production

Let RG be left linear

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

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

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

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

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

Context Free Grammar (CFG)

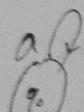
A grammar $G = (V, T, S, P)$ is said to be CFG if all productions in P have the form $\boxed{A \rightarrow x}$

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

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

- RG ∈ CFG not vice versa.

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



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

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

$S \to aa \mid bb \mid ab \mid ba \Rightarrow \underbrace{(a+b)}_{A}.(a+b)$

$A \to a \mid b$

$S \to AA$

$\Rightarrow S \to a \mid b$

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

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

$S \longrightarrow AAB$

$A \longrightarrow a|b$

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

→ CFG not RG

CFG → RG

$S \longrightarrow a|b$

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

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)

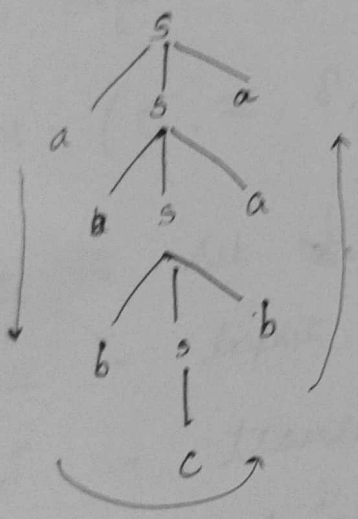
$S \longrightarrow c \mid asa \mid bSb$

$S \longrightarrow aSa$
$\longrightarrow aaSaa$
$\longrightarrow aabSbaa$
$\longrightarrow aabcbaa$

} 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. $L_1 \rightarrow aAbcB$ $\quad L_1 \rightarrow S$

$A \rightarrow c/aAb$

$B \rightarrow c/c$

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

$B \rightarrow c/bB$

$A \rightarrow aAb/\epsilon$

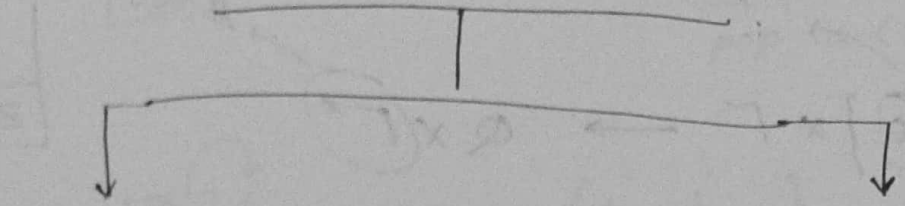
Q3. $L_3 \rightarrow aAbb$ $\quad L_3 \rightarrow S$

$A \rightarrow aAbb/\epsilon$

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

$A \rightarrow aAb/\epsilon$

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



deterministic PDA

non-deterministic PDA

lang: det. cont. free (DCFL)

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

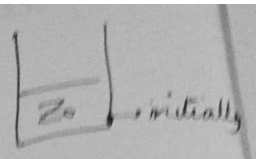
DPDA

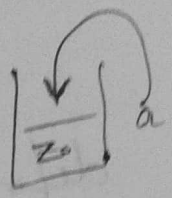
not included in DFA

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

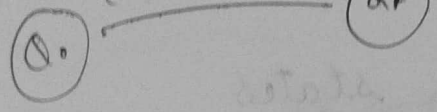
$\therefore \delta$ (curr. state, i/p symbol, symbol on stack top)

→ next state, resultant symbol on stack)





eg: $(Q_0, a, Z_0) \rightarrow (Q, \underbrace{aZ_0}_{\text{push } a})$.

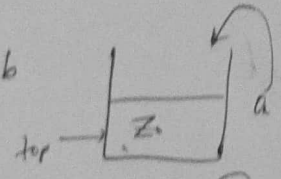
$\therefore$ 

NPDA

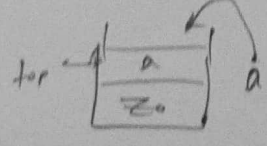
$\delta: Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$

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

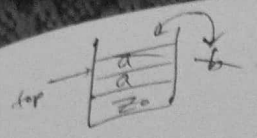
aabb



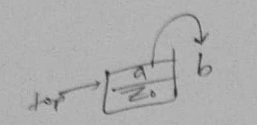
$(Qa, Z_0 / \underbrace{aZ_0}_{\text{push a}})$



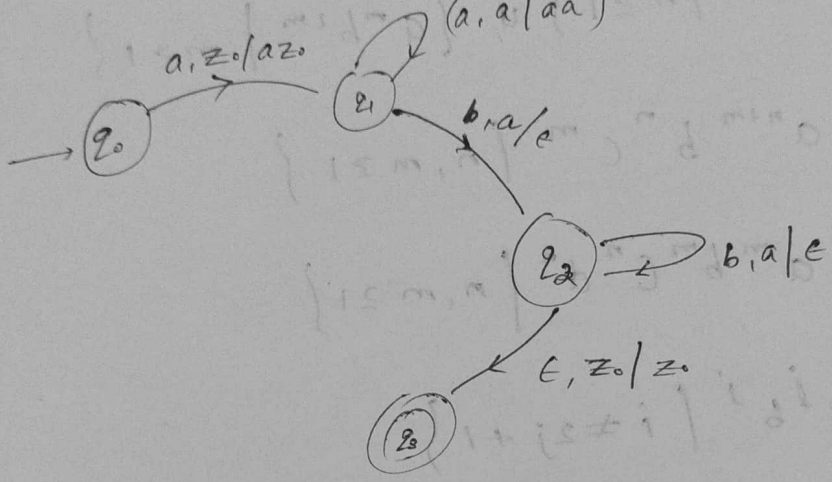
$(a, a / \underbrace{aa}_{\text{push a}})$



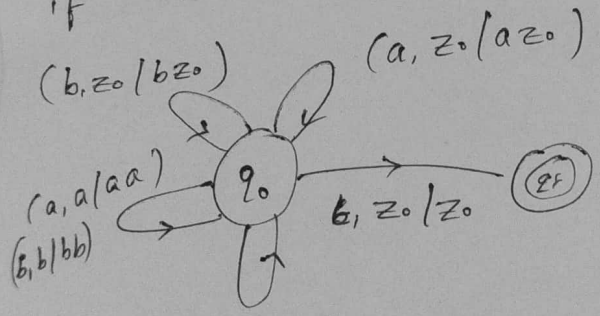
$\longrightarrow (b, a \mid \underline{\epsilon})$ pop a (top)



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



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

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

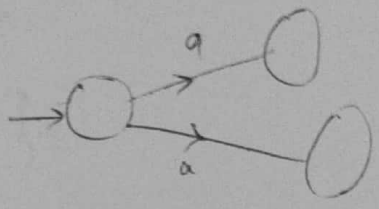
try to write RG in right linear form

CFG

deterministic non-det.

CFG

unambigous Ambigous



CFG (may be)

left recursive right recursive

$A \to Aa$ $A \to aA$

left most & right most derivations

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

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

eg: consider CFG, $\longrightarrow \omega = ab^{2n}$

$S \longrightarrow aAB$

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

$B \longrightarrow bBb \mid \lambda$

TP abbbb ∈ CFG,

left most derivation ⇒ $S \longrightarrow a\overset{\downarrow}{A}B$.

$S \longrightarrow a\lambda B$

$S \longrightarrow ab\overset{\downarrow}{B}b$

$S \longrightarrow abbBbb$

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

right most derivation ⇒ $S \longrightarrow aAB$

$S \longrightarrow aAbBb$

$S \longrightarrow aAbbBbb$

$S \longrightarrow a\underset{\downarrow}{A}bb\lambda bb$

$S \longrightarrow abbbb$

Derivation tree or parse tree

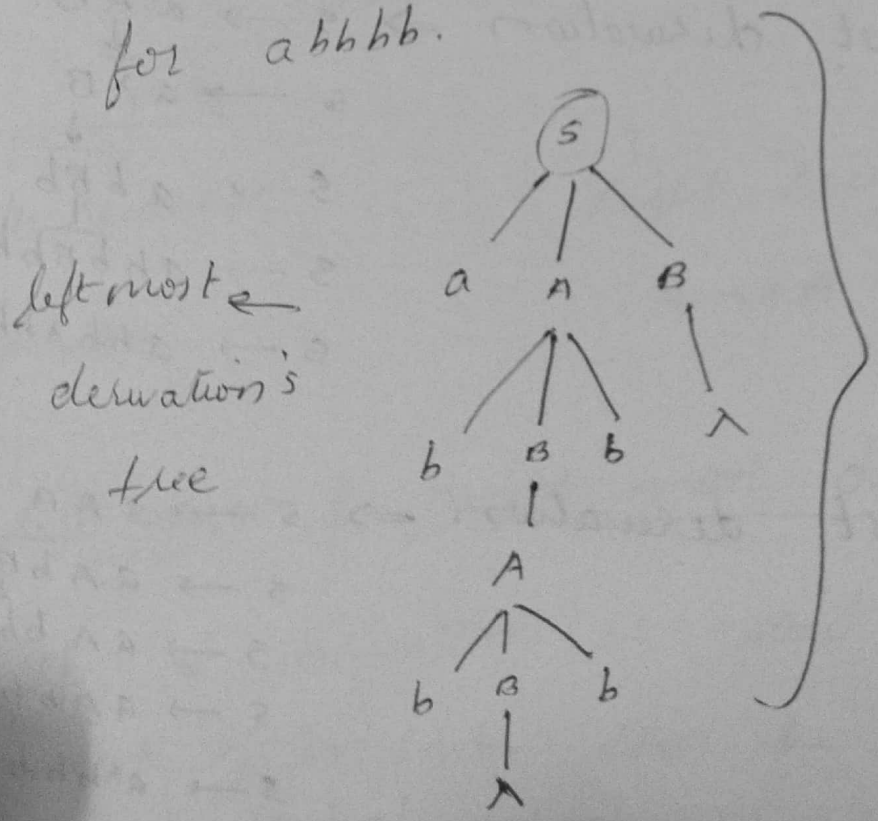
It is a ordered tree in which nodes are labelled with left sides of productions and children of a node represent its right side

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 ambigous

Ambigous grammer.

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

i.e, given CFG is ambigous for w = abbbb

⇒ CFG is ambigous

⇒ 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

$A \to B$
$B \to C$ $\equiv A \to D$
$C \to D$

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

→ production of the form $A \to \lambda$.

→ A is called nullable variable

Elimination

① find a set of all nullable variables

① Write all remaining production 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, Bb a X.

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

Since $A \longrightarrow BC$

$\{A, B, C\}$

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 C, B as C, C as B

λaC: A & B NULL

a: A, B, C as λ

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

$A \longrightarrow BC \mid C \mid B$ (with null var)

(without null var.)

$B \longrightarrow b$ (remove λ) ba

$C \longrightarrow D$

$D \longrightarrow d$.

Q]. $S \rightarrow AB$
$A \rightarrow aAA|\epsilon$
$B \rightarrow bBB|\epsilon$

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

$S \rightarrow A|B|AB.$
$A \rightarrow aAA|aA|a.$
$B \rightarrow bBB|bB|b$

Unit production

Eg: $S \rightarrow A|bc$
$A \rightarrow B|a.$
$B \rightarrow C$

$\equiv$

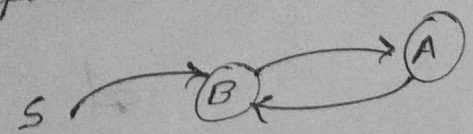
$S \rightarrow A|bc$
$A \rightarrow c|a.$

Eg: $S \rightarrow Aa|B$
$B \rightarrow A|bb$
$A \rightarrow a|bc|B.$

(1) here unit productions are

$S \rightarrow B$
$B \rightarrow A$
$A \rightarrow B$

∴ dependency graph

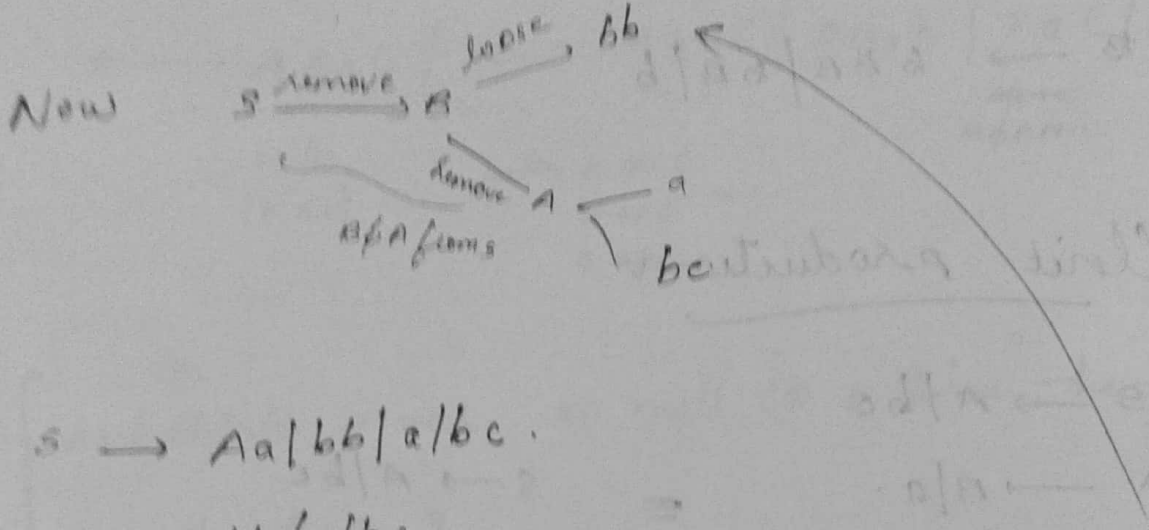


(2) Write CFG with non-unit production

S → Aa

B → bb

A → a/bc



→ S → Aa | bb | a/bc.

B → bb/a/bc
on removing A.

A → a/bc/bb
on removing B from A we loose bb

useless → doesn't produce string at end (goes on...)
eg (C → aCb)

unreachable → can't reach.

(3) Removal of useless & unreachable variable

A prodn is useless if it

removes any useless variable

eg: $A \rightarrow AAD$ bcz D is useless
→ a 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

$S \rightarrow A \rightarrow (a)$

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

not c bcz $C \rightarrow aCb$

aCb

aCb

$\underline{aCb}$ never terminates

Here

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

} Remove rest of symbol

(2) Remove unreachable symbol

i.e B

$\therefore S \rightarrow aS/A$

$A \rightarrow a$

reduce DFA pumping lemma

Membership algo. for CFG

(if w ∈ CFG or not)

CYK:

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

→ Its a universal lang 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

$S \to AB \mid BC$

$A \to BA \mid a$

$B \to CC \mid b$

$C \to AB \mid a$

let b a a b a

1 2 3 4 5

	5	4	3	2	1
1	A,S,C	ϕ	ϕ	A,S	B
2	S,A,C	B	B	A,C	
3	B	S,C	A,C		
4	A,S	B			
5	A,C				

(A) 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 produced string from 1 to 2

12 = ba = (11)(22)

= (B)(A, C)

= BA, BC

= {A, S}

B produce 11

AϕC produce 22

∴ BA ϕ BC produce ba

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

23 = (22)(33)

= {A,C}(A,C)

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

= {AA, AC, CA, CC}

↓ ↓ ↓ ↓

X α α B

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

= B

34 ≡ (33)(44)

≡ (A,C)(B)

= AB, CB → no production

↓

S,C

S produces AB which gives 34

= S,C → 2 non terminals can produce AB.

43 ≡ (44)(55)

= (B)(A,C)

= BA, BC

↓ ↓

A S

= {A,S}

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

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

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

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

$= BB$

$\downarrow$

α.

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

$= AA, AC, SA, SC$

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

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

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

---

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

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

$= AS, AC, CS, CC$

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

$\alpha \quad , \quad \alpha \quad B$

$(B+1)(B+1) = (B)(B) = BB$

$\therefore (2,4) \equiv B.$

$3E = \left\{ \begin{array}{l} (33)(45) \equiv (A,C)(A,B) = \phi \\ (34)^{B1}(35) \equiv (B,C)(A,C) \equiv B \end{array} \right\} B.$

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

$\equiv \phi$

$25 = \left\{ \begin{array}{l} (22)(35) \equiv (A,C)(B) = B,C \\ \quad \text{or} \\ (23)(45) = (B)(A,B) = \phi \\ \quad \text{or} \\ (24)(35) = (B)(A,C) = (A,B) \end{array} \right.$ 25 made from 5 or 6

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

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

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

→ 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 $\in$ CFG else not

here baaba $\in$ 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 subscript

how to show a language ∉ L_{reg}.

$L_{reg} \Rightarrow \{L \mid \exists$ 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, $x \phi z$ can be so)

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

$\Rightarrow$ y can be pumped

③ $|xy| \leq p$

---

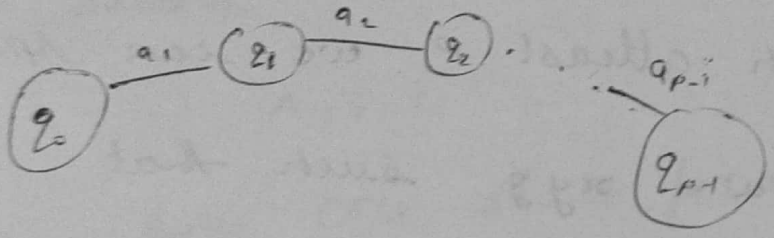
eg: L = string with even no of 1's

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

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

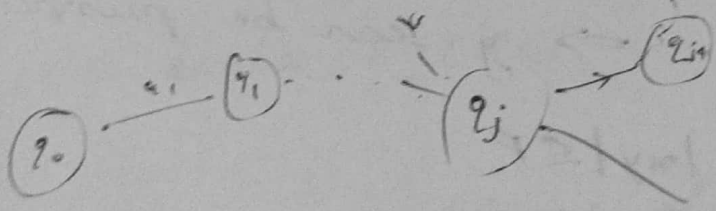
(c) $w = 11$
$x = \phi$
$y = 11$
$z = \phi$
$\Big\}\ xy^i z \in L$

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



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

if pumping is performed, it means



⇒ y = # loop's states (q_{p-1})
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

string here is y

✓. (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,

$\underbrace{00\ldots}_{①}\underbrace{11\ldots}_{②}$ (3)

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 $xy^i z \notin L$ $0\overbrace{0\,1}^{y}1$

$xy^2z \Rightarrow 0\overbrace{0101}^{y^2}1 \in L$

---

eg: L = equal no of zeros & 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 $xy^2z \notin L$

$y = 01$ $\underbrace{000\ldots01}_{p}11\ldots$

It can be pumped

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

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

y: ① $w \in$ palindrome

② $w\ w^R$

③ $w\ w$

④ $w \neq w$

① let $w = 1^p 0 1^p$

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

---

Turing 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