BDMAS
↓ ↓ ↓ ↓ ↓
Bracket

Regular Expression: A way of represent R.L

→ If any language if you develop finite automata of any language then that language is Regular.

→ A language is said to be regular if a Regular expression exit to represent it.

→ Expression of string and operators.

like → i) * (Kleen closure) [a*]

$a^* = \epsilon, a, aa, aaa \ldots$

in sequence to simplify

ii) + positive closure [a+]

$a^+ = a, aa, aaa.$

iii) . concatination [a.b]

iv) + Union [a+b]

→ RE is said to be valid iff it can be derived from the primitive R.E by a finite no. of application of the rule r^*, r^+, $r_1 . r_2$, $r_1 + r_2$

→ If Σ is a ~~primitive~~ given alphabet then ϕ, ϵ/λ ; $a \in \Sigma$ are premative regular expression.

* $a + \phi = a$
* $a + \epsilon = a, \epsilon$

regular expression ↓ Language → R.L. always write in brackets

(1) $r = \phi$, $L(r) = \{\}$ / ϕ

(2) $r = \epsilon$, $L(r) = \{\epsilon\}$

(3) $r = a+b$, $L(r) = \{a, b\}$

(4) $r = a.b$, $L(r) = \{ab\}$

(5) $r = a+b+c$, $L(r) = \{a, b, c\}$

(6) $r = (ab+a).b$, $L(r) = \{abb, ab\}$

(7) $r = a^+$, $L(r) = \{a, a^2, a^3 \ldots\}$

(8) $r = a^*$, $L(r) = \{\epsilon, a, a^2, a^3 \ldots\}$

(9) $r = (a+ba)(b+a)$, $L(r) = \{ab, aa, bab, baa\}$

(10) $r = (a+\epsilon).(b+\phi)$ $= L(r) = \{ab, b\}$

(11) $r = (a+b)^2$, $L(r) = (a+b)(a+b) = \{aa, ab, ba, bb\}$

(12) $r = (a+b)^*$ $= \{\epsilon, a, b, aa, bb \ldots\}$

$\Rightarrow (a+b)^0 = \epsilon$

$(a+b)^1 = a, b$

$(a+b)^2 = aa, ab, ba, bb$

(13) $r = (a+b)^*.(a+b) = \{(a+b)^+\}$

(14) $r = a^*.a^* = \{\epsilon, a, \ldots a^*\}$

(15) $r = (ab)^* = \{\epsilon, ab, abab, ab \ldots\}$

→ ex

5.3

= Ex

c

a)

b)

c)

d)

Ex 7

5.4

Ex

→ every ragular expression generate single regular language.

5.3

Eg → $r_1 = a^*$

$r_2 = a^* + (aa)^*$

Can two different regular expression genrate same Ragular language.

a) $L(r_1) \subseteq L(r_2)$

b) $L(r_1) \supseteq L(r_2)$

c) $L(r_1) = L(r_2)$ $\quad r_1 \cong r_2$

d) $L(r_1) \neq L(r_2)$

Eg → $r = \epsilon^* \Rightarrow L(r) = \{\epsilon^0, \epsilon^1 \ldots \epsilon^1\}$

$= \{\epsilon\}$

$r = \epsilon^+, L(r) = \{\epsilon\}$

$r = \phi^*, L(r) = \{\phi^0, \phi^1, \phi^2 \ldots\}$

$\phi^0 \rightarrow \epsilon$, $\phi^1 \rightarrow$ No, $\phi^2 \rightarrow$ No

$= \{\epsilon\}$

$r = \phi^+ = L(r) = \{\phi^1, \phi^2, \ldots\}$

$= \phi$

$A = \{a, b\}$

$A^0 \rightarrow$ means string of length 0 $= \{\epsilon\}$

$A^1 \rightarrow$ string of length 1 is present $\{a, b\}$

$A^2 = \{aa, bb, ab\}$

5.4

Eg → (i) $r^+ \cup r^* = r^*$

$r^+ \rightarrow r^1, r^2, r^3 \ldots$ $\quad r^* \rightarrow r^0, r^1, r^2$

(ii) $r^+ \cap r^* = r^+$

(iii) $r^* \cdot r^+ = \{\epsilon, r, rr, rrr \ldots\} \cdot \{r, rr, rrr \ldots\}$

ϵ

$= \{r, rr, rrr \ldots\}$

$= r^+$

$r^* = r^*$ → always * is greter.

④ $(r^*)^* = r^*$

⑤ $(r^*)^+ = \{\epsilon, r^1, r^2, r^3 \ldots\}^+ = r^*$

⑥ $(r^+)^* = r^+$

⑦ $((r^*)^+)^* \cdot r^+ = r^+$

⑧ $(a+b)^* = (a^* + b^*)^*$

⑨ $(a+b)^* = (a + b^*)^*$

⑩ $(a+b)^* = (a^* + b^*)^*$

⑪ $(a+b)^* \neq (a.b)^*$

⑫ $(a+b)^* \neq (a.b^*)^*$

⑬ $(a+b)^* = (a^*.b^*)^*$

$(a+b)^*$ → a aninds set which is of biggest set.

5.5 Write Regular Expression for R.L

① Start with ab

$\Sigma = \{a, b\}$

$= ab\,(a+b)^*$

→ end me kuch bhi ho sakta hai.

② Start with bba

$= bba\,(a+b)^*$

③ End with aab

$= (a+b)^* aab$

④ Contain a substring aab

$= (a+b)^* aab\,(a+b)^*$

⑤ Start and end with a

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

↳ only a is also possible

⑥ Start and end with same symbol.

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

⑦ Start and end with diff symbol.

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

⑧ $|w| = 3$

length $= (a+b)(a+b)(a+b) = (a+b)^3$

⑨ $|w| \geq 3$

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

⑩ $|w| \leq 3$

$= \epsilon + (a+b) + (a+b)(a+b) + (a+b)^3$

OR $= (a+b+\epsilon)^3$

5.b Write R.E for R.L

⑪ $|w|_a = 2$ → a string which have 2 a

No. of a is 2 b ho ya na ho koi fark nhi pdta

$= b^*\ a\ b^*\ a\ b^*$

↳ No. of a is 2 and b is any no. of b.

No of a is 2 in string. or more.

(2) $|w|_a \geq 2$

$(a+b)^* \; a \; (a+b)^* \; a \; (a+b)^*$

(3) $|w|_a \not\geq 2 \;=\; b^*(a+\epsilon)\,b^*(a+\epsilon)\,b^*$

OR $= b^* + b^*ab^* + b^*ab^*ab^*$

(4) 3rd symbol from left end is b.

$= (a+b)^2 \; b \; (a+b)^*$

↓ for starting 2 symbol

(5) 2nd symbol from right end is a

2nd position for next 2 symbol from right

$= (a+b)^* \; a \; (a+b)^{2}$

(6) $|w| = 0 \pmod 3$ ↳ string ke length ko agr 3 se divide kre to remender 0.

length 0, 3, 6, 9 are ok

$= [(a+b)^3]^*$

(7) $|w| = 2 \pmod 3$

$= (a+b)^2 \; [(a+b)^3]^*$

↳ remender 2. add kr do. ↳ remender 0

8 $|w|_b = 0 \pmod 2$

$= a^*(a^*ba^*ba^*)^*$

(49)
119(Neso)

63-90 Syllabus

Prove that $(1+00^*+1)+(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

LHS

$= (1+00^*+1)+(0+10^*1)^*(0+10^*1)(0+10^*1)^*$

$= (1+00^*+1)[\epsilon + (0+10^*1)^*(0+10^*1)]$

$\epsilon + \epsilon^* R = R^*$

$= (1+00^*+1)(0+10^*1)^*$

$= (\epsilon \cdot 1 + 00^*1)(0+10^*1)^*$ $\quad \epsilon \cdot R = R$

Neso (50)

Design Regular Expression for the following over $\Sigma = \{a,b\}$

i) L(r) accepting string of length exactly (2)

$\Rightarrow \{(a+b)(a+b)\}$

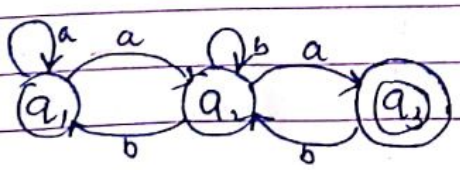
ii) language accept string of length atleast (2)

$\Rightarrow (a+b)(a+b)(a+b)^* + (a+b)(a+b)(a+b)^*$

iii) At most 2.

$\Rightarrow \epsilon + (a+b) + (a+b)(a+b) \equiv (\epsilon+a+b)(\epsilon+a+b)$

(51) Design Regular Expression (NFA → RE)



upcomming transition

write eqn of every state.

$q_3 = q_2 a$ (i/p) $\rightarrow$ ①

$q_2 = q_1 a + q_2 b + q_3 b \rightarrow$ ②

$q_1 = \epsilon + q_1 a + q_2 b$ — ③

↓ starting upcomming

simplify for final state

Simply eqn ① → $q_3 = q_2 a$

$q_3 = (q_1 a + q_2 b + q_3 b) a$

$q_3 = (q_1 a a + q_2 b a + q_3 b a$ — ④

Eg 2

② $q_2 = q_1 a + q_2 b + q_3 b$ (Putting value of q_3 from eqn ①)

$= q_1 a + q_2 b + (q_2 a) b$

$q_2 = q_1 a + q_2 (b + ab)$ ⇒ $R = Q + RP$ Arden's Theorem

R Q R P ∴ $R = QP^*$

$q_2 = q_1 a (b+ab)^*$ → ⑤

Eg 3

③ $q_1 = \epsilon + q_1 a + q_2 b$

$q_1 = \epsilon + q_1 (a + ((q_1 a)(b+ab)) b$ ⇒ $R = Q + RP$

R Q R P ⇒ $R = QP^*$

$q_1 = \epsilon (a + ((q_1 a)(b+ab)) b)^*$ ✓ property $\epsilon \cdot R = R$

$q_1 = (a + (q_1 a)(b+ab)^* b)^*$ → ⑥

Final state →

$q_3 = q_2 a$

$q_3 = (q_1 a (b+ab)^*) a$ Putting value of q_2 from ⑤

q_1 from 6 eqn

$= (a + a(b+ab)^* b)^* a (b+ab)^* a$ by putting value of q_1 from ⑥

Eg DFA to R

↳ Single

$q_1 = \epsilon$

$q_2 = q$

$q_3 = q$

$q_4 =$

Eg ①

$q_1 =$

$=$

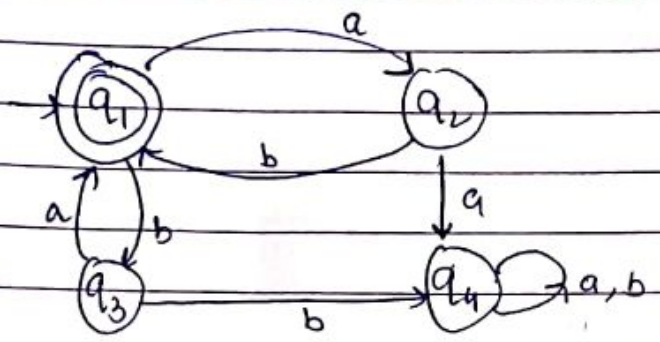
q_1

②

53

52 DFA to Regular Expression :-

↳ Single final state



$q_1 = \epsilon + q_2 b + q_3 a$ — ①

$q_2 = q_1 a$ — ②

$q_3 = q_1 b$ — ③

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$

Eg

①. $q_1 = \epsilon + (q_1 a) b + (q_1 b) a$ → putting q_2, q_3 from eqn ② & ③

$= \epsilon + q_1 (ab + ba)$ $\quad R = Q + RP$

$\quad\quad \downarrow \quad \downarrow \quad\quad \downarrow$ $\quad \therefore R = QP^*$

$\quad\quad Q \quad R \quad P$

$q_1 = \epsilon (ab+ba)^*$ $\quad \epsilon R = R$

$= (ab+ba)^*$

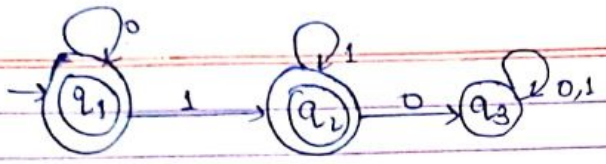
② $q_2 = q_1 a$

→ So q_1 is final state and we get the expression for final state we do not go further.

53 DFA to Regular. ∈ (with Multiple Final State)

↳ for this kind of situation we find expression for both the final state and then union both of them.

$[R = Q + RP$
$R = QP^*$



$q_1 = \epsilon + q_1 0 \quad \text{---(I)}$

$q_2 = q_1 1 + q_2 1 \quad \text{---(II)}$

$q_3 = q_2 0 + q_3 0 + q_3 1 \quad \text{---(III)}$

Final State

(i) can $\quad q_1 = \underbrace{\epsilon}_{Q} + \underbrace{q_1 0}_{R\ P}$ Admans Theorem

$q_1 = \epsilon 0^* \Rightarrow q_1 = 0^* \quad \text{---(IV)}$

(II) em $\quad q_2 = \underbrace{(0^*)1}_{Q} + \underbrace{q_2 1}_{R\ P}$

$R = Q + RP$
$R = QP^*$

$q_2 = ((0^*)1)(1)^*$

$q_2 = (0^*1)(1)^* \Rightarrow \quad q_2 = (0^*1)(1)^* \quad \text{---(V)}$

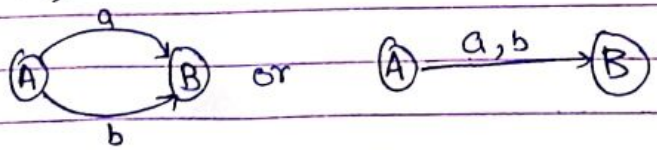
R = union of (IV) & (V)

$R.E = 0^* + 0^* 1 1^*$
$= 0^* (\epsilon + 1 1^*) \quad \rightarrow \epsilon + RR^* = R^*$
$= 0^* 1^*$

54 Conversion of Regular Expression to Finite Automata :-

Rules

i) R.E :- (a+b)



ii) (a.b)

A —

iii) a^*

Eg → (i) (

→A

ii) (a-

* iii)

A

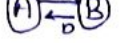
56

Eg → (a

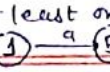
57

Q+RP
= QP*

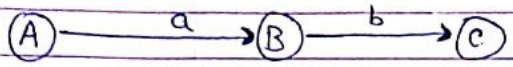
a^* ⇒ 

, $(bc)^*$ ⇒ 

, $(a+b)^*$ ⇒ 

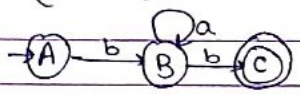
$a^+ = \{a, aa, aaa..\}$ means at least one time more further. 

ii) (a.b)

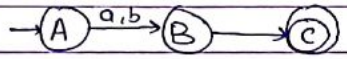


iii) a^* 

Eg → (i) (ba^*b) ⟶ baab, bab, bb.

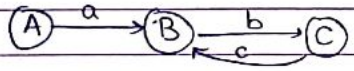


ii) $(a+b)c$



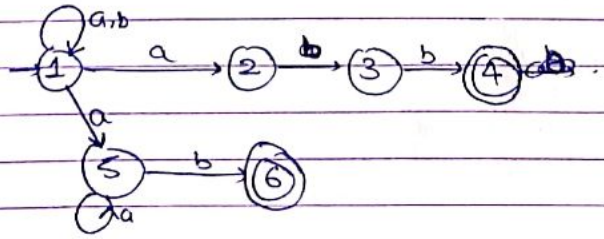
R = Q + RP
R = QP*

* iii) $a(bc)^*$ ⟶ a, abc, abc, abbc, abbcc..



56

Eg $(a+b)^*(abb+a^+b)$

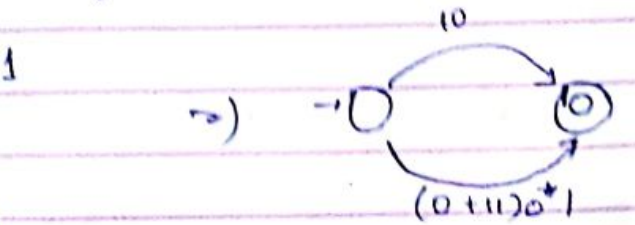


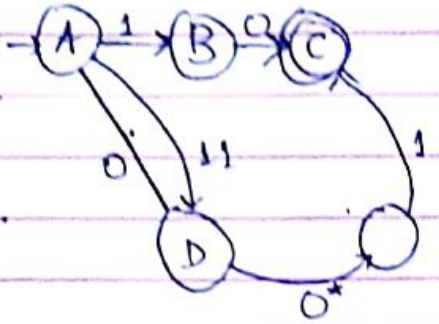
57

Q7 (Part 3)
Convert the following R.E to Equivalent F.A

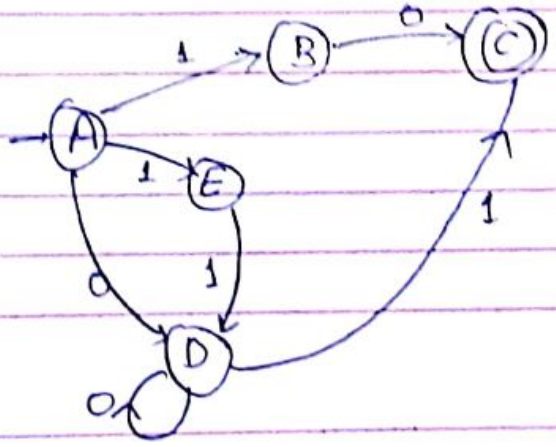
⇒ $10+(0+11)0^*1$

⇒)





⇒)



Equivalance of two finite Automata

↳ which mean on giving two final Automata you find further how you both are same or not.

↓ means two automiata perform same kind of function (the language they accept is same)

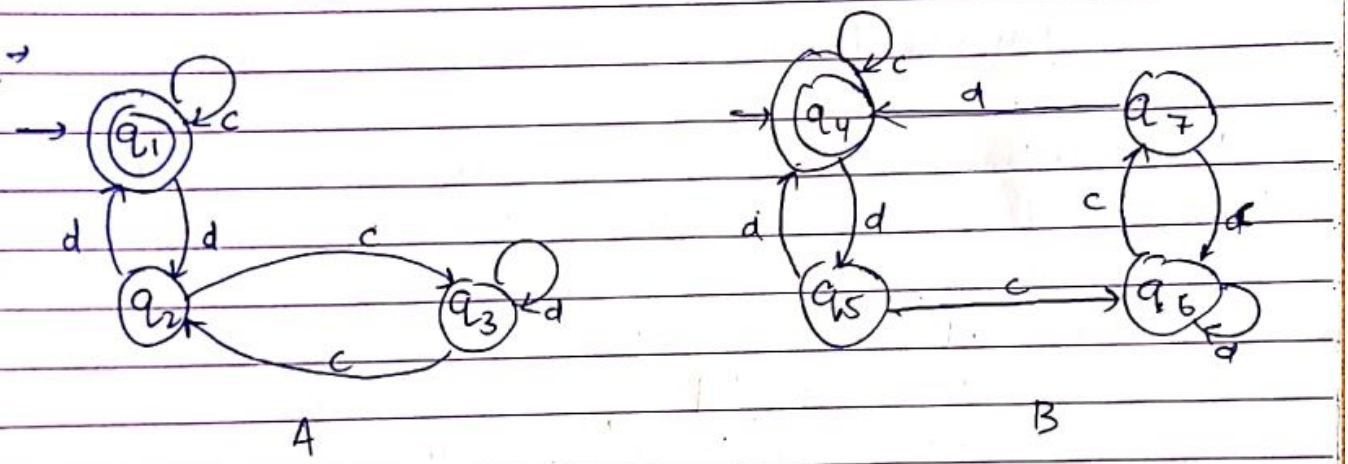
Steps to identify :- equivalance :-

① For any pair of $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$.

The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one Intermediate state and the other is final state.

2) If Intial State is Final state of one automata, then in second automata also Intial State must be final state for them to be equivalent.

Eg →



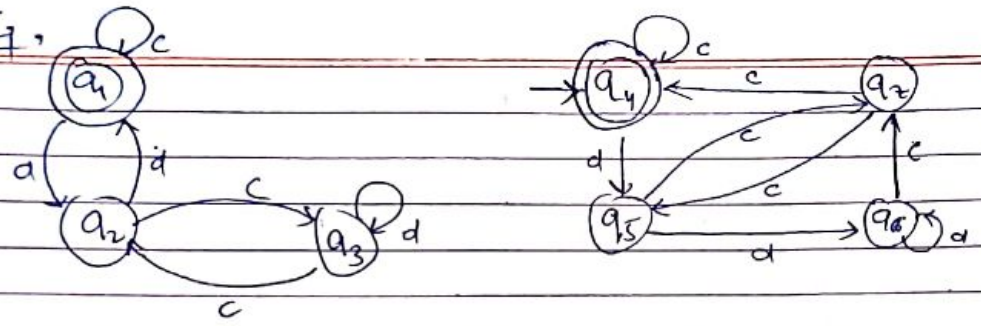
States	c	d
(q_1, q_4)	(q_3, q_4) IS FS	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_6) IS	(q_1, q_4) FS
(q_3, q_6)	(q_2, q_7) IS	(q_3, q_6) IS
(q_2, q_7)	(q_3, q_6) IS	(q_1, q_4) FS

→ A and B both are intial and final state are ~~same~~ intial state verified 2nd rule

→ and by fist rule we check all the states are (final, final) or (intermediate, intermediate)

∴ it satisfies (1) Rule.

Eg:



(60) Pumping lemma (For Regular language)

> Pumping lemma is used to prove that language is NOT REGULAR
→ It is not used to prove that R.L.

If A is a R.L, then A has a pumping length 'P' such that any string 's' where $|s| \geq P$ may be devided into 3 parts $s = xyz$ such that the following condition must be true:

These conditions are true if $s \in A$

1) $xy^iz \in A$ for every $i \geq 0$
2) $|y| > 0$
3) $|xy| \leq P$

To prove that language is not regular. (we prove using Contradiction)

→ Assume that A is Regular
→ It has to have a Pumping length (say P)
→ All string longer then P can be pumped $|s| \geq P$
→ Now find a string 's' in A such that $|s| \geq P$
→ Divide s into xyz
→ Show that $xy^iz \notin A$ some i
→ Then consider all ways that s can be devide into xyz
→ Show that none of these can satisfy all the 3 pumping conditions at the same time

→ S cannot be pumped = = CONTRADICTION

G1

Eg→ Prove $A = \{a^n b^n \mid n \geq 0\}$ is Not Regular.

Assume A is regular.

Pumping length = P

$S = a^P b^P$ ⇒ S = aaaaaaaa bbbbbbbb

x y z

Condition① P = 7

Case 1 - The y is in the 'a' part

aaaaaaaa bbbbbbb

x y z

$xy^iz \Rightarrow xy^2z$

aa aaaaaaaa bbbbbbb

11 ≠ 7 (a)

Case 2 :- The y is in the 'b' part

aaaaaaab bbbbbbb

x y z

(ii) $xy^2z \Rightarrow$

aaaaaaabb bbbbbbb

11 ≠ 7 (b)

case 3 :- The y is in the 'a' and 'b' part

aaaaaaaa bbbbbbbb

x y z

(iii) $xy^2z \Rightarrow$

aaaaa aabbaabb bbbb

Condition③

$|xy| \leq P$ P = 7

Case① 2 ≤ 7 satisfied

Case② 13 ≤ 7 Not satisfied

Case③ 9 ≤ 7 "

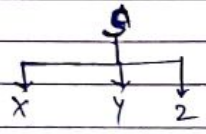
∴ We assume that A is regular bt it can satisfies all the prop ∴ A is non Regular language.

62
Q.2

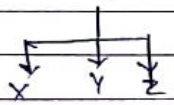
$A = \{ yy \mid y \in \{0,1\}^* \}$

Assume that A is Regular.

then it must have a pumping length $= p$



$S = 0^p 1\ 0^p 1 \Rightarrow p = 7$



Case 1) 0000000 1 00000001

X Y Z

$xy^iz \Rightarrow xy^2z$

⇓

00 0000000 1 00000001

11 7

The string does not lies in the language.

∴ It is Contradiction and it is Not a R.L.

① This not follow yy. $\notin A$

② $|y| > 0$

③ $|xy| \leq p$

$6 \leq 7$

If we assume y upto 1 then this condition is fail.

63

→ Noam Chomsky Grammar which

Grammar Type	Gr
TYPE-3	REG

→ A Grammar 'G' 4 tuples as

V = Set of Var

T = set of

S = Start

P = Productio

Eg: G = ({S, A,

production strings on V V.

V = {S, A,

T = {a, b}

S = S

P = {S →

Regular Gram

Right Linea

if productio form

A → xB

A → x

Regular Grammer :-

→ Noam Chomsky gave a Mathematical model of Grammar which is effective for writing computer lang.

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type-3	Regular Grammar	Regular Grammar	Finite State Automata

→ A Grammar 'G' can be formally described using 4 tuples as $G = (V, T, S, P)$ where,

V = Set of Variables or Non-Terminal symbol
T = Set of Terminal symbol
S = Start symbol
P = Production rules for Terminal and Non-terminals.

Eg → $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

production Rule has the form $\alpha \to \beta$ where α and β are strings on VUT and atleast one symbol of α belongs to V.

$V = \{S, A, B\}$
$T = \{a, b\}$
$S = S$
$P = \{S \to AB, A \to a, B \to b\}$

Eg, Production Rule
$S \to AB$
$\to aB$
$\to ab$

Regular Grammar :-

Right Linear
if productions are of the form
$A \to xB$
$A \to x$
where $A, B \in V$ and $x \in T$

Left Linear
if productions are of form
$A \to Bx$
$A \to x$

→ Right linear grammar if the Non-Terminal symbol is Right side of Terminal symbol.

$A \rightarrow xB$

$A, B \in V$

$x \in T$

Eg → $S \rightarrow abS \mid b$ → Right linear Grammar

$S \rightarrow Sbb \mid b$ → Left " "

(terminal symbol)

64 Derivation from a Grammar:

A set of all strings that can be derived from a grammar is said to be the language generated from that Grammar.

Eg → ① $G_1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \epsilon\})$

$S \rightarrow aAb$ [by $S \rightarrow aAb$]

$\rightarrow aaAbb$ [by $aA \rightarrow aaAb$]

$\rightarrow aaaAbbb$ ["]

$\rightarrow aaabbb$ [by $A \rightarrow \epsilon$]

Eg ② $G_2 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

$S \rightarrow AB$

$\rightarrow aB$ [by $A \rightarrow a$]

$\rightarrow ab$ [by $B \rightarrow b$]

↳ this is the only string from this grammar.

$L(G_2) = \{a, b\}$

So this is the

Eg 3 $G_3 = (\{S, A,$

B

$S \rightarrow AB$

$\rightarrow ab$

$S \rightarrow AB$

$\rightarrow abB$

$\rightarrow abb$

$L(G_3) = \{ab,$

↑ language Generated by G_3. $= \{a^n$

65 Context

→ In formal language language generate

⇒ The set of all CF accepted by Pushdo

⇒ Pushdown Autom Automata

$L(G2) = \{a, b\}$ ← this is the only string generated by the grammar

So this is the language generated by this grammar G2.

Eg3 $G3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB,\ A \to aA/a,\ B \to bB/b\})$

$S \to AB$
$\to ab$

$S \to AB$
$\to aAbB$
$\to aabb$

$S \to AB$
$\to aAb$
$\to aab$

$S \to AB$
$\to abB$
$\to abb$

$L(G3) = \{ab, a^2b^2, a^2b, ab^2\}$ ← language Gererated by G3.

$= \{a^m b^n \mid m \geq 0, n \geq 0\}$

65 Contept Free language:

⇒ In formal language theory, a Context Free language is a language generated by some Context Free grammar.

⇒ The set of all CFL is identical to the set of languages accepted by Pushdown Automata.

⇒ Pushdown Automata is more powerful than Finalstate Automata

$a^n b^n$ → is not a Regular language we see earlier. bcz it could not designed using F.S machine.

Context Free Grammar is defined by 4 tuples as
$G = \{ V, \Sigma, S, P \}$ where

V = Set of Variables and Non-Terminal Symbol
Σ = set of Terminal Symbol
S = Start Symbol.
P = Production Rule.

Production Rule of the form
$A \to a$
where, $a = \{ V \cup \Sigma \}^*$ and $A \in V$
↓ union.

Eg → For generating a language that generates equal number of a's and b's in the form $a^n b^n$. the context free Grammar will be defined as

$$G = \{ (S, A), (a, b), S, (S \to aAb, A \to aAb \mid \epsilon) \}$$

↳ Because context free language is High level language it generate $a^n b^n$.

$S \to aAb$
$\to aaAbb$ (by $A \to aAb$)
$\to aaaAbbb$ (by ")
$\to aaabbb$ (by $A \to \epsilon$)
$\to a^3 b^3 \Rightarrow a^n b^n$

72 Method to find whether a String belongs to a Grammar or NOT

1) Start with the start symbol and choose the closest production that matches to the given string.
2) Replace the variables with its most appropriate production. Repeat the process until the string is generated or until no other production one left.

Eg→ Verify whether the Grammar $S \to 0B/1A$, $A \to 0/0S/1AA/\wedge$, $B \to 1/1S/0BB$ generate the string 00110101

1) $S \to 0B$ ($S \to 0B$)
$\to 00BB$ ($B \to 0BB$)
$\to 001B$ ($B \to 1$)
$\to 0011S$ ($B \to 1S$)
$\to 00110B$ ($S \to 0B$)
$\to 001101S$ ($B \to 1S$)
$\to 00110101B$ ($S \to 0B$)
$\to 00110101$ ($B \to 1$)

∴ We generate the string given using given grammar.

Eg→ Verify whether the grammar $S \to aAb$, $A \to aAb/\wedge$ (empty symbol) generates the string aabbb

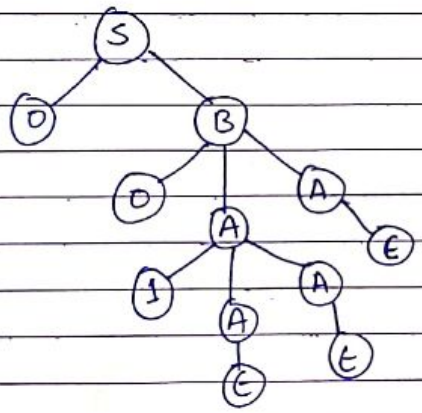
$S \to aAb$ ($A \to aAb$)
$\to aaAbb$
$\to aabb$ ($A \to \wedge$)
$\to aaaAbbb$ ($A \to aAb$)
$\to aaabbb$ ($A \to \wedge$)

∴ this string is not belongs to this grammar.

73 Derivation Tree:

→ A derivation Tree is an ordered rooted tree that graphically represents the semantic info. of string derived from a context free grammar.

Ex For the Grammar $G = \{V, T, P, S\}$ where $S \to 0B$, $A \to 1AA/\epsilon$, $B \to 0AA$



Root Vertex: Must be labelled by the start symbol.

Vertex: Labelled by Non-Terminal Symbol

leaves: Labelled by Terminal symbol or ϵ

Left Derivation Tree

→ Applying production to the leftmost variable in each step.

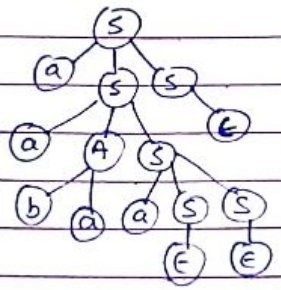
Eg → Generate string aabaa

Grammar: $S \to aAS/ass/\epsilon$

$A \to SbA/ba$

Right Derivation Tree.

→ Applying production to the rightmost variable in each step.



aabaa

⇒ $S \to ass$ [$S \to aAS$]

$\to aaAss$ [$A \to ba$]

$\to aabass$ [$S \to ass, S \to \epsilon$]

⇒ aabaa

Right Deriv

(a)

74 Am

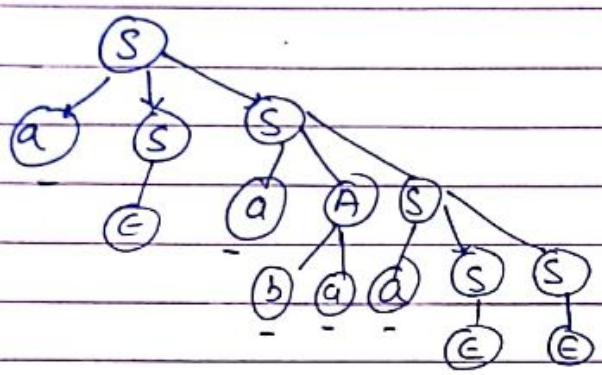
A gramar ... or more deriv... two or more ...

Ex → G = ({S ... S → S+S | ... generate ...

(1) S →

∴

Right Derivation Tree:



a ab aa

74 Ambiguous Grammar :-

A grammar is said to be Ambiguous if there exist two or more derivation tree for a string w (that means two or more left derivation tree).

Ex → G = ({S}, {a+b, +, *}, P, S } where P consist { }
S → S+S | S*S | a | b The string a+a*b can be generated as.

① S → S + S
→ a + S
→ a + S * S
→ a + a * S
→ a + a * b

② S → S * S
→ S + S * S
→ a + S * S
→ a + a * S
→ a + a * b

In both the case we use left derivation tree
∴ G is Ambiguous Grammar.

75

Simplification of Context Free Grammar :-

In CFG sometimes all the production rules and symbols are not needed for the derivation of string. Besides this, there may also be some NULL production and UNIT productions.
Elimination of these productions and symbols is called Simplification of CFG.

Steps :-

① Reduction of CFG ② Removal of Unit productions
③ Removal of NULL production.

Reduction of CFG

Phase 1: Derivation of an equivalent grammar G', from the CFG, G, such that each variable derives some terminal string.

Derivation Procedure :-

Step 1 :- Include all symbols W_1, that derives some terminal and initalize $i = 1$.
Step 2: Include symbol W_{i+1}, that derives W_i
Step 3: Increment i repeat step 2, until $W_{i+1} = W_i$
Step 4: Include all production rules that have W_i in it

Phase 2: Derivation of an equivalent grammar G'', from the CFG, G' such that each symbol appears in a sentential form.

Step 1: Include

Eg:- Find a reduced
having production

V → A,C
Σ → a
S → S
P → S

Phase 1:

T = { a, c,
W_1 = { A,
W_2 = { A,
W_3 = { A,

G' = {(A, C, E

P: S → AC
Production Rule

Phase 2: Y_1 = {S}
Y_2 = {S, A
Y_3 = {S A
Y_4 = {S,

G'' = { (A,

P: S → A C

symbols are not
there may also

called

productions

i', from the CFG,
terminal string.

ne terminal and

Wi
li in it

as G'', from
appears in

Eg:- Find a reduced grammar equivalent to Grammar G, having production rules P: S→AC/B, A→a, C→c/BC, E→aA/e

V→ A,C, B, E
Σ→ a,c,e
S→ S
P→ S→Ac/B

Phase 1:

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

$W_1 = \{A, C, E\}$

$W_2 = \{A, C, E, S\}$

$W_3 = \{A, C, E, S\}$

$G' = \{(A, C, E, S), (a, c, e), P, (S)\}$

P: S → AC, A → a, C → c, E → aA/e

Production Rule

see in this G'

Phase 2: $Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$

$Y_3 = \{S, A, C, a, c\}$

$Y_4 = \{S, A, C, a, e\}$

$G'' = \{(A, C, S), \{a, c\}, P, \{S\}\}$

P: S → AC, A → a, C → c

Simplified and Reduce form of CFG.

Simplification of Context Free Grammar:

Removal of Unit Production :-

Any production rule of the form $A \to B$ where $A, B \in$ Non terminal is called Unit production.

Procedure of Remove :-

Step 1 → To remove $A \to B$, add production $A \to x$ to the grammar rule whenever $B \to x$ occurs in the grammar. [$x \in$ Terminal, x can be Null]

Step 2 : Delete $A \to B$ from the grammar

Step 3 : Repeat from Step 1 until all units productions are removed.

Eg :- Remove Unit Production From the Grammar

P : $S \to XY$, $X \to a$, $Y \to z \mid b$, $z \to M$, $M \to N$, $N \to a$

$z \to M$, ~~$M \to$~~ , $M \to N$

i) $M \to N$

Since $N \to a$, we add $M \to a$

P : $S \to XY$, $X \to a$, $Y \to z \mid b$, $z \to M$, $M \to a$, $N \to a$

ii) $z \to M$

Since $M \to a$, we add $z \to a$

P : $S \to XY$, $X \to a$, $Y \to z \mid b$, $z \to a$, $M \to a$, $N \to a$

iii) $Y \to z$, we change $Y \to a$

P : $S \to XY$, $X \to a$, $Y \to a \mid b$, $z \to a$, $M \to a$, $N \to a$

Remove the unreachable symbol →

P: $S \to XY$, $X \to a$, $Y \to a \mid b$

Removal of Null Production:

In a CFG, a non-Terminal Symbol 'A' is nullable variable if there is a production $A \to \epsilon$ or there is a derivation that starts at 'A' and lead to ϵ. (like $A \to \dots \to \epsilon$)

Procedure:

Step 1: To remove $A \to \epsilon$, look for all productions whose right side contains A.

Step 2: Replace each occurance of 'A' in each of these productions with ϵ

Step 3: Add the resultant productions to the Grammar

Eg → Remove Null Productions from the following

$S \to ABAC$, $A \to aA/\epsilon$, $B \to bB/\epsilon$, $C \to c$

$A \to \epsilon$, $B \to \epsilon$

i) To element $A \to \epsilon$

$S \to ABAC$

$S \to ABC \mid BAC \mid BC$

$A \to aA$

$A \to a$

~~$S \to BC$~~

ii) ~~$B \to \epsilon$~~ / To element

~~$S \to ABAC$~~

New production: $S \to ABAC \mid ABC \mid BAC \mid BC$, $A \to aA \mid a$

$B \to bB \mid \epsilon$, $C \to c$

ii) To eleminate $B \to \epsilon$

$S \to ABAC$, $S \to AAC \mid AC \mid C$

~~$S \to A\,BAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$~~

$B \to bB$

$B \to b$

New production :

$S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$

$A \to aA \mid a$

$B \to bB \mid b$

$C \to c$

78

Chomsky Normal Form :

In Chosky Normal form (CNF) we have a restriction on the length of RHS ; which is ; element in RHS should either be two variable or a Terimal.

→ A CFG is a CNF if the production are in the following form :

$$A \to a$$
$$A \to BC$$

Where A, B, C are non-Terminal and a is terminal.

CFG to CNF

Step1 : If the start symbol S occur on some right side, create a new start symbol S' and a new production $S' \to S$

Step2 : Remove Null productions.

Step3 : Remove unit productions.

Step4 : Replace

where

two or

Step5 : If

and

the

Eg →

Soln

1)

2) Rem

After

After

3) Rem

Step 4: Replace each production $A \to B_1 \dots B_n$ where $n \geq 2$, with $A \to B_1 C$ where $C \to B_2 \dots B_n$. Repeat this step for all production having two or more symbol on the right side.

Step 5: If the right side production is form $A \to aB$ where ~~is the~~ to then the production is replaced by $A \to XB$ and $X \to a$. Repeat the steps for every production which is of the form $A \to aB$.

Eg → P: $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$

Sol:

1) P: $S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$

2) Remove Null prod. $B \to \epsilon$, $A \to \epsilon$

After Removing $B \to \epsilon$: P: $S' \to S$, $S \to ASA \mid aB \mid a$, $A \to B \mid S \mid \epsilon$, $B \to b$

After Removing $A \to \epsilon$: P: $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA \mid S$, $A \to B \mid S$, $B \to b$.

3) Remove Unit production:

Removing $S \to S$: P: $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA$

Removing $S' \to S$: P: $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA$

$S' \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to b \mid S$, $B \to b$

Removing $A \to B$: P: $S' \to S$, $S' \to ASA \mid aB \mid a \mid AS \mid SA$,

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to b$, $B \to b$

Removing $A \to S$: P: $S' \to S$ $S' \to$ —,,—

$S \to$ —,,—

$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$

$B \to b$

4) Find out more than TWO variable in RHS

$S' \to ASA$, $S \to ASA$, $A \to ASA$

After Removing these : $S' \to AX \mid aB \mid a \mid AS \mid SA$

$S \to AX \mid aB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$

$B \to b$

$X \to SA$

5) Now change productions $S' \to aB$, $S \to aB$, $A \to aB$

P: $S \to AX \mid YB \mid a \mid AS \mid SA$

$S' \to AX \mid YB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$

$B \to b$

$X \to XA$

$Y \to a$

↑ no this is CNF (Chomsky Normal form)

Geibeach Normal form :-

A CFG is GNF frm if the production in the form $\to$ $A \to b$

$A \to bC_1 \dots C_n$] For GNF

C, $A \to$ Non-terminal , $b \to$ Terminal

① Check if the CFG has Unit or Null production Remove it.

② Check whether CFG is Already in Chomsky NF and convert it to CNF if it is not.

③ Change the name of Non-Terminal Symbol into A_i in order

Eg :- $S \to CA \mid BB$ → By Assending order of this.
$B \to b \mid SB$
$C \to b$
$A \to a$

Replace the name

⇒ Step 3 → direct, ① Swith A_1
② C with A_2
③ A with A_3
④ B with A_4

we get →

$A_1 \to A_2 A_3 \mid A_4 A_4$
$A_4 \to b \mid A_1 A_4$
$A_2 \to b$
$A_3 \to a$

Step 4: Alter the rule so that Non-Term are in assending order, such that If Production is of the form $A_i \to A_j x$ then $i < j$ and should never $i \geq j$

$A_4 \to b \mid A_1 A_4$

Replace → $A_4 \to b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$
$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4$
$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4$

Left Recursion

Step 5 → Remove left Recursion
Introduce a New Variable to remove the left Recursion

$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4$
$Z \to A_4 A_4 Z \mid A_4 A_4$
$A_4 \to b \mid b A_3 A_4 \mid b Z \mid b A_3 A_4 Z$

Pumping lemma (for CFL)

Pumping lemma (for CFL) is used to prove that a language is NOT Context Free.

» If A is a Context free language, then A has a pumping length 'p' such that any string 's', where $|s| \geq p$ may be divided into 5 pieces $s = uvxyz$ such that some condition must true :-

1) $uv^i xy^i z$ is in A for every $i \geq 0$
2) $|vy| > 0$
3) $|vxy| \leq p$

⇒ So that none of these can satisfy all the 3 pumping condition at the same time.

Eg → Show that $L = \{ a^N b^N c^N \mid N \geq 0 \}$ is Not Context Free

1) → Assume L is Context free.
2) → L must have pumping length (P)
3) Now we take a string S such that $S = a^P b^P c^P$
4) We divided S into parts uvxyz

Eg P = 4 So ⇒ $S = a^4 b^4 c^4$ •

Case I: vfy contains only one type of symbol

aaaa abbbb cccc
u v x y z

⇒ $uv^i xy^i z$ (i = 2)

$uv^2 xy^2 z$

⇒ a aaaa abbbbc cc

∴ ⇒ $a^6 b^4 c^5$ is not in language form $a^N b^N c^N$

Eg a^6

Condition

Case II: Eit

aaaa b
u v

⇒ aa aabb

⇒ $a^4 b^2 a^2 b^2$

∴ $a^N b^N$

Eg 2 Show

→ Assu
→ L m
→ Now
→ We

Eg → P = 5

Case 1: vx

00000
u

uv^i x

⇒ uv^2 x

$a^6 b^4 c^5 \notin L$

Condition 1 is not satisfied.

Case II : Either V or Y has more than one kind of symbol.

aaaa bbbb cccc

u v x y z

$uv^i xy^i z$

$i=2 \quad uv^2 xy^2 z$

→ aa aabbaabb b bb cccc

→ $a^4 b^3 a^2 b^2 b^3 c^4 \notin L$

∴ $a^N b^N c^N$ condition 1. is not satified

Eg 2 Show that $L = \{ ww \mid w \in \{0,1\}^* \}$ is Not a Context Free

→ Assume Context free
→ L must have pumping length (P)
→ Now we take a string S such that $S = 0^P 1^P 0^P 1^P$
→ We divide S into parts uvxyz

Eg → $P=5$ So, $S = 0^5 1^5 0^5 1^5$

Case 1 : vxy does not standdle a bondary.

Boundaries

00000 | 11111 | 00000 | 11111

u vxy z

$uv^i xy^i z \qquad i=2$

$\Rightarrow uv^2 xy^2 z$

$a^6 b^4 c^5 \notin L$

Condition 1. is not satisfied.

Case II: Either V or Y has more than one kind of symbol.

aaaa bbbb cccc

u v x y z

$uv^i xy^i z$

$i=2 \quad uv^2xy^2z$

⇒ aa aabbaabb b bb cccc

⇒ $a^4b^2a^2b^2b^3c^4 \notin L$

∴ $a^N b^N c^N$ condition 1. is not satified

Eg 2 Show that $L = \{ ww \mid w \in \{0,1\}^* \}$ is Not a Context Free

→ Assume Context free
→ L must have pumping length (P)
→ Now we take a string S such that $S = 0^P 1^P 0^P 1^P$
→ we divide S into parts uvxyz

Eg → P=5 So, $S = 0^5 1^5 0^5 1^5$

Case 1: vxy does not standdle a bondary.

Boundaries

00000 11111 00000 11111

u vxy z

$uv^i xy^i z \quad i=2$

⇒ uv^2xy^2z

$\Rightarrow$ 1111
$\Rightarrow$ vxy

uv^2xy^2z

$\Rightarrow$ 00000 11 1 11 1000001111
v^2 y^2

$0^5 1^7 0^5 1^5$ does not belongs to language L.
$\neq$

Case 2a: vxy straddles the first boundary.

00000 1111 1 1 00000 1 1 1 1 1
u v x y z

$\Rightarrow uv^ixy^iz \Rightarrow i=2 \quad uv^2xy^2z$

$\Rightarrow 0^7 1^7 \quad 0^5 1^5$ does not belongs to language
$\neq$

Case 2b: vxy straddles the third boundary.

00000 1111 1 00000 1 1 1 1 1
u vxy z

v = 00
x = 1
y = 11

$\Rightarrow uv^2xy^2z$
$\Rightarrow 0^5 1^5 \quad 0^7 1^7 \notin L$
$\neq$

Case 3: vxy straddles the midpoint

00000 1111 1 00000 1 1 1 1 1
u vxy z

$\Rightarrow uv^2xy^2z$
$\Rightarrow 0^5 1^7 \neq 0^7 1^5 \notin L$

Pushdown Automata:

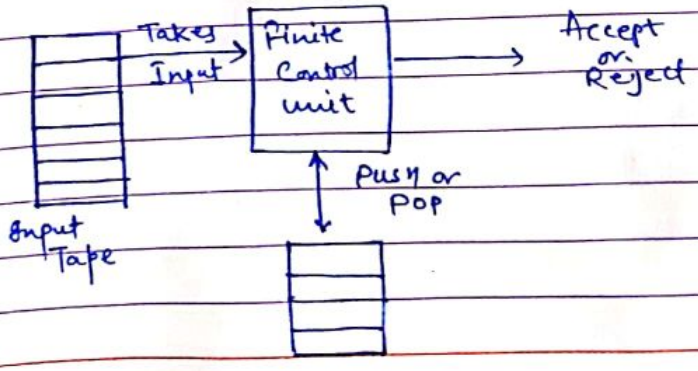
A pushdown Automata (PDA) is a way to implement a Context Free Grammar in a similar way we design Finite Automata for Regular Grammar.

→ It is more Powerful than FSM
→ FSM has a very limited memory but PDA has more memory.
→ PDA = Finite State Machine + A stack.
↳ ∞ memory, so this PDA is more powerful than FSM

A stack is a way we arrange elements are on top of another
A stack does two basic operations:
PUSH : add at the Top
POP : Top element of stack is read and removed.

A Pushdown Automata has 3 components:

1) An input tape
2) A Finite Control unit
3) A stack with infinite size



PDA (Formal definition)

A Pushdown Automata is formally defined by 7 tuples as shown below:

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Where:

- $Q \rightarrow$ Finite set of states
- $\Sigma \rightarrow$ A finite set of Input symbol
- $\Gamma \rightarrow$ A finite state of Alphabet
- $\delta \rightarrow$ The transition function
- $Z_0 \rightarrow$ The start Stack symbol
- $q_0 \rightarrow$ The state start
- $F \rightarrow$ The set of Final / Accepting states.

→ δ takes as argument a tiple $\delta(q, a, X)$ where:

i) q is a state in Q

ii) a is either an Input Symbol in Σ or $a = \epsilon$

iii) X is a stack symbol, that is member of Γ

→ The output of δ is finite set of pairs (p, γ) where:

p is a new state

γ (gamma) is a string of stack symbol that replaces X at top of the stack.

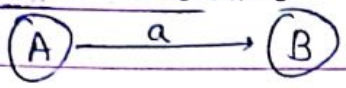
Eg → If $\gamma = \epsilon$ then the stack is popped

If $\gamma = X$ then Stack is unchanged

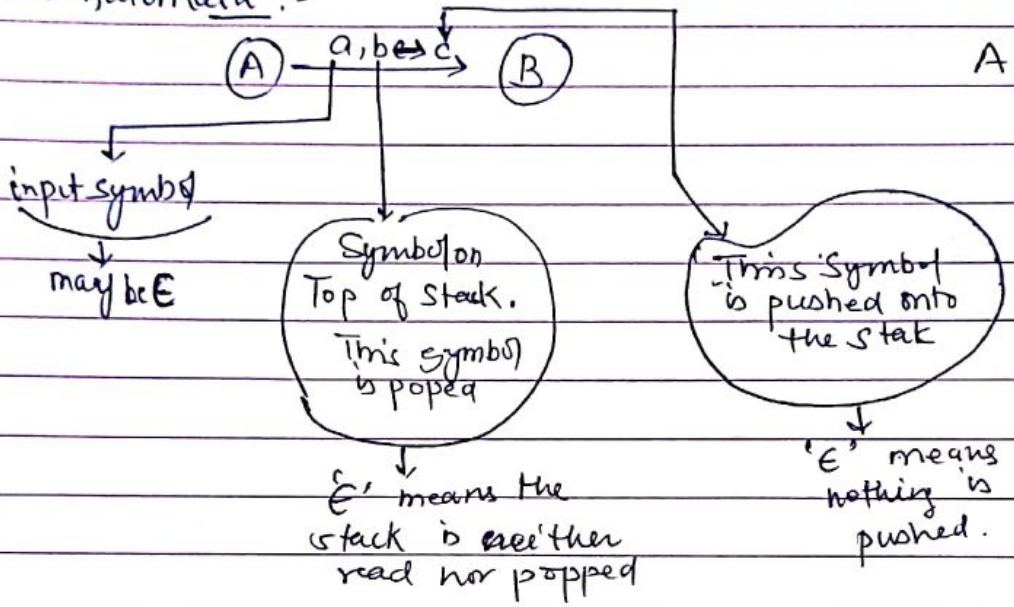
If $\gamma = YZ$ then X is replaced by Z and Y is pushed into the stack

87: Pushdown (Graphical Notation)

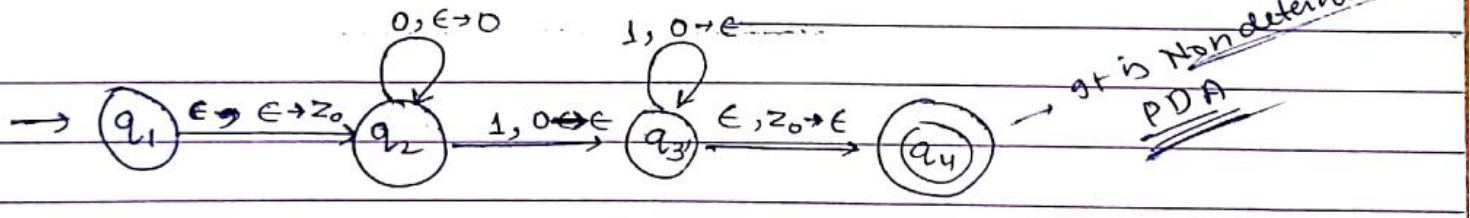
Finite State Machine

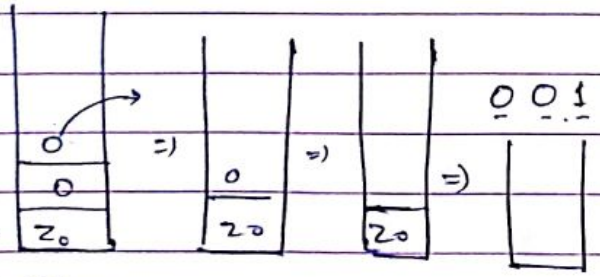


Pushdown Automata :-



Eg:- Construct a PDA that accept $L = \{ 0^n 1^n \mid n \geq 0 \}$

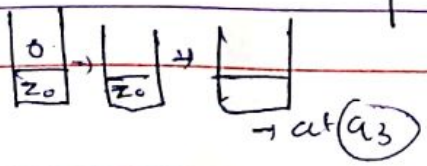




$ → Bottom most element of stack

In PDA there is 2 ways to string is accepted 1) - is to reach final state
2) case ~~2nd~~ is to find stack is empty.

or



94:

Turning Machine:-

→ Recursively Enumerable Languages.

→ Content Free Language

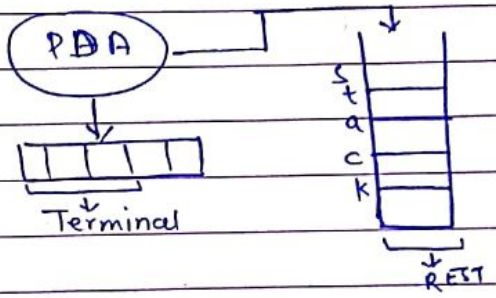
→ Regular Languages

FSM → The i/p string

a	a	a	a	b	a	b	b

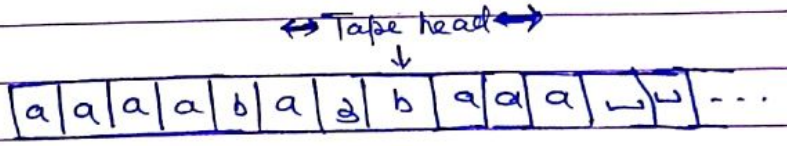
PDA → The i/p string

→ A stack



Turing Machine:

→ A Tape



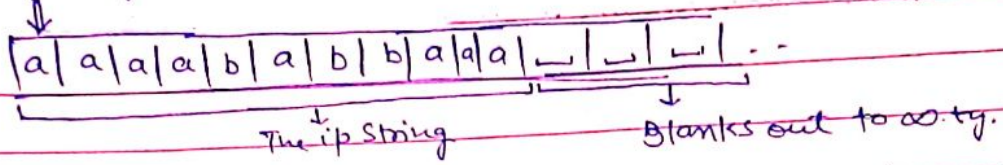
Tape Alphabets : $\Sigma = \{0, 1, a, b, x, z.\}$

The Blank ␣ is a special symbol ␣ $\notin \Sigma$

The blank is a special symbol used the fill the infinite tape

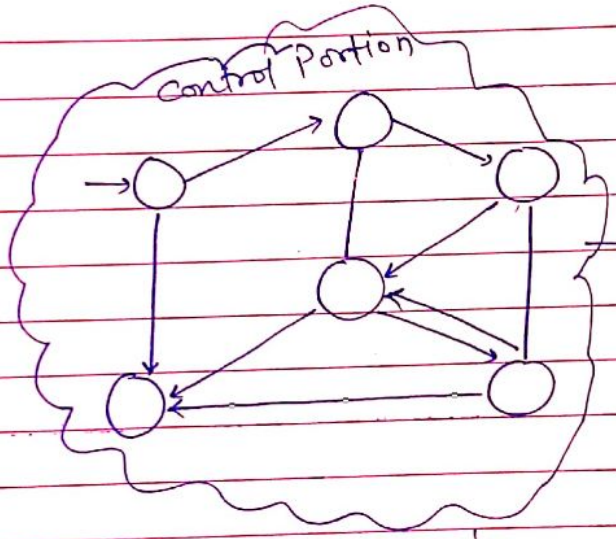
→ Tape is an infinite sequence and the tape head is at the position where the current controled is present and empty sequence is fill with blank symbol.

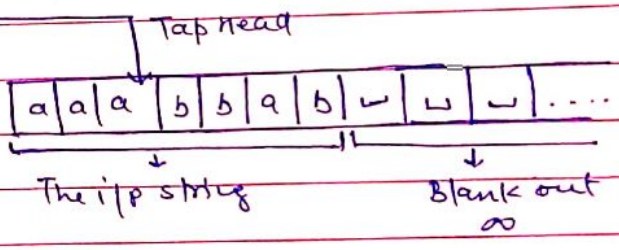
Initial Configuration :



Operations on the tape :

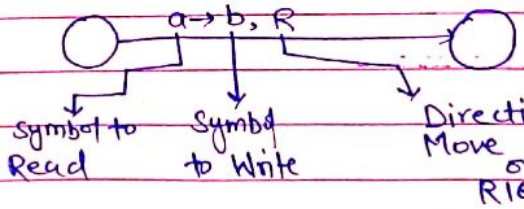
→ Read / scan symbol below the Tape Head
→ Update / Write a symbol below the Tape Head
→ Move the Tape Head one step LEFT
→ Move the " " RIGHT



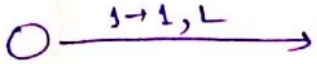


The Control portion similar to FSM or PDA
The Program
It is deterministic

If we are at the left of the tape, and trying to Move LEFT, then do not move stay at the left end.



⇒ If you dont want to update the cell, JUST WRITE THE SAME SYMBOL.



Rule of operation -2:

→ Control is with a sort of FSM
→ Initial state
→ Final States: (there are two final states) ← in Turning Mah.
    1) The ACCEPT STATE
    2) The REJECT STATE

→ Computation can either.
    1) HALT and ACCEPT
    2) HALT and REJECT
    3) LOOP (the machine fail to HALT)

96

Formal Definition:

A turning Machine can be defined as a set of 7 tuples.

$$(Q, \Sigma, \Gamma, \delta, q_0, b, F)$$

Q → Non empty set of states
Σ → Non Empty set of Symbol (i/p symbol)
Γ → Non empty set of Tape symbol
δ → Transition function defined as
$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$
q_0 → Initial States
b → Blank symbol
F → Set of final states (Accept state & Reject state)

Thus, the production rule of Turning Machine will be written as $\delta(q_0, a) \rightarrow (q_1, Y, R)$

at q_0 — get i/p a — goto q_1 — with Y — move R, L

Turing's ... be ... perfor...

few a...

1) M... by ...

2) ... computer...

A lan... mac...

Eg (1)

Turing's Thesis: States that any computation that can be carried out by mechinal means can be performed by some turing Machine.

Few arguments for accepting this theis are:

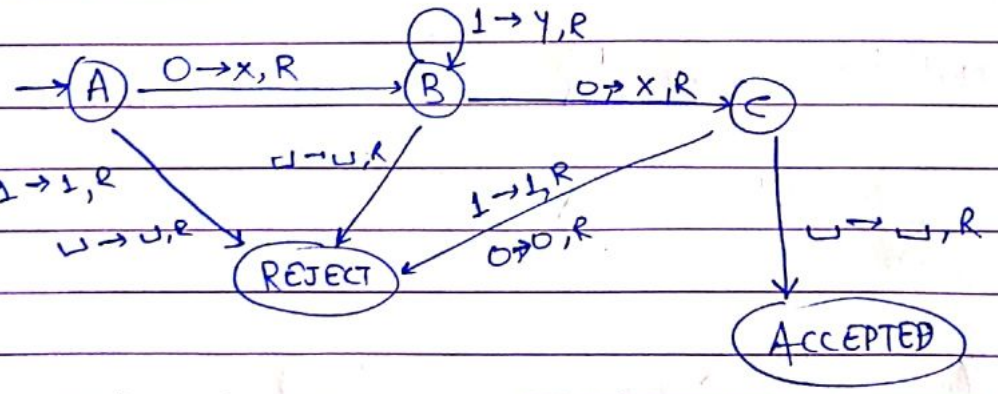
1) No one has yet been able to suggest a problem solvable by what we consider on algorithm, for which a Turing Machine perform program cannot be written.

2) Anything that can be done on existing digital computer can also done by Turning Machine.

A language is accepted by turning machine than that machine is called Recursively Enumrable.

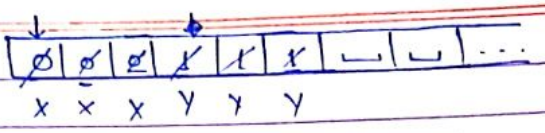
Eg ① Design Turning Machine which Recognize the Lang.

$L = 01^*0$



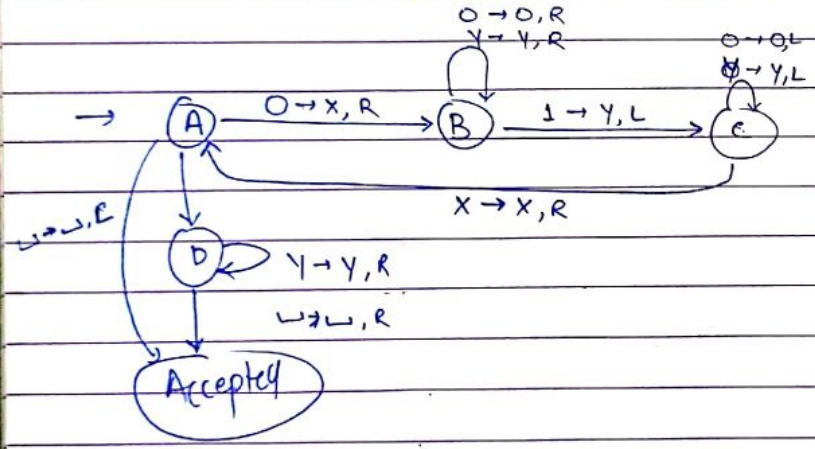
Eg 0110

Eg2:

$L = 0^N 1^N$ → 

Algorithm:

- Change 0 to 'X'
- Move Right to First '1'
  If none: Reject
- Change "1" to 'Y'
- Move LEFT to leftmost "0"
- Repeated the above steps until no more '0's
- make sure no more '1's remain.



Given Palindrome → PDA

$L = \{ w w^R \mid w = (a+b)^+ \}$ (Reverse; → it cannot be empty)

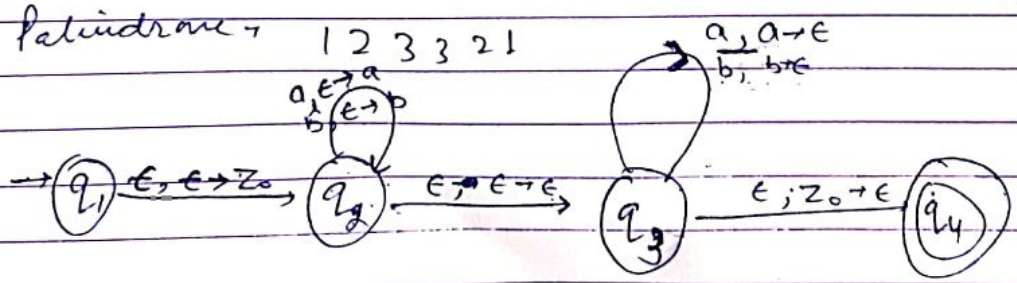
Palindrome: 1 2 3 3 2 1

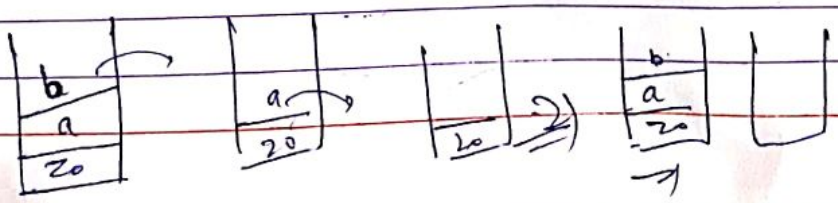
a b b a

a b a b

↓

No processed





Turning Machine (Even Palindrome)

$\Sigma = \{a, b\}$