__/__/____

Module 1: Introduction to the theory of computation and Finite Automata.

Module 2: Regular Language, Regular Grammar and properties of regular Language.

Module 1 Revision.

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots \cup \Sigma^n$

$\Sigma^* = \{\lambda\} \cup \{0,1\} \cup (00, 01, 10, 11) \cup \ldots$

(O) → Double circle indicates final state

(2) → starting pt of state

Every DFA can be defined using five tuples

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

Q → set of inputs of all states

Σ → inputs

q_0 → start state/initial state (1) → S

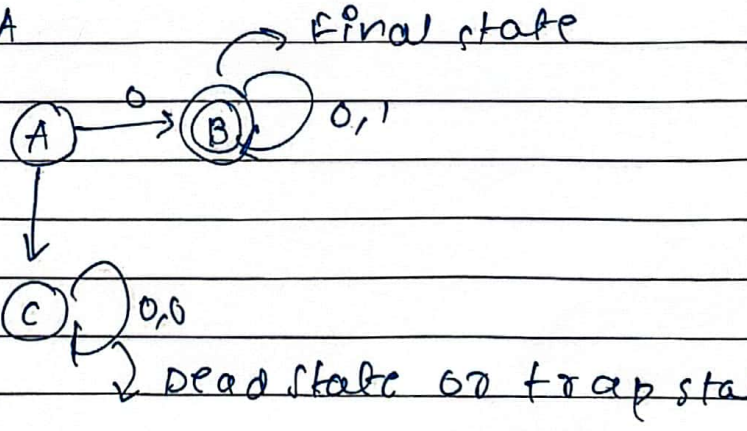
F → set of final state (O) → F

δ → Transition function from $Q \times \Sigma \to Q$

$Q = \{A, B, C, D\}$ $F = \{D\}$

$\Sigma = \{0, 1\}$

$q_0 = A$

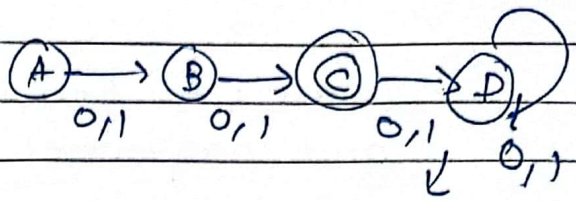


__/__/____

$\Sigma = \{0,1\}$ of length 2

$L = \{01, 00, 11, 10\}$

$\Sigma = \{0,1\}$



Regular - Languages

→ A Language is said to be Regular Language if and only if some finite state machine recognize it

Not Regular - Languages

→ Not recognized by Finite state - Machines

→ Requires Memory

✱ Memory of FSM is very - Limited

✱ It cannot store or count

Example :- $a^N b^N$

aaa bbb → cannot be counted FSM

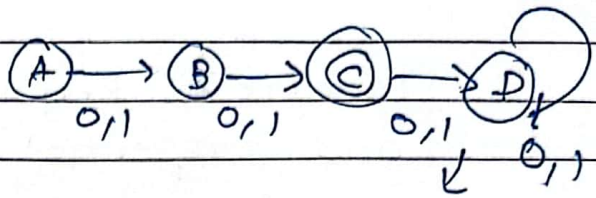
Not a regular - Language

__/__/____

$\Sigma = \{0,1\}$ of length 2

$L = \{01, 00, 11, 10\}$

$\Sigma = \{0,1\}$



Regular - Languages

→ A Language is said to be Regular Language if and only if some finite state machine recognize it

Not Regular - Languages

→ Not recognized by finite state - Machine

→ Requires Memory

* Memory of FSM is very - Limited
* It cannot store or count

Example :- $a^N b^N$

aaa bbb → cannot be counted FSM

Not a regular - Language

__/__/____

operations on Regular Languages

Union :- $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

concantenation - $A \circ B \{xy \mid x \in A \text{ and } y \in B\}$

STAR - $A^* = \{x_1, x_2, x_3 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

Eg:- $A = \{pq, r\}$ $B = \{t, uv\}$

1. Union operation $A \cup B = \{pq, r, t, uv\}$
2. concantenation $A \circ B = \{pqt, pquv, rt, ruv\}$
3. STAR - $A^* = \{\varepsilon, pq, r, pqr, rpq, pqpq, rr, pqpqpq, rrr \ldots\}$

Theorem 1: The class of Regular languages is closed under union.

Theorem 2 :- The class of Regular languages is closed under concantenation

__/__/____

Regular Expressions are used for representing certain sets of strings in an algebraic function.

Rules

1) Any terminal symbol $\in \Sigma$ a, b, c, $\wedge$ ϕ Including $\wedge$ and ϕ are regular exp.

2) Union of two regular expressions is also a regular expression

R_1, R_2 $R_1 + R_2$ $R_1 \rightarrow$ Regular

$R_2 \rightarrow$ Regular

$R_1 \cup R_2 \rightarrow$ Regular

3) The concatenation of two regular expression is also a regular expression. $R_1, R_2 \rightarrow (R_1 \cdot R_2)$

4) iteration (closture) of a regular expression is also a regular expression

$R \rightarrow R*$ at = $\wedge, a, a, aaa$

5) Regular expression over Σ are precisely those obtained recursively by the application of the above rules once or several times

$L(R_1 + R_2) = L(R_1) \cup L(R_2)$

$L(R_1 \cdot R_2) = L(R_1)\, L(R_2)$

$L((R_1)) = L(R_1)$ $\quad L(R_1^*) = (L(r_1))^*$

Regular Expression - Examples:-

1) $\{0, 1, 2\}$ 0 or 1 or 2

$R = 0 + 1 + 2$ $\quad + \rightarrow$ or symbol

2) $\{\wedge, ab\}$

$R = \wedge ab$

3) $\{abb, a, b, bba\}$

$R = abb + a + b + bba$

4) $\{\wedge, 0, 00, 000 \ldots\}$ closure of 0

$R = 0^*$

5) $\{1, 11, 111, 1111\}$

$R = 1^+$

Identities of Regular Expression

$\emptyset + R = R$

$\emptyset R + R\emptyset = \emptyset$

$\varepsilon R = R\varepsilon = R$

$\varepsilon^* = \varepsilon$ and $\emptyset^* = \varepsilon$

$R + R = R$

$R^* R = R^*$

$RR^* = R^* R$

$(R^*)^* = R^*$

$\varepsilon + RR^* = \varepsilon + R^* R = R^*$

$(PQ)^* P = P(QP)^*$

$(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$

$(P + Q)R = PR + QR$

$R(P + Q) = RP + RQ$

* (kleene closure)

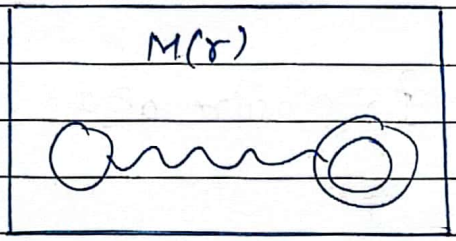
\+ (positive closure)

. concatenation

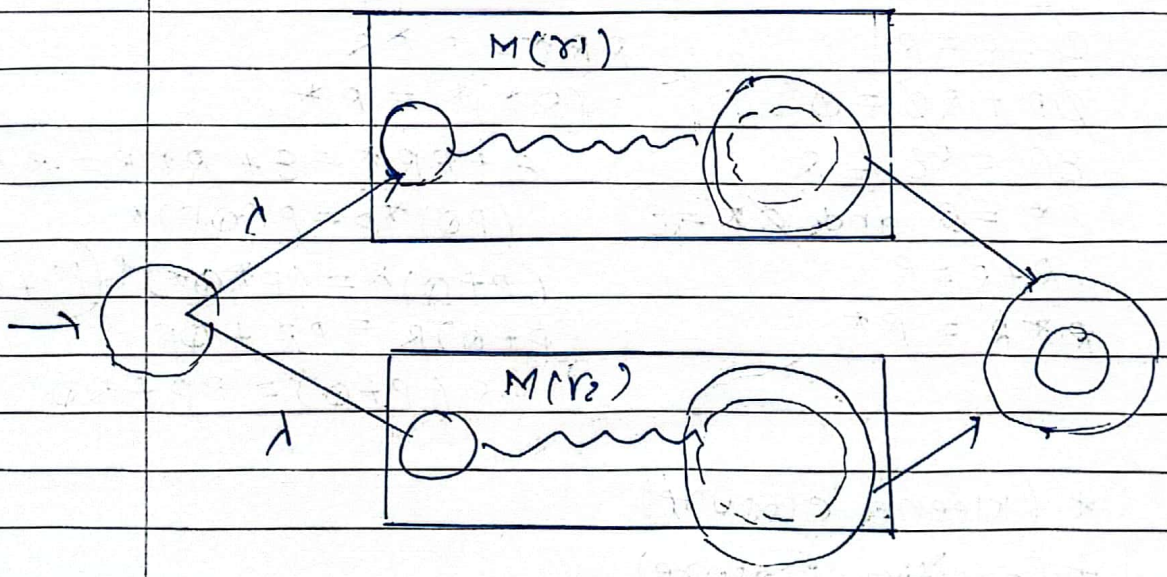
: union.

* Connection between Regular Expressions and Regular Language

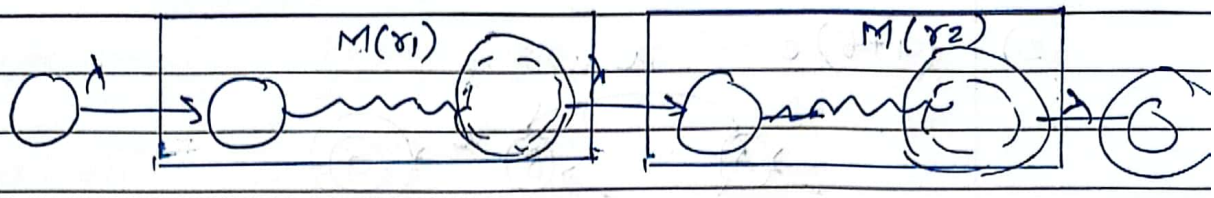
For Every regular language there is a regular expression and for every regular expression there is a regular language.



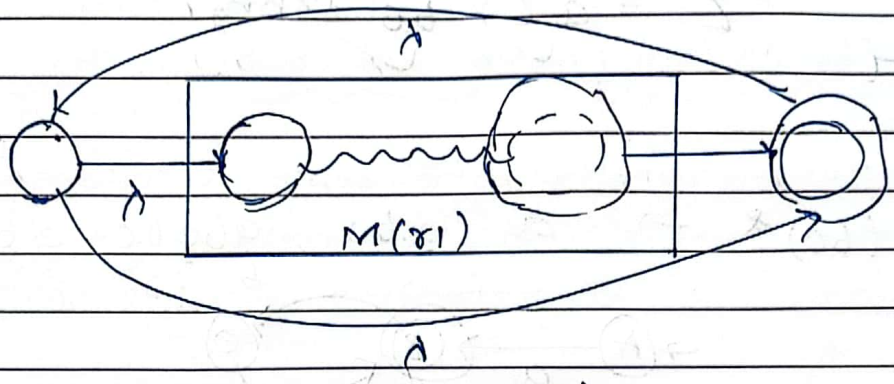
schematic rep. of an non-determistic finite automata accepting L(r)



Automation for $L(r_1 + r_2)$

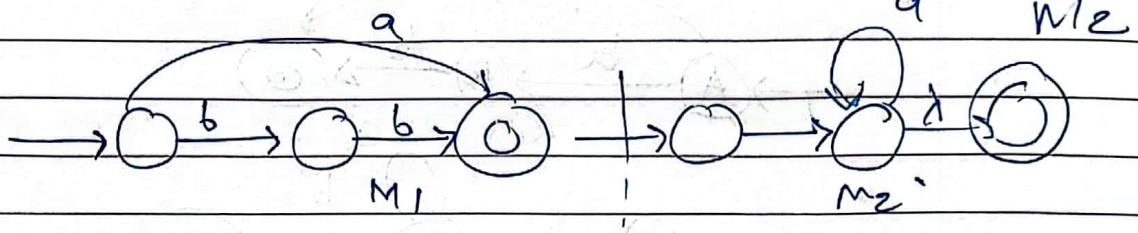


Automation for $L(r_1, r_2)$



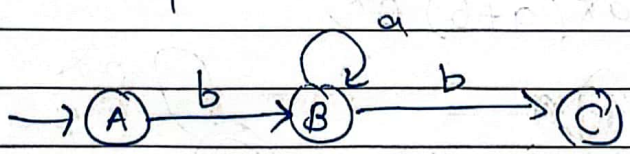
Automation for $L(r_1^*)$

find an nfa that accepts $L(r)$
where $r = (a+bb)^*(ba^* + \lambda)$



a) $ba^*b \rightarrow bb, bab, baab$

$a^* = \{\lambda, a, aa, aa, aaa\}$



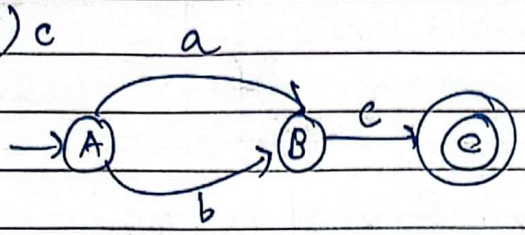
$ba^*b = \{bb, bab, baab\}$

a^* → closure indicates self loop

$a = \{\cdot$

$a \cup b =$

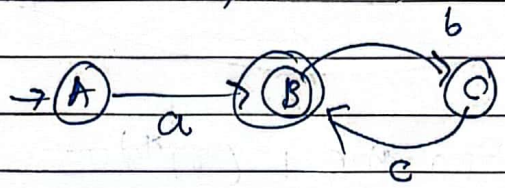
2) $(a+b)c$



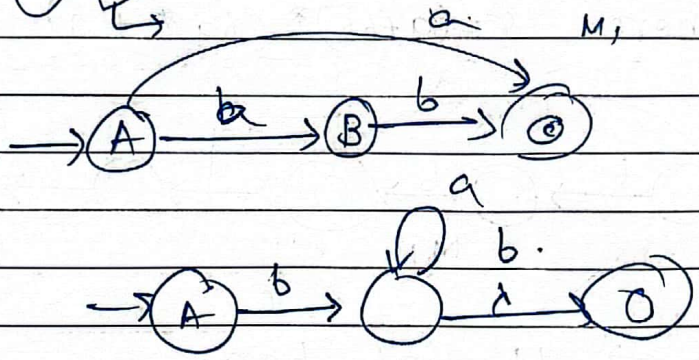
$\Sigma = a$ & b string

$a\{\cdot$

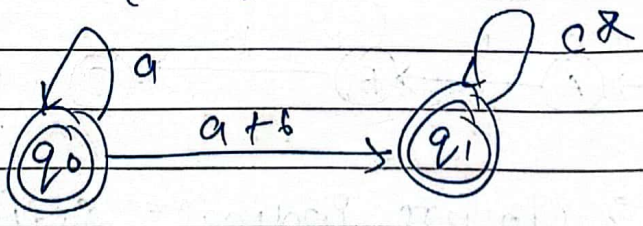
$a(bc)^* \rightarrow a, abc, abcbc, abcbcbc$



$r = (a+bb)^*(ba^*+\lambda)$



$L(a^* + a^*(a+b)c^*)$



__/__/__

Theorem 1 $R(a+b)^* = \{a, b\}^*$

$(a+b)^*$ $(a+b)^3$

$(a+b)(a+b)(a+b)$

↓ ↓ ↓

b b a

Theorem 2: Two regular expression P and Q are equivalent ($P = Q$) if they accept the same set of strings

$r_1 = a^* = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$

$r_2 = a^* + (aa)^* = \{a, aaaa, aa, a \ldots\}$

R.L → r_1, r_2

1. $R = \{a\}$ $L = \{a\}$
2. $R = \{a+b\}$ $L = \{a, b\}$
3. $R = \{a+b+c\}$ $L = \{a, b, c\}$

Concatenation

$R = \{a, b\} = L = \{a.b\}$

$R = \{a.b + a\}b$ $L = \{abb, ab\}$

Design a regular expression that represent all strings over the alphabet $\Sigma = \{a, b\}$ where every accepted string 'w' starts with substring $s = \{abb\}$

__/__/____

Q1. $w = \{(a+b)^* bab$

Q2. $w = (a+b)^* aba (a+b)^*$

Q3. $w = a + a(a+b)^* a + b + b(a+b)^* b$

Algebraic properties of regular expression

(1) Closure property

$r_1 + r_2$

$r_1 \cdot r_2$

r_1^*

r_1^+

(2) Associative property

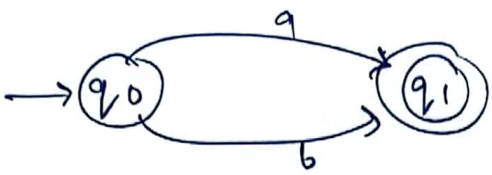
$(r_1 + r_2) + r_3 \square r_1 + (r_2 + r_3)$

$(r_1 \cdot r_2) \cdot r_3 \square r_1 \cdot (r_2 \cdot r_3)$

Identity property

$r \cdot [\varepsilon] = r$

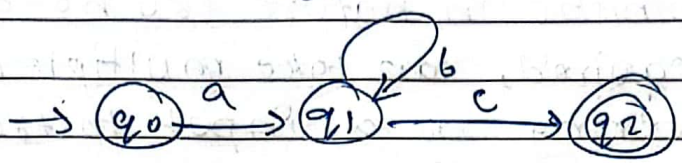
$r + [\phi] = r$



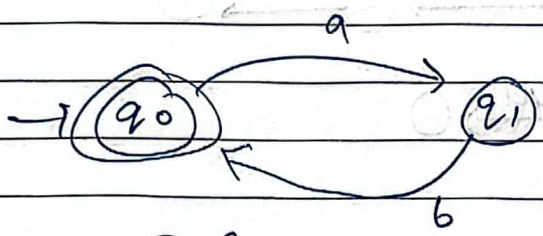
__/__/____

$R(L) = \{ a + b \}$

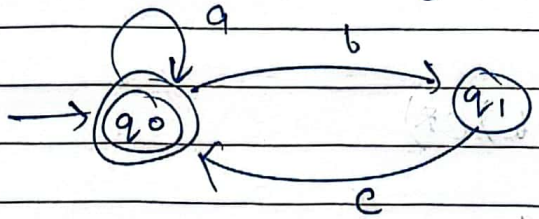
$a.b$



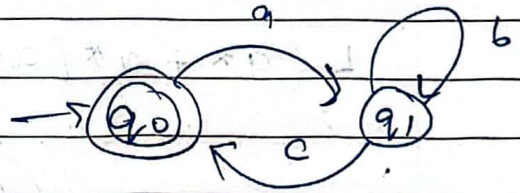
$(ab^*)c \quad ab^*c$



$(a \cdot b)^*$

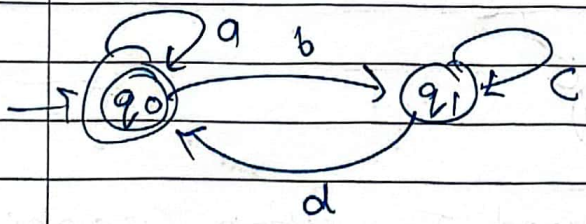


$(a + b \cdot c)^*$



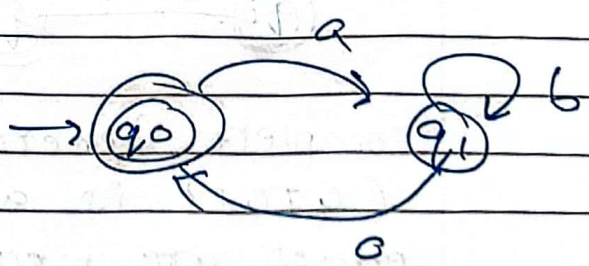
~~$a + (b \cdot (a)^*)$~~

$a(b + ca)^* c$



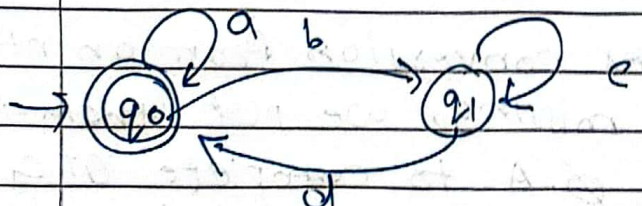
~~$a + b + c \cdot d$~~

$(a + bc^* d)^*$



~~$a.b^*$~~

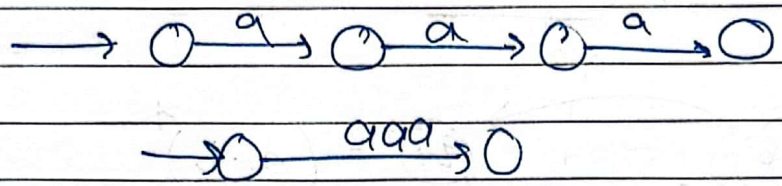
$a(b^* a)^* c$



$(a + b e^* d)^*$

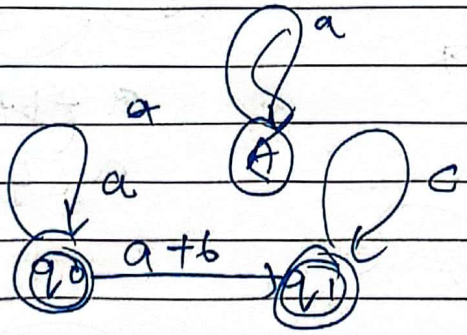
Transition Graph.

1. Same as e-nfa have more than one initial state.
2. Non-deterministic in nature (so no-dead state is required, can take multiple moves can have string as a input to transform from one state to another.



Example 8.8

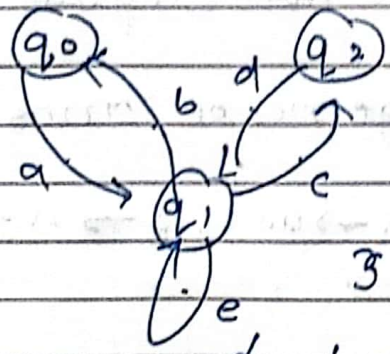
$L(a^* + a^*(a+b)c^*)$



Complete Generalized Transition Graphs (GTGs) is a graph in which all edges are present.

If a GTG after conversion from an nfa, has some edges missing, we put them in and label them ϕ. A to complete GTG with $|V|$ vertices has exactly $|V|^2$ edges

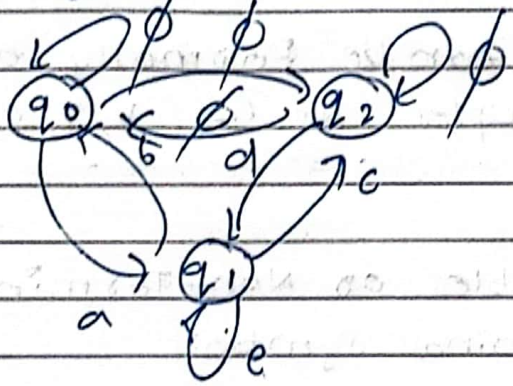
Y - vertices $|V|^2$ edges



In GTG(a) is not complete complete it

3 - vertices $|3|^2 = 9$ edges

$9 - 5 = 4$



__/__/____

chapter3. (Regular and Non-Regular Grammar)

constraint Languages contain huge no. of string

If $G = (\{S\}, \{0,1\}, \{S \to 0S1, S \to \Lambda\}, S)$
find $L(G)$

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

V = set of variables or Non-Terminal symbols
T = set of Terminal symbols
S = Start symbol
P = Production rules for Terminals

production rules form of $\alpha \to \beta$
α and β are strings on VUT and atleast one symbol of belong to V

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

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

Regular Grammar

↙ ↘

Right Linear Grammar | Left Linear Grammar

A grammar is said to be Right Linear if all production are of the form

$A \rightarrow xB$

$A \rightarrow x$ $\forall A, B \in V$ and $x \in T$

Left-Linear Grammar iff all productions are of the form

$A \rightarrow Bx$

$A \rightarrow x$ $\forall A, B \in V$ and $x \in T$

Example :- $S \rightarrow abS \mid b$ → Right linear

$S \rightarrow Sbb \mid b$ → Left linear

__/__/____

Theorem 4.1 $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 L_2$, L_1 and L_1^* are closed under the family of regular language.

Defination 4.1
Suppose Σ and Γ are alphabets then a function

$h : \Sigma \to \Gamma^*$ is called homomorphism

Homomorphism means substitution in which a single letter is replaced with a string.

$w = a_1 a_2 \ldots a_n$

then

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

IF L is a language on Σ then its homomorphic image is

$$h(L) = \{ h(w) : w \in L \}.$$

Example let $\Sigma = \{a, b\}$ $\Gamma = \{a, b, c\}$

define $h(a) = ab$

$h(b) = bbc$

then $h(aba) = ab\ bbc\ ab$

The homomorphic image of $L = \{aa, aba\}$ is

$\{abab, abbbcab\}$

__/__/____

Example 4.3 :-

$\Sigma = \{a, b\}$ and $\Gamma = \{b, c, d\}$ Define h by

$h(a) = dbcc$,

$h(b) = bdc$

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

$r = (dbcc + (bdc)^*)(dbccdbcc)^*$

denotes the regular language $h(L)$

Theorem: If L is a regular-Language then its homomorphic image $h(L)$ is also regular.

L_1 and L_2 be languages on same alphabet

$L_1/L_2 = \{x : xy \in L_1 \text{ for some } y \in L_2\}$

closture properties of regular Language

L_1/L_2 $\{y \mid xy \in L_1 \text{ for some } x \in L_2\}$ Left

$L_1/L_2 = \{x \mid xy \in L_1 \text{ for some } y \in L_2\}$ Right

$L_1 = \{0^*\}$ $L_2 = 0^*$

pigeon-Hole - principle

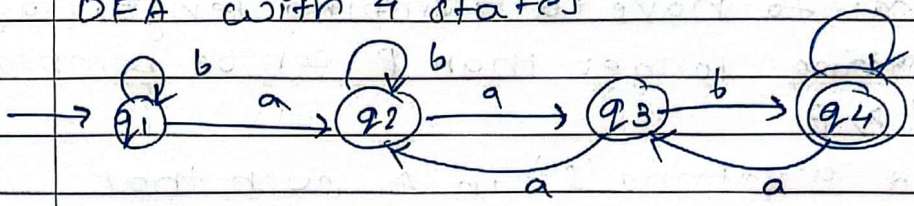
__/__/__

pigeon principle:-
It states that if n pigeons fly into m pigeon hole and $n > m$ then atleast one hole must contain two or more pigeons.

In automata theory, pigeons are the strings of a's, the pigeonhole are the states and the correspondence associated each string with the state to which A goes when string is i/p.

In Automata it is used to prove to say that certain infinite language are not regular.

DFA with 4 states



a, aa, aab → no state is repeated

aabb, bbaa, abbabb, abbbabbabb - state is repeated

If string w has length $|w| \geq 4$ then the transition of string w are more than the state of DFA

$w \geq$ no. of states
q in the walk of w

__/__/____

pumping lemma

pumping lemma is used to prove that a language is NOT regular.

If A is a Regular language, then A has a pumping length 'P' such that any string 's' where $|s| \geq P$ may be divided into 3 parts $s = xyz$ such that

(1) $xy^iz \in A$ ii) $|y| > 0$ iii) $|xy| \leq P$

proof using contradiction

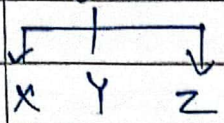
→ Assume that A is regular
→ It has to have a pumping length (say P)
→ All strings longer than P can be pumped $|s| \geq P$
→ find a string 's' in A such that $|s| \geq P$
→ Divide s into xyz
show that $xy^iz \notin A$ for some i
→ Then consider all ways that s can be divided into xyz
→ show that none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

$A = \{yy \mid y \in \{0,1\}^*\}$ is Not regular

A is Regular Language.

pumping length = P

$S = 0^P 1 0^P 1$



$P = 7$

$\underbrace{00}_{X}\underbrace{0000}_{Y}\underbrace{0100000001}_{Z}$

$XY^iZ \Rightarrow XY^2Z$

$= 000000100000001$

$\underbrace{00\ 0000\ 0000}_{\text{11 zeros}}\ 1\ \underbrace{0000001}_{\text{6 zeros}}$

$11 \neq 6$

$\notin A$