Ulman

Page No.
Date

① Nondeterministic
② Deterministic

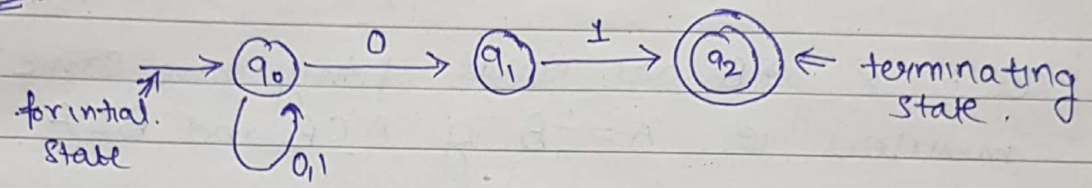
THEORY OF COMPUTATION

Tutorial

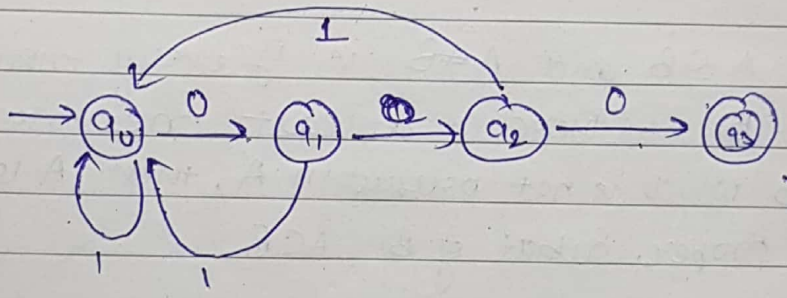
Q1 Construct an NFA to accept all strings terminating in 01.

Q2 Construct a NFA to accept those strings containing consecutive zeroes.

Soln 1:



Soln 2

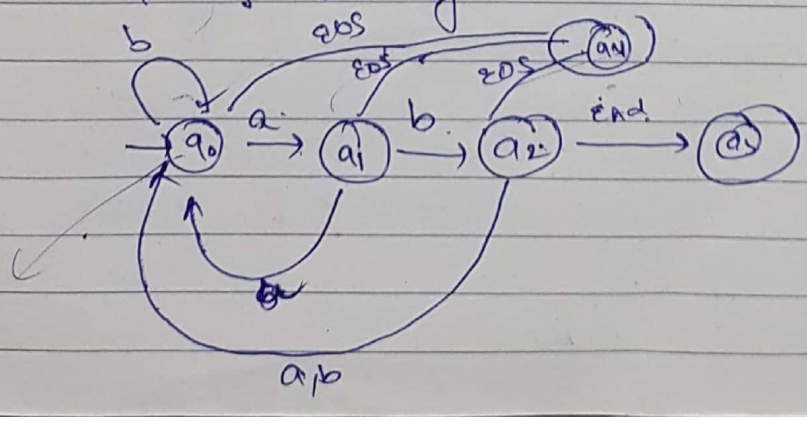


Transition table

$f(q_1, q_2, q_3) = \Sigma$ 01

$f(q_1 q_2 q_3 q_4) = \Sigma$ 0 0 0

Q3 Construct a DFA accepting all strings over $\{a, b\}$ ending in ab.



Q Construct a DFA to accept all strings containing even number of zeroes & even number of ones

SETS

1) If every member of a set A is a member of set B, then ~~set A is a~~

$A \subseteq B$ i.e. A is contained in B

2) Set A & set B are equal if they have same members ie. $A = B$ if $A \subseteq B$ and $B \subseteq A$

3) If $A \subseteq B$ and $A \neq B$ is. If every member of A is in B and there is atleast one member of B which is not present in A, then A is proper subset of B $A \subset B$

Page No.
Date

THEORY OF COMPUTATION

Equivalence of DFA and NFA.

For every NDFA there exist a DFA such that.

Date

on application of a on q_0.,

m can reach after application of a.

M' has to remember all these possible states at any instant of time. Hence, the states of M' are defined as subsets of Q as M starts with initial q_0, q' can be defined [q_0], a string w belong to T(M) if a final state is one of the possible states M reaches upon processing w.

So a final state M' is any subsets of Q containing some final states of M.

iv) $\delta'([q_1 q_2 \ldots q_i], a) =$

$\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a)$.

Equivalently,

$\delta'([q_1 q_2 \ldots q_i], a) = \{R_i \ldots P_j\}$

$L = T(M')$

$\delta'(q_0', x) = [q_1 \ldots q_i]$
if $\delta(q_0, x) = \{q_1 \ldots q_i\}$ for all x in Σ^*

Note: Kleenstar
$\Sigma = \{x\}$
$\Sigma^* = \{\lambda, x, xx, \}$
Closure of the alphabet.

$|x|$, the y part

$\delta'(q_0', x) = [q_1, q_2 \dots q_i]$ if

$\delta(q_0, x) = \{q_1, q_2 \dots q_i\} \rightarrow (2)$

$|x| = 0$, $\delta(q_0, \lambda) = \{q_0\}$.

$\delta(q_0', \lambda) = q_0' = [q_0]$

so, (2) is true for $|x| = 0$.

(2) Assume (2) $\forall y$ with $|y| \leq m$

let x be $m+1$, $x \Rightarrow$

$x = ay$,

$|y| = m$ & $a \in \Sigma$

let $\delta(q_0, y) = \{p_i \dots p_j\}$

and

$\delta(q_0, ay) = \{r_1, r_2 \dots r_k\}$

$\delta'(q_0', y) = [p_i \dots p_j]$

$\{r_1 r_2 \dots r_k\} = \delta(q_0, ay) = \delta(\delta(q_0, y), a)$

$= \delta(\{p_i \dots p_j\}, a)$

$\delta'([p_i \dots p_j], a) = [r_1 r_2 \dots r_k]$

Here, $\delta'(q_0', ya) = \delta(\delta'(q_0, y), a) =$

$\delta'([p_i \dots p_j], a) = [r_1 r_2 \dots r_k]$

second is true for all strings x, i.e., $\hat{\delta}(q_0, x)$ is established

$x = ya$

$x \in T(M)$ if and only $\hat{\delta}(q_0, x)$ contains a state of x.

$\hat{\delta}(q_0, x)$ contains a state of x if and only $\hat{\delta}'(q_0', x)$ is

$x \in T(M)$ if and only $x \in T(M')$.

DFA M' accepts L.

Q Construct a DFA equivalent to

$M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_0\})$

State / Σ =	0	1
q_0	q_0	q_1
q_1	q_1	q_0, q_1

(1) The states are subsets of Q.

$\phi, [q_0], [q_1], [q_0, q_1]$

(2) Initial state is $[q_0]$

(3) Final state $[q_0]$, $[q_0, q_1]$

(4) $\hat{\delta}$(M has n states corresponding finite automata has 2^n states, we need not to construct $\hat{\delta}$

Date

for all states 2^n but only for those whichever from $[q_0]$ because our interest in constructing M_1 accepting $T(M)$.

state/Σ	0	1
ϕ	ϕ	ϕ
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

Q Construct a DFA equivalent to

$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\})$.

state/Σ	a	b
q_0	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3	ϕ	q_2

(1) States of Q are

$\phi, q_0, q_0q_1, q_0q_2, q_0q_3, q_0q_1q_2, q_0q_1q_3, q_0q_2q_3, \ldots$

(2) Initial state $[q_0]$

(3) Final state $[q_3, q_3q_4]$

FINITE AUTOMATA WITH OUTPUTS.

1) The mathematical model of finite state is represented. by 5 tuple method $A(Q, \Sigma, \delta, q_0, F)$ which doesn't include the information about output but now we will be considering finite automata with outputs There are 2 machines we can be formulated by finite automata.
(1) Moore
(2) Mealy
Both of them are not working as finites automata. (These can't be used for language acceptance). These machine provides some output only.

The finite automata which we have considered have binary outputs ie. they accept the strings or don't accept the strings. This acceptability was decided on the basis reachability of final state from initial state. Now we remove this restriction.
Consider the model where outputs can be choosen from some other alphabet. The value of output function $Z(t)$ in a most general case is a function of present state q and a present input $x(t)$.

$$Z(t) = \lambda(q(t), x(t))$$

λ is the output function.

This generalised model is usually called as mealy machine.

If the output function $Z(t)$ depends only on present state and is independent of current input. then.

$$Z(t) = \lambda(q(t))$$

This is called Moore machine.

Moore Machine: A moore machine is a 6 tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$.

Δ is the output occurred

λ is the mapping from $q_0 \to \Delta$ giving the output associated with a state

Mealy machine is a 6 tuple

$$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

$\lambda \to$ is the output function mapping

$$Q \times \Sigma \longrightarrow \Delta$$

* → In mealy machine we get an output only on the application of an input signal.

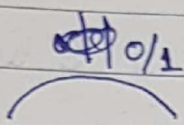
1) For input string is null, the output is only null.

2) A finite automata can be converted into a moore & machine by introducing $\Delta = \{0, 1\}$ and defining $\lambda(q) = 0$ if $q \notin F$
$\lambda(q) = 1$ if $q \in F$

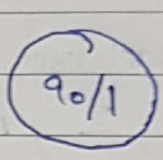
3) For moore machine if input string is of length n, the output string is of $n+1$.
The first output is $\lambda(q_0)$ for all output strings.

In case of mealy machine, if input is of length n, then output is of same length n.

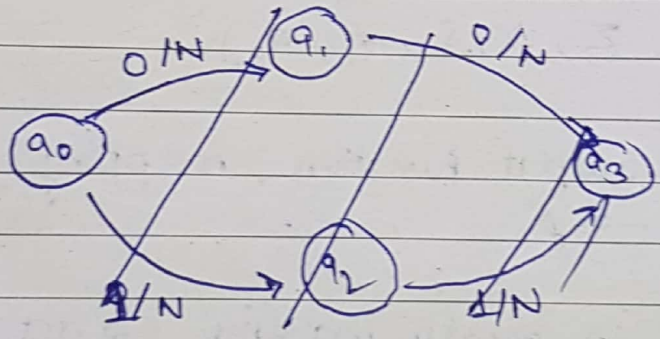
Mealy

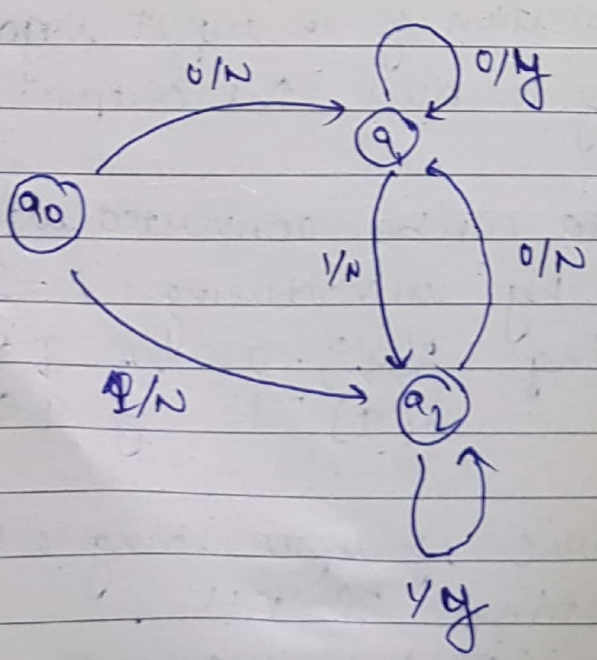


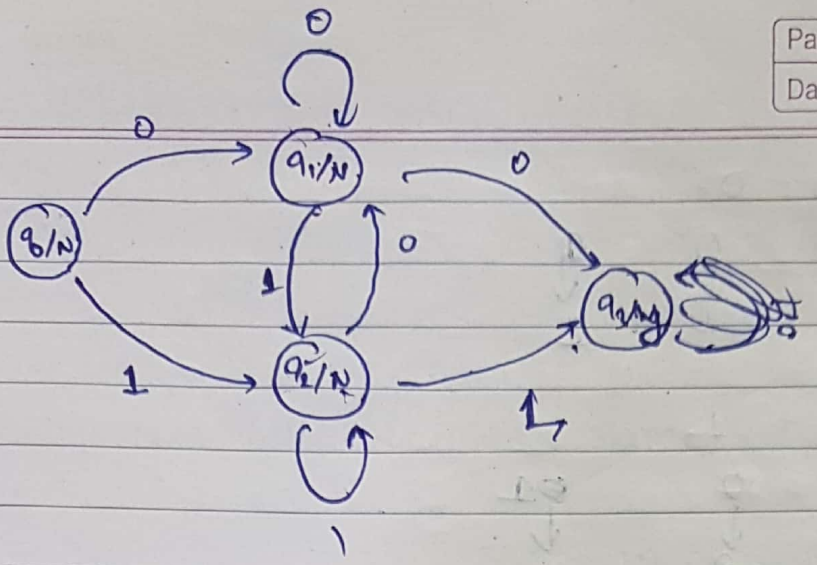
Moore

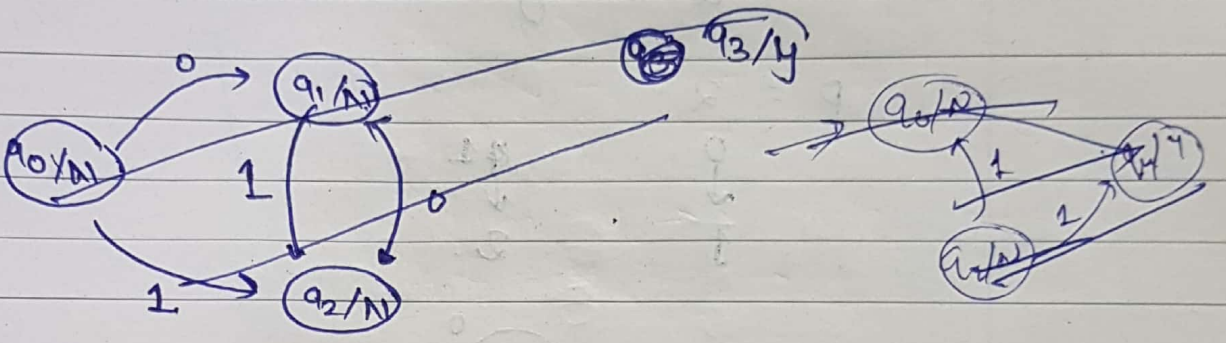


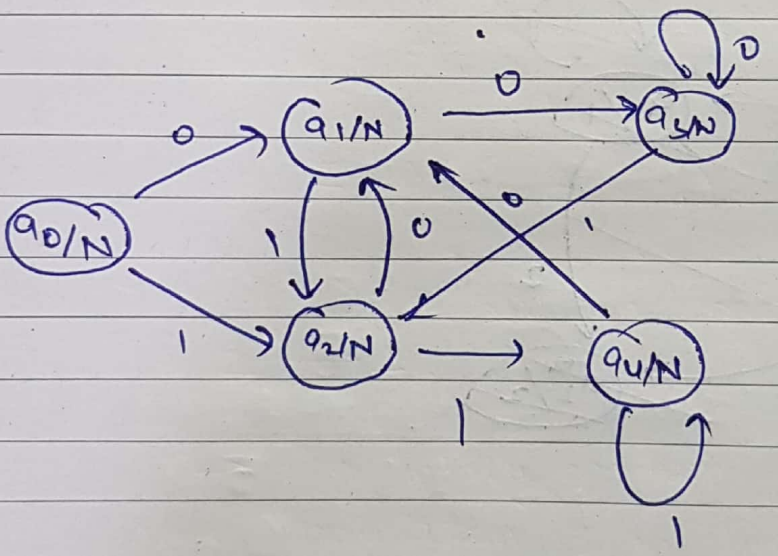
Q) Make/construct mealy or moore machine that produces Y if input ends with 00 or 11 and N otherwise.











Q Residue mod 3 of binary number using mealy & moore machine.

$i \to 0 \quad 2i$

$i \to 1 \quad 2i+1$

$Y3 \to p = 0, 1, 2.$

$0 \to 2i/3 \to 2p \bmod 3$

$1 \to (2i+1)/3 \to 2p+1 \bmod 3$

Date

$p = 0$

0 → 0

1 → 1

$p = 1$

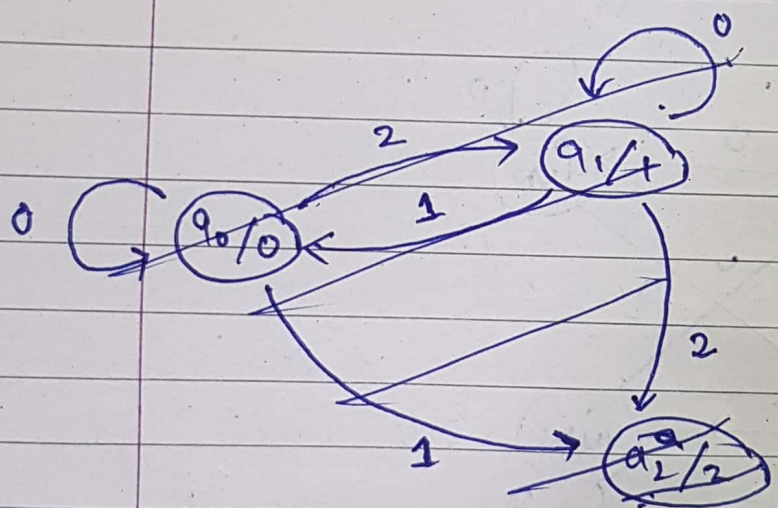
0 → 2

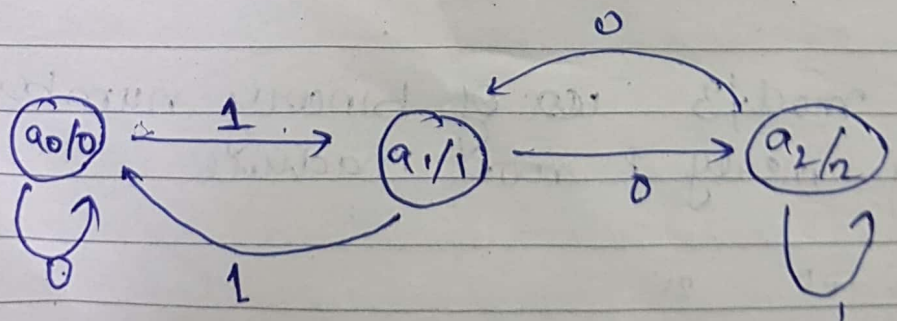
1 → 0

$p = 2$

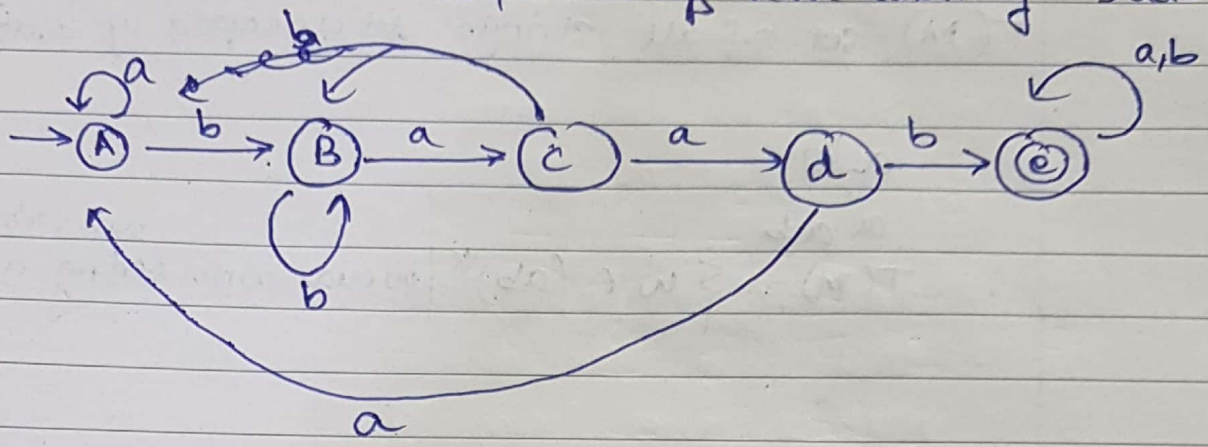
0 → 1

1 → 2

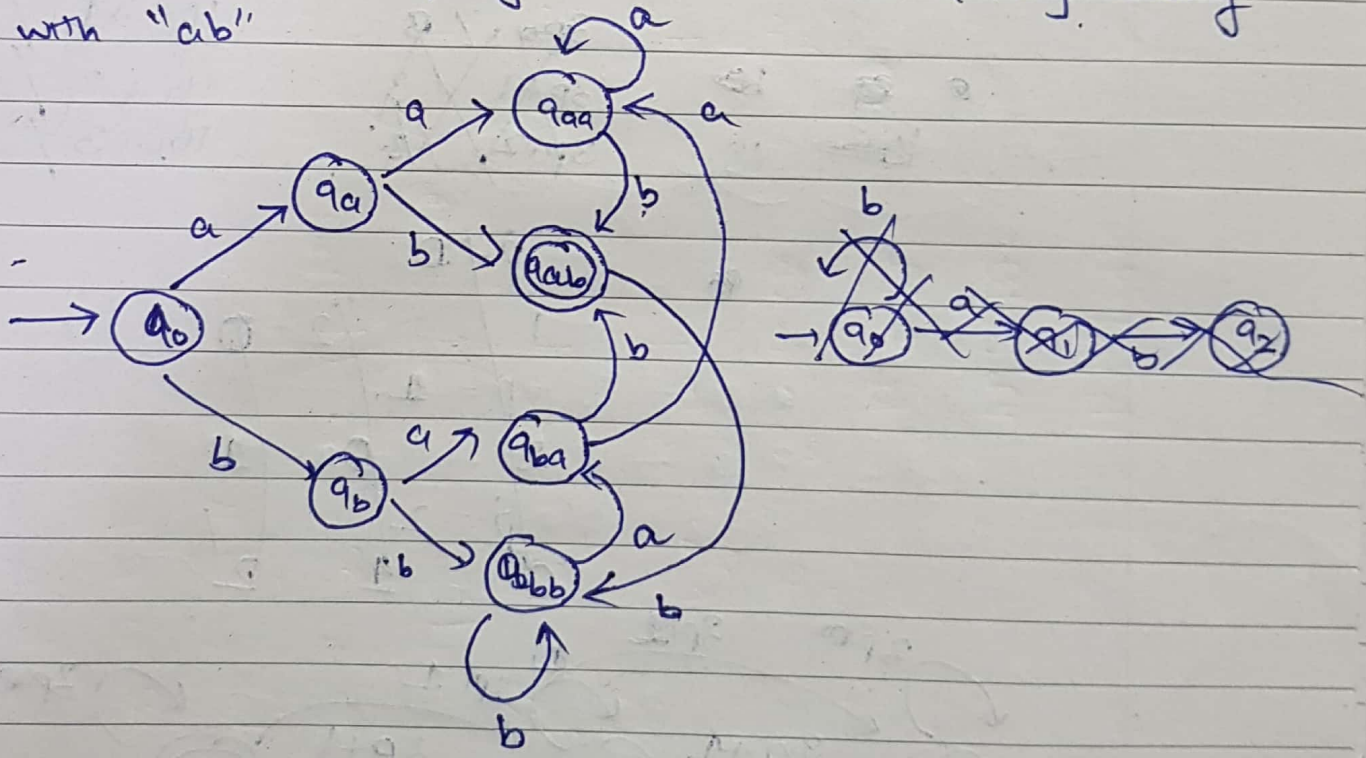




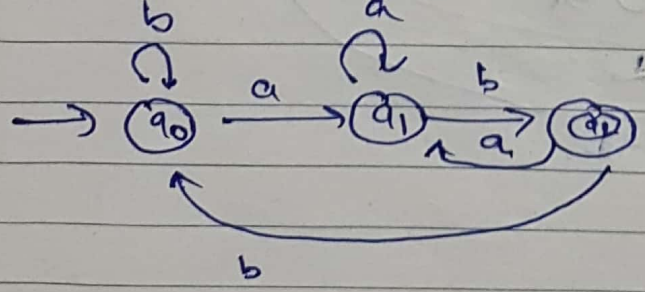
Q Construct a DFA which accepts strings have substring "baab".



Q Construct DFA accepting all values over {a,b} ending with "aab"



Q Find T(M) for the DFA M below



	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

L(M).

M=

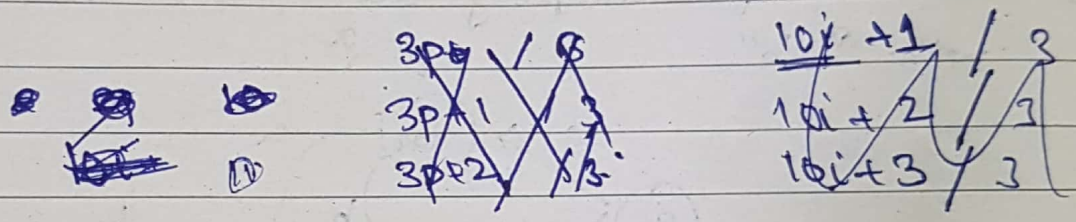
T(M) set of all strings accepted by automata

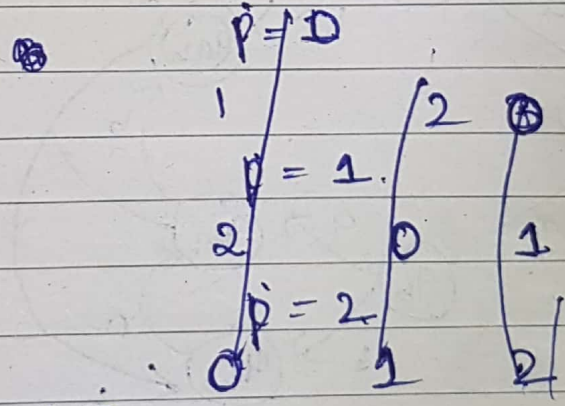
ab

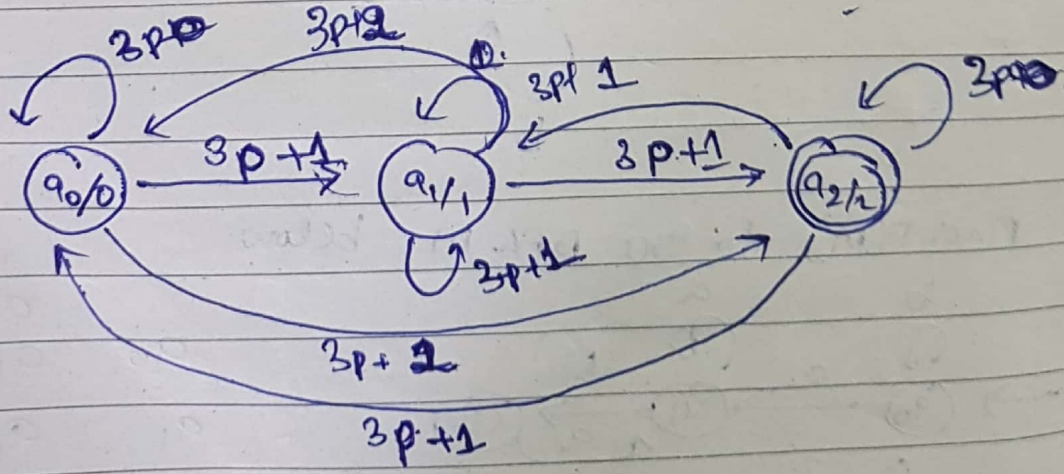
aab

∴ $T(M) = \{ w \in (ab)^* \mid w \text{ ends with substring } ab \}$

Q Residue mod 3 of decimal number using mealy & moore.







ϵ and ϕ

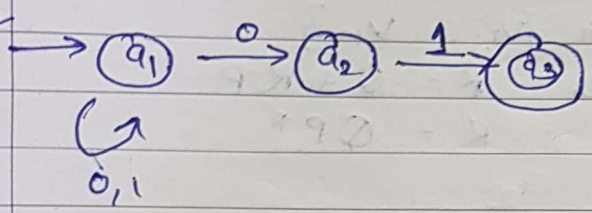
ϵ → the unique string of length 0. It is called as an empty or null string (it is represented as input in automata)

ϕ – it is represented as blank space.

Page No.

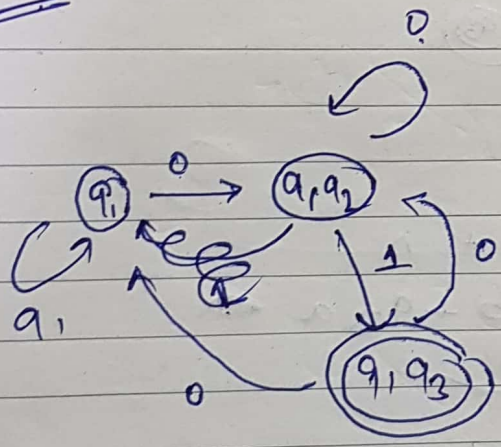
Q) For given NFA construct equivalent DFA.

NFA



	0	1
q_1	q_1, q_2	q_1
q_2	ϕ	q_3
q_3	ϕ	ϕ

DFA



	0	1
q_1	q_1, q_2	q_1
q_1, q_2	q_1, q_2	q_1, q_3
q_1, q_3	q_1, q_2	q_1

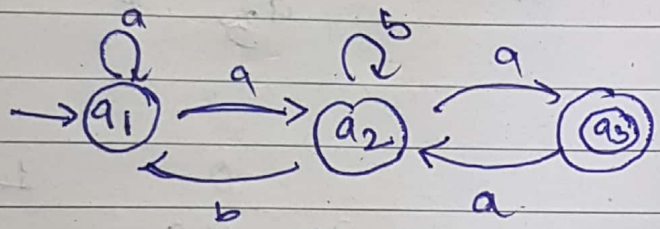
Q) Construct a mealy & moore m/c in which if string ends with (0) → A (011) → B and else → C.

Page No.
Date

Algebraic method using ARden's theorem following assumptions are made regarding transition systems :

1. Transition graphs does not have null moves
2. It has only one initial state.
3. Its vertices are from $\{v_1, \ldots v_n\}$
4. Arden theorem states $R = Q + RP$

$R = QP^*$



$q_1 = q_1a + q_2b + \wedge$

$q_2 = q_1a + q_2b + q_3a$

$q_3 = q_2a$

$q_2 = q_1a + q_2(b+aa)$

$Q \quad R \quad P$

$q_2 = QP^*$

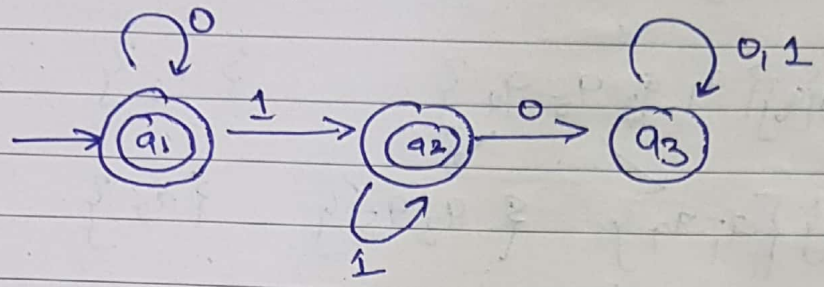
$= q_1a(b+aa)^*$

~~$q_3 = q_1aa + q_2ba + q_3aa$~~

~~$q_2 = q_1$~~

$q_3 = q_1 aa(b+aa)^*$

Q =



$q_1 = q_1 0 + \wedge$

$q_2 = q_1 1 + q_2 1$

$q_3 = q_3(0+1) + q_2 0$

$q_2 = q_1 1 + q_2 1$

$q_2 = q_1 1(1)^*$

$q_3 = q_2 0 (0+1)^*$

$q_3 = q_1 1(1)^* 0(0+1)^*$

MINIMISATION OF AUTOMATA

state / Σ	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_1
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

π bonds

$\Phi = \{Q_1, Q_2\}$

$Q_1 = \{\ \}$

$Q_2 \{q_2\}$

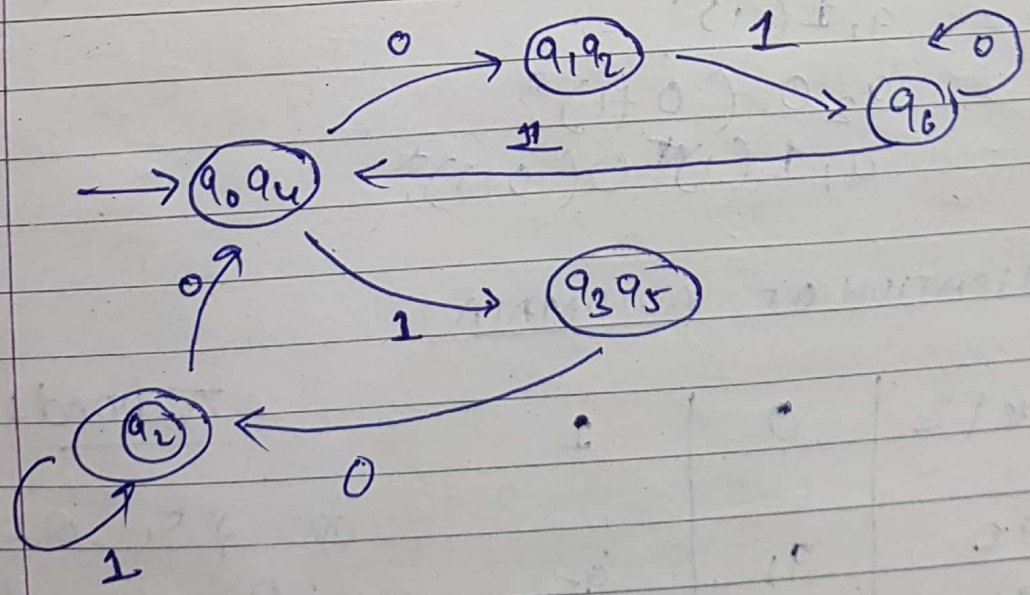
$Q_0 = \{q_0 q_1 q_2 q_3 q_4 q_5 q_6 q_7\}\ \{q_2\}$

$Q_1' = \{q_0 q_4, q_6\}\ \{q_1, q_3, q_5 q_7\}\ \{q_2\}$

$Q'' = \{q_0 q_4\}\{q_6\}\{q_1 q_3 q_5 q_7\}\ \{q_2\}$

$Q''' = \{q_0 q_4\}\{q_6\}\{q_1 q_7\}\ \{q_3 q_5\}\ \{q_2\}$

	0	1
→ $q_0 q_4$	$q_1 q_7$	$q_3 q_5$
$q_1 q_7$	q_6	q_2
(q_2)	$q_0 q_4$	q_2
$q_3 q_5$	q_2	q_6
q_6	q_6	$q_0 q_4$



TCA

Page No.
Date

Minimise

$R = (a/b)^* abb$

↓

NFA → DFA → MINIMISATION

IMP TOPICS:

1. Regex ↔ finite automata
2. NFA ↔ DFA
3. Minimisation.

Page No.
Date

At the first stage the procedure so that both machines accept exactly the same input sequences. We look into the next state column for any state q_i and determine the number of different outputs associated with q_i in that column. We split q_i into several different states The number of such states being equal to the number of output associated with q_i.

Present state	Next state a=0	o/p	Next state a=1	o/p
q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Present state	a=0 state	o/p	a=1 state	o/p
q_1	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

So the pair of states and output in the next column can be re write as Moore table of m/c

Present state	Next state		output
	a=0	a=1	
→ q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Q Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a moore m/c the following procedure is adopted to convert following mealy m/c

Q

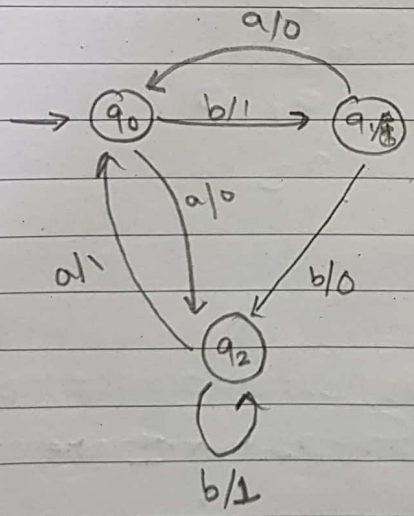
Present state	Next state		o/p
	a=0	a=1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

In the case of the moore m/c for every state pair we construct the pair of the next state & the corresponding o/p we reconstruct the table for mealy machine

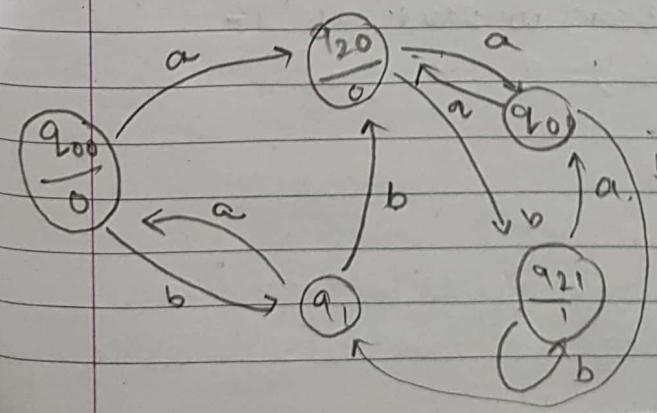
Present state	Next state $a=0$ state	o/p	$a=1$ state	o/p
q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	1	q_3	0
q_3	q_3	0	q_0	0

Note: if two have identical transition (ie rows corresponding to 2 states are identical), then we can delete one of them.

ques Construct given mealy to moore m/c.



PS	NS $a=0$ state	o/p	NS $a=1$ state	o/p
q_0	q_2	0	q_1	1
q_1	q_0	0	q_2	0
q_2	q_0	1	q_2	1



PS	NS $a=0$	NS $a=1$	o/p
q_{00}	q_{20}	q_1	0
q_{01}	q_{00}	q_1	1
q_1	q_{00}	q_{20}	1
q_{20}	q_{01}	q_{21}	0
q_{21}	q_{01}	q_{21}	1

Date

Q Construct a Mealy m/c which is equivalent to Moore m/c. given in table.

Present state	Next state a=0	Next state a=1	o/p
→ q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

NS

PS	a=0 state o/p	a=1 state o/p
q_0	q_1 0	q_2 1
q_1	q_3 1	q_2 1
q_2	q_2 1	q_1 0
q_3	q_0 1	q_3 1

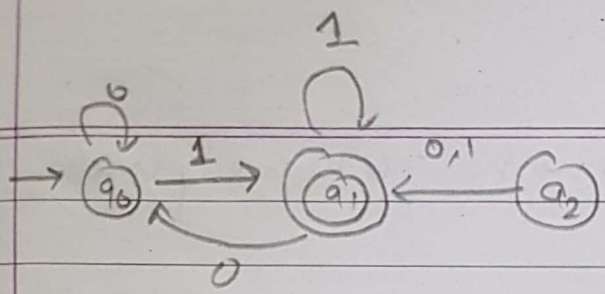
Q What is dead state?

→ All those non-final states which transit to itself for all input signals in Σ are called as dead state.

$\Sigma = \{0,1\}$, q is said to dead state if $\delta(q,0) = q$, $\delta(q,1) = q$ & $q \notin F$

Q What is Inaccessible state?

All those state which can never be reached from initial state are called inaccessible state

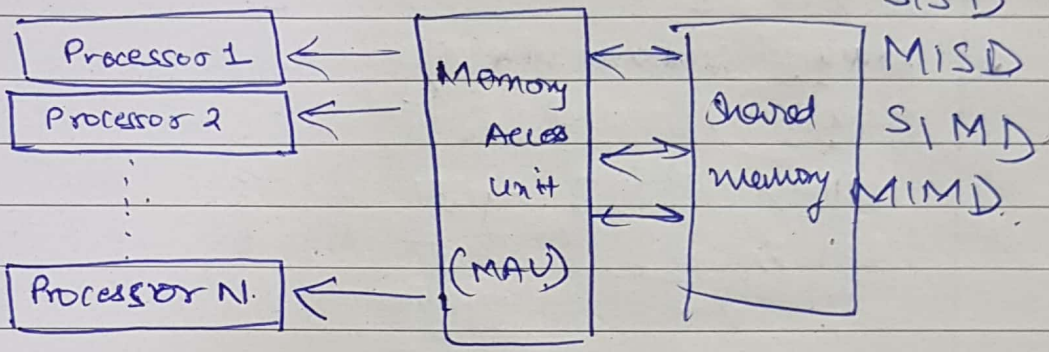


q_2 is inaccessible state.

HPC

PRAM

RAM
SISD
MISD
SIMD
MIMD



TOC

Page No.
Date

Minimization of Finite Automata:

The pair of states (p, q) are distinguishable state if the transition goes to final states & non-final states respectively.

The pair of states are non-distinguishable state if both transition goes to final state group or non-final group

How to minimise the final automata, all the non-distinguishable states are to be merged and used as a singe state.

Two methods are:

1) Hopcraft's method.
2) Mufit Nerode theorum

Equivalence class: There are the group of equivalent states. For a DFA M the minimum no. of states in a equivalent DFA is a same as the number of equivalence classes of M's state. So if we can find equivalence classes, we can use these as the states of the smallest equivalent machine

Hopcraft's Method: of construction of minimum automata:

1) Construction of π_0 :-

By defn. of (0-equivalence), π_0

$$\pi_0 = \{Q_1^0, Q_2^0\}$$

where Q_1^0 is the set of all final states & Q_2^0 is equal $Q - Q_1^0$.

2) Construction π_{k+1} & π_k

Let Q_{ik} be any subset in π_k. If q_1 & q_2 are in Q_{ik}, they are (k+1) equivalent provided

$\delta(q_1, a)$ & $\delta(q_2, a)$ are k equivalent. And our whether $\delta(q_1, a)$ & $\delta(q_2, a)$ are in the same equivalence class in π_k for every $a \in \Sigma$, if so q_1 & q_2 are (k+1) equivalent. In this way Q_{ik} is further devided in (k+1) equivalent classes.

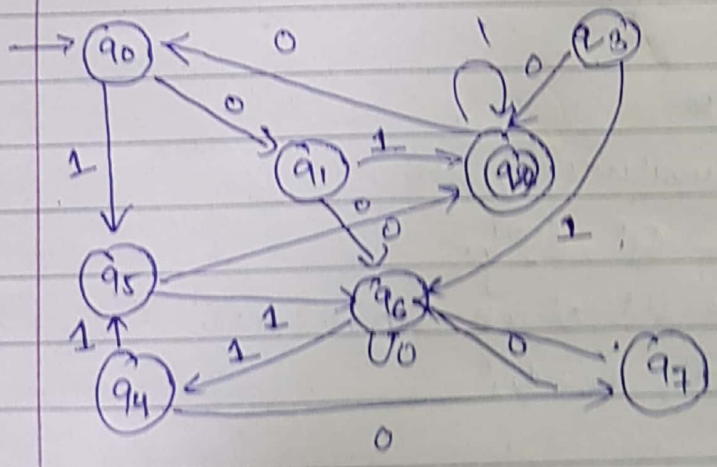
Repeat this for every δik in Πk to get all the elements of $\Pi k+1$

3) Construct Πk for $k = 1, 2, 3, \ldots$ till

$$\boxed{\Pi k = \Pi k+1}$$

4) Construction of minimum automata: for the required minimum automata, the states are equivalent classes obtained in step 3. The state table is replaced [by] by the corresponding equivalent class q.

Q Construct a minimum state automata equivalent to given automata.



state / i/p	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_6	q_2
q_3	q_1	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

$Q_1^0 = \{q_2\}$

$Q_2^0 = \{q_0\ q_1\ q_3\ q_4\ q_5\ q_6\ q_7\}$

$\pi_0 = \{q_2\}\ \{q_0\ q_1\ q_3\ q_4\ q_5\ q_6\ q_7\}$

$\because$ q_0, q_1 are not 1 equivalent because.

$\delta(q_1, 1)$ & Q_1^0 & $\delta(q_0, 1)$ is Q_2^0

Simply q_0 is not 1 equivalent with $q_3\ q_5\ q_7$.

q_0 & q_4 are 1 equivalent.

similarly q_0, q_6 is 1 equivalent to q_6

Repeat the construction using q_3, q_5, q_7.

~~$\pi_1 = \{q_2\}\ \{q_0\ q_4\ q_6\}\ \{q_1\ q_3\ q_5\ q_7\}$~~

~~q_4 is not~~ $Q_2 = \{q_0, q_4, q_6\}$

$Q_3' = \{q_1, q_7\}$

the elements left in Q_0' are $\{q_3\ q_5\}$

$Q_4' = \{q_3, q_5\}$.

$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$.

q_0, q_6 are not 2 equivalent.

$\pi_2 = \{\{q_2\}, \{q_0\ q_4\}, \{q_6\}, \{q_1\ q_7\}, \{q_3, q_5\}\}$

$\pi_3 = \pi_2$ i.e., they same.

So the π_2 gives us the equivalence classes, so

$M' = (Q', \{0, 1\}, \delta', q_0', F')$

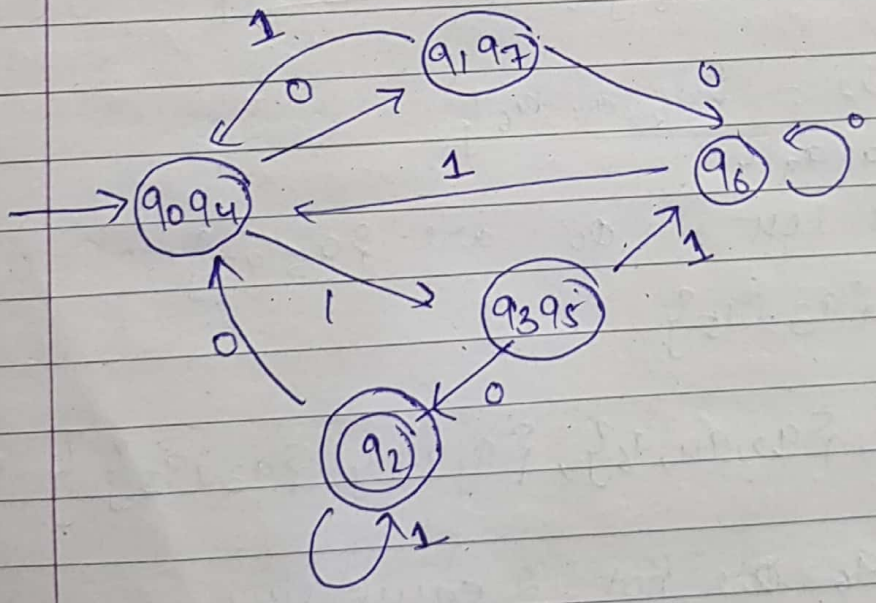
Page No.
Date

where

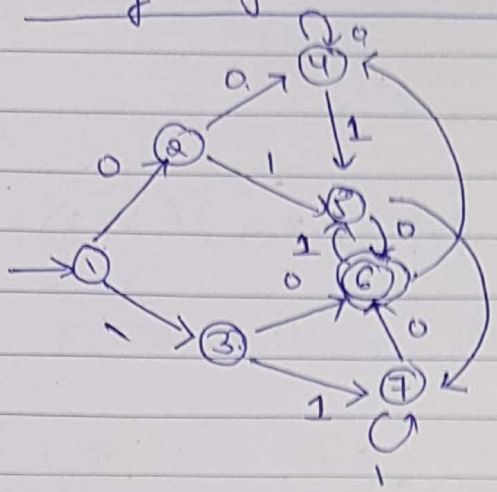
$Q' = \{[q_2], [q_0q_4], [q_6], [q_1q_7], [q_3q_5]\}$

$q_0' = [q_0q_4]$ $\quad F' = [q_2]$.

state / Σ	0	1
$[q_0q_4]$	$[q_1q_7]$	$[q_3q_5]$
$[q_2]$	$[q_0q_4]$	$[q_2]$
$[q_1q_7]$	$[q_6]$	$[q_0q_4]$
$[q_3q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0q_4]$



Using Myhill Nerode theorum



fill row & columns of final state as x.

	1	2	3	4	5	6	7
2	✓						
3	x	x					
4	✓	✓	~~x~~				
5	x	x	✓	x			
6	x	x	x	x	x		
7	x	x	✓	~~x~~	✓	x	

(1,7) → 0 → (2,6) x
(1,7) → 1 → (3,7)

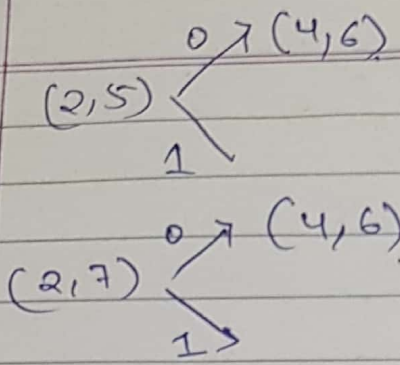
(1,2) → 0 → (2,4) → 0 → (4,4) ✓
(2,4) → 1 → (3,5) → 0 → (6,6) ✓
(3,5) → 1 → (7,7) ✓

(1,5) → 0 → (2,6) x
(1,5) → 1 → (3,7)

(1,4) → 0 → (2,4)
(1,4) → 1 → (3,5)

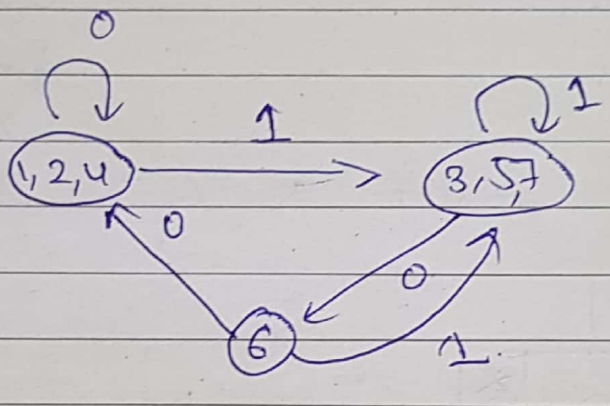
(1,3) → 0 → ~~(3,6)~~ (2,6)
(1,3) → 1 → (3,7)

(2,3) → 0 → (4,6)
(2,3) → 1 →



(1,7), (1,4) (2,4) (3,5) (3,7) (5,7)

We combine States 1,2,4 & 3,5,7.



8. Construct a minimum state automata equivalent to an automata whose transition table is

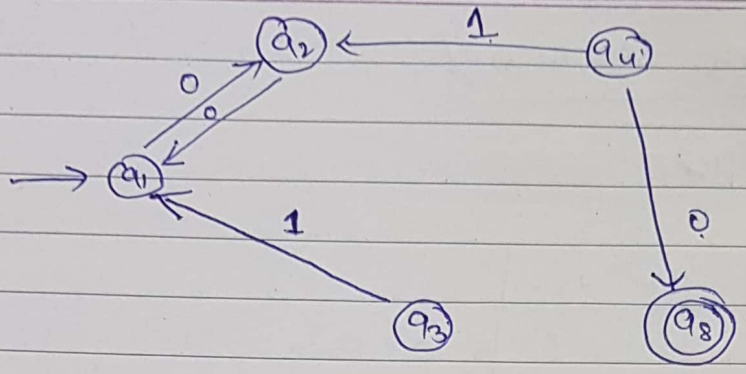
State	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_4
q_3	q_1	q_5
q_4	q_4	q_2
(q_5)	q_6	q_6

It is an NFA first convert it to DFA & then minimise

9. Construct a minimum state automata equivalent

Peter Rinse

Page No.
Date



(q6)

Σ	0	1
→ q_1	q_2	q_6
q_2	q_1	q_6
q_3	q_7	q_1
q_4	q_8	q_2
q_5	q_1	q_7
(q6)	q_8	q_3
(q7)	q_1	q_4
(q8)	q_1	q_3

Q. Equivalence of two Finite Automata

$\{q_1 q_2 q_3 q_4 q_5\}$ $\{q_6 q_7 q_8\}$

~~$\{q_3\}$ $\{q_4\}$ $\{q_1 q_2\}$~~ $\{q_6\}$ $\{q_7\}$

$\{q_1 q_2 q_5\}$ $\{q_3\}$ $\{q_4\}$ $\{q_6\}$ $\{q_7 q_8\}$

$\{q_1 q_2\}$ $\{q_5\}$ $\{q_3 q_4\}$ $\{q_6\}$ $\{q_7 q_8\}$

Page No.
Date

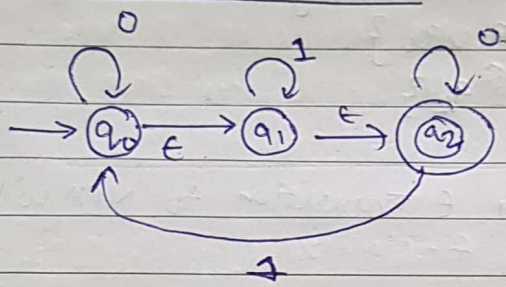
Production systems:

comprises of

1) set of valid rules
2) one or more knowledge space
3) control strategy (allow motion & should be systematic)
4) rule applied

TOC.

NFA with ϵ-Transition



It is possible to make transition from q_0 to q_1 without consuming input.

Five tuple set denotation:

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

$$\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$$

Acceptance of a string by an NFA with ϵ-Transition

A string w in Σ^* in by NFA with ϵ transition, if there exist at least one path corresponding to w, which starts in an initial state & ends in one of the final state. Since this path formed by ϵ-transition as well as non-ϵ-transition, to find whether w is accepted or not by the NFA with ϵ-moves, we

Page No.
Date

require to define a function ϵ-closure (q) where q is the a state of the automata.

ϵ-closure (q) of a state is the set of all states we reach by following the transition function from the given state that are labelled ϵ.

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$
ϵ-closure $(q_1) = \{q_1, q_2\}$
ϵ-closure $(q_2) = \{q_2\}$

Conversion of NFA with ϵ Transition to NFA without ϵ-Transition

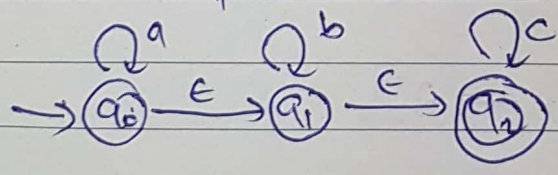
Q $(Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b, c\}$ and ϵ moves.

q_0 = initial state

F = final state.



Q/Σ	a	b	c	ε
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

Find the states of NFA without ε-transition including initial state & final state as follows

(a) The language accepted by above NFA with set of strings {a,b,c} followed by any no. of b's & c's.

(b) There will be same no. of states but the names can be constructed by writing the state name as a set of states in ε-closure(c).
Initial state will be ε-closure of intial state of NFA with ε-transition

$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$$

new initial state of NFA without ε-transition.

$$\varepsilon\text{-closure}(q_1) = \{q_1, q_2\} \Rightarrow \text{new state.}$$
$$(q_2) = \{q_2\}$$

The final state of NFA without ε-transition are all those new states which contain final states of NFA with ε-transition as member.

NFA without ε-transition.

$$M' = (Q', \Sigma, \delta', q_0', F')$$
$$Q' = (\{q_0 q_1 q_2\}, \{q_1 q_2\}, \{q_2\})$$
$$q_0 = (\{q_0 q_1 q_2\})$$
$$F' = (\{q_0 q_1 q_2\}, \{q_1 q_2\}, \{q_2\})$$

Page No.
Date

Rule 2: δ'

Consider each old machine transition in each row, we can ϕ entry & ϵ-transition column in the old machine

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

$$\delta'(\{q_0 q_1 q_2\}, a) = \{q_0 q_1 q_2\}$$

this is because the new machine accepts the same language as the old machine & must atleast have the same transitions for the new state names.

so,

$$\delta'(\{q_0, q_1, q_2\}, a) = \epsilon\text{-closure}(\delta(q_0 q_1 q_2), a))$$

$$= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\}$$

$$= \epsilon\text{-closure}(\delta(q_0, a) \cup \phi \cup \phi)$$

$$= \epsilon\text{-closure}(\delta(q_0, a))$$

$$= \{q_0, q_1, q_2\}$$

$$\delta'(\{q_0, q_1, q_2\}, b) = \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2), b)]$$

$$= \epsilon\text{-closure}(\delta(q_1, b))$$

$$= \{q_1, q_2\}$$

$$\delta(\{q_0, q_1, q_2\}, c\} = \{q_2\}$$

$$\delta'(\{\{q_1, q_2\}, a)\} = \phi \qquad \delta'(\{q_2\}, a) = \phi$$

$$\delta'(\{\{q_1, q_2\}, b)\} = \{q_1, q_2\} \qquad \delta'(\{q_2\}, b) = \phi$$

$$\delta'(\{q_1, q_2\}, c\} = \{q_2\} \qquad \delta'(\{q_2\}, c) = \{q_2\}$$

Page No.
Date

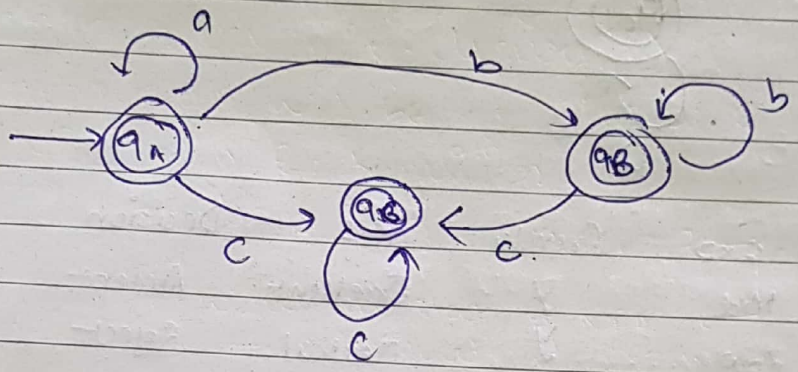
Q.

Q/Σ	a	b	c
$\{q_0, q_1, q_2\}$	$\{q_0 q_1 q_2\}$	$\{q_1 q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	ϕ	$\{q_1 q_2\}$	$\{q_2\}$
$\{q_2\}$	ϕ	ϕ	$\{q_2\}$

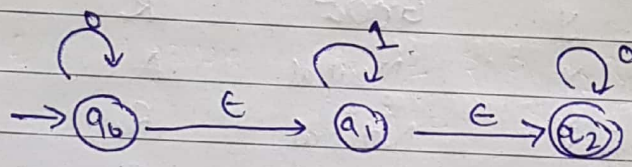
let $\{q_0 q_1 q_2\} \rightarrow q_A$

$\{q_1 q_2\} \rightarrow q_B$

$\{q_2\} \rightarrow q_C$



Ans



Equivalent NFA without ϵ-transition.

ϵ - closure (q_0) = $\{q_0 q_1 q_2\}$ — initial state.

ϵ - closure (q_1) = $\{q_1 q_2\}$

ϵ - closure (q_2) = $\{q_2\}$

$\delta(\{q_0 q_1 q_2\}, 0) = (\delta\{q_0, 0) \cup \delta(q_2, 0))$

$= \{q_1 q_2 q_0\}$

$\delta(\{q_0 q_1 q_2\}, 1) = \{q_1 q_2\}$

$\delta(\{q_1 q_2\}, 0) = \{q_2\}$

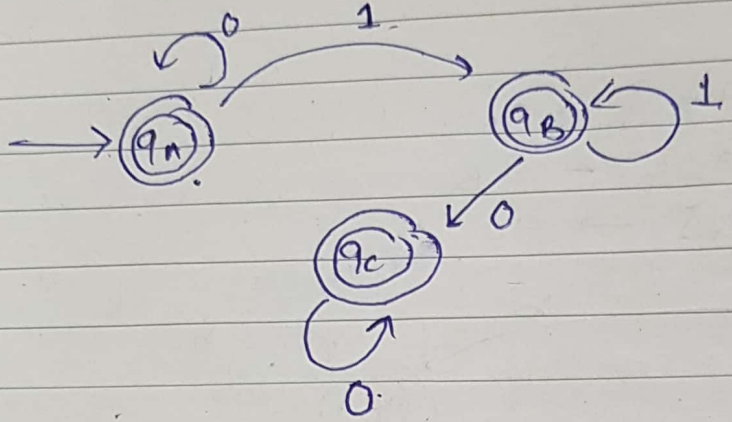
$\delta(\{q_1 q_2\}, 1) = \{q_1 q_2\}$

$\delta(\{q_2\}, 0) = q_2$ $\quad \delta(\{q_2\}, 1) = \phi$

jflap 7.1 Version

Page No.
Date

Q/Σ	0	1
$\{q_0 q_1 q_2\}$	$\{q_0 q_1 q_2\}$	$\{q_1 q_2\}$
$\{q_1 q_2\}$	$\{q_2\}$	$\{q_1 q_2\}$
$\{q_2\}$	$\{q_2\}$	ϕ



SC

U	Dep	Exp	French	Ref	Decision
x_1	MBA	Mid	Y	Excellent	Accept
x_2	MBA	Low	Y	Neutral	Reject
x_3	MCE	Low	Y	Good	R
x_4	MSc	High	Y	Neutral	A
x_5	MSc	Mid	Y	Neutral	R
x_6	MSc	High	Y	Excellent	A
x_7	MBA	High	N	Good	A
x_8	MCE	Low	N	Excellent	R

$X_{accept} = x_1 x_4 x_6 x_7$

$X_{reject} = x_2 x_3 x_5 x_8$

$X_{DEFR}: x_1 x_4 x_6 x_7$

$X_{DEFR\ (Reject)} = x_2 x_3 x_5 x_8$

SOFT COMPUTING

Date

TOC

Formal language:

Grammar: A phase structure grammar comprises set of (V_n, Σ, P, S) where

$V_n \rightarrow$ a finite non empty set whose elements are called variables

$\Sigma \rightarrow$ a finite non empty set whose elements are called terminals

$S \rightarrow$ start symbol, element of V_n

$P \rightarrow$ is finite set whose elements are from $\alpha \rightarrow \beta$ where α, β are strings on $V_n \cup \Sigma$, α has atleast one symbol from V_n. Elements of P are called production or production rules or rewriting rules.

① $\alpha \rightarrow \beta$ is a production in a grammar G. and γ, δ are any two strings on $V_n \cup \Sigma$

$$\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta.$$

This process is called one step-derivation.

② The language denoted by Grammar L(G) is defined as $\{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$ or L(G) is the set of all terminal strings derived from start symbol S.

③ If $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$, then α is called sentential form. The elements of L(G) are sentential forms.

④ G_1 and G_2 are two equivalent if $L(G_1) = L(G_2)$

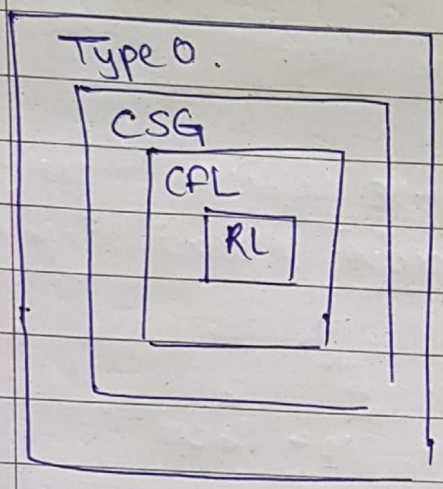
Page No.
Date

⑤ The string generated by the most application of production is called as the working string.

⑥ The derivation of a string is complete when the working string cannot be modified. If the final string doesnot contain any variable, it is called as a sentential form.

Chomsky classification of languages

Type 0
Type 1
Type 2
Type 3



The defination of G
Vn & Σ are sets of symbols
S is the elment of Vn, so
If we want to classify grammar we have to do it by considering the form of production.
Chomsky classified grammar in 4 types depending production -

1) Type 0
2) Type 1
3) Type 2
4) Type 3

Type 0: Accepted by ~~Turing~~ Turing machine (Recursively enumerable grammar)

Type 1: Context sensitive (~~LBA~~ Linear Bounded Automata)

Type 2: Context free (Push Down Automata)

Type 3: Regular Expression - FA < DFA / NFA.

Date

① Type 0: 'G' is an any phase structure grammar without any restriction. The production of the form ϕ

$\phi A \psi \longrightarrow \phi \alpha \psi$

ϕ → left context, A → variable, ψ → right context, α → replacement string.

Ex1 $abA\underline{bcd} \longrightarrow ab\underline{ABbcd}$

left context: ab; A; right context: bcd

$\alpha = AB$

Ex2 $AC \longrightarrow A$

left context: A

null → right context

$\alpha = \wedge$ (null)

This production erases C when left context is A & right context is $\wedge$.

② Type 1: A production of the form.

$\phi A \psi \longrightarrow \phi \alpha \psi$

is called type 1 production where $\alpha \neq \wedge$. In type 1 production, erasing of A is not permitted.

Ex1 $AB \longrightarrow AbBc$

A - left context

$\wedge$ - Right context

$\alpha = bBc$.

The grammar is called Type 1 or context sensitive or context dependent. If all its production are type 1 production. The

Page No.
Date

(3) ~~Type 0~~: production $S \to \wedge$ is also allowed in Type 1. But in this case S doesn't appear on the right hand side of any production

✓ $S \to aA/\wedge$

✗ $S \to aAS/\wedge$

(2) ~~T~~ The language generated by Type 1 is called a context-sensitive language.

(3) Type 2: production is a production of the form $A \to \alpha$ where A is the element of V_N and α is the element of $\alpha \in (V_N \cup \Sigma)^*$. In other words LHS has no left context & right context.

$S \to Aa$.

$A \to a$

$B \to abc$.

$A \to \wedge$

A grammar is called Type 2, if it contains only Type 2 production, is called as context free grammar. A language generated by context free grammar is called Type-2 or context free language.

(4) Type 3: production of the form

$A \to a$

$A \to aB$

$A, B \in V_N$

$a \in \Sigma$

is called type 3 th production If the production rules are the form in which RHS contains at most one non-terminal symbol is known as Type 3 grammar. The language generated by Type-3 grammar is recognised by FA & are identical of regex. grammar.

TOC

Formal languages :

Q $L_1 = \{a^n b^n \mid n \geq 1\}$

$G_1 = (\{S\}, \{a, b\}, P, S)$

where $P \rightarrow aSbb \quad S \rightarrow abb$

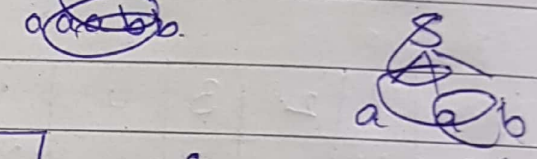
Q Construct the grammar generated by

$L_2 = \{a^m b^n \mid m > n, m, n \geq 1\}$

$S \rightarrow aS \mid aA$

$A \rightarrow aAb$

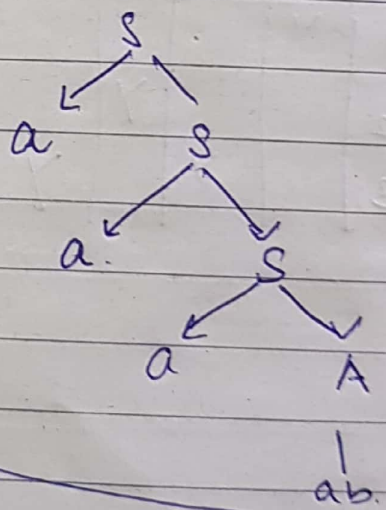
$A \rightarrow ab$



Q $L = \{a^n b^n c^i \mid n \geq 1, i \geq 0\}$

Q $L = \{a^i b^n c^n \mid n \geq 1, i \geq 0\}$

Q $L = \{0^n 1^n 2^n \mid n \geq 1\}$



Q Construct a grammar which generates all the even integers upto 998.

Q Find L(G) for $G(\{A, S\}, \{0, 1\}, P, S)$

where P is $S \rightarrow 0A \mid 1S \mid 0 \mid 1$

$A \rightarrow 1A \mid 1S \mid 1$

Generally there are three approaches to implement lexical analyser

1) Use a lexical analyser generator such as next compiler to produce lexical analyzer from a regular expression based specification in this case the generator provides routine for reading & buffering the input

2) Write the lexical analyser in the conventional ~~system~~ program language using the input & output des ... to read the input.

3) Write the lexical analyser in assembly language & explicitly manage the ~~lor~~ reading of input.

2) Sentinels
If we use the above input buffering scheme we must check each time the look ahead pointer that we have not moved ~~to~~ one half of the buffer. If we do then we must reload the other buffer when end of file of second half is generated, then only the first half will be erased

3) Lexeme : It is a sequence of characters in a program i.e. matched by the pattern of token

Q different b/w tokens & lexeme.?

THEORY OF COMPUTATION

Regular
The language accepted by FA is described by simple expressions called regular expression.
Eg. A regular language over Σ^* is one that can be obtained from these language using the operation of union, concatenation, kleen *.

Regular set : A set represented by a regular expression.

eg. $a, b \in \Sigma$, (i) $\{a\}$
(ii) $a+b = \{a, b\}$
(iii) $ab \quad \{ab\}$
(iv) $a^* \quad \{\epsilon, a, aa \ldots\}$
(v) $(a+b)^* \quad \{a, b\}^*$

The set represent by R is denoted by $L(R)$.

Describe the following set by regex.
(i) $\{101\}$
$\{1\}, \{0\}$
↓ ↓
1 0
concatenating 101

(ii) $\{abba\}$
abba

(iii) $\{01, 10\}$
$\{01\}, \{10\} \rightarrow 01+10$

(iv) $\{\lambda, ab\} \rightarrow \lambda + ab$

(5) $\{1, 11, 111 \ldots\}$
Any element in this set is obtained by concatenating 1 by $\{1\}^*$
$1 1^*$

We have following operation on regex.

1. Union

   $L_1 \cup L_2$

   $L_1 = \{ab, ba\}$ $L_2 = \{bb, aa\}$

   $L_1 \cup L_2 = \{aa, ab, ba, bb\}$

2. Concatenation

   $\{abaa, abbb, baaa, babb\}$

A regular language can be described by explicit formula, we think of a regular expression as representing the most typical string in corresponding language.

Identities:

Two regular expression P and Q are equivalent if P & Q represent the same set of string.

(i) $\phi + R = R$

(ii) $\phi R + R\phi = \phi$

(iii) $\wedge R = R\wedge = R$

(iv) $\wedge^* = \wedge$ and $\phi^* = \wedge$

(v) $R + R = R$

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

(vii) $RR^* = R^*R$

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

(ix) $\wedge + RR^* = R^* = \wedge + R^*R$

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

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

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

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

Imp: to prove these identities.

ARDEN'S THEORUM

Let P and Q be two regular expression over Σ. If P does not contain $\wedge$, then following equation in R, $R = Q + RP$ has a unique solution given by

$$R = QP^*$$

Proof:

$$\begin{aligned} R &= Q + QP^*P \\ &= Q(1 + P^*P) \\ &= QP^* \end{aligned}$$

Hence first is satisfied when $R = QP^*$.
$\therefore R = QP^*$ is the solution of first.

To prove uniqueness consider (1) here by replacing R by $Q + RP$ on RHS.

$$\begin{aligned} R &= Q + (Q + RP)P \\ &= Q + QP + RPP \\ &= Q + QP + QP^2 + RP^3 \\ &= Q + QP + QP^2 \dots + QP^i + RP^{(i+1)} \\ &= Q(\wedge + P + P^2 \dots + P^i) + RP^{(i+1)} \end{aligned}$$

from (1), $R = Q(\wedge + P + P^2 \dots + P^i) + RP^{(i+1)}$ — (2)

We now should that any solution of (I) is equivalent to QP^*. Suppose R satisfies (i) then it satisfies (ii), Let w be a string of length i in Set R, then w belongs to the set $Q(\wedge + P + P^2 \dots + P^i) + RP^{(i+1)}$

As P doesn't contain null, R_{p+1} has no string of length less than $|P|$.
∴ w is not in the set R_{p+1}. This mean that $w \in Q(\wedge + P^1 \cdots + P^i)$
∴ $w \in QP^*$

Consider a string w in set QP^*. If w is in QP^*, then w is in the set $Q(\wedge + P^1 \cdots + P^k)$ where $k \geq 0$.

~~Hence~~

So w is in the RHS of second. So w is in R.
R and QP^* represent the same set. This proves the uniqueness of first.

Q Prove

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

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

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

$$1(\wedge+00^*)1(0+10^*1)^*$$

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

$0+0P = Q(1+P)$

$\wedge + PP^* = P^*$

$\wedge + PP^* = P^*$

→ closure properties of Regular set.

The regular sets are closed under union, concatenation & Kleene closure (here closed means if a & b belong to set A and a+b also belongs to set A, then a is closed under addition operation).

Let R_1 and R_2 denote any two regex. Any string in R_1+R_2, is a string from R_1 or a string from R_2. Thus the set represented by R_1+R_2 is a union of set represented by R_1 & R_2. Hence the regular set are closed under union.

Let R_1 & R_2 denote any two regex. Any string in R_1R_2 is a string from R_1 followed by a string from R_2. Thus the set represented by R_1R_2 is a concatenation of the set represented by R_1 & R_2. Hence by def. regular set are closed under concatenation

Let R_1 denote a regex. Any string in R^* is a string obtained by concatenating elements for some $n \geq 0$. Thus the set represented by R is $\{w_1 w_2 \ldots w_n \mid w_i\}$ is, w_i is the set represented by R & $n \geq 0$. Hence by def. the regular sets are closed under kleen closure.

(Mention kleene closure)

FINITE AUTOMATA (TOC)

Page No.
Date

Transition system containing ∧-moves, the ~~gen~~

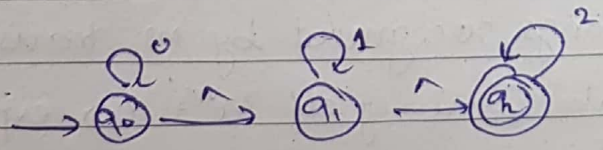
the transition system can be generalised by committing null ~~no~~ input (no input).

It is possible to convert a transition system with null ~~moves~~ moves into an equivalent system without ∧ moves.

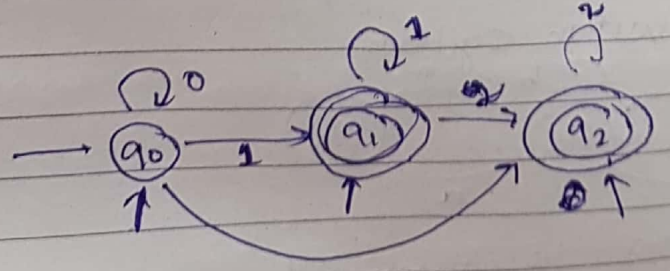
Suppose we want to replace ∧ move from vertex v_1 to vertex v_2, we proceed as.

1. Find all the edges starting from v_2.
2. Duplicate all these edges starting from v_1, without changing the edge labels
3. If v_2 is an initial state, Make v_2 also an initial state
4. If v_2 is a final state, make v_1 also as the final state.

Consider a finite automata with ∧ moves, obtain an equivalent automata without ∧ moves.



we first eliminate the ∧ move from q_0 to q_1.



→ Algebraic method using Arden's theorem.

The following method is an extension of arden's theorem. It is used to find a regular expression recognised by transition system.

Following assumptions are made—

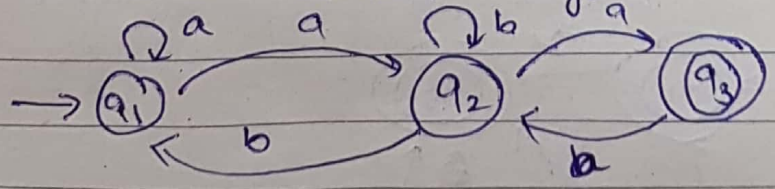
1) The transition graph doesn't have ∧ moves
2) It has only one initial state.
3) Its vertices are $v_1 \ldots v_n$
4) v_i, the regular expression, represents the set of strings accepted by system even though q_i is the final state.
5) α_{ij} represents the regex representing the set of labels of edges from q_i to q_j when there is no such edge $\alpha_{ij} = \phi$.

so we get

$$v_1 = v_1 \alpha_{11} + v_2 \alpha_{21} \cdots + v_n \alpha_{n1} + \wedge$$
$$v_2 = v_1 \alpha_{12} + v_2 \alpha_{22} \cdots + v_n \alpha_{n2}$$
$$\vdots$$
$$v_n = v_n \alpha_{1n} + v_2 \alpha_{2n} \cdots + v_n \alpha_{nn}.$$

For getting the set of strings recognised by the transition system, we have to take union of all v_i's corresponding to final states

Q Consider the transition system



prove that the string recognised are

$$(a + a(b + aa)^* b)^* a(b + aa)^* a.$$

$q_1 = q_1 a + q_2 b + \wedge$

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

$q_3 = q_2 a$

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

$q_2 = q_1 a (b + aa)^*$

$q_1 = q_1 a + q_1 a (b + aa)^* b$

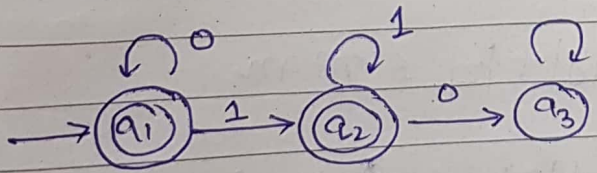
$q_1 = q_1 (a + a (b + aa)^* b)$

$q_1 = (a + a (b + aa)^* b)^*$

$q_2 = (a + (a + (b + aa)^* b)^* a (b + aa)^*$

$q_3 = (a + (a (b + aa)^* b))^* a (b + aa)^* a.$

Q. Describe the set accepted by finite automata



$q_1 = q_1 0 + \wedge$

$q_2 = q_1 1 + q_2 1$

$q_3 = q_2 0 + q_3 (0+1)$

$q_1 = 0^*$

$q_2 = 0^* 1 1^*$

$q_3 = 0^* 1 1^* + q_3 (0+1) \Rightarrow 0^* 1 1^* (0+1)^*$

$q_1 + q_2 = 0^* (\wedge + 1 1^*) = 0^* 1^*$

Concatenation: If R_1 and R_2 denote any two regex, a string in R_1R_2 is a string from R_1 followed by a string from R_2.
Thus, the set represented by R_1R_2 is the concatenation of the sets represented by R_1 and R_2. Hence, by definition regular sets are closed under concatenation.

Kleene closure: Let R denote a regex. The string in R^* is a string obtained by concatenating for some $n \geq 0$. Thus the set represented by R^* is $\{w_1, w_2, \ldots, w_n\}$ for w_i w_i is the set represented by R and $n \geq 0$.
Hence, by definition, regular sets are closed under Kleene closure [mention definition of Kleene closure].

08/09/2018
Thursday

Pumping Lemma

Here we give a neccessary condition to an FP

Theorem:
The result is called the pumping lemma as gives a method of pumping many FP strings from a given string. As pumping lemma gives a necessary condition,

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automata with n-states. Let L be the regular set accepted by L. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, $\exists\ x, y, z$ such that $w = xyz$ where $y \neq \wedge$ and $xy^iz \in L\ \forall\ i \geq 0$

DELTA Notebook

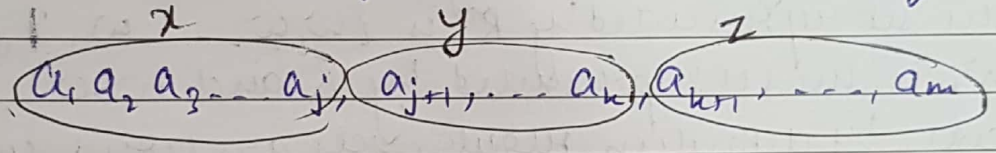
Dt.: Pg.: Delta

Proof Let $w = a_1 a_2 a_3 \ldots a_m$, $m \geq n$

$\delta(q_0, q_1, q_2, \ldots q_i) = q_i$ for $i = 1, 2, \ldots m$

$Q_1 = \{q_0, q_1 \ldots q_m\}$

Q_1 is the sequence of states in the path with path value $w = a_1 a_2 a_3 \ldots a_m$. As there are only 'n' distinct states, at least two states in q_1 must coincide among the various pairs of repeated states, we take the first pair. Let us take them as q_j and q_k such that $(q_j = q_k)$, then 'j' and 'k' satisfy the condition $\rightarrow$ $0 \leq j < k \leq n$. The string 'w' can be decomposed into three substrings.

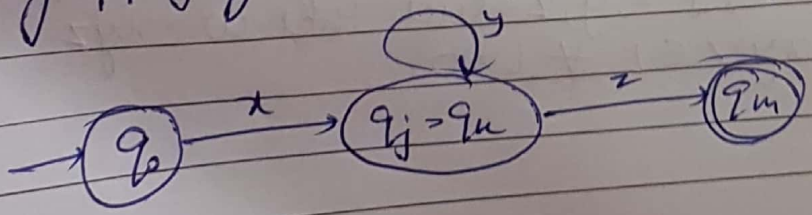


Let x, y, z denote three strings respectively.

As $k \leq n$

$|xy| \leq n$ and $w = xyz$

The path with path value 'w' is the transition path of m. The automata M starts with initial state q_0. On applying string x, it reaches $q_j = q_k$. On applying string 'y', it comes back to $q_j = q_k$. After the application of y^i $\forall$ $i \geq 0$, the automata is in the same state q_j. On applying 'z', it reaches q_m - a final state. Hence, $xy^iz \in L$ as every state in q_1 is obtained by applying an I/P symbol. So, $y \neq \wedge$.



NOTE: The decomposition is valid only for strings of length ≥ no. of states.

Dt.:
Pg.:
Delta

Applications

1) Used to prove that certain sets are not regular. The following steps are needed for proving that L

Assume that L is regular.
Let 'n' be the no. of states in the corresponding finite automata.
Choose a string 'w' such that $|w| \geq n$.
Choose $w = xyz$ such that $|xy| \leq n$, $|y| > 0$.
Find a suitable integer 'i' such that $xy^iz \notin L$.
This contradicts our assumption. Hence, L is not regular.

NOTE: The important part is to find 'i' such that $xy^iz \notin L$
In some cases, we prove this by considering $|xy^iz| \notin L$.
In some cases, we have to use the structure of strings in L.

Q) Show that set L is not regular.

$L = \{a^{i^2} \mid i \geq 1\}$

Sol) Let L be regular
Let n be the no. of states in finite automata accepting L.
Let $w = a^{n^2}$, $|w| = n^2 > n$
$w = xyz$, $|xy| \leq n$, $|y| > 0$
Let $i = 2$

$\Rightarrow w \Rightarrow xy^2z$

$\Rightarrow |w| = |xy^2z|$
$= |x| + 2|y| + |z| > |xyz| = |x| + |y| + |z|$
$= |x| + 2|y| + |z| > |x| + |y| + |z|$

$|y| > 0$
$n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$

DELTA Notebook

As $|xy| < n$, $|y| \leq n$

$\therefore |xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$

$\therefore n^2 < |xy^2z| \leq n^2 + n < n^2 + n + n + 1$

Hence $|xy^2z|$ lies strictly b/w n^2 and $(n+1)^2$ but is not equal to any of them.

$\Rightarrow |xy^2z|$ is not a perfect square.

$\therefore |xy^2z| \notin L$

This is a contradiction.

$\therefore$ L is not regular.

Q) $L = \{0^i 1^i \mid i \geq 1\}$

Sol) Let L is regular

Let n be the no. of states in the finite automata accepting L.

Let $w = 0^n 1^n$

$|w| = 2n > n$

$w = xyz$, $|xy| \leq n$, $|y| \neq 0$

Choose i such that $xy^iz \notin L$

String y can be of any type -

Case 1: $y = 0^k$, $k \geq 1$

Case 2: $y = 1^l$, $l \geq 1$

Case 3: $y = 0^k 1^j$, $k, j \geq 1$

Case 1: $i = 0$

$xyz = 0^n 1^n$, $xz = 0^{n-k} 1^n$, $k \geq 1$, $n - k \neq n$, $xz \notin L$

Conversion of regular expression to finite automata

⇨ (a/b)* abb

(a/b) :

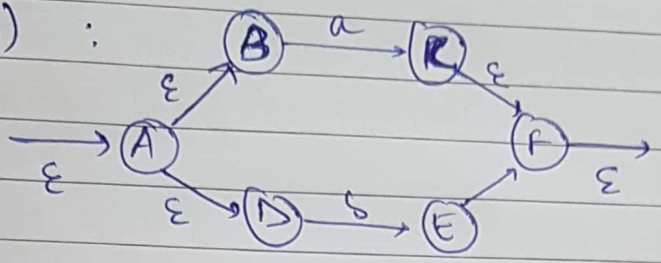
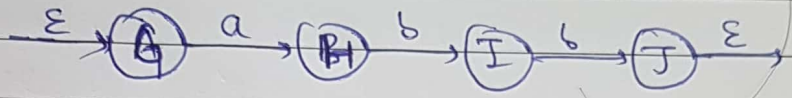
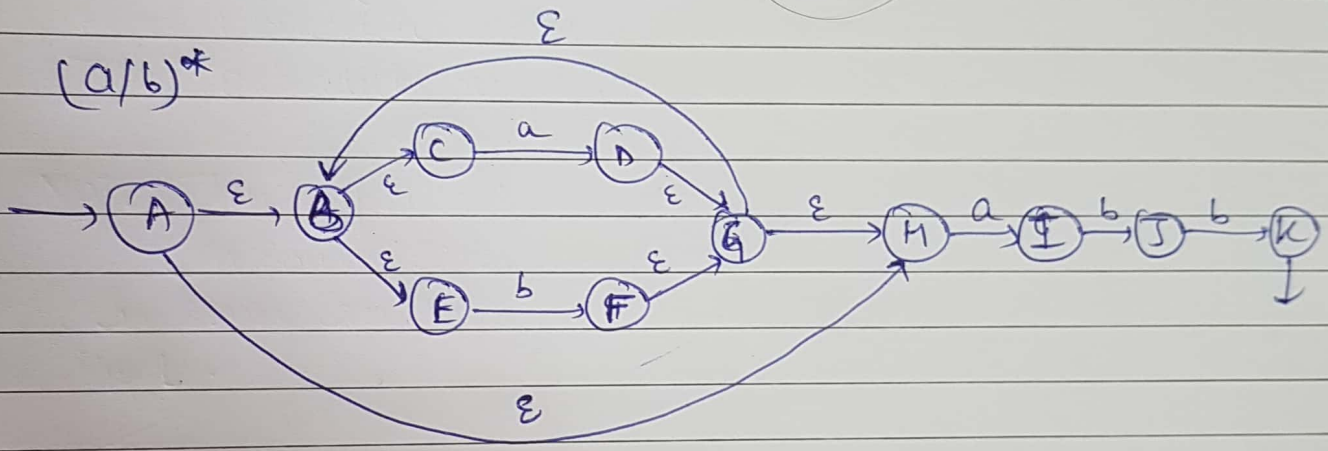


abb :



(a/b)*



	ε^*	a	ε^*
A	A	∅	∅
	B	∅	∅
	H	a	H
B	B	∅	∅
	C	D	G
	E	∅	∅
C	C	D	G
D	D		
	G		
E	E		
F	F		
	G		
G	H		

DELTA Notebook