THEORY OF COMPUTATION

Neso Academy
Notes

Prerequisites

- Symbol → eg a, b, c, 0, 1 --- are all symbol
- Alphabet (Σ) → collection of symbols
  eg $\{a, b\}$, $\Sigma = \{d, e, f\}$
- String → Sequence of symbols
  eg a, b, abcd, 12ab3c.
- Language → Set of strings
  eg L_1 = set of all strings from $\Sigma = \{a, b\}$ of length 2.
  $= \{ab, ba, aa, bb\}$ (finite)

  L_3 = Set of all strings that begin with 0 over $\Sigma = \{0, 1\}$
  $= \{0, 00, 01, 000, 010, 001, 011, 0000, \ldots\ldots\}$
  (infinite Language)

- Power of Σ : Let $\Sigma = \{0, 1\}$

Σ^0 = set of all strings of length 0 : $\Sigma^0 = \{\epsilon\}$

Σ^1 = set of all strings of length 1 : $\Sigma^1 = \{0, 1\}$

Σ^2 = set of all strings of length 2 : $\Sigma^2 = \{00, 11, 01, 10\}$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 - - - \Sigma^n \qquad n \to \infty$$

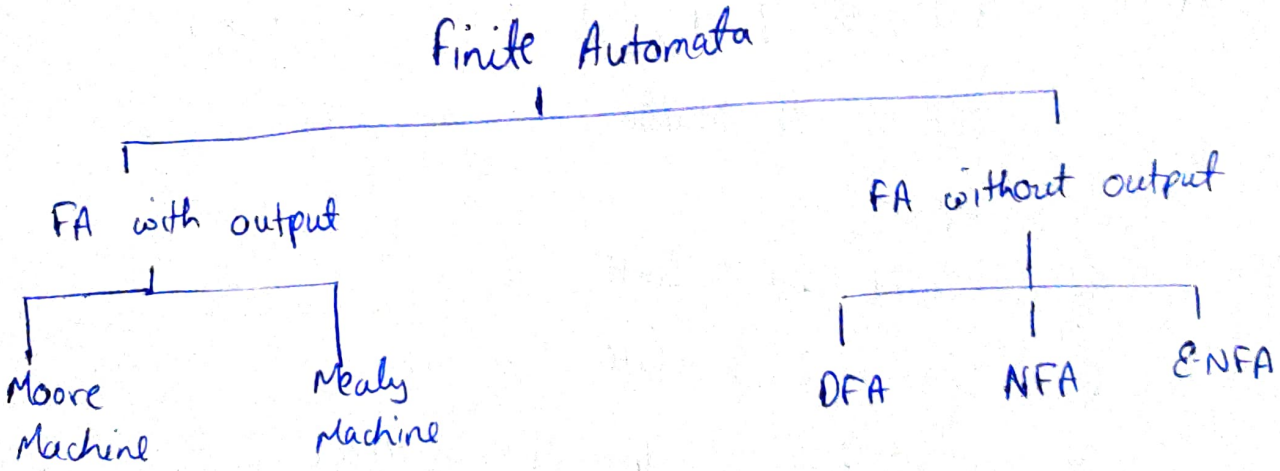
- Cardinality :- no. of elements in a set

for $\Sigma^n = 2^n$

eg $\Sigma^2 = 2^2 \Rightarrow$ cardinality is 4 for Σ^2.

for binary strings

Finite State Machine



they are simplest model of computation
have very limited memory.

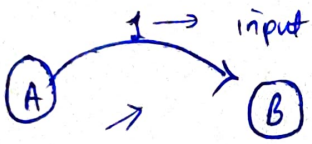
DFA (Deterministic finite Automata)

The finite automata are called deterministic if machine read an input string one symbol at a time. In DFA, there is only one path for specific input from current state to next state.

Diagram

Starting state

finite State. (final state)



transition from state A to B on getting input 1.

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

Q = set of all states

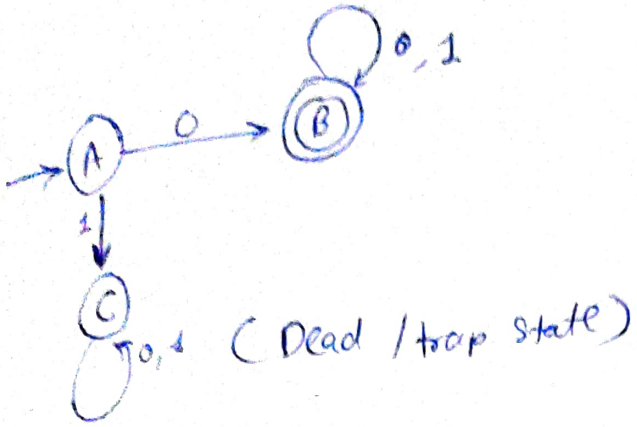
Σ = inputs

q_0 = start / initial state

F = set of final state

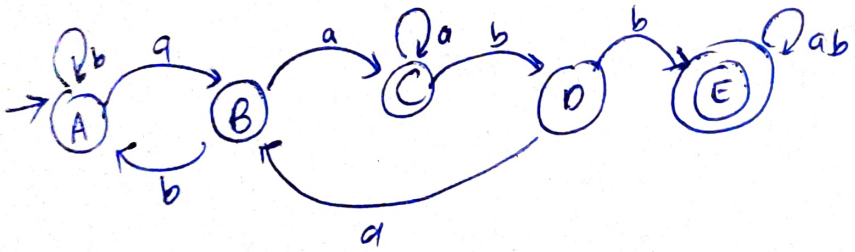
δ = transition Function from $Q \times \Sigma \to Q$

Q1) Set of all strings that start with 0.



Q2 Construct a DFA that accepts any string over {a, b} that does not contain string aabb in it.

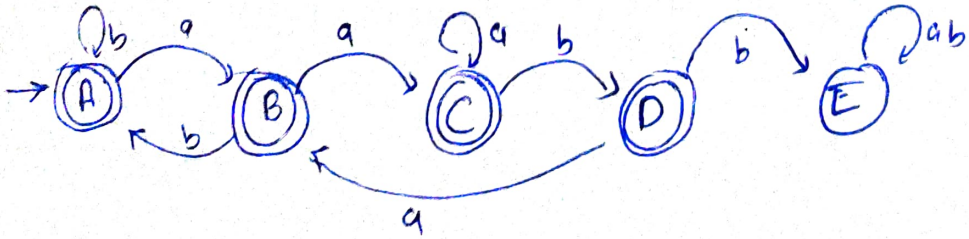
So let us construct a DFA that accepts all strings over {a, b} that contains strings aabb in it.



now want doesnot contain aabb

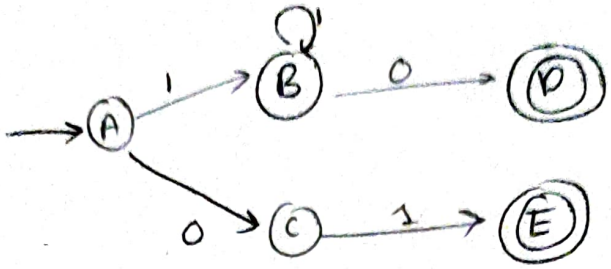
do it by Flipping the state

→ Make final state into non final state & non final state into finite state.



eg3

How to figure out what a DFA recognises ?



It is NFA (as some transitions not shown)

Some strings it accepts are

10 ✓

11110 ✓

A B D

01

C E

So

L = { Accepts string 01 or string of atleast one '1' followed by a '0' }

Regular Language

- A Language is said to be a Regular language if and only if some finite state machine recognizes it.

- So what languages are NOT REGULAR ?
  - » Not regnized by finite state machine
  - » requires memory.

eg abbab abbab

So rule is first 5 char. are repeated
So it requires memory
hence it is non-regular.

eg 2 $a^N b^N$
is not-regular as finite state machine is not having count feature
So it is non-regular.

Operations on Regular language

UNION $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$

CONCATENATION $A \circ B = \{ xy \mid x \in A \text{ and } y \in B \}$

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

eg $A = \{pq, r\}$, $B = \{t, uv\}$

- $A \cup B = \{pq, r, t, uv\}$
- $A \circ B = \{pqt, pquv, rt, ruv\}$
- $A^* = \{\epsilon, pq, r, pqpq, rr, pqrr, pqrpq, \cdots\}$

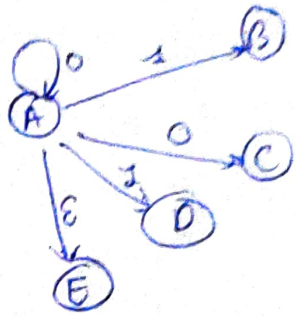
Non-Deterministic Finite Machine

(NFA)

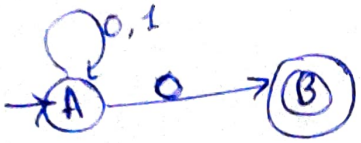
> In NFA, given the current state there could be multiple next states.

» The next state may be choosen at random

» All the next state may be chosen in parallel.



eg 2)



all strings end with 0.

here at B we didn't wrote any trainsition means it stay at B & if anything added so dead state.

NFA = $(\varphi, \Sigma, q_0, F, \delta)$

here $\delta = \varphi \times \Sigma \to 2^{\varphi}$

as 2^{φ} as → for A 1 → A, B, AB, ∅

So for 2 state A, B it can go to 2^{φ} states.

here $\varphi = \{A, B\}$

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

$q_0 = A$

$F = B$

δ: A x 0 → A

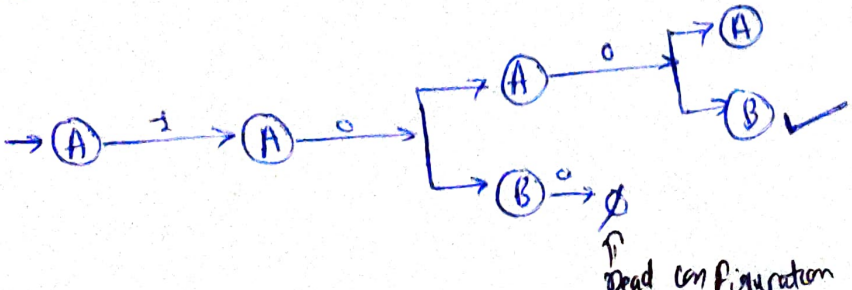
A x 0 → B

A x 1 → A

B x 0 = ∅

B x 1 = ∅

here let for 100



Even at least one way/path accepts string then NFA accepts the string

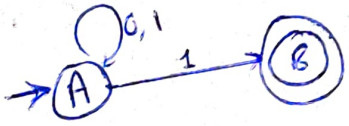
eg2) Construct NFA that accepts sets of all strings on {0, 1} of length 2.

Conversion of NFA to DFA

- Every DFA is an NFA but not vice versa
  But there is an equivalent DFA for every NFA

eg L = {Set of all strings over (0, 1) that ends with '1'}

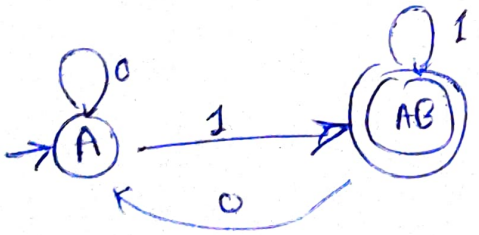
NFA =



Transition table

	0	1
A	{A}	{A, B}
B	∅	∅

its DFA



ist this

then this

consider {A, B} as one state in DFA

	0	1
A	{A}	{AB}
AB	{A}	{AB}

also

- Can't reach B from A anyway so no need to make.
- AB on 0 goes to $(A \xrightarrow{0}) \cup (B \xrightarrow{0})$
- Final state is one with atleast one final state of NFA, here {AB}

eg2) Find equivalent DFA for NFA given by $M = [\{A,B,C\}, \{a,b\}, \delta, A, \{C\}]$ where

$\delta =$

	a	b
→A	A,B	C
B	A	B
©C	-	A,B

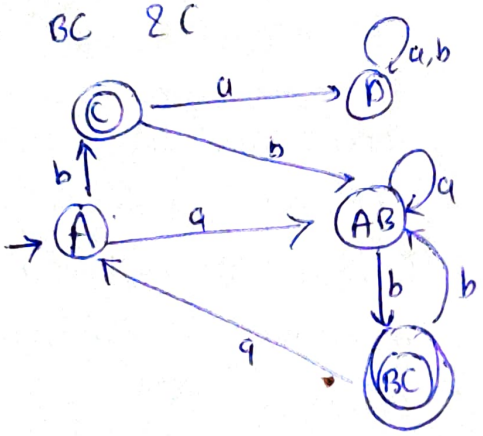
for DFA ⇒

	a	b
→A	{AB}	C
AB	AB	BC
©BC	A	AB
©C	D	AB
D	D	D

for $- \equiv D$ where D is dead state

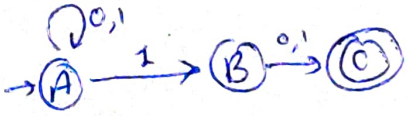
Dead state remains in dead state.

- final state in NFA is C
so for DFA = BC & C



Q3 Design a NFA for a language that accepts all strings over {0,1} in which second last symbol is always '1'. Then convert it to its equivalent DFA.

NFA

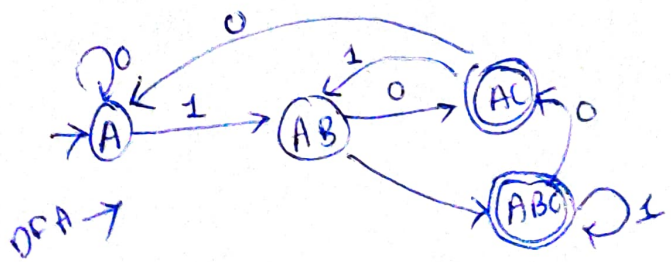


NFA

	0	1
→A	A	A,B
B	C	C
©C	∅	∅

DFA →

	0	1
→A	A	{AB}
AB	AC	ABC
©ABC	AC	ABC
©AC	A	AB



Minimization of DFA

- Minimization of DFA is required to obtain minimal version of any DFA which consists of minimum number of states possible.

To do this we combine two or more states which are equivalent.

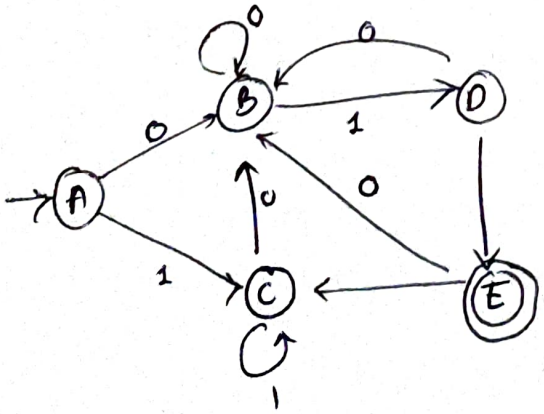
equivalent states.

$\delta(A,x) \rightarrow F$ and $\delta(B,x) \rightarrow F$ OR $\delta(A,x) \not\rightarrow F$ and $\delta(B,x) \not\rightarrow F$

x is any input string
F Final state

- If $|x| = 0$, then A & B are said 0 equivalent
- if $|x| = 1$, " —— " —— " 1 equivalent
- if $|x| = n$, then A & B are said n equivalent.

Q Minimize given DFA



first make transition table

	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

Start from 0 equivalence & continue

→ 0 equivalence : {non final state} , {final state}

here

◉ 0 Equivalence {A,B,C,D} {E}

→ 1 equivalence : if like for checking AA

if $A \xrightarrow{0} =$
$A \xrightarrow{1} =$
$B \xrightarrow{0} C$
$B \xrightarrow{1} =$

} this values lie in {ABCD} of 0 equivalence then AB is 1 equivalent

*for C check either with A or B

◉ 1 equivalence {A,B,C} {D} {E}

→ 2 equivalence new set to compare for output is {ABC}

$B \xrightarrow{1} D$ so D not in {ABC}

but A,C all output in {ABC}

◉ 2 equivalence {AC} {B} {D} {E}

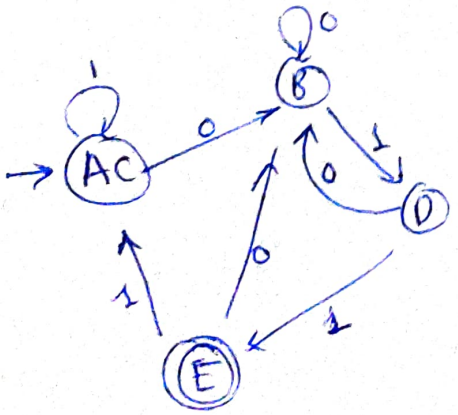
→ 3 equivalence {AC} {B} {D} {E}

Both output same (may not be in {AC} but same set {B} & {AC}

So (outputs from same given sets)

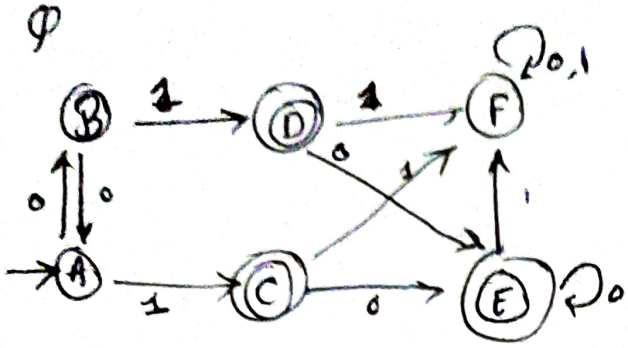
So 3 equivalent.

Stop when p {p-1 equivalence are same (here 3 {2 same)



Minimization of DFA - Table filling Method (Myhill - Nerode Theorem)

Q



STEPS :-

1) Draw table for all pairs of states (P, Q)
2) Mark all pairs where P ∈ F and Q ∉ F
3) If there are any unmarked pairs (P, Q) such that [δ(P, x), δ(Q, x)] is marked, then mark [P, Q] where 'x' is an input symbol.
   REPEAT THIS UNTIL NO MORE MARKINGS CAN BE DONE
4) Combine all unmarked pairs & make them a single state in minimized DFA.

	A	B	C	D	E	F
A						
B						
C	✓	✓				
D	✓	✓				
E	✓	✓				
F	✓	✗	✓	✓	✓	

remove as if include this
So AB & BA both include but we don't need it

- not include AB as no one is in final state
include CA is (C), AD is not path but ✓

③ Unmarked → (B,A), (D,C), (E,C), (E,D), (F,A), (F,B)

(B,A): δ(B,0) = A, δ(B,1) = D; δ(A,0) = B, δ(A,1) = C → DC is unmarked
AB is also unmarked.

(D,C): ×

(E,C): ×

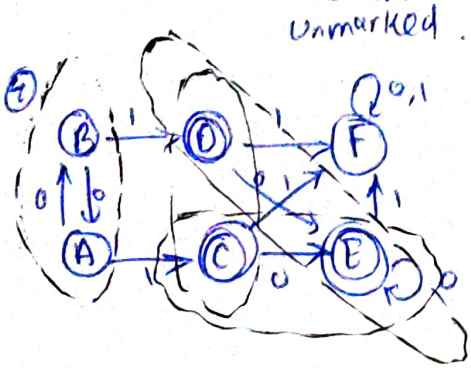
(E,D): ×

(F,A): δ(F,0) = F, δ(F,1) = F; δ(A,0) = B, δ(A,1) = C → FC is marked
So mark this also.

(F,B): δ(F,0) = F, δ(F,1) = F; δ(B,0) = A, δ(B,1) = D → marked FB

(Repeat process 3 till no δ(P,x), δ(Q,x) pair is found)

④



Unmarked AB DC EC ED.

Finite Automata with Outputs

Mealy Machine

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

Q = Finite set of states

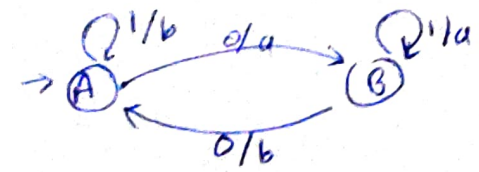
Σ = Finite non-empty set of Input Alphabets

Δ = Set of Output Alphabets

δ = Transition fn : $Q \times \Sigma \to Q$

λ = Output fn : $\Sigma \times Q \to \Delta$

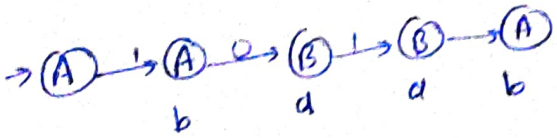
q_0 = Initial state.



1/b
↑ output

→ output depends on State & input

Eg 1010



output (baab)

|input| = |output|

n = 4

Moore Machine

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

Q = finite set of states

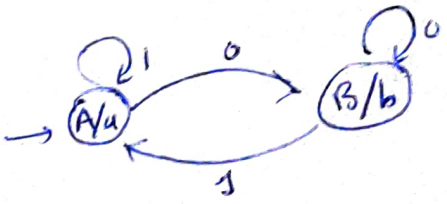
Σ = finite non empty set of Input alphabets

Δ = set of output Alphabets

δ = Transition fn $Q \times \Sigma \to Q$

λ = Output fn : $Q \to \Delta$

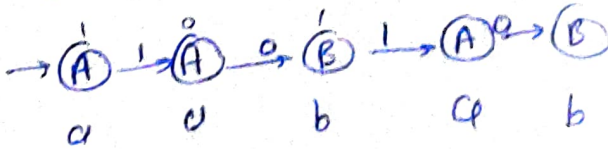
q_0 = Initial state



A/a (at state A reaching produce a)

→ output depends on state

Eg 1010



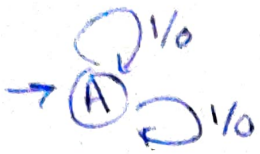
output = aabab

|output| = |input| + 1.

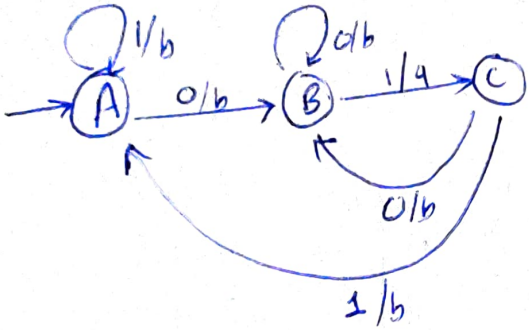
Construction of Mealy Machine

eg) Construct a Mealy Machine that produces 1's complement of any binary input string

eg 10100
01011



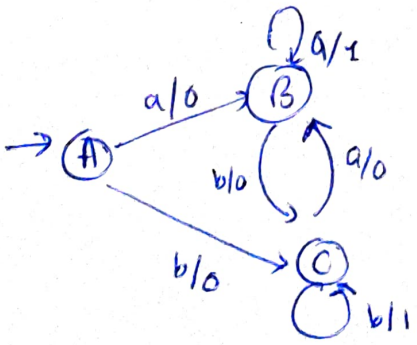
eg) Construct Mealy Machine that prints 'a' whereever sequence '01' is encountered in any input binary string.



here just case of a on getting 01 left cases fill by b.

eg) Design Mealy Machine accepting language consisting of strings from Σ^*, where $\Sigma = \{a, b\}$ & states should end with either aa or bb.

So for accepting aa → 1 print 1
bb → 1



here aa
0 1 ✓
accept

transition table

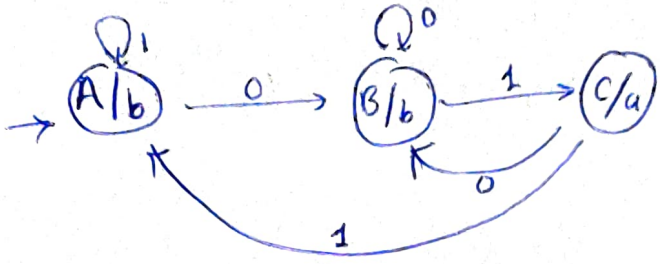
	a	b
A	B, 0	C, 0
B	B, 1	C, 0
C	B, 0	C, 1

Construction of Moore Machine

Construct Moore Machine that prints 'a' whenever sequence '01' is encountered in any input binary string.

$\Sigma = \{0,1\}$ $\Delta = \{a,b\}$

on getting 01 print a



MOORE → states pe output

for 0 1 1 0

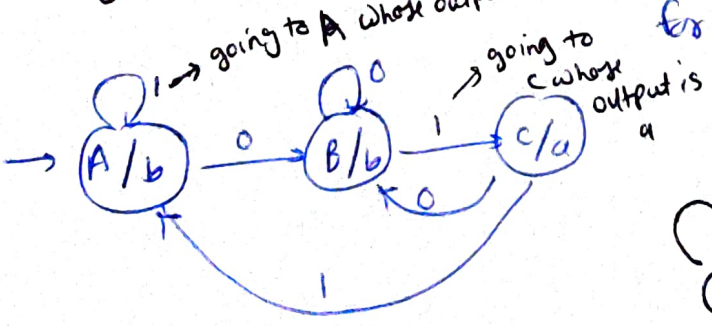
A	B	C	A	B
b	b	a	b	b

transition table →

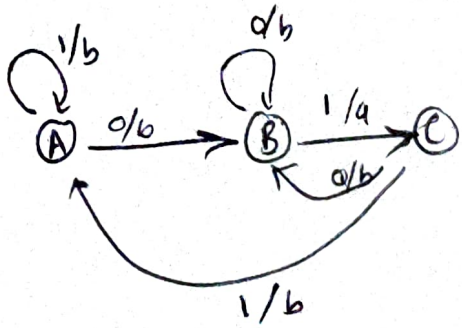
	0	1	output (associate to state)
A	B	A	b
B	B	C	b
C	B	A	a

Conversion of Moore to Mealy Machine

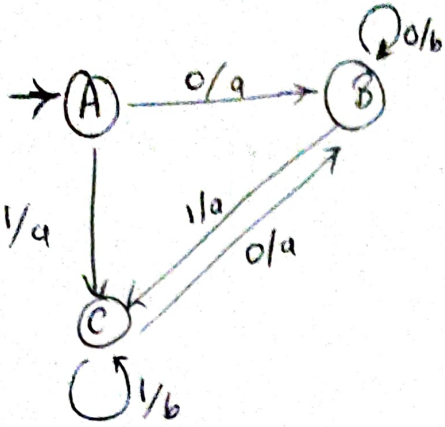
Construct Moore Machine that prints 'a' whenever '01' is encountered in input binary string & convert it to equivalent MEALY MACHINE.



for its mealy



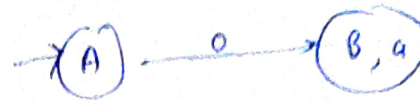
Conversion of Mealy Machine to Moore Machine



for

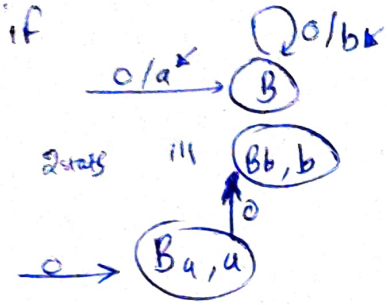
Mealy

III

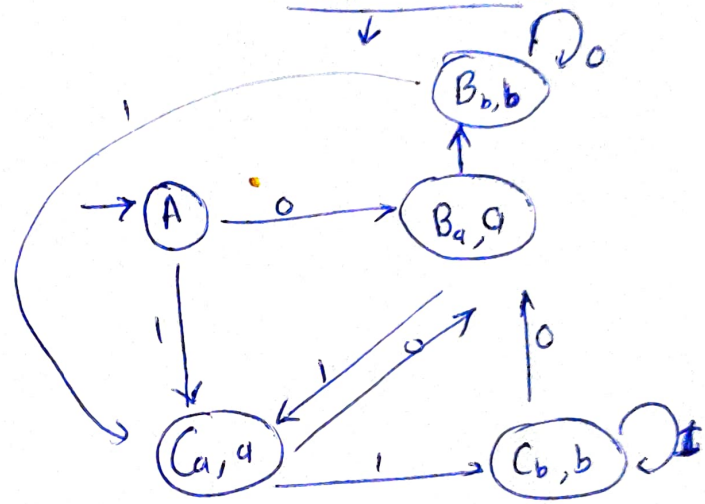


moore

if



Mealy machine to Moore machine

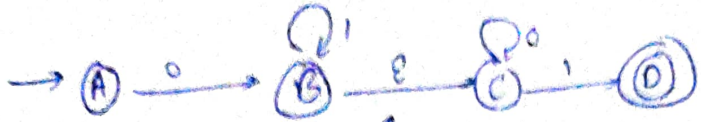


for this see C of mealy

- Moore → Mealy ⇒ No of states were same
- Mealy → Moore ⇒ no. of states increased

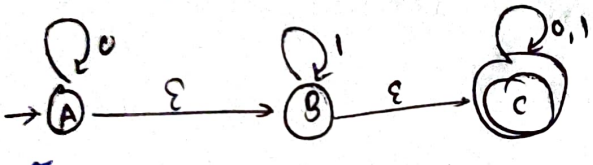
Epsilon (ε) - NFA

here $\delta: Q \times (\Sigma \cup \varepsilon) \to 2^Q$



without getting any input it can go to C from B.

ε-NFA → NFA

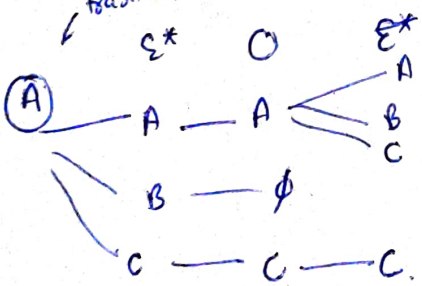


using ε can go to A, B, C

ε closure (ε*) = All state that can be reached from particular state only by seeing ε symbol.

(X is epsilon closure of (itself)

transition to 0 :

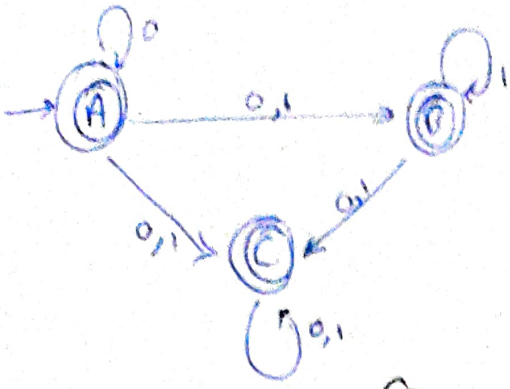


So $A \xrightarrow{0} \{A, B, C\}$

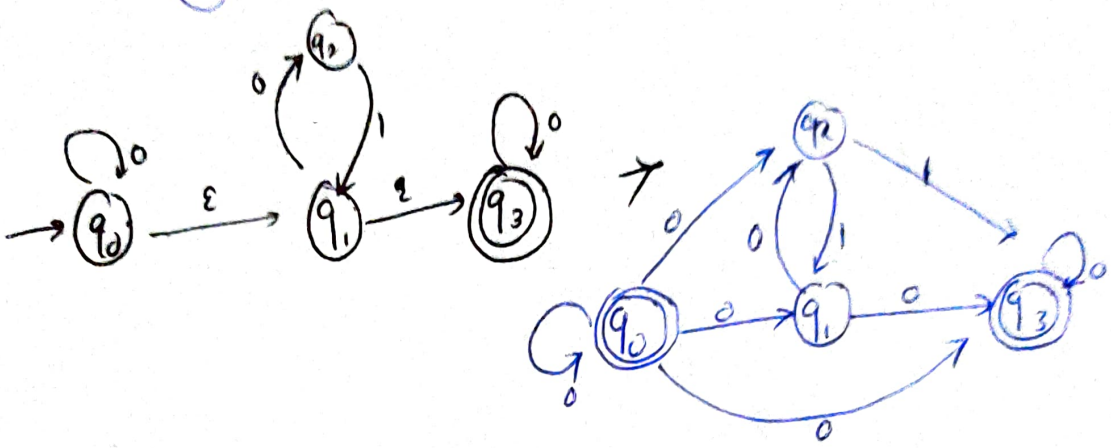
	0	1
→ (A)	{A BC}	{B, C}
(B)	{C}	{B, C}
(C)	{C}	{C}

	ε*	1	ε*
→ A	A	ϕ	
	B	B	B, C
	C	C	C

final states = States which can reach final state using ε only.



eg 2)



- difference betn ε & ϕ

  coell ε = 0 length string whereas phi is a null, ie no string.

  eg: you can insert any no. of epsilons (ε) betn 2 alphabets of input string aεεεεb. It won't make a difference.

  but ϕ acts as 0.

  so $a\phi = \phi$

Regular Expression

- Regular Expressions are used for representing certain sets of strings is an algebric expression.

→ The regular expression over Σ are precisely those obtained recursively by application of above rules once or several times

→ any terminal symbol eg sybols ε Σ including ∧ & ∅ are also regular expression. (∧ → empty set)

Q Write regular expression for following :

1) {0, 1, 2} R = 0 + 1 + 2 + → or 0 or 1 or 2

2) {∧, ab} R = ∧ ab

3) {abb, a, b, bab} R = abb + a + b + bab

4) {∧, 0, 00, 000 --} R = 0* (closure of 0) (includes ∧)

5) {1, 11, 111 - - -} R = (1)(1)* or 1(1)* must include one 1

(concatenate)

Identities of Regular Expression

1) ∅ + R = R

2) ∅R + R∅ = ∅ (concatenate)

3) ε* = ε & ∅* = ε

5) R + R = R

6) R* R* = R*

7) (R*)* = R*

8) (P∅)* P = P(∅P)*

9) (P + ∅)* = (P*∅*)* = (P* + ∅*)*

10) (P + ∅) R = PR + ∅R

11) ε + RR* = R*

12) ∅R* or ∅R = ∅.

∅ ≈ 0, ε, ∧ ≈ 1

∅R = ∅ εR or ∧R = R.

Arden's Theorem :- If P & Q are R regular expressions over Σ, and if P does not contain ϵ, then following equation in R given by $R = Q + RP$ has unique soln i.e $R = QP^*$

Proof

$$R = Q + RP \qquad R = QP^*$$
$$= Q + QP^*P$$
$$= Q(\epsilon + P^*P)$$
$$= Q(\epsilon + P^+) = Q(P)^* \quad \text{Proved}$$

*Q Prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

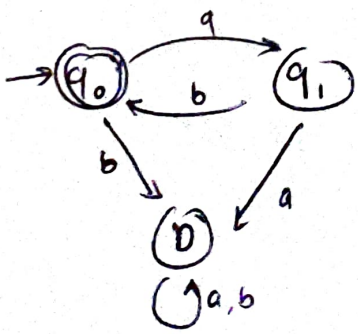
$$\text{LHS} = (1+00^*1)[\epsilon + \underbrace{(0+10^*1)^*}_{P^*}\underbrace{(0+10^*1)}_{P}]$$

$$\epsilon + P^*P = \epsilon + P^* = P^*$$

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

Q Write regular grammers for DFA

$N = \{q_0, q_1, D\}$, $T = \{a, b\}$, $S = q_0$, P:



P →
$q_0 \to aq_1$
$q_0 \to bD$
$q_1 \to bq_0$
$q_1 \to aD$
$P \to aD$
$D \to bD$
$q_1 \to b$
$q_0 \to \epsilon$ → as starting state

putting q_0 in bq_0

Q Design Regular expression for language over {a, b}

1) Language accepting strings of length exactly 2.
2) Language accepting strings of length atleast 2.
3) Language accepting string atmost length 2.

Soln 1) $L = \{aa, ab, ba, bb\}$

$R = aa + ab + ba + bb$

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

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

2) $L = \{aa, ab, ba, bb, aaa, aab \cdots\}$

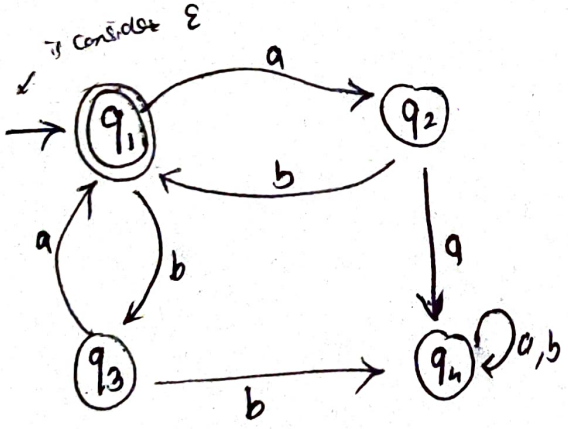
$R = (a+b)(a+b)(a+b)^*$

3) $L = \{\varepsilon, a, b, ab, ba, aa, bb\}$

$R = \varepsilon + a + b + aa + ab + ba + bb = (\varepsilon + a + b)(\varepsilon + a + b)$

Regular Expression for given DFA

Same method for NFA also (Also use of Arden's Theorem)



$q_1 = \varepsilon + q_2 b + q_3 a$ → (1)

$q_2 = q_1 a$ → (2)

$q_3 = q_1 b$ → (3)

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$ → (4)

(1) $q_1 = \varepsilon + q_2 b + q_3 a$

putting values of q_2 & q_3 from 2 & 3

$q_1 = \varepsilon + q_1 ab + q_1 ba$

$q_1 = \varepsilon + q_1(ab + ba)$

$R = \varphi + R\rho$

$R = \varphi \rho^*$

Arden's theorem

$q_1 = \varepsilon (ab + ba)^*$ $\qquad \varepsilon R = R$

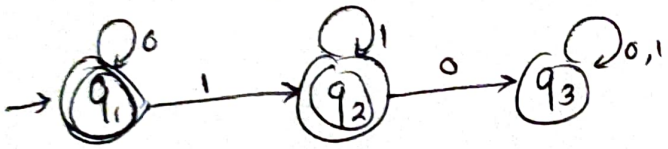
$q_1 = (ab + ba)^*$

since q_1 is final state so this is ans, if q_1 would not have been final state so using this find expression for final state.

Ans = eqn for final state = of q_1 = $(ab + ba)^*$

- **for DFA with multiple final state**

To do → find final eqn for all final state & take their union.



$q_1 = \varepsilon + q_1 0$ → (1)

$q_2 = q_1 1 + q_2 1$ — (2)

$q_3 = q_2 0 + q_3 0 + q_3 1$ — (3)

Final state q_1

$q_1 = \varepsilon + q_1 0$ $\qquad R = Q + RP$

$= \varepsilon 0^*$ $\qquad ans = QP^*$

$= 0^*$

for q_2

$q_2 = q_1 1 + q_2 1 = 0^* 1 + q_2 1$ $\qquad (R = Q + RP)$

$q_2 = (0^* 1)(1)^*$ $\qquad R = QP^*$

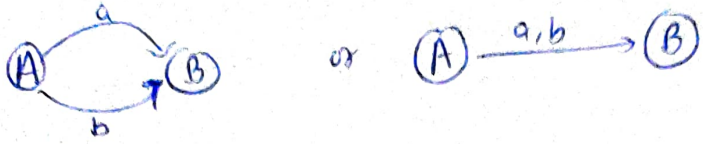
R = union of both final state $\qquad (\varepsilon + RR^* = R^*)$

$= 0^* + (0^* 1)(1^*) = 0^* (\varepsilon + 11^*) = 0^* 1^*$

$= (01)^*$

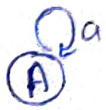
Conversion of Regular Expression To finite Automata

1) (a+b) can go to next state on getting either of a or b



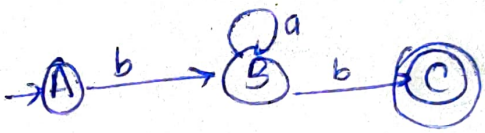
2) (ab)

3) a^*

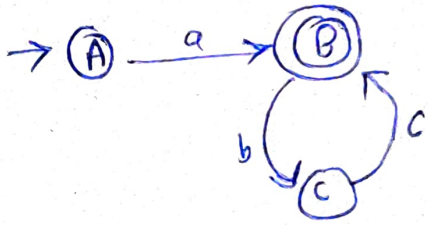


Q convert following to finite Automata.

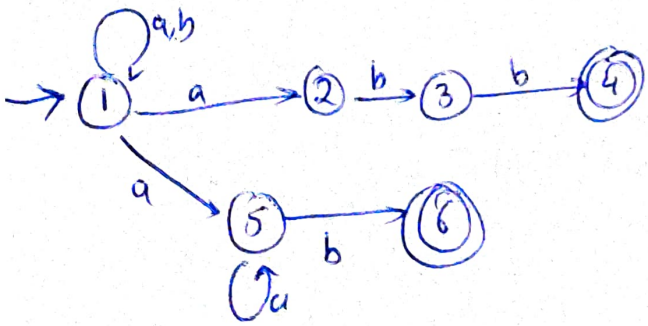
1) $b\,a^*b$



2) $a(bc)^* = a, abc, abcbc$



3) $(a|b)^*(abb|a^*b)$ $\{| \equiv + \text{ as } a \text{ or } b = a+b = a|b\}$



$a^+ = (a, aa, aaa, \ldots)$

$a^* = (\varepsilon, a, aa \ldots)$

Equivalence of two Finite Automata

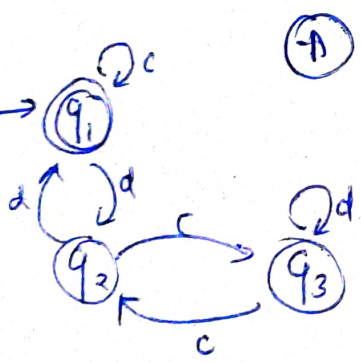
Steps

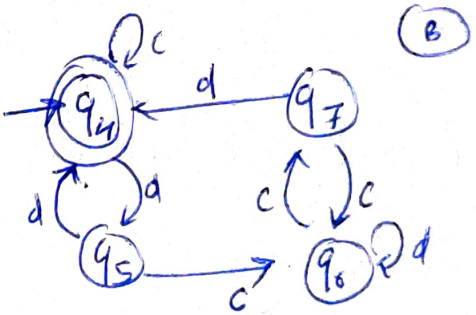
1) for any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta(q_i, a) = q_a$ & $\delta\{q_j, a\} = q_b$

The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is INTERMEDIATE State & other is FINAL state.

2) If initial state is final state of one automaton, then in second automaton also Initial state must be final state for them to be equivalent.

eg





States	for c	d
• (q_1, q_4) Condition 2 ✓	$(q_1\ q_4)$ Final state ↓ FS	(q_2, q_5) IS → Intermediate state (IS)
• (q_2, q_5)	$q_3\ q_6$ IS IS	(q_1, q_4) FS FS
• $(q_3\ q_6)$	$(q_2\ q_7)$ IS IS ↑ new pair	$(q_3\ q_6)$ IS IS
• $(q_2\ q_7)$	$(q_3\ q_6)$ IS IS	$q_1\ q_4$ FS FS

↙ Check for all pairs (even pairs made in betn)

So A & B are equivalent

Pumping Lemma (for Regular Language)

- Pumping Lemma is used to prove that language is NOT regular.
- But it can not be used to prove language is regular.

IF A is regular language, then A has a pumping length 'p' such that string 's' where $|s| \geq p$ may be devided into 3 parts $s = xyz$ such that following condn's must be true:

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

% prove that language is not Regular using Pumping lemma.
(we prove by contradiction)

→ Assume A is Regular
→ It has to have a Pumping length (say P)
→ All strings longer than P can be pumped $|s| \geq p$
→ Now find string 's' in A such that $|s| \geq p$
→ Devide s into x y z
→ Show that $xy^iz \notin A$ for some i
→ Then consider all ways that s can be devided into xyz
→ Show that none of this can satisfy all 3 pumping conditions at same time
→ S can not be Pumped =- CONTRADICTION

Q Using Pumping lemma prove $A = \{a^n b^n \mid n \geq 0\}$ is not regular.

Proof

Assume that A is Regular

Pumping length = p.

$S = a^p b^p$

$x \; y \; z$

let $p = 7$ $\quad S = aaaaaaaa bbbbbbbb$

y can be given in 3 ways

Case I / The y is in 'a' part $\quad S = \underbrace{aa}_{x}\underbrace{aaaaaa}_{y}\underbrace{bbbbbbb}_{z}$

Case II / y is in 'b' part $\quad S = \underbrace{aaaaaaaa}_{x}\underbrace{bbbbbb}_{y}\underbrace{b}_{z}$

Case III / y is in both 'a' & 'b' $\quad S = \underbrace{aaaaa}_{x}\underbrace{aabbb}_{y}\underbrace{bbbbb}_{z}$

now check for

1) xy^iz lie in A or not let $i = 2$.

xy^2z for case I $= \overbrace{aa}^{x}\overbrace{aaaaa}^{y^2}\overbrace{aaaa\,a\,bbbbbbb}^{z}$

$a = 11 \quad b = 7 \quad |a| \neq |b|$ so not in A

2) xy^2z for case II $= aaaaaaaa\,bb\;bbbb\;bbbb\;b$

$7 \neq 11$ not in A

3) xy^2z for case III $= \underbrace{aaaaa}_{x}\;\underbrace{aabb\;aabb}_{y^2}\;\underbrace{bbbbbb}_{z}$

not of pattern $a^n b^n$ not in A.

So it is not regular.

also

case 3 $|x| = 5 \quad |y| = 4$ (aabb)

$|xy| = 9 \not\leq |p| = 7$

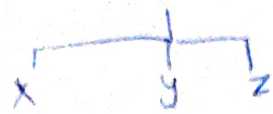
Q) Using Pumping Lemma prove that $A = \{yy \mid y \in \{0,1\}^*\}$ is not regular.

Proof

Assume A is regular

then it must have pumping length = p

take string from A → $S = 0^p 1 0^p 1$



let p = 7 $S = \underbrace{0000000}_{x}\underbrace{0}_{y}\underbrace{10000000}_{z}1$

now,

$xy^1z \Rightarrow xy^2z = \underbrace{00}_{x}\underbrace{00000000}_{y^2}\underbrace{01000000}_{z}1$

this not in is A as

11→0 , 1 , 7→0 , 1

$|y| > 0$ ✓

$|xy| \leq p$ ✓

So it can not be pumped

So our assumption was wrong.

Questions

Q for $\Sigma = \{0,1\}$ give regular expression r such that

$L(r) = \{\omega \in \Sigma^* : \omega$ has at least one pair of consecutive zeros$\}$

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