THEORY OF COMPUTATION

Symbol :- Basic (smallest) building block in the subject of TOC. It can be anything — alphabets / numbers / special symbols etc.

Example :- a, b, c, 0, 2, ... 

Alphabet :- Set / collection of symbols is called alphabet.

Denoted by "Σ"

(Alphabet is always finite)

Example :- $\Sigma = \{a, b\}$ $\quad |\Sigma| = 2$

String :- Sequence of symbols in a given alphabet.

Example :- If $\Sigma = \{a, b\}$

Strings over Σ = a, b, aa, ab, bc, bb, ...

> How many strings of length n are possible from alphabet of size m ?
>
> Ans :- m . m . m . .... n times
>
> $= m^n$

Language :- Collection (set) of strings is called language

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

L_1 = set of all strings of length 2

$= \{aa, ab, ba, bb\}$ (finite)

L_2 = set of all strings of length 3

$= \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$ (finite)

L_3 = set of all strings where each string starts with 'a'.

$= \{a, aa, ab, aaa, aab, aba, abb, ...\}$ Infinite

Powers of Σ

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

Σ^1 = set of all strings over Σ of length 1.

$= \{a, b\}$

Σ^2 = set of all strings of length 2

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

$\vdots$

Σ^0 = set of all strings of length 0.

$= \{\epsilon\}$ → epsilon

$|\epsilon| = 0$

size of $\epsilon = 0$.

substitute any no. of staring here 0

Σ^*

$= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots$

$= \{\epsilon\} \cup \{a, b\} \cup \{aa, ab, ba, bb\} \ldots$

(Infinite)

All languages are subset of Σ^*.

$L_1 \subseteq \Sigma^*$

$L_2 \subseteq \Sigma^*$

$L_3 \subseteq \Sigma^*$

C programming language = Set of all valid C programs

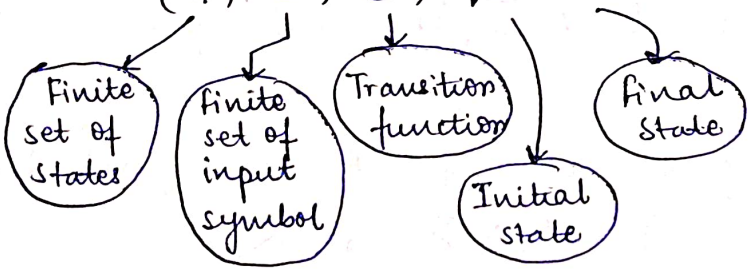
Finite automata

Finite automata are the simplest machines to recognize patterns

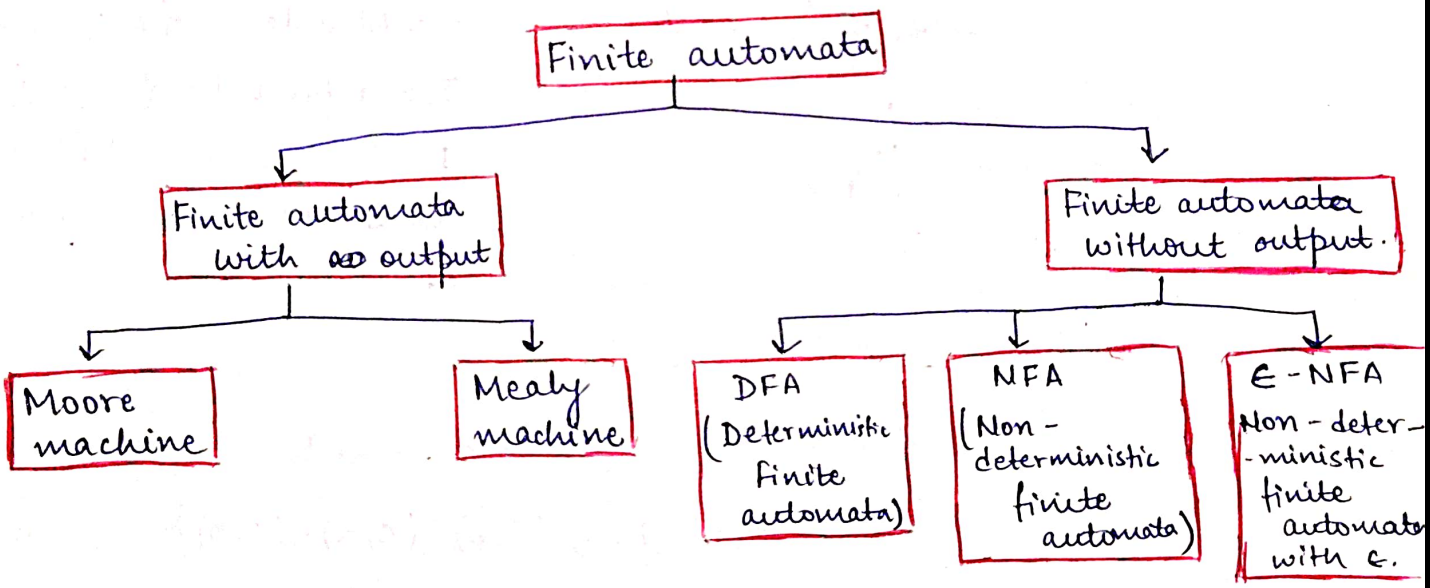
It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, ~~it changes~~ transition occurs

A finite automation is a collection of 5 tuples

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



Classification



Deterministic Finite Automata (DFA)

→ The finite automata are called DFA if the machine reads an input string one symbol at a time.

→ In DFA, there is only one path for specific input from the current state to the next state.

→ DFA does not accept null value i.e. it cannot change state without any input character.

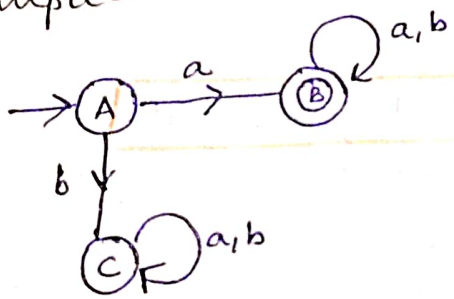
→ Can contain multiple final states

→ Used in Lexical Analysis in Compiler.

Transition function of DFA is defined as

$$\delta = Q \times \Sigma \longrightarrow Q$$

Example -



Q = All states = {A, B, C}

Σ = alphabet = {a, b}

F = Final states = {B}

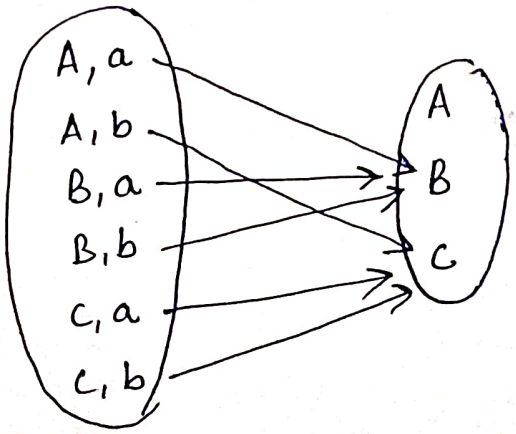
q_0 = Initial state = {A}

$$\delta = Q \times \Sigma \longrightarrow Q$$

$$= \{A, B, C\} \times \{a, b\} \longrightarrow Q$$

$$= \{(A,a)\ (A,b)\ (B,a)\ (B,b)\ (C,a)\ (C,b)\} \longrightarrow \{A, B, C\}$$

=

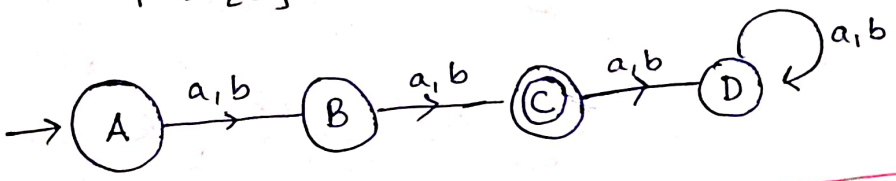


Ques:- Construct a DFA that accepts set of all strings over {a, b} of length 2.

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

L = set of all strings of length 2
= {aa, ab, ba, bb}

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



A string is said to be accepted under Finite Automata if we can reach final state from initial state after scanning each character of the input string.

* String acceptance

Upon scanning a string completely, if we reach final state from initial state, then, the string is said of be accepted.

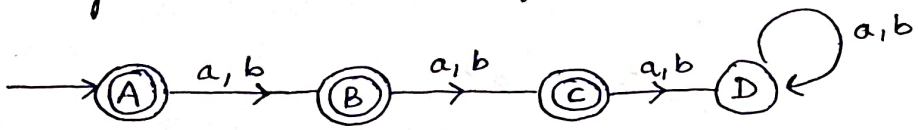
* Language acceptance

A FA is said to accept a language if all

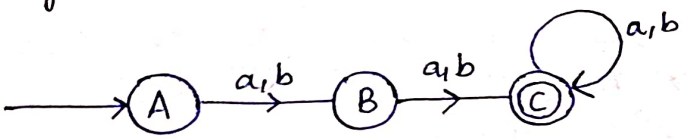
① all the strings in the language are accepted by the FA.

② all the strings that are not in the language are rejected by the FA.

→ DFA that accepts a set of all strings over {a, b} of length less than or equal to 2.



→ DFA that accepts a set of all strings over {a, b} of length ~~less~~ greater than or equal to 2.



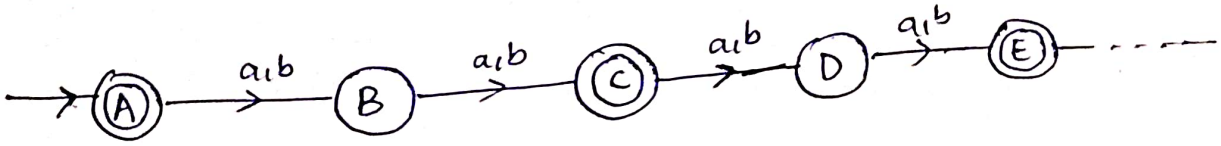
In general,

$\|w\| = n$	No. of states = $n+2$
$\|w\| \leq n$	No. of states = $n+2$
$\|w\| \geq n$	No. of states = $n+1$

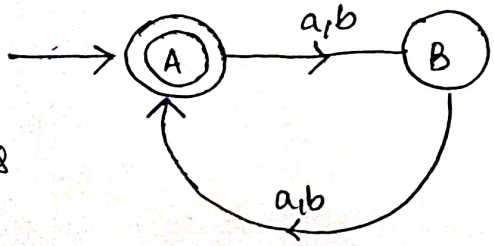
→ DFA that accepts a set of all strings over {a, b} whose length is even.

$L = \{\epsilon, aa, ab, ba, bb, aaaa, \ldots bbbb \ldots\}$

$|w| \bmod 2 = 0$



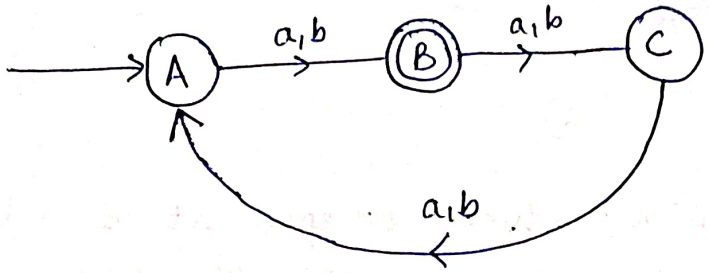
Better solution
Minimal states



→ DFA that accepts a string over {a, b} ~~whose~~ which follow following property—

$|w| \bmod 3 = 1$

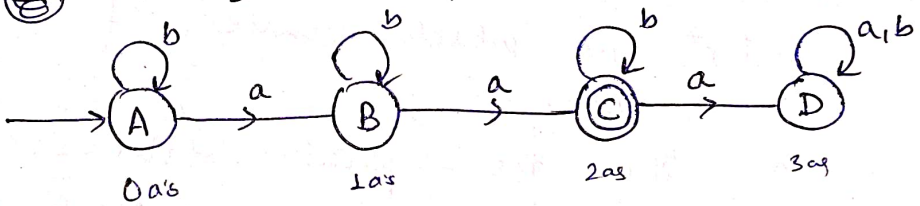
$L = \{\epsilon, aaa, aaaaa\ldots bbb, \ldots\}$



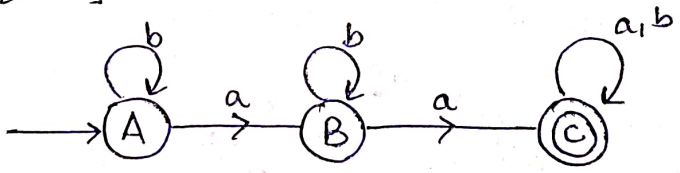
→ In general, $|w| \bmod n$ contains n states.

→ Minimal DFA which accepts set of all strings over $w \in \{a,b\}^*$ where numbers of a in $w = 2$ $(n_a = 2)$

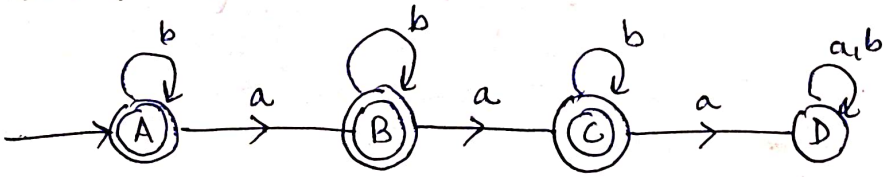
$L = \{aa, baa, aba, aab, bbaa, \ldots\}$



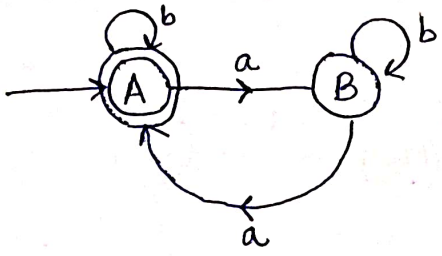
→ Minimal DFA which accepts set of all strings over $w = \{a,b\}^*$ where $n_a \geq 2$



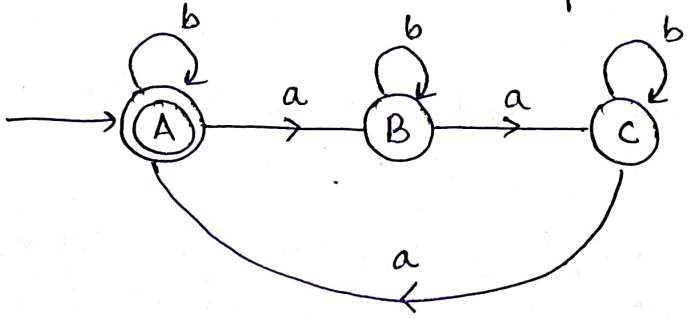
→ Minimal DFA which accepts set of all strings over $w = \{a,b\}^*$ where $n_a \leq 2$



→ Minimal DFA which accepts set of all strings over {a, b}* in which number of a's is even.



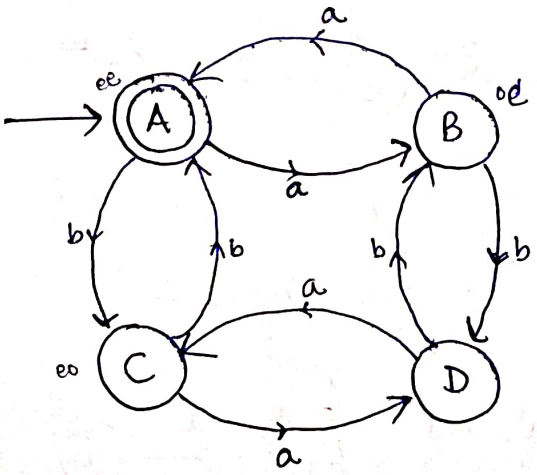
→ Minimal DFA which accepts set of all strings over {a, b}* in which number of a's is multiple of 3.



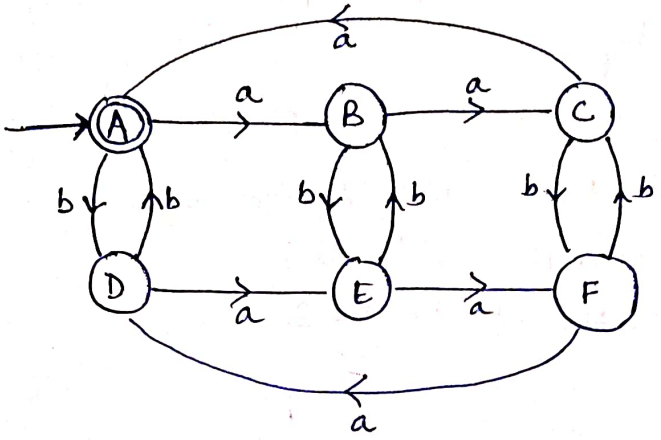
* → Minimal DFA which accepts set of all strings over {a, b}* in which number of a's and no. of b's is even.

There are 4 possible states —

No. of a's is	No. of b's is
even	even
odd	even
even	odd
odd	odd



→ Minimal DFA which accepts set of all strings over {a, b}
in which $n_a(w) \cong 0 \mod 3$ AND
$n_b(w) \cong 0 \mod 2$



A = 0 a's 0 b's
B = 1 a's 0 b's
C = 2 a's 0 b's
D = 0 a's 1 b's
E = 1 a's 1 b's
F = 2 a's 1 b's

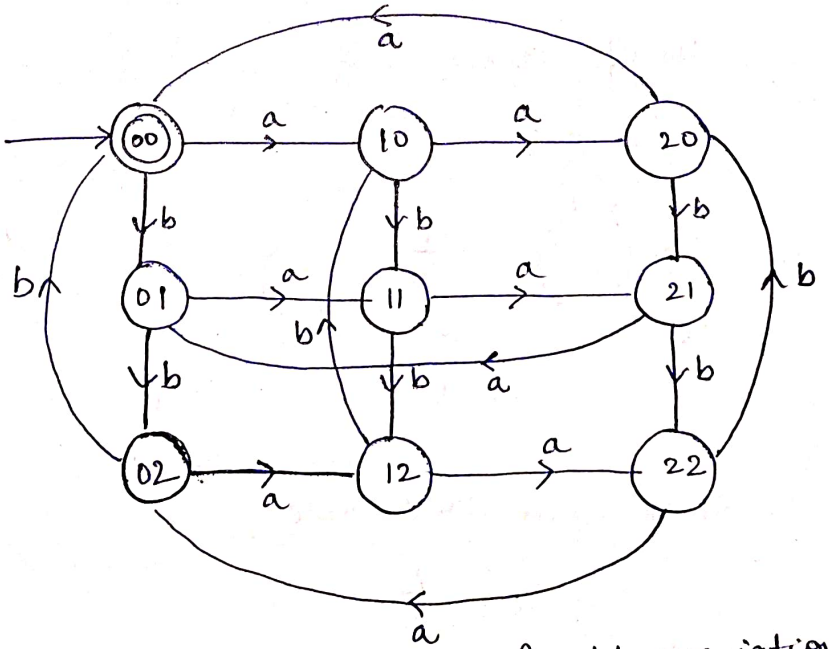
Possible variations —

$n_a(w) \mod 3 \leq n_b(w) \mod 2$

$n_a(w) \mod 3 \geq n_b(w) \mod 2$.

$n_a(w) \cong 0 \mod 3$ or $n_b(w) \cong 0 \mod 2$

→ Minimal DFA which accepts set of all strings over {a, b}
in which $n_a(w) \cong 0 \mod 3$
$n_b(w) \cong 0 \mod 3$.



Possible variations —

$n_a(w) \mod 3 = 1$ && $n_b(w) \mod 3 = 2$

same ★ $n_a(w) \mod 3 > n_b(w) \mod 3$

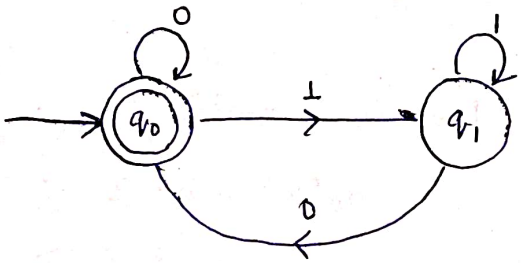
same ★ $(n_a(w) > n_b(w)) \mod 3$

→ Minimal DFA that accepts set of all strings over {0,1}, which when interpreted as binary number is divisible by 2.

$\Sigma = \{0, 1\}$ $\quad w \in \{0,1\}^*$

Any number when divided by 2 gives 2 remainders — 0 and 1

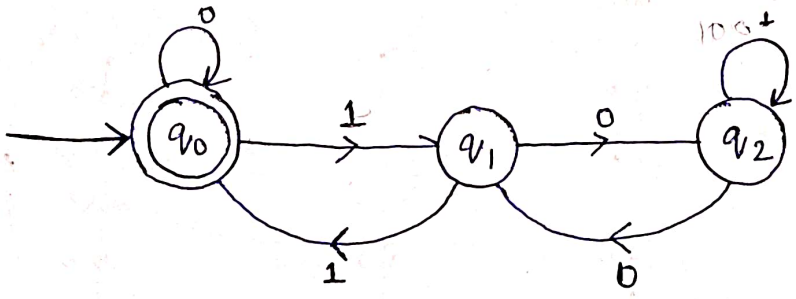
∴ No. of states = 2



→ Minimal DFA which accepts set of all strings over {0,1} which when interpreted as binary number is divisible by 3.

$\Sigma = \{0, 1\}$ $\quad w \in \{0,1\}^*$

No. of states = 3



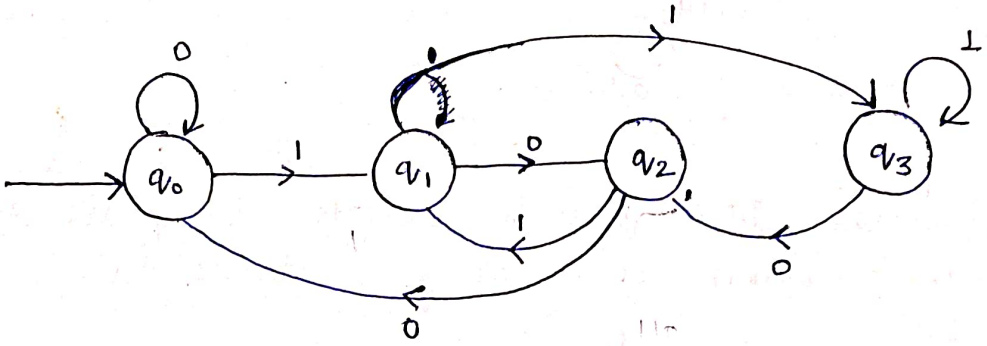
State Transition table

		0	1
→	* q_0	q_0	q_1
	q_1	q_2	q_0
	q_2	q_1	q_2

→ Minimal DFA which accepts set of all strings over $\{0,1\}$ which when interpreted as a binary number is divisible by 4.

$\Sigma = \{0,1\}$ $\quad w \in \{0,1\}^*$

No. of states = 4.

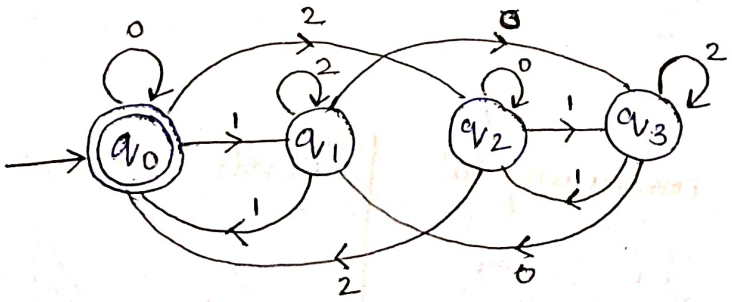


Transition table

	0	1
$\rightarrow^* q_0$	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

→ Minimal DFA which accepts set of all strings over $\{0,1,2\}$ which when interpreted as ternary string are divisible by 4.

$\Sigma = \{0,1,2\}$ $\quad w \in \{0,1,2\}^*$



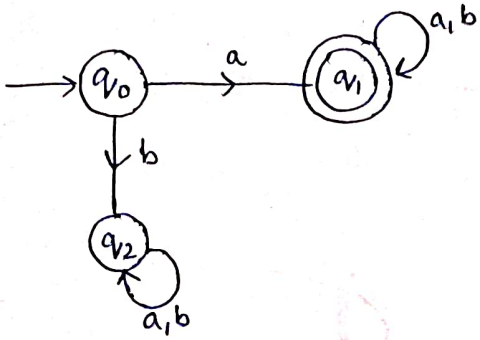
Transition table

	0	1	2
$\rightarrow^* q_0$	q_0	q_1	q_2
q_1	q_3	q_0	q_1
q_2	q_2	q_3	q_0
q_3	q_1	q_2	q_3

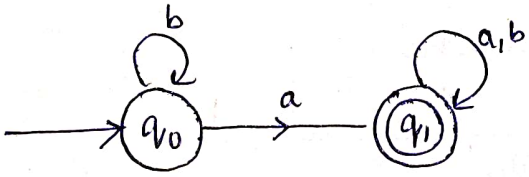
→ Minimal DFA which accepts set of all strings over {a, b} where each string starts with an 'a'.

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

$L = \{a, aa, ab, aaa, \ldots\}$



→ Minimal DFA which accepts set of all strings over {a, b} where each string contains 'a'.

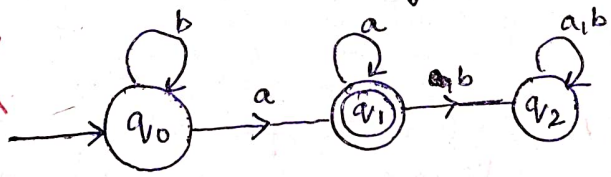


$\Sigma = \{a, b\}$

$L = \{a, ba, aa, \ldots\}$

→ Minimal DFA which accepts set of all strings over {a, b} where each string ends with an 'a'.

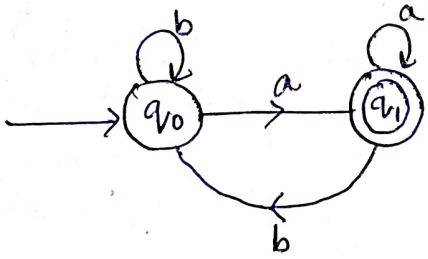
used extra state.



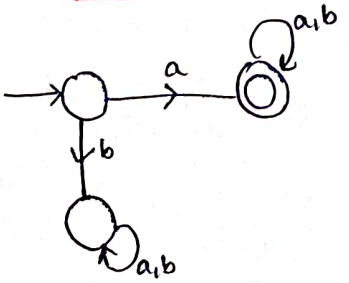
$\Sigma = \{a, b\}$

$L = \{a, aa, ba \ldots\}$

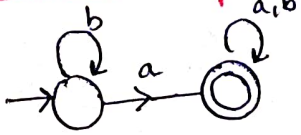
Best solution.



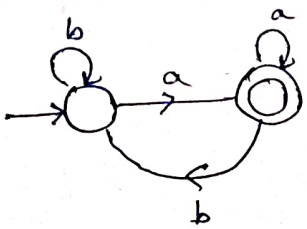
Starts with 'a'



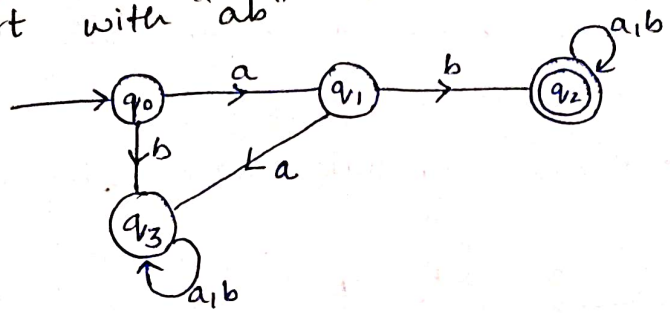
containing 'a'



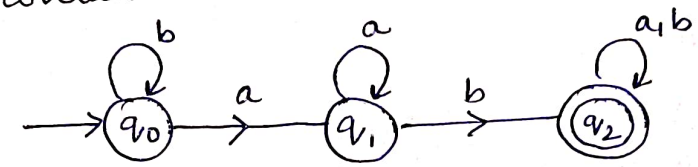
Ends with 'a'



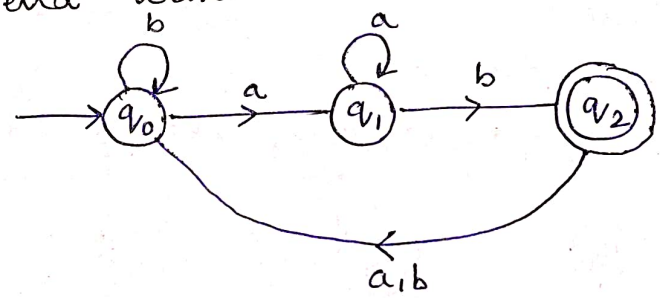
→ Minimal DFA which accepts set of all strings over {a,b} which start with "ab"



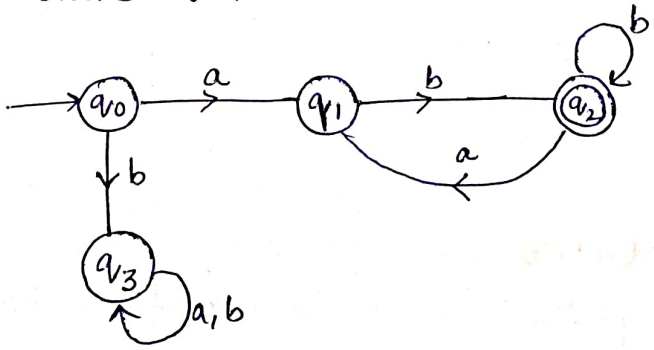
→ Minimal DFA which accepts set of all strings over {a,b} which contain "ab"



→ Minimal DFA which accepts set of all strings over {a,b} which end with "ab"



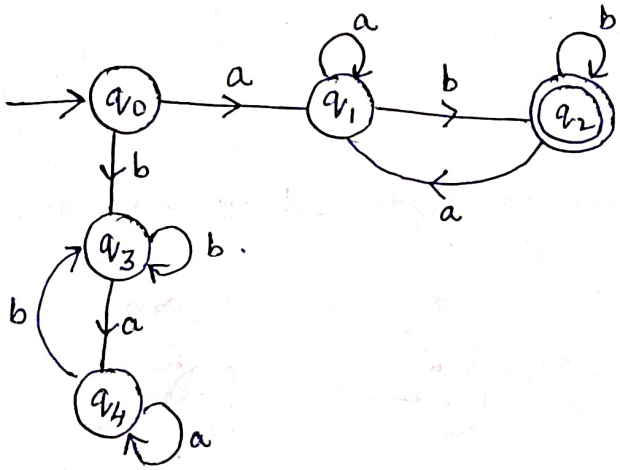
→ Minimal DFA which accepts set of all strings over {a,b} which starts with 'a' and ends with 'b'



→ Minimal DFA which accepts set of all strings over {a, b} which starts and ends with different symbol.

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

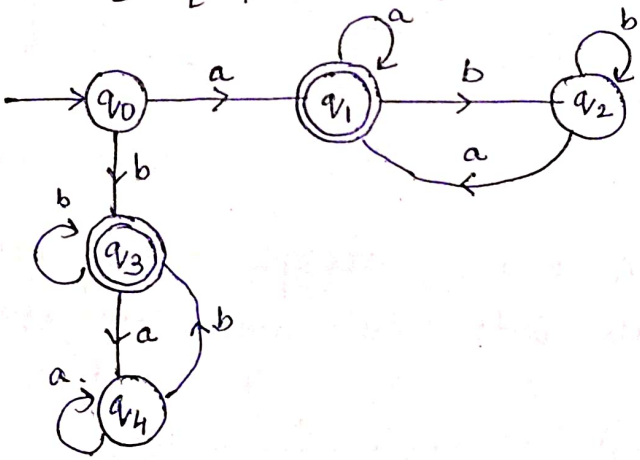
$L = \{ab, ba, aab, baa, bba, abb, \ldots\}$



→ Minimal DFA which accepts set of all strings over {a, b} which starts and ends with the same symbol.

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

$L = \{a, aa, aba, abba, b, bb, \ldots\}$



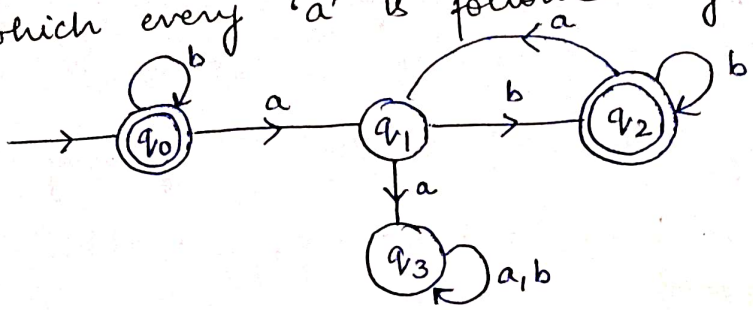
Complementation

→ Converting final states to non-final states and vice versa.

→ applicable for only DFAs not NFAs.

→ If DFA is $(Q, \Sigma, q_0, \Sigma, F)$, its complement will be $(Q, \Sigma, q_0, \Sigma, Q-F)$

→ Minimal DFA which accepts set of all strings over {a, b} in which every 'a' is followed by a 'b'.

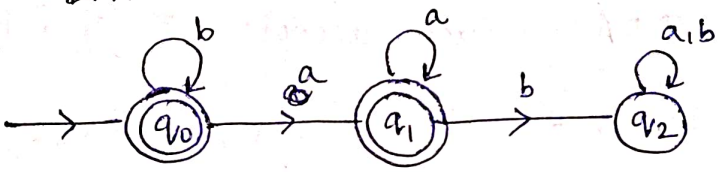


L = {ϵ, ab, abab, ... b }

→ Minimal DFA which accepts set of all strings over {a, b} in which every 'a' should never be followed by a 'b'

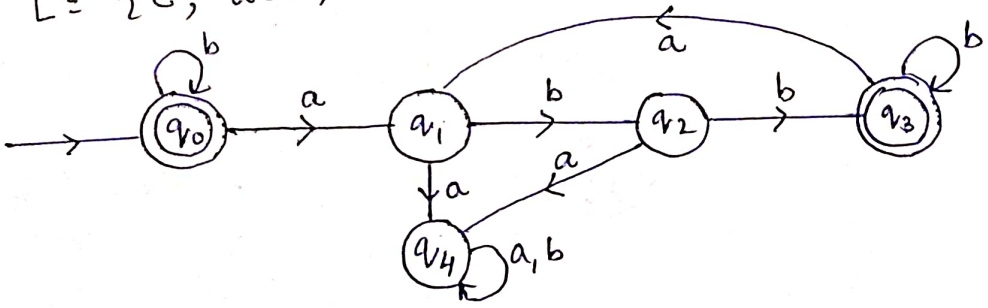
L = { ϵ, a, aa, aaa, ... b, bb, ... , ba, bba.... }

Since there are elements in the intersection of language, therefore, complementation of previous DFA is not correct.



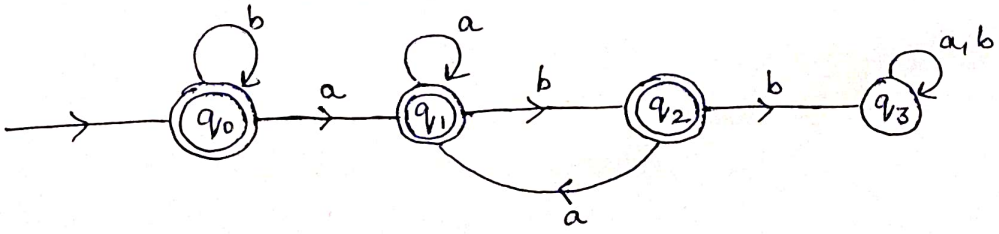
→ Minimal DFA which accepts set of all strings over {a, b} in which every 'a' should be followed by a 'bb'

L = {ϵ, abb, abbabb, --- babb, bb, ... }



→ Minimal DFA which accepts set of all strings over {a, b} in which every 'a' should never be followed by a 'bb'

L = {ϵ, a, ab, aba, ~~abb~~, ... b, bb, bbb.... }



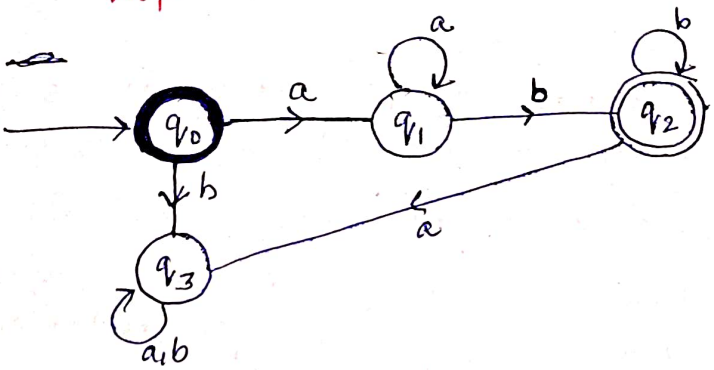
→ Minimal DFA which accepts

$$I = \{ a^n b^m : n, m >= 1 \}$$

L = any number of a's followed by any number of b's

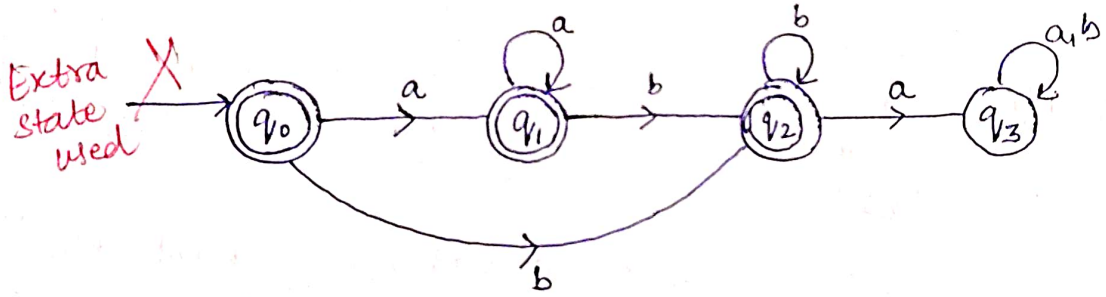
L = { ~~ε~~, ab, aab, aaab, ... aaabb, ... abbb, ... }

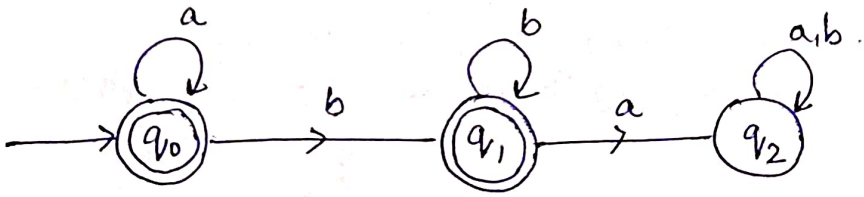
not present



→ Minimal DFA which accepts $I = \{ a^n b^m \mid n, m >= 0 \}$

L = { ε, a, b, aab, aa, abb, bb, ... }

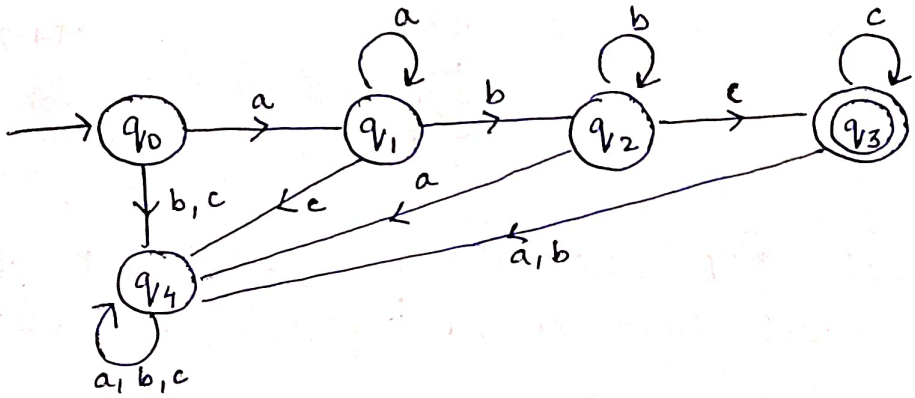




< Complement of language which contains ba >

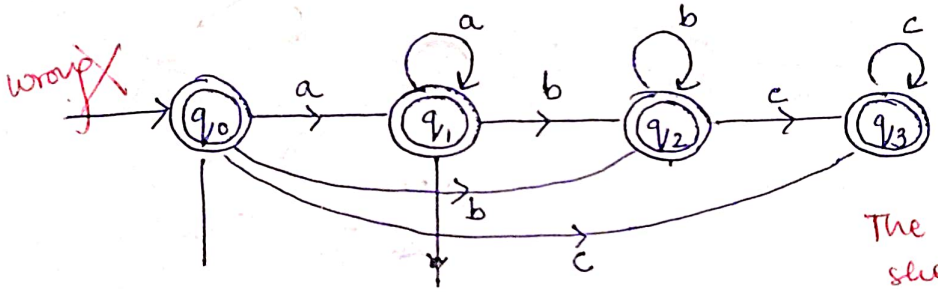
→ Minimal DFA which accepts set of all strings over {a, b, c} which satisfy the following property.

$$I = \{ a^n b^m c^l \mid n, m, l >= 1 \}$$

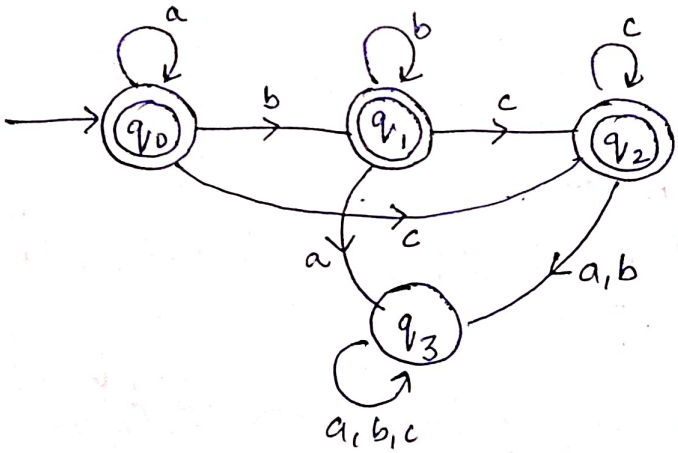


→ Minimal DFA which accepts

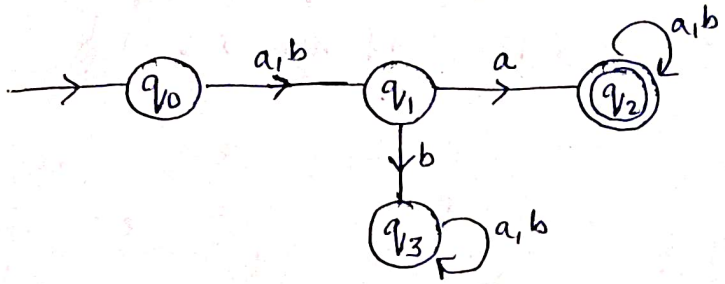
$$I = \{ a^n b^m c^l \mid n, m, l >= 0 \}$$



The language should not contain ba, ca, cb

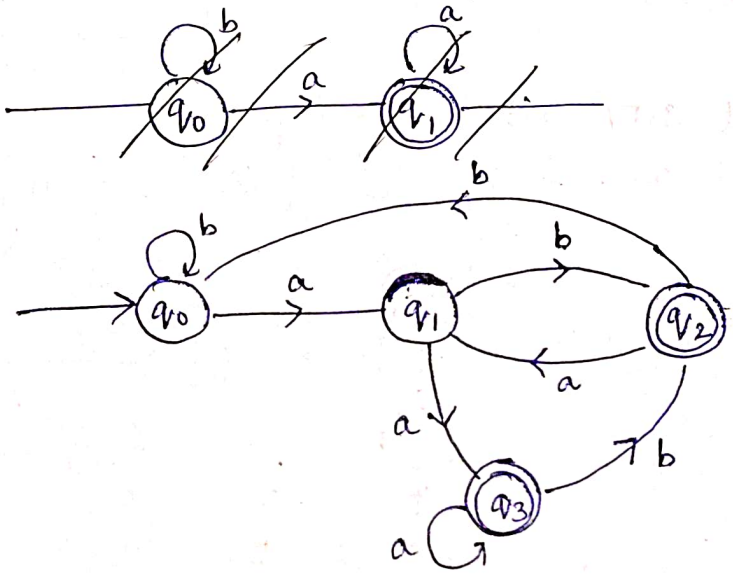


→ Minimal DFA which accepts set of all strings over {a, b} such that second symbol from LHS is 'a'

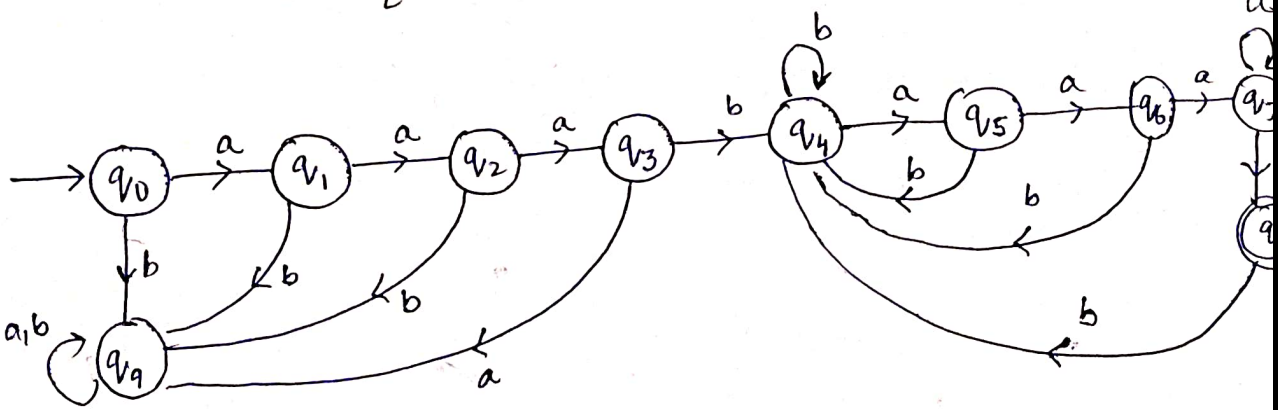


n+2 states for nth symbol

→ Minimal DFA which accepts set of all strings over {a, b} such that second symbol from RHS is 'a'



→ Minimal DFA which accepts set of all strings over {a, b} such that $L = \{ a^3 b w a^3 \mid w \in \{a, b\}^* \}$



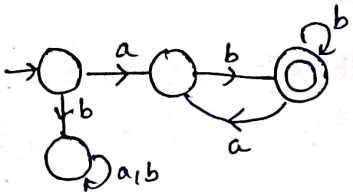
Operations on DFA

1. Union :- starts and ends with different symbols.

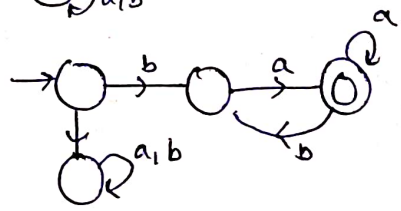
$\Sigma = \{a, b\}$ $L_1 = \{ab, aab, abb, \ldots\}$

$L_2 = \{ba, bab, bba, \ldots\}$

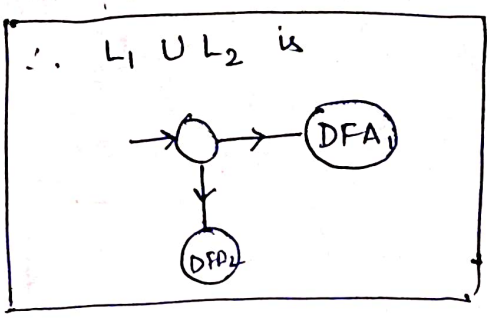
DFA$_1$:-



DFA$_2$:-



$\therefore L_1 \cup L_2$ is



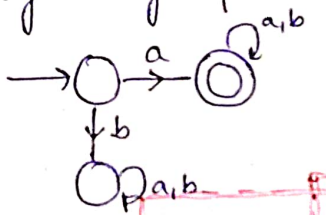
2. Concatenation :- starts with 'a' ends with 'b'

$L_1 = \{$set of all strings starting with 'a'$\} = \{a, aa, ab, aaa, \ldots\}$

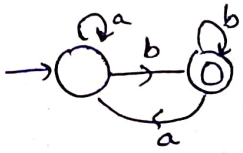
$L_2 = \{$set of all strings ending with 'b'$\} = \{b, bb, ab, bbb, \ldots\}$

Take any string of L_1 and concatenate with L_2.

DFA for L_1 —

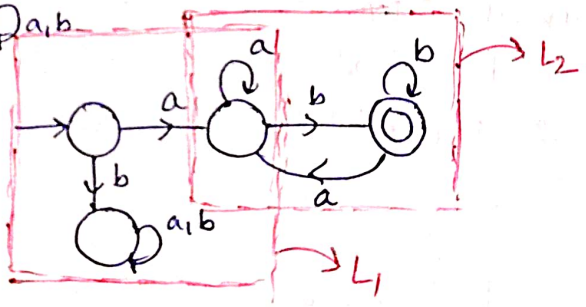


DFA for L_2 —



★ Final state in the first DFA and Initial state in the second DFA should be merged to get concatenated DFA

$L_1 . L_2$:-



3. Cross product :- Even no. of a's and even no. of b's.

I KNOW IT !!

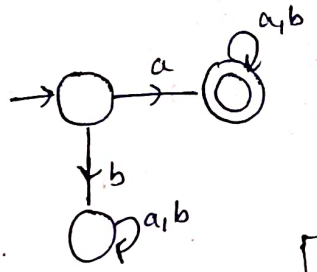
Complement - Does not contain 'a'

I KNOW THIS TOO !!

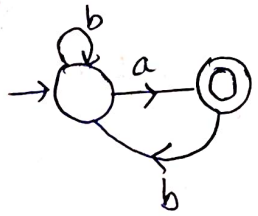
Reversal - $L_1 = \{$ starts with a$\} = \{a, aa, ab, aaa, \dots\}$

$L_2 = \{$ ends with a$\} = \{a, aa, ba, aaa, \dots\}$

L_1 :-



L_2 :-



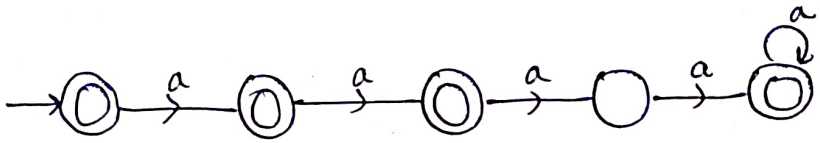
Resultant of Reversal of a DFA can give DFA or NFA.

① Reverse all the transitions
② Make initial state final and make final state initial.
③ Remove state which will never be reached.

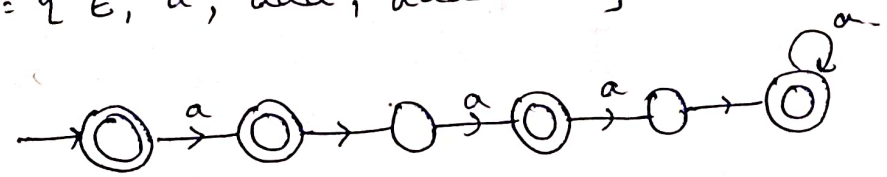
→ Minimal DFA over $\{a\}$

1. For $\{a^n \mid n>=0, n!=3\}$
2. For $\{a^n \mid n>=0, n!=2, n!=4\}$

1. $L = \{\epsilon, a, aa, aaaa, aaaaa \dots\}$



2. $L = \{\epsilon, a, aaa, aaaaa \dots\}$



NFA (Non-deterministic Finite Automata)

→ NFA stands for Non-deterministic finite automata.

→ A finite automata is called NFA when there exists many paths from specific input from the current state to the next state.

→ Every NFA is not DFA but every DFA is an NFA. Each NFA can be translated into DFA.

Transition function : $\delta: Q \times \Sigma \to 2^Q$

→ NFA V/S DFA

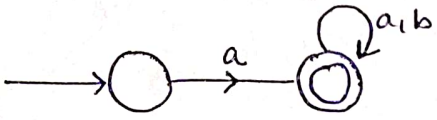
↳ NFA can contain multiple next states.
DFA contains only 1 next state.

↳ NFA contains ϵ transition
DFA does not contain ϵ transition

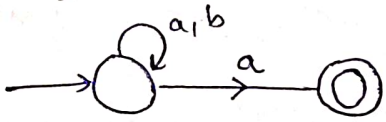
↳ NFA contains dead configuration
DFA contains dead state.

Examples—

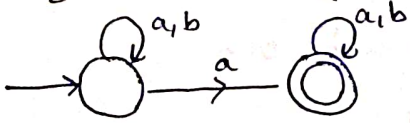
(1) L_1 = { starts with 'a' }



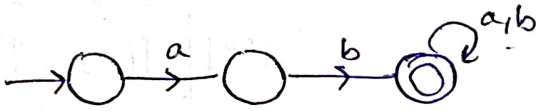
L_2 = { ends with 'a' }



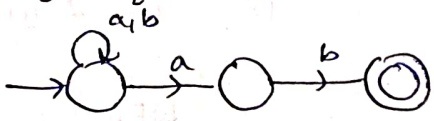
L_3 = { contains 'a' }



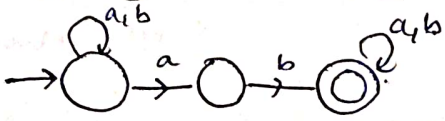
L_4 = { starts with 'ab' }



L_5 = { ends with 'ab' }



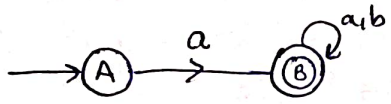
L_6 = { contains 'ab' }



Conversion of NFA to DFA

① L = { all strings starting with 'a' }

NFA :-



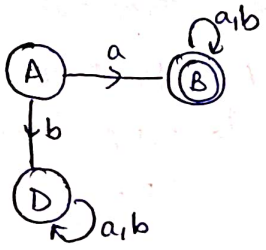
State Transition table

	a	b
A	B	ϕ
B	B	B

State transition table for DFA —

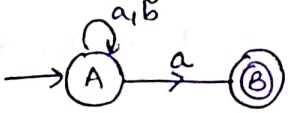
	a	b
→ A	B	D
B	B	B
D	D	D

DFA :-



② L = { all strings ending with 'a' }

NFA :-



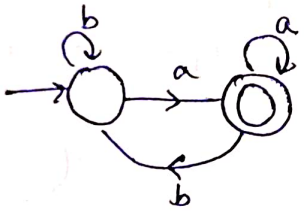
State transition table

	a	b
→ A	{A,B}	{A}
* B	ϕ	ϕ

State Transition Table for DFA

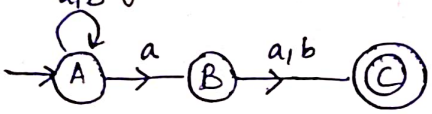
	a	b
[A]	[AB]	[A]
[AB]	[AB]	[A]

DFA :-



③ L = { all strings where second symbol from R.H.S. is a }
L = {aa, ab, aaa ...}

NFA :-

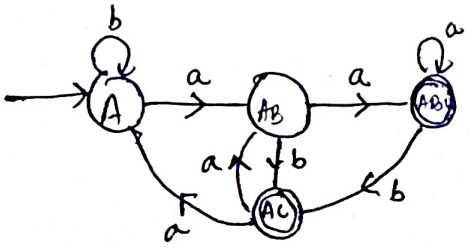


Transition table

	a	b
→ A	{A,B}	{A}
B	{C}	{C}
* C	{ }	{ }

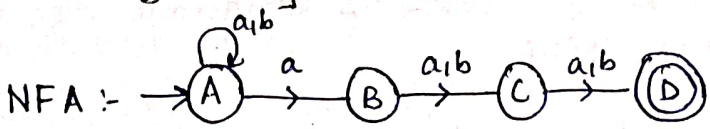
State Transition Table for DFA

	a	b
→ A	[AB]	[A]
[AB]	[ABC]	[AC]
* [ABC]	[ABC]	[AC]
* [AC]	[AB]	[A]



(3) L = {all strings in which third symbol from RHS is 'a'}

NFA :-

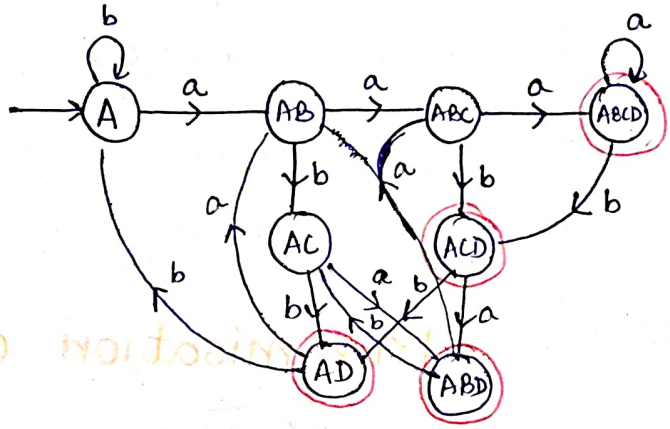


Transition table

	a	b
→ A	{A,B}	{A}
B	{C}	{C}
C	{D}	{D}
* D	{}	{}

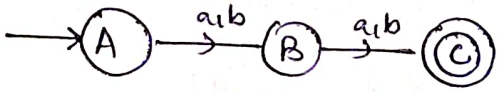
Transition table for DFA —

	a	b
[A]	[AB]	[A]
[AB]	[ABC]	[AC]
[AC]	[ABD]	[AD]
* [AD]	[AB]	[A]
[ABC]	[ABCD]	[ACD]
* [ABD]	[ABC]	[AC]
* [ACD]	[ABD]	[AD]
* [ABCD]	[ABCD]	[ACD]

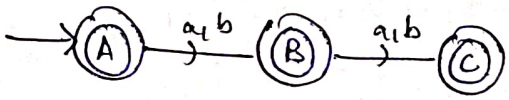


(4) NFA for → strings of length 2
→ strings of length atmost 2
→ strings of length atleast 2

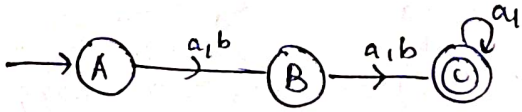
i)



ii)

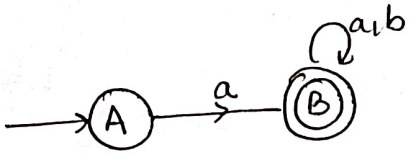


iii)



No. of states in NFA ≤ No. of states in DFA.

∴ if minimum no. of states in finite automata are asked, always go for NFA.

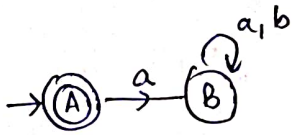


For DFA, answer to both of these questions is same

What is the complement of language accepted by this NFA.

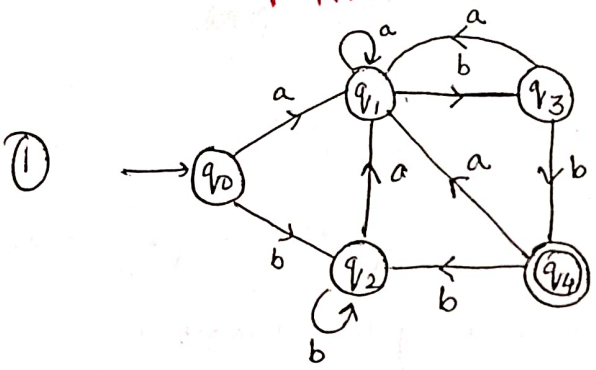
$L = \{\epsilon, b, bb, ba, bbb \ldots\}$

What is the language accepted by complement of this NFA?

Complement of NFA = 

$L = \{\epsilon\}$.

Minimisation of DFA



State Transition Table

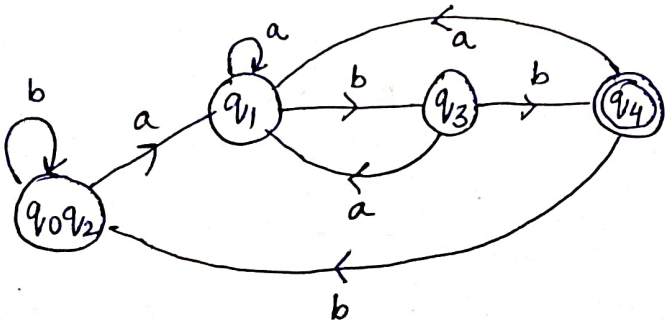
	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	$*q_4$
$*q_4$	q_1	q_2

0 equivalent → $[q_0\ q_1\ q_2\ q_3][q_4]$

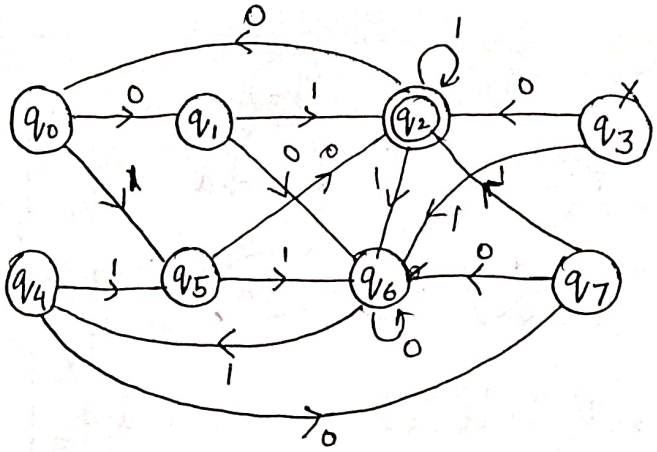
1 equivalent → $[q_0\ q_1\ q_2][q_3][q_4]$

2 equivalent → $[q_0\ q_2][q_1][q_3][q_4]$

3 equivalent → $[q_0\ q_2][q_1][q_3][q_4]$



(2)



State Transition table

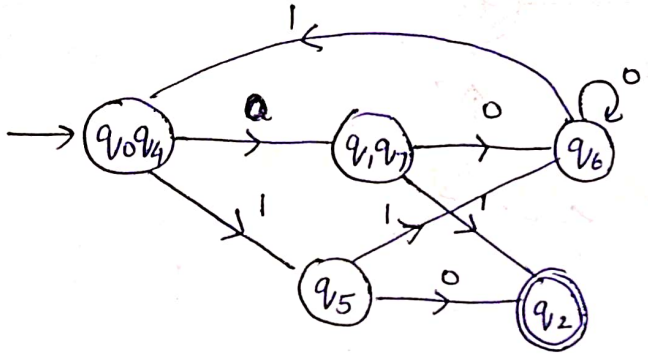
	0	1
→ q_0	q_1	q_5
q_1	q_6	$*q_2$
$*q_2$	q_0	$*q_2$
q_4	q_7	q_5
q_5	$*q_2$	q_6
q_6	q_6	q_4
q_7	q_6	$*q_2$

0 equivalent → $[q_0 q_1 q_4 q_5 q_6 q_7][q_2]$

1 equivalent → $[q_0 q_4 q_6][q_1 q_7][q_5][q_2]$

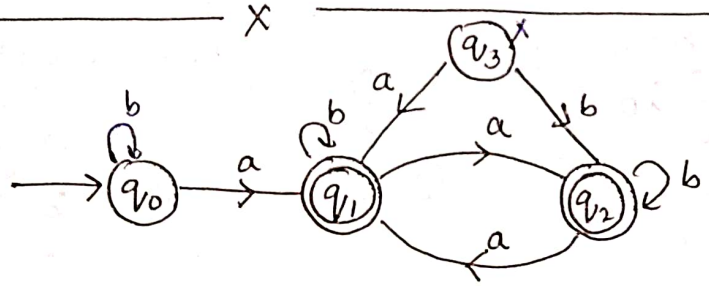
2 equivalent → $[q_0 q_4][q_6][q_1 q_7][q_5][q_2]$

3 equivalent → $[q_0 q_4][q_6][q_1 q_7][q_5][q_2]$



---

(3)



State transition Table

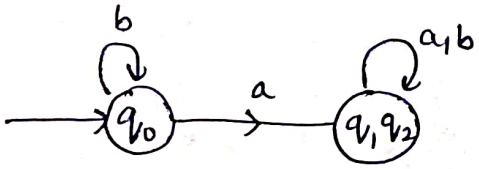
	a	b
→ q_0	$*q_1$	q_0
$*q_1$	$*q_2$	$*q_1$
$*q_2$	$*q_1$	$*q_2$

~~0 equivalent :- $[q_0 q_1][q_2]$~~

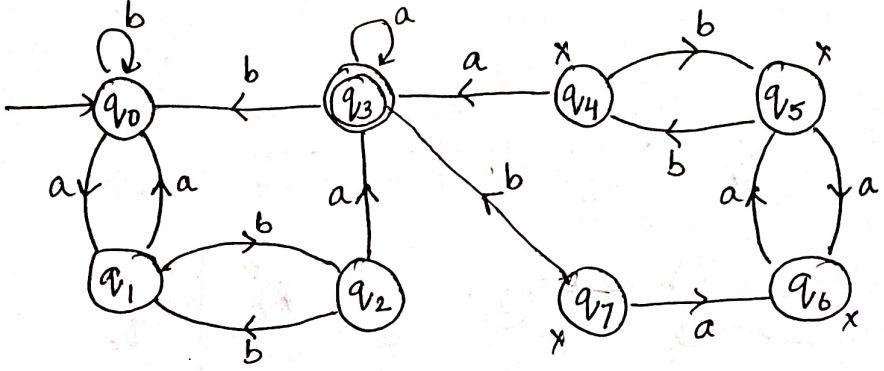
~~1 equivalent :- $[q_0][$~~

0 equivalent :- $[q_0][q_1 q_2]$

1 equivalent :- $[q_0][q_1 q_2]$



(4)



State Transition Table

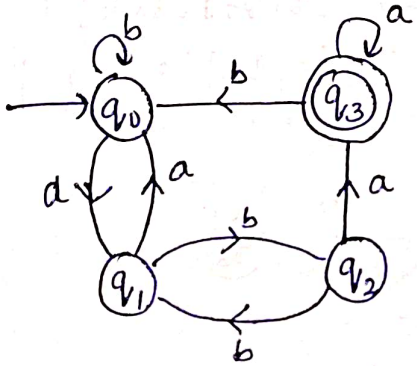
	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_0	q_2
q_2	$*q_3$	q_1
$*q_3$	q_3	q_0

0 equivalent :- $[q_0 q_1 q_2][q_3]$

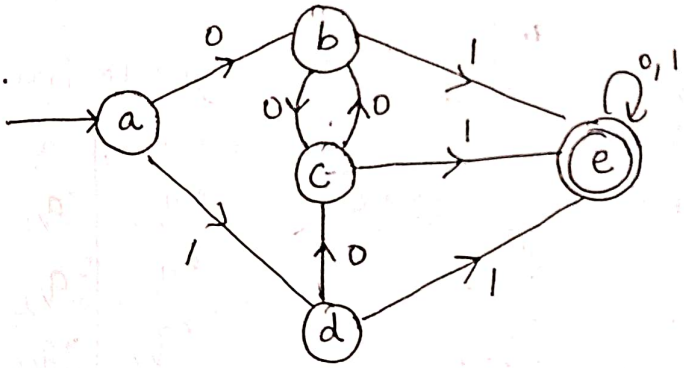
1 equivalent :- $[q_0 q_1][q_2][q_3]$

2 equivalent :- $[q_0][q_1][q_2][q_3]$

3 equivalent :- $[q_0][q_1][q_2][q_3]$



(5)



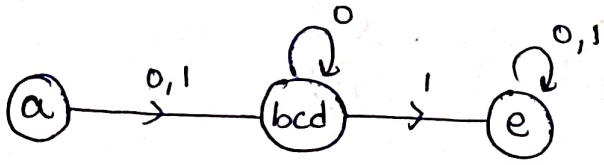
State Transition Table

	0	1
$\rightarrow$ a	b	d
b	c	e
c	b	e
d	c	e
* e	e	e

0 equivalent :- [abcd][e]

1 equivalent :- [a][bcd][e]

2 equivalent :- [a][bcd][e]



Introduction to Moore and Mealy machines

Finite automata with o/p — Deterministic

- Moore machine
- Mealy machine

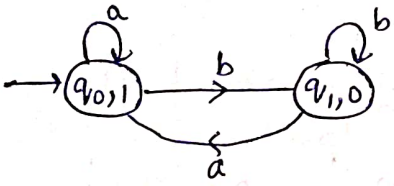
$(Q, \Sigma, \delta, q_0, \Delta, \lambda)$

Q = finite set of states
Σ = i/p alphabets
δ = Transition function $Q \times \Sigma \to Q$
q_0 = Initial state
Δ = o/p alphabet
λ = o/p function

Moore machine

$\lambda : Q \to \Delta$

For every state, output is given

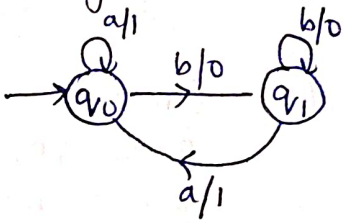


For n bit input, n+1 bit output is produced.

Mealy machine

$\lambda : Q \times \Sigma \to \Delta$

For every state and input, output is given



For n bit input, n bit output is produced.

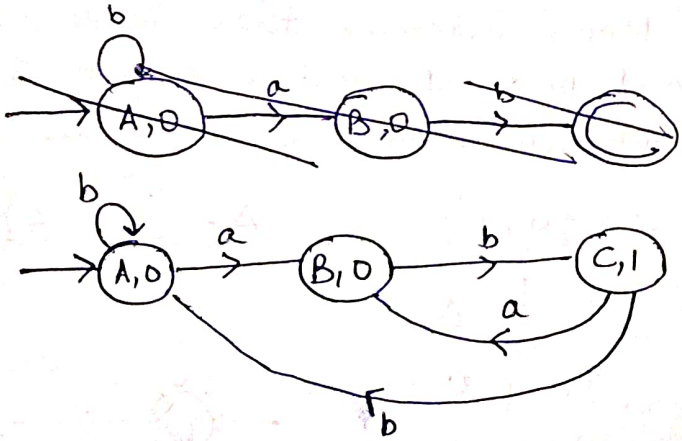
Examples

(1) Construct a Moore machine that takes set of all strings over $\{a, b\}$ as i/p and prints '1' as a o/p for every occurance of 'ab' as a substring.

Counting no. of ab.

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

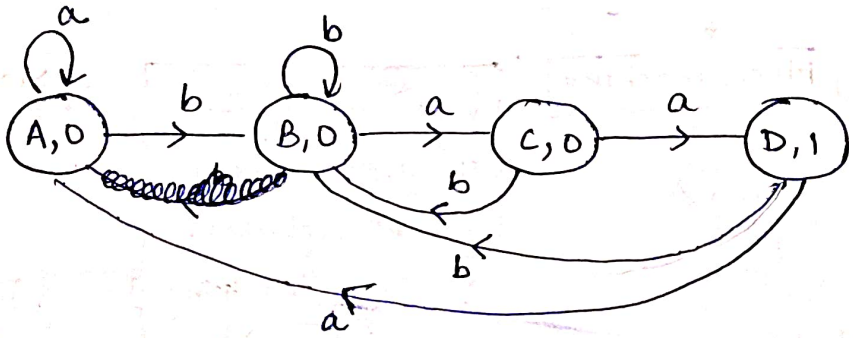
$\Delta = \{0, 1\}$



<Construct DFA for strings ending with ~~abaa~~ baa>

baaa

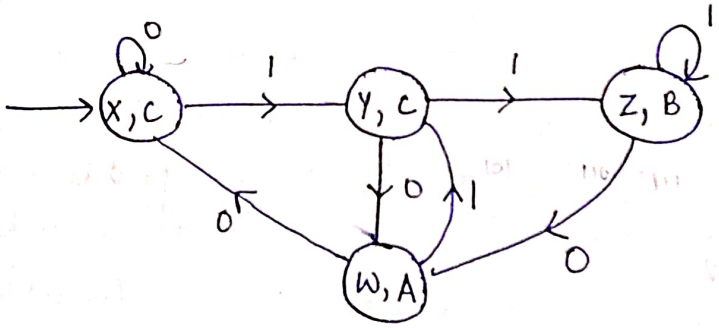
② Counting occurance of substring baa



③ Construct a Moore machine that takes set of all strings over {0,1} and produces 'A' as output if input ends with '10' and produces 'B' as o/p if i/p ends with '11' otherwise produces c.

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

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

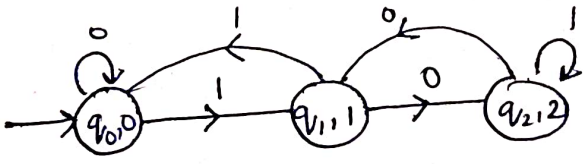


④ Construct a Moore machine that takes input binary numbers and produces residue modulo 3 as o/p.

[i.e. prints remainder when no. is divided by 3]

$\Sigma = \{0, 1\}$ $\Delta = \{0, 1, 2\}$

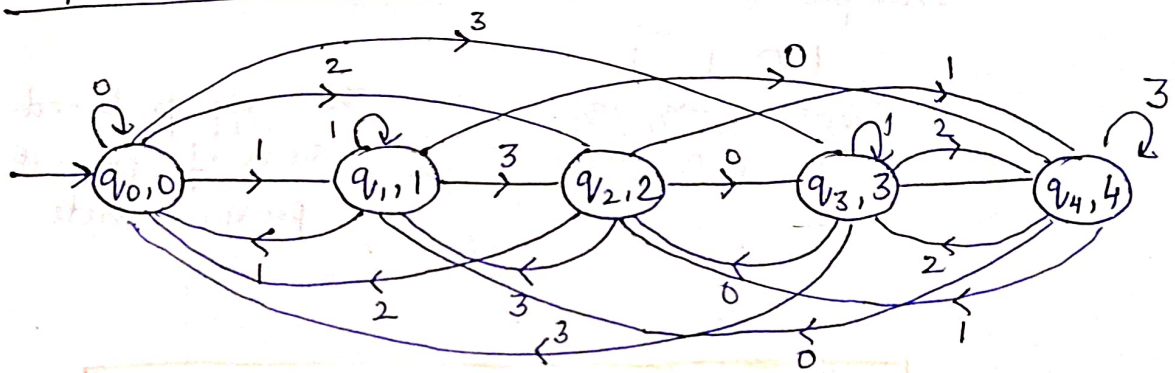
	0	1	Δ
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2



(5) Construct a Moore machine that takes numbers in base4 as i/p and produces residue modulo 5 as o/p.

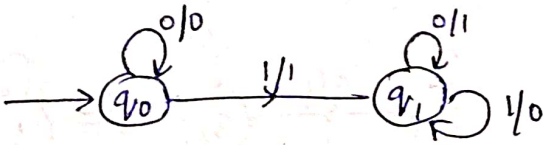
$\Sigma = \{0, 1, 2, 3\}$ $\Delta = \{0, 1, 2, 3, 4\}$

	0	1	2	3	Δ
q_0	q_0	q_1	q_2	q_3	0
q_1	q_4	q_0	q_1	q_2	1
q_2	q_3	q_4	q_0	q_1	2
q_3	q_2	q_3	q_4	q_0	3
q_4	q_1	q_2	q_3	q_4	4

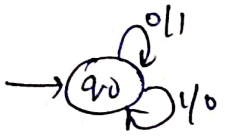


(6) Construct a Mealy machine that takes binary number as i/p and produces 2's complement of that number as o/p. Assume the string is read from LSB to MSB and end carry is discarded.

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

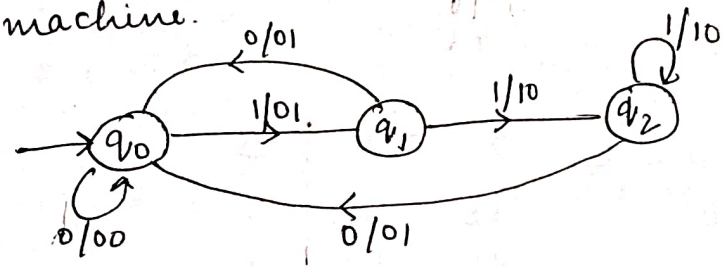


2's complement



1's complement.

(7) What is the o/p produced by the following state machine.



a) 11 → 01

b) 10 → 00

c) Sum of present and previous bits.

d) NOT

Example:-

1 0 1 1

$q_0 \to q_1 \to q_0 \to q_1 \to q_2$

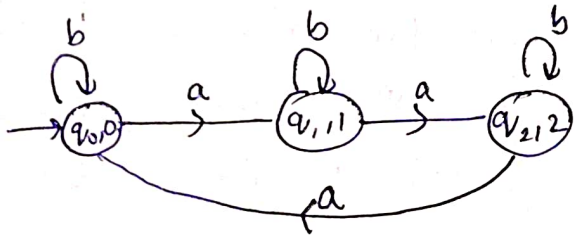
01 01 01 10

∴ The o/p produced is the sum of present and previous bits.

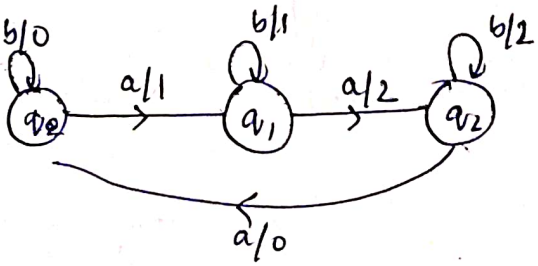
Mealy machine to Moore machine conversion and vice-versa.

Moore → Mealy

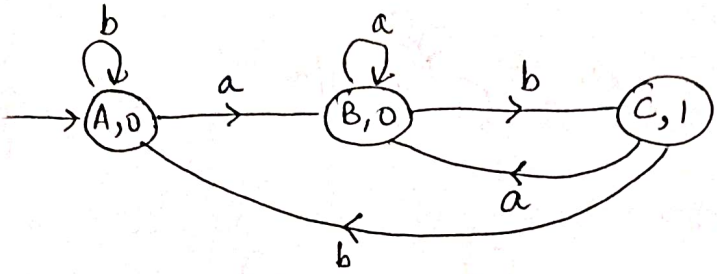
Moore machine:-



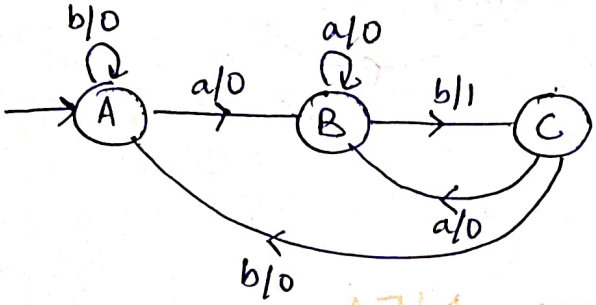
Mealy machine:-



(2)

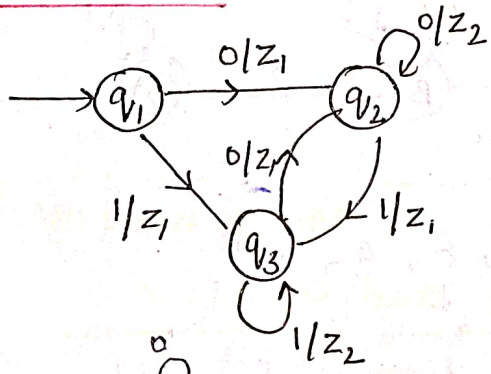


Moore machine

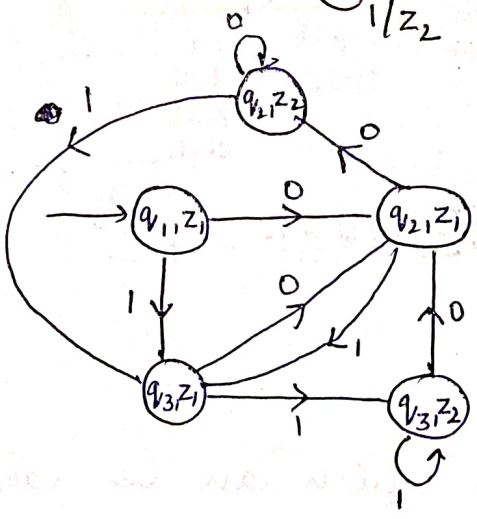


Mealy machine

Mealy to Moore

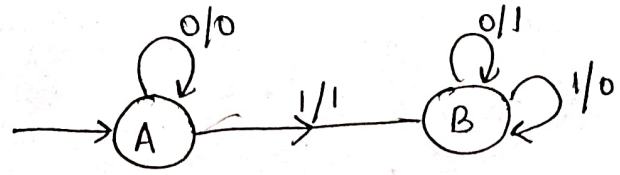


Mealy machine



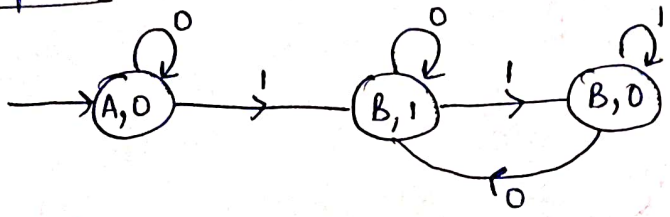
Moore machine

(2)



Mealy machine

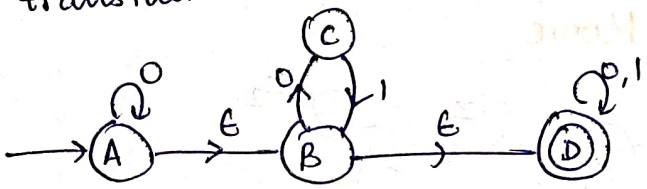
2's complement



Moore machine

Epsilon NFA

A transition can even contain ϵ (null string)



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

- Q: finite set of states
- Σ: input alphabets
- δ: Transition function
- q_0: Initial state
- F: finite set of final states.

$$\delta : Q \times \Sigma \cup \{\epsilon\} \longrightarrow 2^Q$$

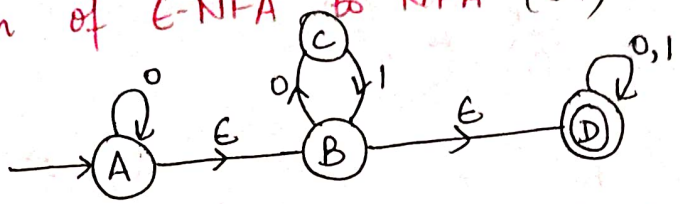
ϵ closure(A)

set of all states which can be reached from A only on seeing ϵ.

ϵ closure (A) = $\{A, B, D\}$

Conversion of ϵ-NFA to NFA (57)

(1)



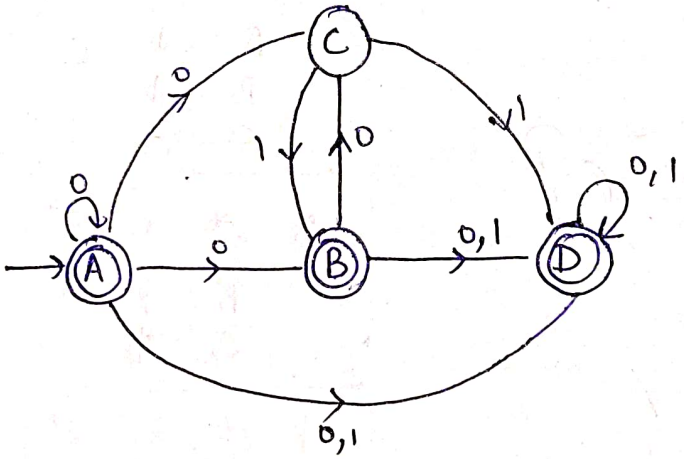
	0	1
A	{ABCD}	{D}
B	{CD}	{D}
C	ϕ	{B,D}
D	{D}	{D}

A

ϵ^*	0	ϵ^*
A	A	A
		B
B	C	D
D	D	C
		D

	ϵ^*	0	1	ϵ^*
A	A	ϕ		
	B	ϕ		
	D	D	D	

∴ NFA is



B

ϵ^*	0	ϵ^*
B	C	C
D	D	D

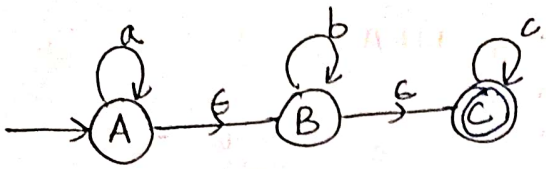
B	ϵ^*	1	ϵ^*
	B	ϕ	
	D	D	D

C	ϵ^*	0	ϵ^*
	C	ϕ	ϕ

C	ϵ^*	1	ϵ^*
	C	B	B,D

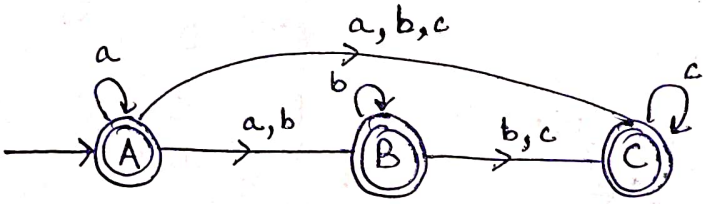
D	ϵ^*	0	ϵ^*
	D	D	D

(2)



	a	b	c
A	{ABC}	{B,C}	{C}
B	{ϕ}	{BC}	{C}
C	{ϕ}	{ϕ}	{C}

∴ NFA is



A	ϵ^*	a	ϵ^*
	A	A	A
	B	ϕ	B
	C	ϕ	C

A	ϵ^*	b	ϵ^*
	A	ϕ	B
	B	B	C
	C	ϕ	

B	ϵ^*	c	ϵ^*
	B	ϕ	C
	C	C	

A	ϵ^*	c	ϵ^*
	A	ϕ	C
	B	ϕ	
	C	C	

B	ϵ^*	a	ϵ^*
	B	ϕ	ϕ
	C	ϕ	

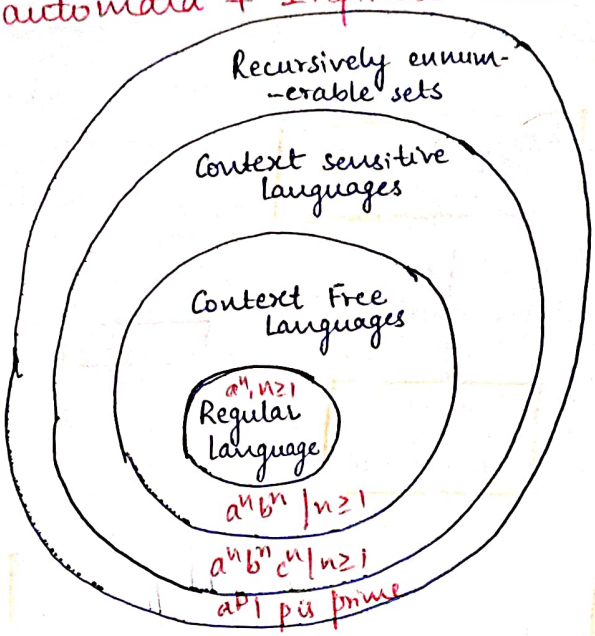
B	ϵ^*	b	ϵ^*
	B	B	B
	C	ϕ	C

C	ϵ^*	a	ϵ^*
	C	ϕ	ϕ

C	ϵ^*	b	ϵ^*
	C	ϕ	ϕ

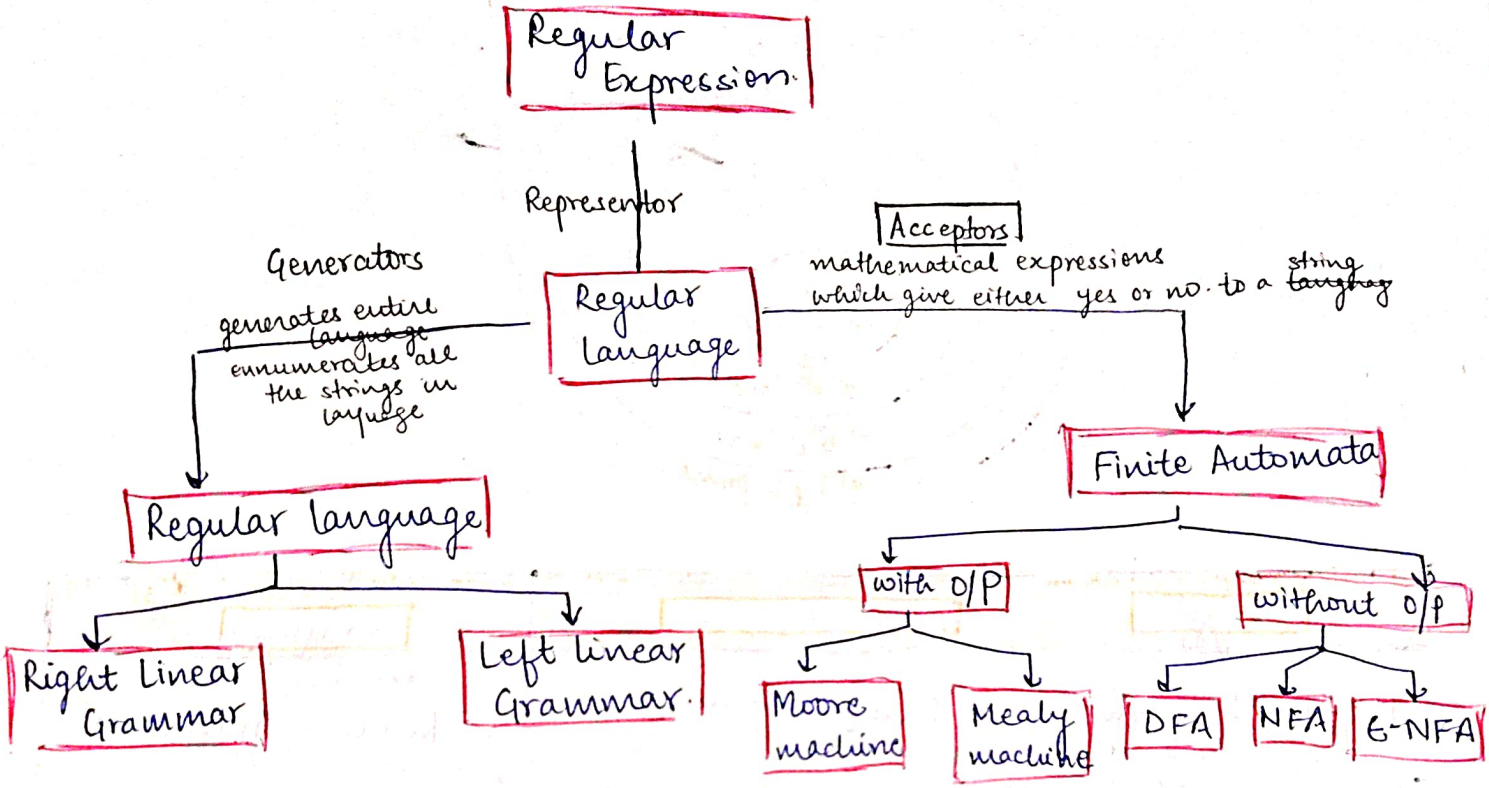
C	ϵ^*	c	ϵ^*
	C	C	C

Finite automata + Infinite memory = Push Down Automata



Language	Machine name	Grammar
Regular language	Finite Automata	Regular grammar
Context Free language	Push Down Automata	context Free Grammar
Context Sensitive language	Linear Bounded Automata	Context sensitive grammar
Recursively Enumerable set.	Turing machine.	Unrestricted grammar.

Regular Languages



Regular expression

Regular expressions are the representation of languages that are exactly accepted by Finite Automatas

Contains 3 operators

- Union (+)
- Concatenation (.)
- Kleene closure (*)

Primitive Regular expression

a) ϕ , ϵ , $a \in \Sigma$

ϕ → $\{\}$; ϵ → $\{\epsilon\}$

b) Any operators applied to primitive regular expression are regular expressions.

c) Take a and b any no. of times. The result gives regular expression.

$\phi = \{\}$

$\epsilon = \{\epsilon\}$

$a = \{a\}$

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

$a^+ = a \cdot a^* = \{a, aa, aaa, \ldots\}$

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

$\Sigma(a,b)$

① L_1 = {set of all strings whose length is exactly 2}

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

Regular expression

$= (a+b)^2$

② L_2 = length is atleast 2.

$L_2 = \{aa, ab, ba, bb, aaa, aaab, \ldots\}$

Regular expression —

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

③ L_3 = length is atmost 2

$L_3 = \{\epsilon, a, b, aa, ab, ba, bb\}$

Regular expression

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

④ Even length strings

$L = \{\epsilon, aa, ab, ba, bb, \ldots\}$

Regular expression

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

⑤ Odd length strings.

$L = \{a, b, aaa, aab, \ldots\}$

Regular expression

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

⑥ length is divisible by 3

$L = \{\epsilon, aaa, aab, \ldots bbb, \ldots\}$

Regular expression

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

(7) Length of string $\cong$ 2 mod 3
i.e. when divided by 3, remainder is 2.

Regular expression

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

(8) set of strings in which number of a's is exactly 2

$L = \{aa, aab, aba, baa, \ldots\}$

Regular expression

$b^* a b^* a b^*$.

(9) String in which number of a's is atleast 2

Regular expression

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

(10) String in which number of a's are atmost 2

Regular expression

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

(11) No. of a's are even.

~~Regular expression~~

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

Regular expression

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

or $(b^* a b^* a)^* . b$

(12) set of all strings which start with a

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

(13) set of all strings which end with a.

$(a+b)^* . a$

Set of all strings which contain 'a'

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

(14) Starting and ending with different symbols.

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

(15) starting and ending with same symbol.

$L = \{\epsilon, a, b, aa, bb \ldots\}$

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

★ No 2 a's should come together

$L = \{\epsilon, b, bb, bbb, a, ab, \ldots\}$

Regular expression

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

$(b+ab)^*$ — strings ending with b

$(b+ab)^*a$ — strings ending with a

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

OR

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

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

No 2 a's and no 2 b's should come together.

$L = \{\epsilon, a, b, ab, ba, aba, bab, \ldots\}$

Starts with	ends with	
a	a	$(ab)^*a$ / $a(ba)^*$
a	b	$(ab)^*$ of $a\|ba$
b	a	$(ba)^*$ / $b(ab)^*$
b	b	$(ba)^*b$ / $b(ab)^*$

Regular expression —

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

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

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

OR

$(\epsilon+a)(ba)^*(b+\epsilon)$

IDENTITIES OF REGULAR EXPRESSIONS

$\phi + R = R + \phi = R$

$\phi \cdot R = R \cdot \phi = \phi$

$\epsilon \cdot R = R \cdot \epsilon = R$

$\epsilon^* = \epsilon$

$\phi^* = \epsilon$

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

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

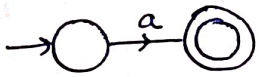
$= (a^* b^*)^* = (a^* + b)^*$

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

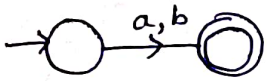
Conversion of Regular expression to Finite Automata.

ϕ :

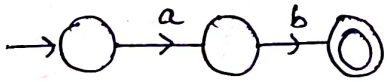
a :



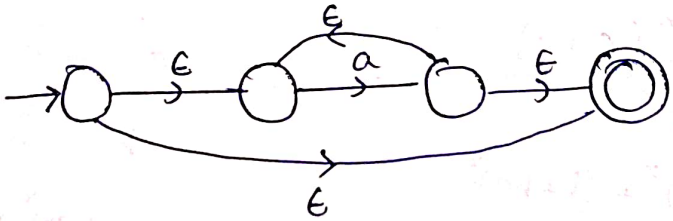
$a+b$:



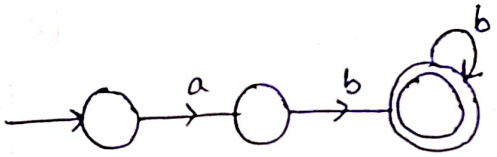
$a.b$:



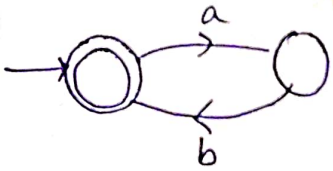
a^* :



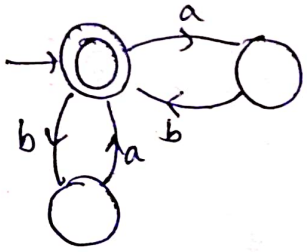
ab^* :



$(ab)^*$:



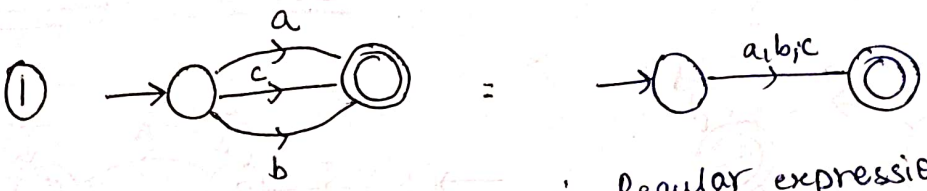
$(ab+ba)^*$:



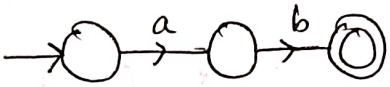
Conversion of Finite Automata to Regular Expression

State Elimination Method

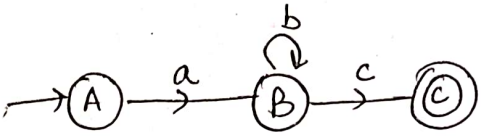
① No incoming edge to initial state.
② No outgoing edge from final state.
③ Only one final state.
④ Remove remaining states

① 

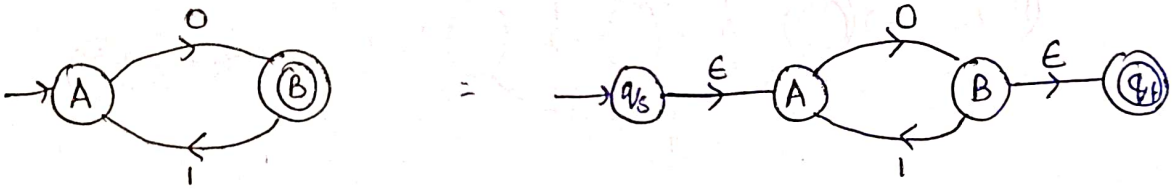
∴ Regular expression = a+b+c.

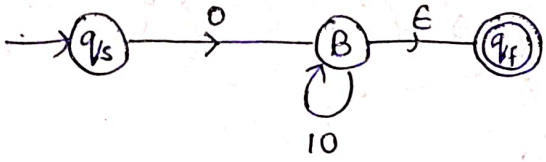
② 

Regular expression = a·b

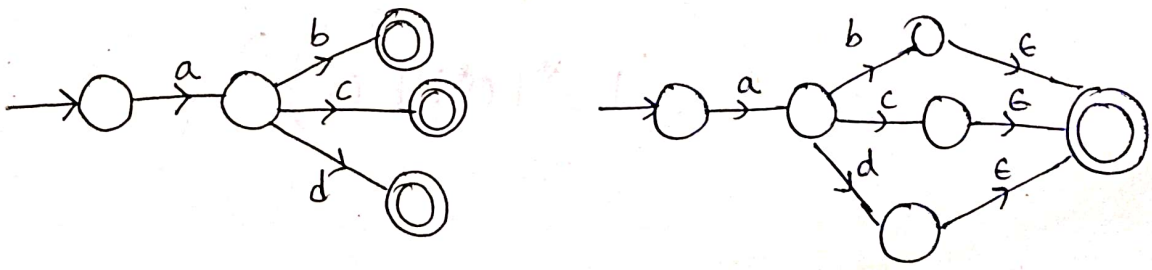
③ 

ab^*c

④ 

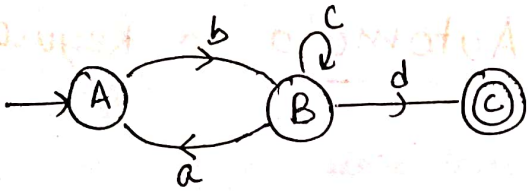


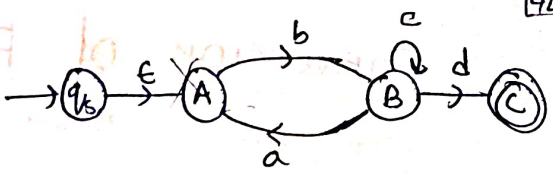
$0(10)^*$

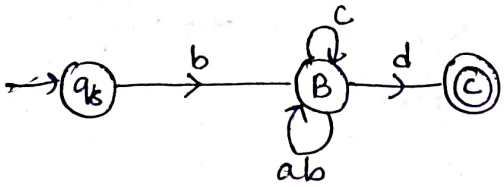
⑤ 

$a\cdot(b+c+d)$

(6)



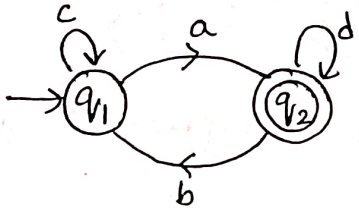


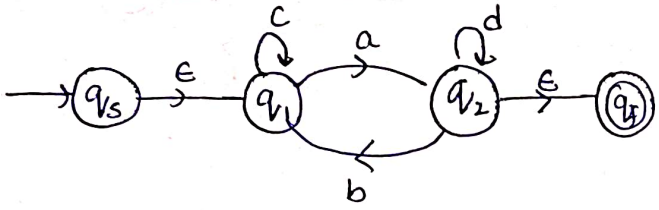


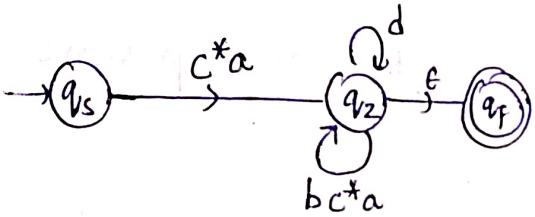
~~$b(ab)^*d$~~

$= b \cdot (ab+c)^* d$

(7)

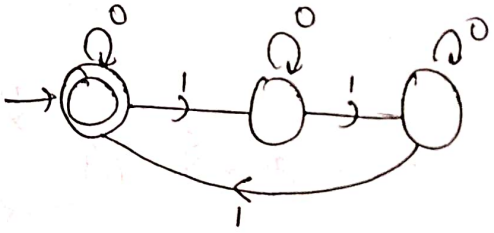


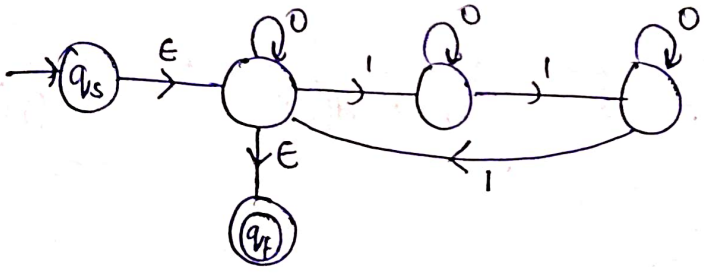




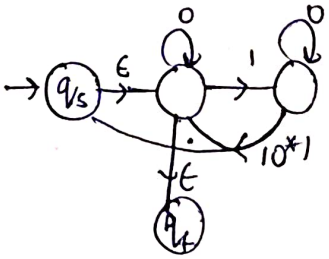
$= c^* a (d + bc^* a)$

(8)





=



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

Question:- How many n state DFAs can be constructed with a designated initial state over the alphabet Σ containing m symbols.

$|Q| = n \qquad |\Sigma| = m$

	1	2	3	4	.....	m
1						
2						
3						
⋮						
n						

No. of ~~fin~~ combinations with 0 final state = ${}^nC_0$

No. of " " 1 final states = ${}^nC_1$

No. of " " 2 final states = ${}^nC_2$

⋮

No. of " " n final states = ${}^nC_n$

$\therefore$ No. of combinations possible = ${}^nC_0 + {}^nC_1 + {}^nC_2 + \dots {}^nC_n$

$= 2^n$

In 1 combination, no. of ways of filling up the table = $n^{n \times n}$

$\therefore$ Total no. of DFAs possible = $\underline{2^n \cdot n^{n \times m}}$

Regular grammar

used to generate the entire language.

Grammar is represented by

(V, T, P, S)

- V: set of all vertices
- T: set of all terminals
- P: set of all productions
- S: start symbol

Example

$$P = \begin{cases} S \rightarrow aSB \\ S \rightarrow aB \\ B \rightarrow b \end{cases}$$

$V = \{S, B\}$

$T = \{a, b\}$

Derivation :-

Generates language $a^n b^n \mid n \geq 1$

Ques. Construct grammar that accepts set of all strings of length 2.

$L = \{aa, ab, ba, bb\}$

$S \rightarrow aa \mid ab \mid ba \mid bb$

$S \rightarrow AA$

$A \rightarrow a \mid b$

2. $a^n \mid n \geq 0$

$L = \{a, aa, aaa, \ldots\}$

$S \rightarrow aA$

$A \rightarrow a \mid \epsilon$

$A \rightarrow Aa \mid \epsilon$

3. $(a+b)^*$

$S \rightarrow aS \mid bS \mid \epsilon$

4. Set of strings of length atleast 2

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

$S \rightarrow AAB$
$A \rightarrow a \mid B$
$B \rightarrow aB \mid bB \mid \epsilon$

5. set of strings of length atmost 2

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

$S \rightarrow BB$
$B \rightarrow a \mid b \mid \epsilon$

6. set of strings starting with a and ending with b.

$a(a+b)^* b$

$S \rightarrow aBb$
$B \rightarrow aB \mid bB \mid \epsilon$

7. set of strings starting and ending with different symbols.

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

$S \rightarrow aBb \mid bBa$
$B \rightarrow aB \mid bB \mid \epsilon$

8. set of all strings starting and ending with same symbol

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

$S \rightarrow aAa \mid bAb \mid a \mid b \mid \epsilon$
$A \rightarrow aA \mid bB \mid \epsilon$

9. $a^n b^n \mid n \geq 1$

$S \rightarrow aSb \mid ab$

10. set of all pallindromes

$ww^R \cup waw^R \cup wbw^R$

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$

11. even length strings

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

$S \rightarrow BS \mid \epsilon$
$B \rightarrow AA$
$A \rightarrow a \mid b$

12. $a^n b^m \mid n, m \geq 1$

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

13. $a^n b^n c^m \mid n, m \geq 1$

$S \rightarrow AB$
$A \rightarrow aAb \mid ab$
$B \rightarrow cB \mid c$

14. $a^n c^m b^n \mid n, m \geq 1$

$S \rightarrow aSb \mid aAb$
$B \rightarrow cB \mid c$

15. $a^n b^n c^m d^m \mid n, m \geq 1$

$S \rightarrow AB.$
$A \rightarrow aAb \mid ab$
$B \rightarrow cBd \mid cd$

16. $a^n b^n c^n \mid n \geq 1$

Not possible

17. $a^n b^{2n} \mid n \geq 1$

$S \rightarrow aSbb \mid abb$

18. $a^n b^m c^m d^n \quad n, m \geq 1$

$S \rightarrow aSd \mid aAd$
$A \rightarrow bAc \mid bc$

19. $a^n b^m c^n d^m \mid n, m \geq 1$

Not possible

20. $a^{m+n} b^m c^n$

$= a^n a^m b^m c^n$

$S \rightarrow aSc \mid aAc$
$A \rightarrow aAb \mid ab$

21. $a^n b^{n+m} c^m \quad n, m \geq 1$

$a^n b^n b^m c^m$

$S \rightarrow AB$
$A \rightarrow aAb \mid ab$
$B \rightarrow bBc \mid bc$

22. $a^n b^m c^{n+m}$

$= a^n b^m c^m c^n$

$S \rightarrow aSc \mid aAc$
$A \rightarrow bAc \mid bc$

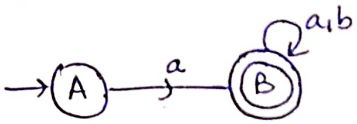
Classification of grammar.

Chomsky classification

- **Type 3 grammar**
  - Production is of the form
  - R.L.G. $A \to \alpha B \mid \beta$ or
  - L.L.G. $A \to B\alpha \mid \beta$
  - $A, B \in V$
  - $\alpha, \beta \in T$.
  - Regular language
- **Type 2 grammar**
  - Production of the form
  - $A \to \alpha$
  - $A \in V$
  - $\alpha \in (V \cup T)^*$
- **Type 1 grammar**
- **Type 0 grammar**

② Finite automata (L) $\longrightarrow$ Right Linear grammar (L) $\xrightarrow{R}$ Left linear grammar (L^R)

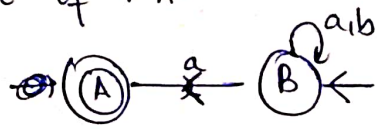
Example –



$A \to aB$
$B \to aB \mid bB \mid \epsilon$] Right Linear grammar (L)

$A \to Ba$
$B \to Ba \mid Bb \mid \epsilon$] Left linear grammar (L^R)

Reverse of F.A.



L^R

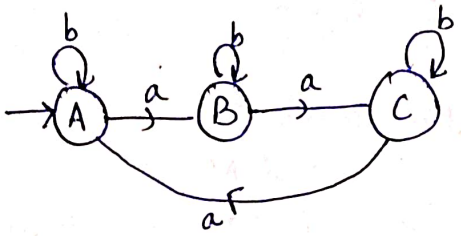
~~A~~ $B \to aB \mid bB \mid aA$
$A \to G$] Right linear grammar L^R

$B \to Ba \mid Bb \mid Aa$
$A \to \epsilon$] Left linear grammar L

Convert the following regular grammar to F.A.

$A \to aB \mid bA \mid b$
$B \to aC \mid bB$
$C \to aA \mid bC \mid a$

Right linear grammar



Elimination of epsilon productions, unit productions and useless symbols from CFG.

→ Elimination of ϵ-production

① Find nullible variable (variable that directly gives ϵ)
② Write production with & without the nullible variable
③ Remove the variable.

① $S \to AB$
$A \to aAA \mid \epsilon$
$B \to bBB \mid \epsilon$

Nullible variables = {S, A, B}

$S \to AB \mid B \mid A \mid \epsilon$
$A \to aAA \mid aA \mid a$
$B \to bBB \mid bB \mid b.$

② $S \to AbaC \mid baC \mid Aba \mid ba$
$A \to BC \mid B \mid C$
$B \to b \mid \epsilon$
$C \to D \mid \epsilon$
$D \to d$

Nullible variables = {B, C, A}

$S \to AbaC \mid baC \mid Aba \mid ba$
$A \to BC \mid B \mid C$
$B \to b$
$C \to D$
$D \to d$

→ Elimination of unit production

(1) $S \to AB$
$A \to a$
$B \to C \mid b \mid a$
$C \to D \mid a$
$D \to E \mid a$
$E \to a$

$B \to C \to D \to E \to a$
$C \to D \to E \to a$
$D \to E \to a$

∴ After elimination

$S \to AB$
$A \to a$
$B \to b \mid a$
$C \to a$
$D \to a$
$E \to a$

→ Elimination of useless symbols

$S \to AB \mid AC$
$A \to aAb \mid bAa \mid a$
$B \to bbA \mid aaB \mid AB$
$C \to abCA \mid aDb$
$D \to bD \mid aC$

Useful symbols = $\{a, b, A, B, S\}$

Remove useless symbols ($\{C, D\}$)

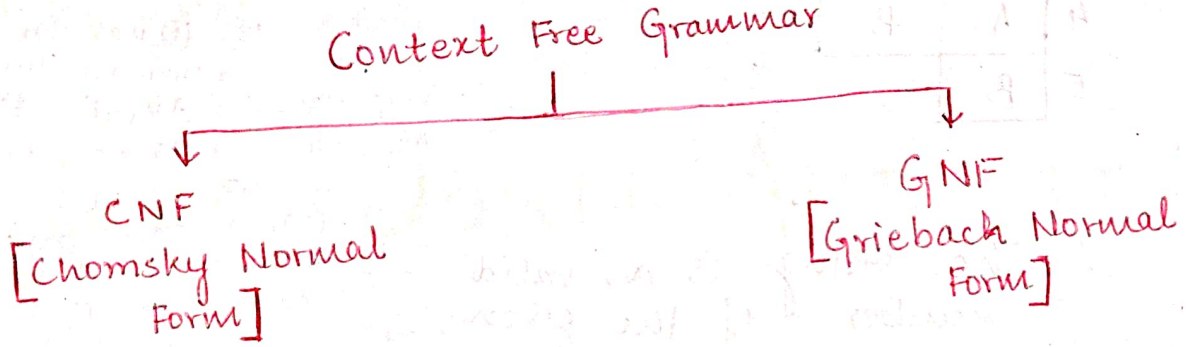
Remove all transitions which contain useless symbols and also remove unreachable symbols.

$S \to AB$
$A \to aAb \mid bAa \mid a$
$B \to bbA \mid aaB \mid AB$

$S \rightarrow ABC \mid BaB$
$A \rightarrow aA \mid BaC \mid aaa$
$B \rightarrow bBb \mid a$
$C \rightarrow CA \mid AC$

Useful symbols = $\{a, b, A, B, S\}$

$S \rightarrow BaB$
$B \rightarrow bBb \mid a$



Chomsky Normal Form

If all the productions are of the form $A \rightarrow BC$ and $A \rightarrow a$ where A, B, C are variables and a is a terminal, then, the grammar is CNF.

Advantages

- Length of each production is restricted
- Derivation tree or parse tree obtained from CNF is always binary tree.
- The number of steps required to derive a string of length $|w|$ is $2|w| - 1$
- It is easy to apply CYK membership algorithm

CYK membership
↓
tests membership in $O(|w|^3)$ time universal in nature

Check whether the string "aabbb" is a valid member of following grammar or not.

$S \rightarrow AB$
$A \rightarrow BB \mid a$
$B \rightarrow AB \mid b$

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

∴ The string is a valid member of the given grammar.

1 2 3 4 5
a a b b b

1,4
(11)(24) (12)(34) (13)
AA ϕA SB B

aa = 12 = (11)(12) AA A SB BB
= AA
23 = (22)(33)
= AB
S, B
34 = (33)(44)
= BB

(2,4)
(22)∪(34) + (23)(44) 45 = (44)(55)
[A][A] + (S, B) ∪ B = BB

AA SB BB
3,5
(34)(55) (33)(45) = A ∪ (S, B)
AB BA = AS AB B

2,5
(22)(35) (23)(45) (24)(55)
(A)(SB) (SB)A A B
AS AB SA BA AB

1 2 3 4 5
(11)(25) | (12)(35) | (13)(45) | (14)
(A)(SB) | ϕ(SB) | (SB)A | AB
AS AB SA BA AB
S, B

1,4
AA ϕA SB B

1 2 3 4

2, 3, 4
1 | 2 3

12 ∪ 3
(12) ∪ (33)
ϕ ∪ B

2 3 5
(22)(35) (23)(45) (24)(55)

Greibach Normal Form

If all the productions are of the form $A \to a\alpha$ where $\alpha \in V^*$, then, it is known as GNF.

Example:- $A \to aBCDE$
$B \to a$

Advantages –

① The no. of steps required to generate a string of length $|w|$ is $|w|$.

② useful to convert CFG to PDA.

PUSH DOWN AUTOMATA

7 tuples –

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

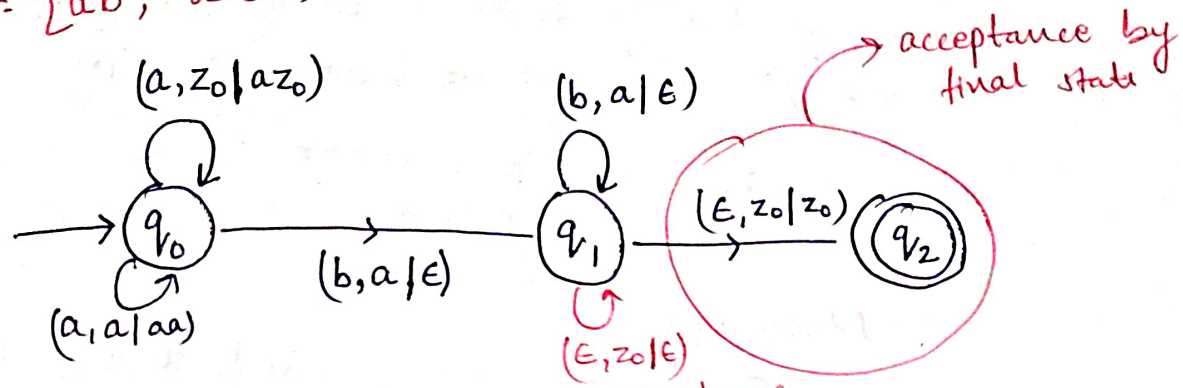
- Q : Finite set of states
- Σ : Input symbol
- δ : Transition Function
- q_0 : Initial state
- Z_0 : Bottom of the stack
- F : set of final states
- Γ : Stack alphabet

Deterministic PDA: $Q \times \{\Sigma \cup \epsilon\} \times \Gamma \longrightarrow Q \times \Gamma^*$

Non Deterministic PDA: $Q \times \{\Sigma \cup \epsilon\} \times \Gamma \longrightarrow Q \; 2^{(Q \times \Gamma^*)}$

① $L = \{a^n b^n \mid n \geq 1\}$

$L = \{ab, aabb, aaabbb, \ldots\}$



Transition Function

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

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

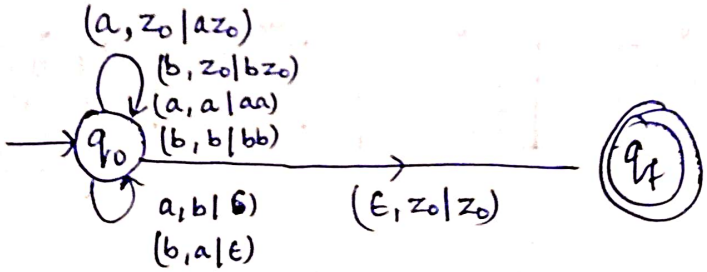
$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) \rightarrow (q_2, Z_0)$

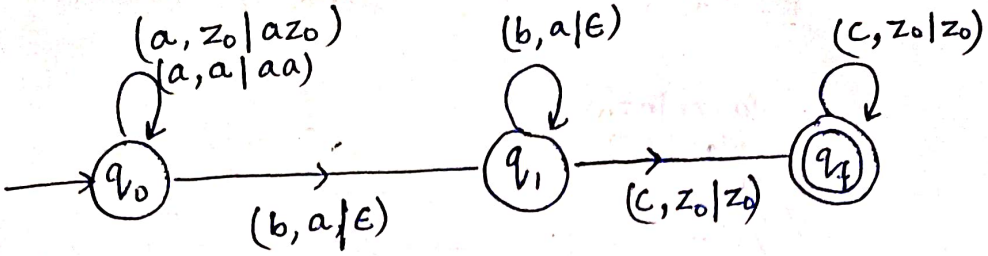
② $L = \{w \mid n_a(w) = n_b(w)\}$

$L = \{\epsilon, ab, ba, aabb, abba, baba \ldots\}$

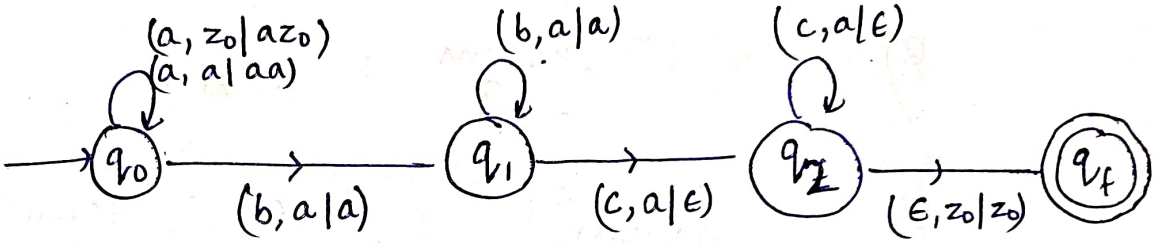


BS

(3) $L = \{ a^n b^n c^m \mid n, m \geq 1 \}$

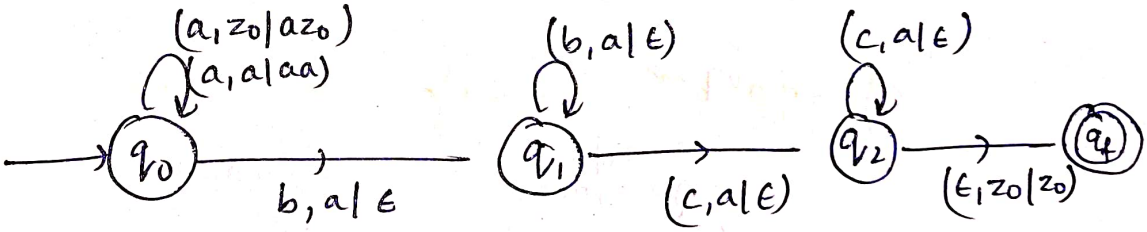


(4) $L = \{ a^n b^m c^n \mid n, m \geq 1 \}$



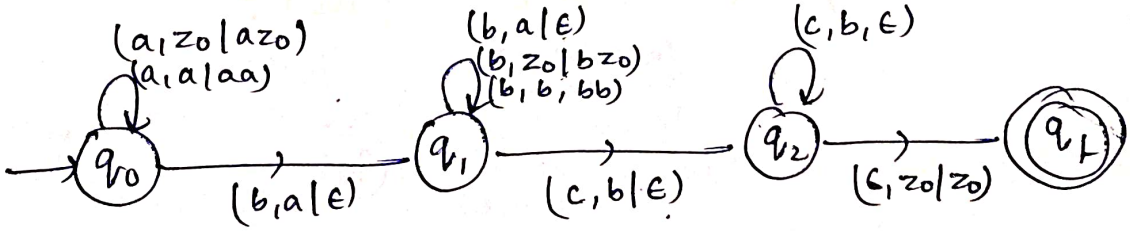
(5) $L = \{ a^{m+n} b^m c^n \mid m, n \geq 1 \}$

$L = a^{m+n} b^m c^n = a^n a^m b^m c^n$



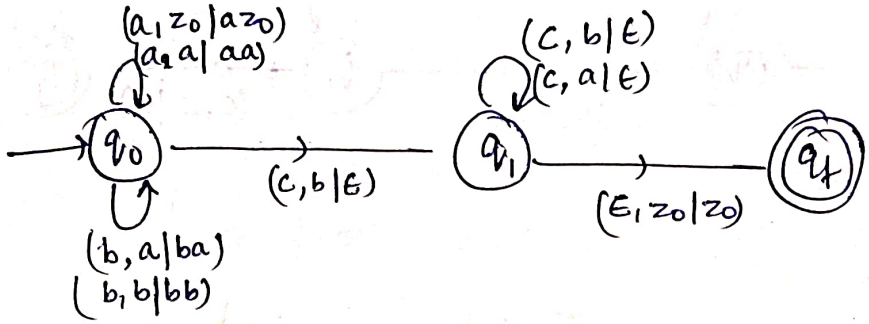
(6) $L = \{ a^n b^{m+n} c^m \mid n, m \geq 1 \}$

$L = a^n b^{m+n} c^m = a^n b^n b^m c^m$

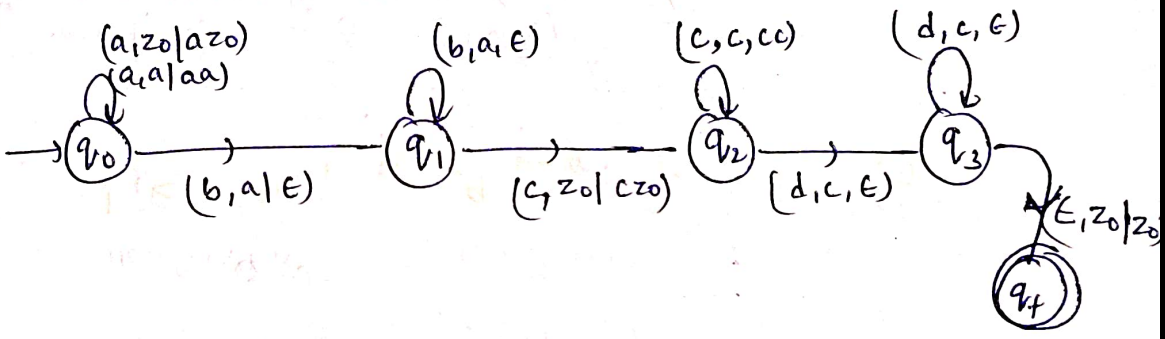


7) $L = \{ a^n b^m c^{n+m} \mid n, m \geq 1 \}$

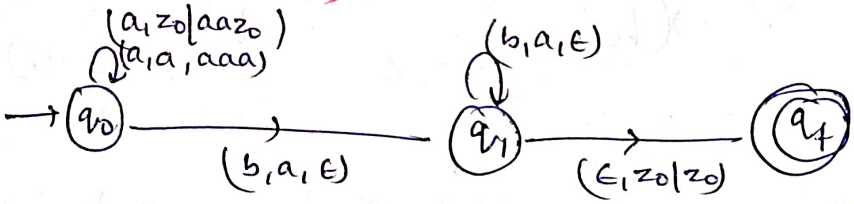
$L = a^n b^m c^{n+m}$



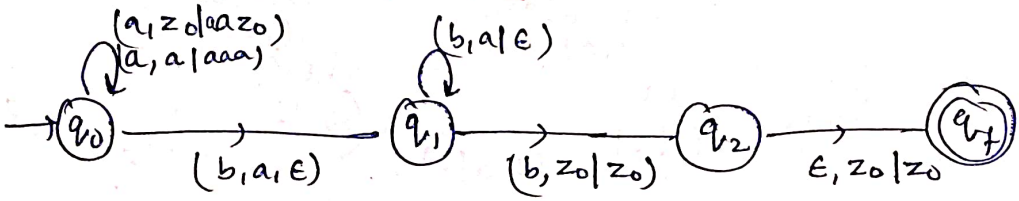
8) $a^n b^n c^m d^m \mid n, m \geq 1$



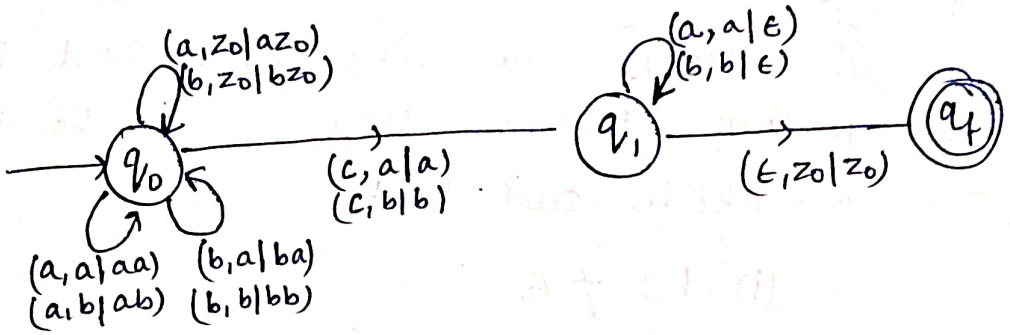
9) $L = \{ a^n b^{2n} \quad n \geq 1 \}$



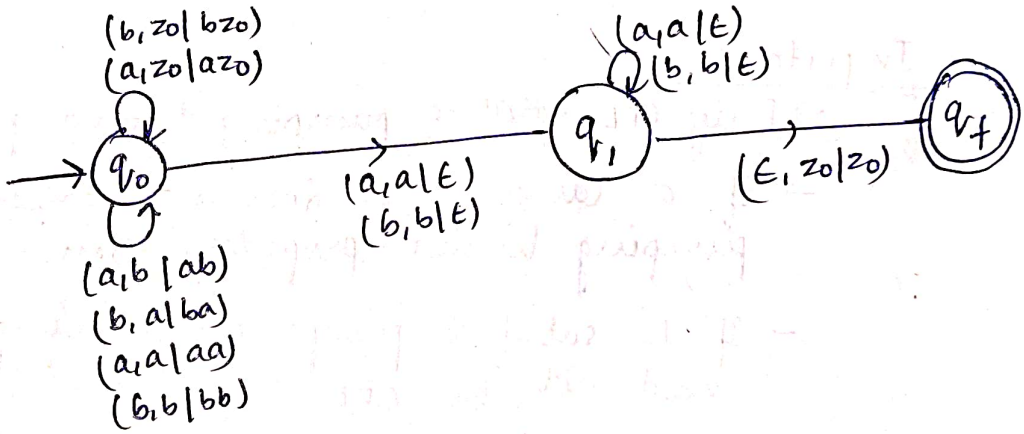
10) $L = \{ a^n b^{2n+1} \mid n \geq 1 \}$



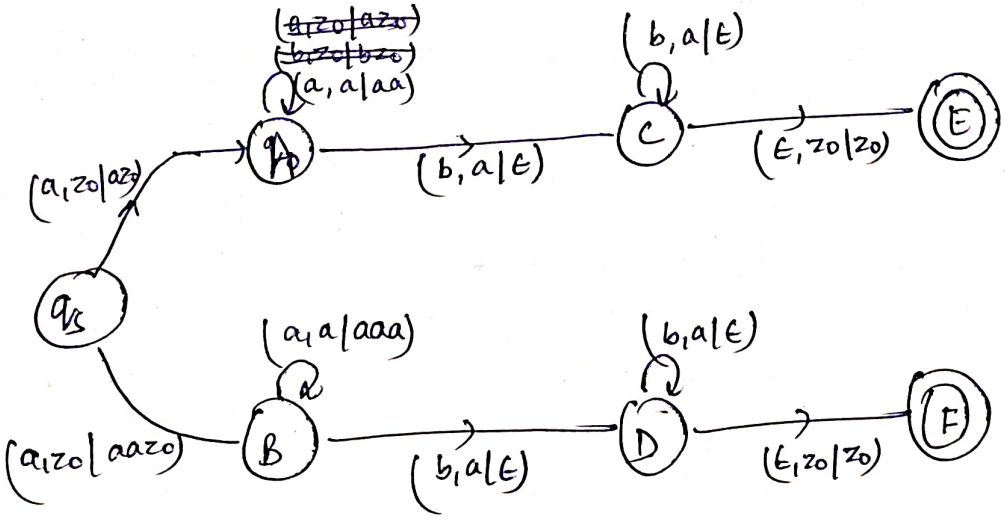
11) $L = \{ wcw^R \mid w \in (a,b)^+ \}$ palindromes in which center is known.



12) $L = \{ ww^R \mid w \in (a,b)^+ \}$ even length palindromes



13) $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ a^n b^{2n} \mid n \geq 1 \}$



Pumping lemma for context free language.

If L is a CFL and $w \in L$ such that $|w| \geq n$, for some positive integer n, then, w can be decomposed as $w = abcde$ such that

(i) $bd \neq \epsilon$

(ii) $|bcd| \leq n$

(iii) For all $i \geq 0$, the string ab^icd^ie is also in L

Important

→ Every CFL satisfies pumping lemma property.

→ If a language L does not ~~satisfies~~ satisfy pumping lemma property, then, L is not CFL.

→ If L satisfies pumping lemma property, L need not be CFL.

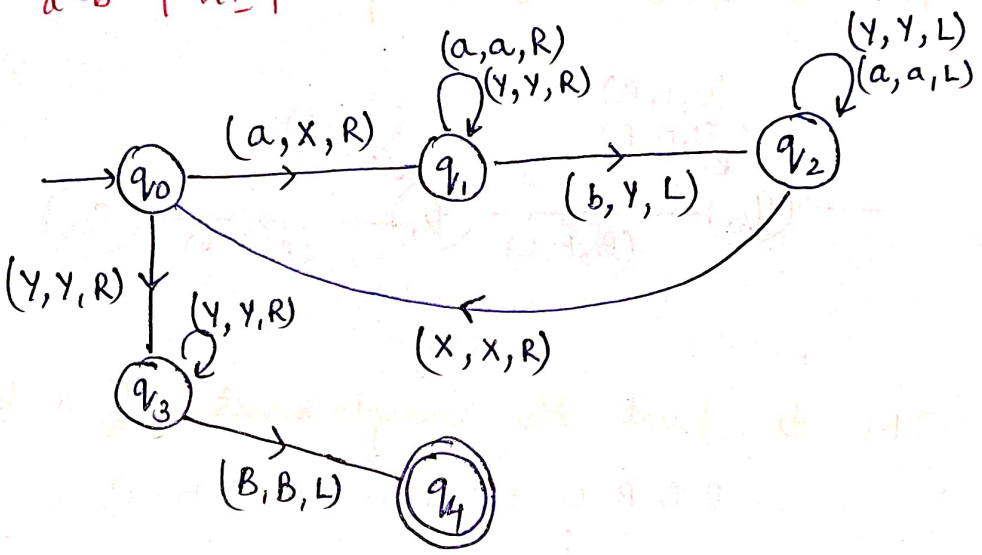
→ Every context free language satisfies pumping lemma property.

Turing Machine

FA and PDA can move in only 1 direction. Turing machine can move both forward and backward.

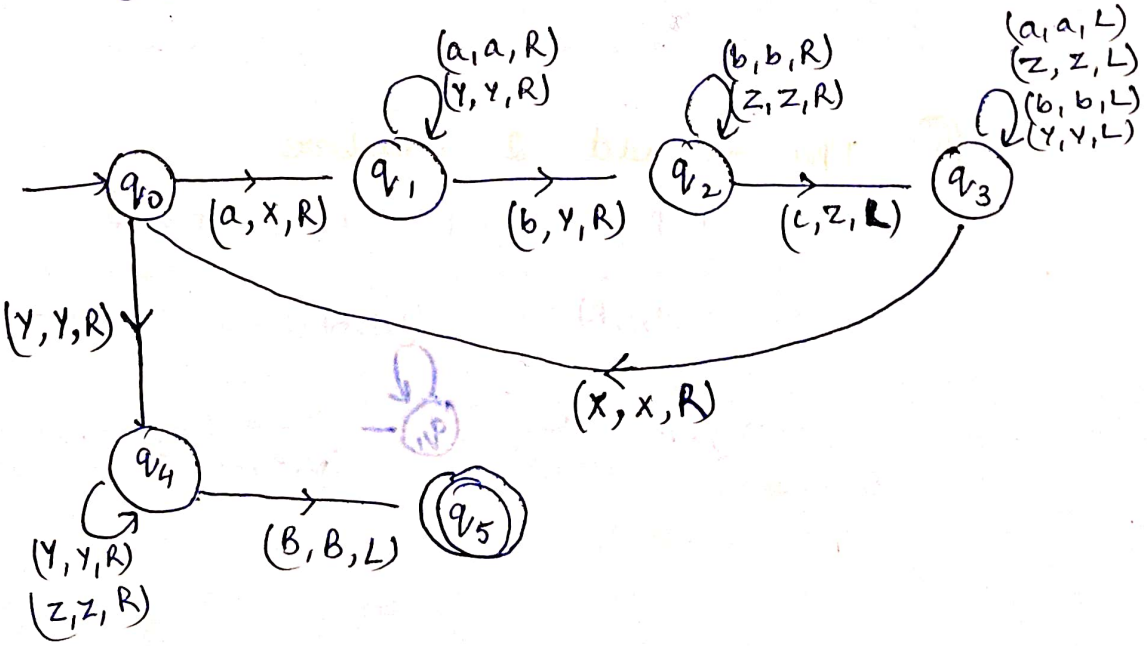
FA and PDA cannot modify the input tape. Turing machine can read as well as write on the input tape.

(1) $a^n b^n \mid n \geq 1$



(2) $L = \{a^n b^n c^n \mid n \geq 1\}$

B B a a a b b b c c c B B

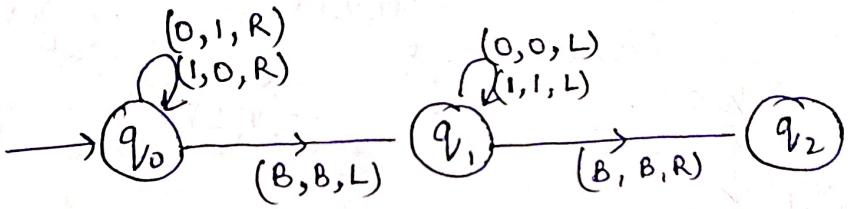


Acceptor :- tells whether a string belongs to a language or not.

Transducer :- gives output.

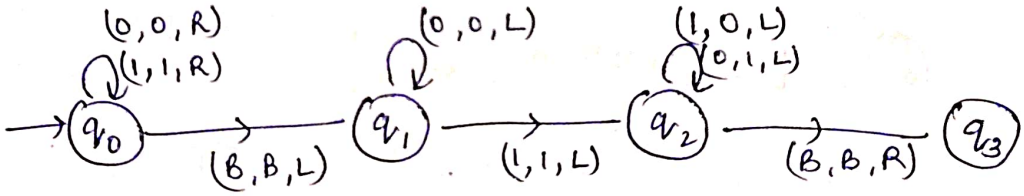
Turing machine can act as an acceptor as well as transducer

(3) TM to find 1's complement of a number



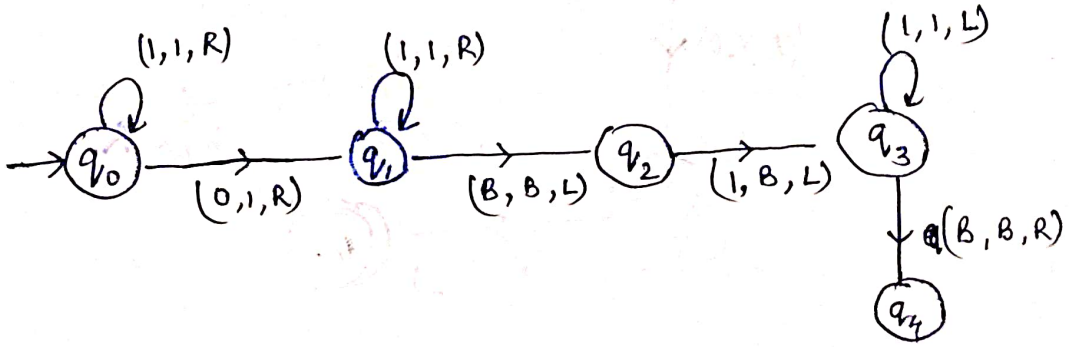
(4) TM to find 2's complement of a binary number

B B B 0 1 1 1 0 0 0 B B B



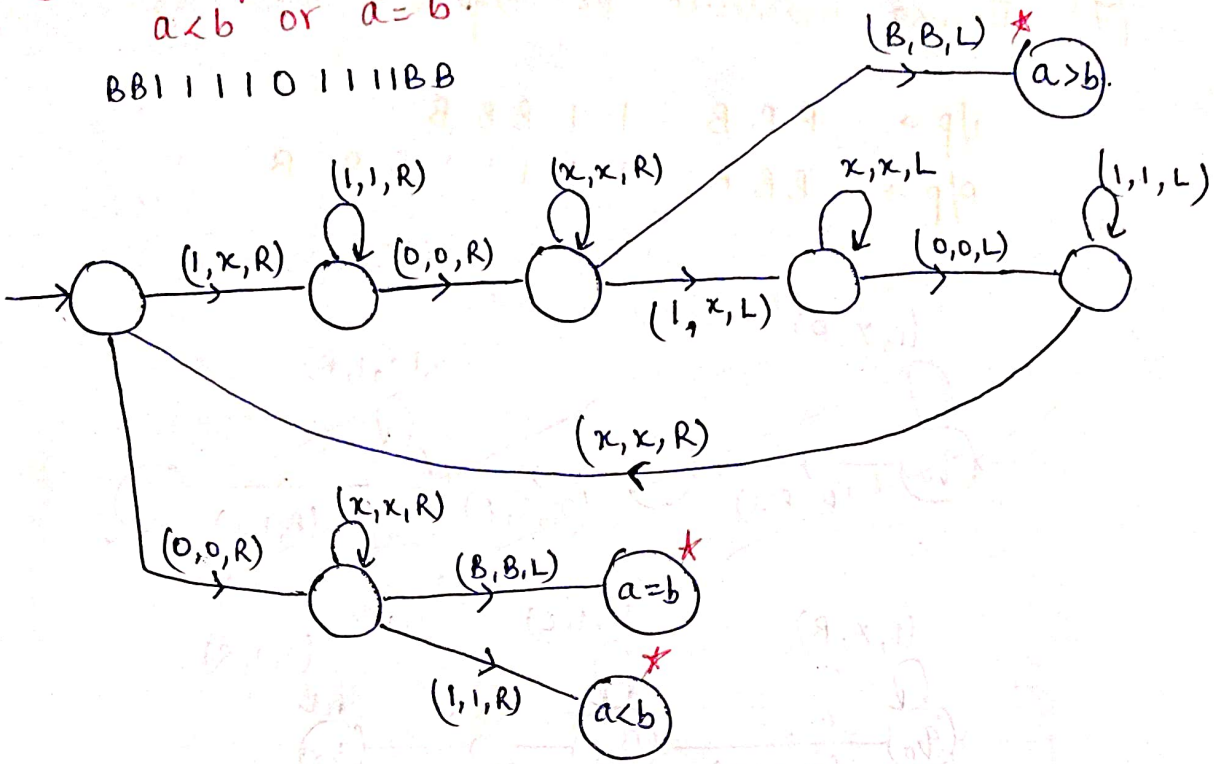
(5) TM to add 2 numbers

B B B 1 1 1 0 1 1 1 1 B B B



⑥ Turing machine which tells whether $a > b$ / $a < b$ or $a = b$.

BB1111011111BB



The standard Turing Machine.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

- Q — Finite set of states
- Σ — Input alphabet
- Γ — Tape alphabets
- δ — Transition Function
- q_0 — Initial State
- B — Blank symbol
- F — Set of final state

$$\delta : (Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\})$$

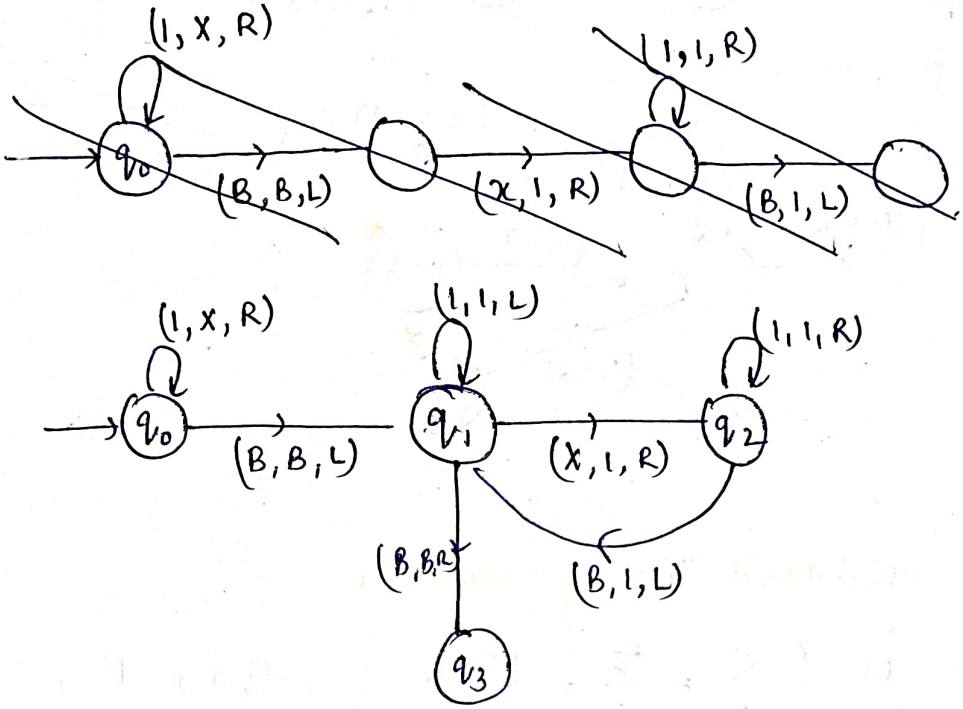
Summary

- Tape is unbounded, so any number of left and write moves are possible.
- It is deterministic (atmost one move for each configuration)
- No special i/p or o/p file (everything is on tape)

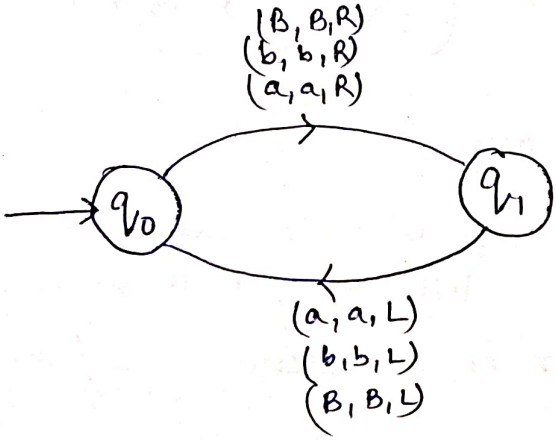
(7) Turing machine as copier

i/p → B B B 1 1 1 B B B

o/p → B B B 1 1 1 1 1 1 B B B



Non Halting Turing Machine



Turing Thesis

Modifications in Turing Machine

- TM with stay option
- TM with semi-infinite tape
- Offline TM
- Multitape TM

$$\delta: Q \times \Gamma^n \longrightarrow Q \times \Gamma^n \times \{L, R\}^n$$

- Jumping TM
- Non erasing TM
- Always writing TM.
- Multidimensional TM

$$\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, U, D\}$$

- Multihead TM
- Atomata with a queue.
- ★ TM with only 3 states
- ★ Multitape TM with stay option and atmost 2 states
- ★ Non Deterministic TM

$$\delta: Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

- A NPDA with 2 independent stacks

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

TM + Tape (input size) = LBA

TM + Tape (stack) = PDA

TM + Tape (finite size) = FA

FA → equivalent

PDA → Non deterministic is more powerful than determ

LBA → undecided

TM → both are equivalent

Languages accepted by LBA

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

$L = \{ a^{n!}, n \geq 0 \}$

$L = \{ a^n : n = m^2, m \geq 1 \}$

$L = \{ a^n \mid n \text{ is a prime} \}$

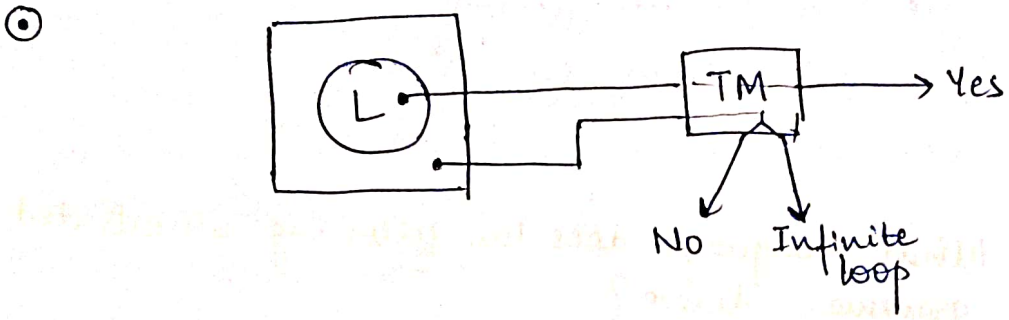
$L = \{ a^n \mid n \text{ is not a prime} \}$

$L = \{ ww \mid w \in \{a,b\}^+ \}$

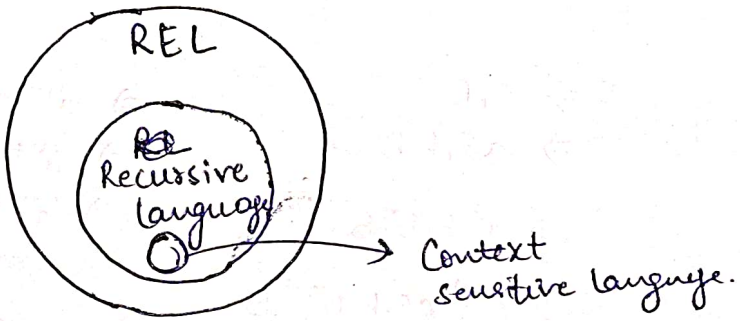
$L = \{ w^n \mid w \in \{a,b\}^+, n \geq 1 \}$

$L = \{ www^R \mid w \in \{a,b\}^+ \}$

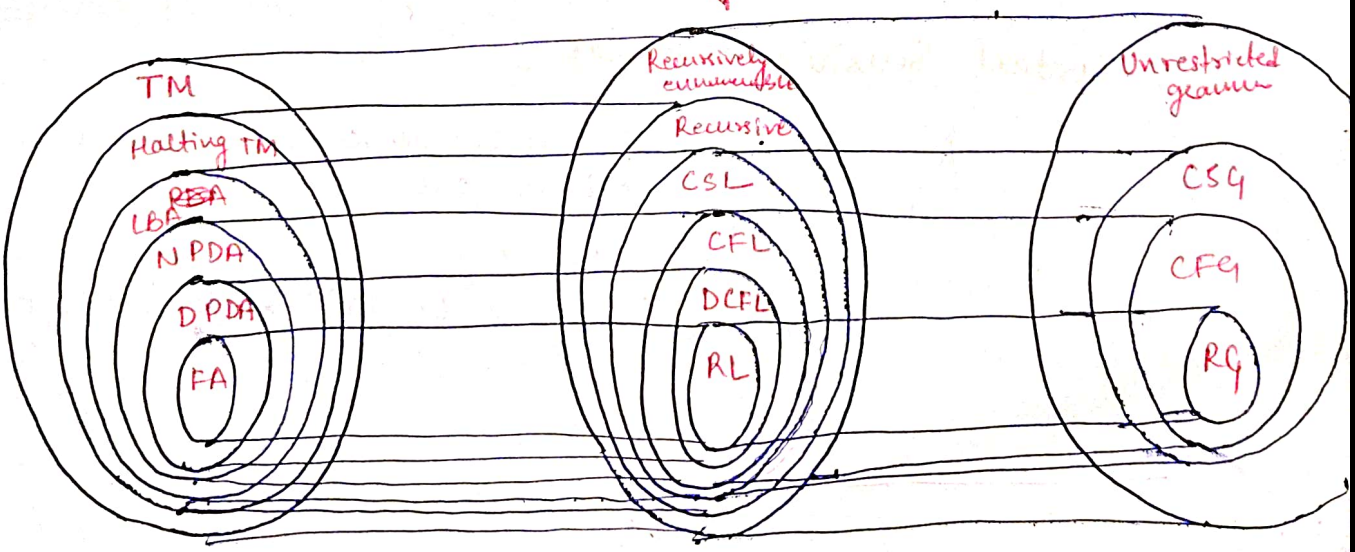
- Language accepted by a TM is called 'Recursively enumerable Language'



- Languages for which halting TM can be constructed (which either gives yes or no as output never falls into infinite loop) are known as Recursive languages



TOC - The Big Picture



Unrestricted grammar

A grammar is called unrestricted grammar if all the productions are of the form $\boxed{u \to v}$

where u is in $(V \cup T)^+$

v is in $(V \cup T)^*$

Q:- What language does the following unrestricted grammar derive?

$S \to S_1 B$

$S_1 \to aS_1 b$

$bB \to bbbB$

$aS_1 b \to aa$

$B \to \epsilon$

$S \Rightarrow S_1 B$

$\Rightarrow aS_1 bB$

$\Rightarrow a^n S_1 b^n B$

$\Rightarrow a^{n-1} \underline{aS_1 b}\, b^{n-1} B$

$\Rightarrow a^{n-1} aa\, b^{n-1} B$

$\Rightarrow a^{n+1} b^{n-1} B$

$\Rightarrow a^{n+1} b^{n-2} \cdot bB$

$\Rightarrow a^{n+1} b^{n-2} b^k$ where k = odd no.

$\Rightarrow a^{n+1} b^{n+(k-2)}$

$\Rightarrow a^{n+1} b^{n+m}$

$m = -1, 1, 3 \ldots$

$n = 1, 2, 3 \ldots$

Context Sensitive grammar

A grammar is said to be context sensitive if all the productions are of the form

$$x \to y$$

where $x, y \in (V \cup T)^+$ and $|x| \leq |y|$

Ques:- What is the language generated by the following CSL

$S \rightarrow abc \mid aAbc$

$Ab \rightarrow bA$

$Ac \rightarrow Bbcc$

$bB \rightarrow Bb$

$aB \rightarrow aa \mid aaA$

$S \Rightarrow aAbc$
$\Rightarrow abAc$
$\Rightarrow abBbcc$
$\Rightarrow aBbbcc$
$\Rightarrow aaAbbcc$
$\Rightarrow aabAbcc$
$\Rightarrow aabbAcc$
$\Rightarrow aabbBbccc$
$\Rightarrow aabBbbccc$
$\Rightarrow aaBbbbccc$
$\Rightarrow aaabbbccc$

$\therefore$ Language = $a^n b^n c^n \mid n \geq 1$

Theorem:

If a language L and its complement $\bar{L}$ are both recursively enumerable, then, both the languages are recursive.

If L is recursive, then, $\bar{L}$ is also recursive and consequently both are recursively enumerable.

Countable set

A set is said to be countable if there exists an ennumeration method using which all the elements of the set can be generated; and for any particular element, it takes only finite number of steps to generate it.

The finite number of steps taken to generate an element can be used as its index and hence a mapping into natural numbers.

Every subset of countable set is either finite or countable.

Every turing machine can be encoded into $\{0,1\}^*$

Set of all ~~alphabets~~ strings over $\{a,b\}^*$ are countable

set of all turing machines are countable

Set of all languages is uncountable.

∴ There must be some languages which are not recursively enumerable.
ie. Turing Machines are not possible for every language

If S is a countable set, 2^S is uncountable

→ If S_1 and S_2 are countable sets, then, $S_1 \cup S_2$ is countable and $S_1 \times S_2$ is countable.

→ The cartesian product of finite number of countable sets is countable.

→ The set of all languages that are not recursively enumerable is uncountable.

Turing Machine halting problem

Given the description of a TM 'm' and an i/p 'w', does 'm' when started with 'w' as its i/p eventually halts?

Theorem

If the halting problem were decidable, every RE language would be recursive.
Consequently, the halting problem is undecidable.

Recursively enumerable language	Recursive language
① TM exists	① Halting TM exists
② No membership algorithm exists	② Membership algorithm exists.

Undecidable problems

① **State entry problem**
Given a TM, a state $q \in Q$ and $w \in \Sigma^+$, decide whether or not q is ever entered when M is applied to w.

② Given a TM M, whether or not M halts if started with a blank tape.

③ Almost any problem related to RE language is undecidable

Decidability Table

Problem	RL	DCFL	CFL	CSL	Recursive language	REL
Does $w \in L$ (Membership P)	D	D	D	D	D	UD
Is $L = \phi$ (Emptiness P)	D	D	D	UD	UD	UD
Is $L = \Sigma^*$ (Completeness P)	D	UD	UD	UD	UD	UD
Is $L_1 = L_2$ (Equality P)	D	UD	UD	UD	UD	UD
Is $L_1 \subseteq L_2$ (Subset P)	D	UD	UD	UD	UD	UD
$L_1 \cap L_2 = \phi$	D	UD	UD	UD	UD	UD
Is L finite or not (Finiteness P)	D	D	D	UD	UD	UD
Is Complement of L a language of same type or not	D	D	UD	D	D	UD
Is intersection of 2 languages of same type	D	UD	UD	UD	UD	UD
Is L Regular language	D	D	UD	UD	UD	UD

1. ⊙ Every subset of a countable set is countable.

∵ N is countable
∴ set of prime numbers is countably infinite.

2. ⊙ Union of 2 countable sets is countable.

$\Sigma \cup \Sigma^* \rightarrow$ countable

$(N \times N) \cup \Sigma^* \rightarrow$ countable.

$CI \cup CI \cup CI \ldots\ldots\ldots = CI/C$

↳ countably infinite times.

↳ can be countable can be countable infinite.

3. ⊙ Union of countable number of countable sets is countable.

4. ⊙ Intersection of 2 countable sets = Countable.

$CI \cap CI = CI.$

↳ $A \cap B \subseteq A$ ∵ A is countable, ∴ $A \cap B$ is also countable

[Subset of any countable set is countable]

5. ⊙ Cartesian product of 2 countable sets is countable.

↳ finite family of countable sets.

$C \times C = C$

$C \times C \times C = C$

$C \times C \times C \times \ldots\ldots \times C = C$

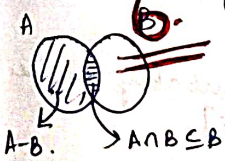
$C \times C \times C \times \ldots\ldots = C$ may / maynot be countable

↳ infinite times / countably infinite times

$UC - C = UC$

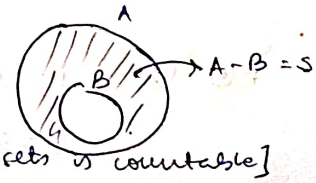
6. ⊙ $A \rightarrow$ uncountable $B \rightarrow$ countable

$A - B$ = uncountable.



$A = (A \cap B) \cup (A - B)$

countable ↳ uncountable.



Let S be countable. Then,

$A = S \cup B$ [union of countable sets is countable]

∴ A should be countable

Contradiction

∴ ~~S is ∞~~ ∴ A − B is uncountable

* Set of irrational nos is uncountable. $\mathbb{R} = R - Q$.

Uncountable ∪ Countable = Uncountable

Uncountable ∪ Uncountable = Uncountable

$R - Irr = \overset{\text{rational}}{Q}$

$R - (0,1) = R^*$ (uncountable)

Uncountable − Countable = Uncountable

Uncountable − Uncountable = may be countable/uncountable.

Countable − Countable = Countable

Countable − Uncountable = Countable [coz subset of countable]

Uncountable × Non empty set = uncountable

Uncountable × ϕ = ϕ countable.

Cantor's Theorem

Let S be any set (finite, infinite, countable, uncountable whatever)

$|S| < |P(S)|$ → There are many levels of infinity.

↳ Power set of S.

$\therefore |N| < |P(N)|$

Natural numbers

$\therefore P(N)$ is uncountable

⊙ For every infinite set S, P(S) is always uncountable

↳ for finite sets, P(S) is countable (common sense!!)

★⊙ Set of all languages is uncountable.

$L = P(\Sigma^*)$ → uncountable [Power set of countably infinite set]

→ A power set is infinite iff it is uncountable.

Interesting ⊙ A power set can never be countably infinite.

Set of all finite subsets of N	→ countable
Set of all possible subsets of N	→ uncountable.

Language $\subseteq \Sigma^*$ Since Σ^* is countable

$\therefore$ L is countable.

6. ⊙ Every Language is countable (reg / CFL/CSL/ Rec/ RE/ ~~Not RE~~).

Understand the difference !!!!!

Every Language is countable	$\because L \subseteq \Sigma^*$
Set of all languages is uncountable	$\because S = P(\Sigma^*)$

→ Set of all DFAs are countable

→ Set of all Turing machines are countable

→ Set of all NFAs / PDAs / CFGs / DPDAs are countable

} binary encoding exists. → $\subset \Sigma^*$

Set of all grammars	} Countable.
Set of all automata	
Set of all regular expressions	

Set of all computer programs → countable.

★ For any infinite language L, there is atleast one subset that is not RE

→ uncountable no. of subsets which are not RE

Set of all languages	→ uncountable
Set of regular languages	→ countable
$\therefore$ set of non regular languages	= uncountable.

For any infinite language L, there are uncountable number of subsets

$|L| < |P(L)|$

(L: infinite; P(L): infinite of higher level uncountable)

* Let L be a language for which every subset is a RE language

Then, L is finite
↳ regular / decidable / RE.

① Set of non RE languages → countable (coz it is language)
② Set of all non RE languages → uncountable (uncount - countable)

CLOSURE PROPERTIES

		Regular (closed in all)	DCFL (closed in complement & ∩ with regular only)	CFL (not closed in ∩, ~)	CSL	Rec (closed in all)	RE (not closed in ∩, ~)
All except DCFL	Union		N				
All except CFL, DCFL	Intersection		N	N			
All except DCFL, CFL, RE	~~Complem~~ Set Difference		N	N			N
all except RE & CFL	Complement			N			N
All are closed	Intersection with Reg. language						
All except DCFL	Kleene star		N				
All except DCFL	Kleene plus		N				
All except DCFL	Concatenation		N				
all except DCFL	Reversal		N				

not closed means may or may not be