L-1

FLAT

* Alphabet - $\{a,b\}$ $\{a,b,c\}$ $\{0,1\}$ $\{0,1--9\} = \Sigma$

* String - Ex: a, b, aa, ab, bc ...

* If the Alphabet set Σ contains 'n' alphabets i.e the no. of alphabets in the set are 'n' (denoted by $|\Sigma|$) the the no. of possible strings of length 'n' possible over Σ is $|\Sigma|^n$.

* Language = collection of strings

Let, $\Sigma = \{a,b\}$ then Language

L_1 = {set of all strings of length 2} → Finite Language

= {aa, ab, ba, bb}

L_2 = {set of all strings of length 3} → Finite Language

= {aaa, aab, aba, abb, baa, bab, bba, bbb}

L_3 = {set of all strings where each string start with 'a'}

= {a, aa, aaa, ab, abb, aba ---- } → Infinite Language

Powers of Σ

Let the Alphabet set be $\Sigma = \{a,b\}$

Σ^1 = {set of all strings of Length exactly one} = {a, b}

$\Sigma^2 = \Sigma . \Sigma$ = {set of all strings of Length exactly 2} = {aa, ab, ba, bb}

$\Sigma^3 = \Sigma \Sigma . \Sigma$ = {set of all strings of length exactly 3} = {aaa, aab, aba, abb, baa, bab, bba, bbb} = 8

Σ^0 = {set of all strings of length '0'} = $\{\epsilon\}$ Epsilon = Null string [$|\epsilon| = 0$]

$\Sigma^* = \{\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 ---\} = \{\{\epsilon\} \cup \{a,b\} \cup \{aa, ab, ba, bb\} ----\}$

= {set of possible strings over (a,b) of all lengths}

= { Universal set }

PRACTICAL SCENARIO

Now, consider the c language $\Sigma = \{a,b \cdots z, A,B \cdots Z, 0 \cdots 9, +, \cdots x, \cdots\}$

Now,

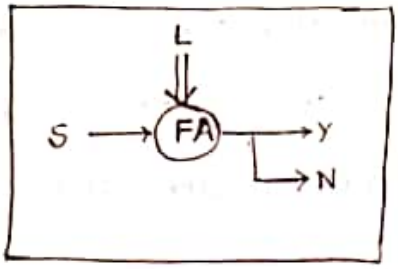
```
Void main()
{
    int a,b;
    :
}
```

} program in 'c' but in TOC it is a string

Now, C- programming language = {set of all "valid" programs}

$= \{P_1, P_2, P_3 \cdots\}$ (P_n)

Now, Given any string and to find out whether the string belongs to the Language is a heavy task if the Language is Infinite

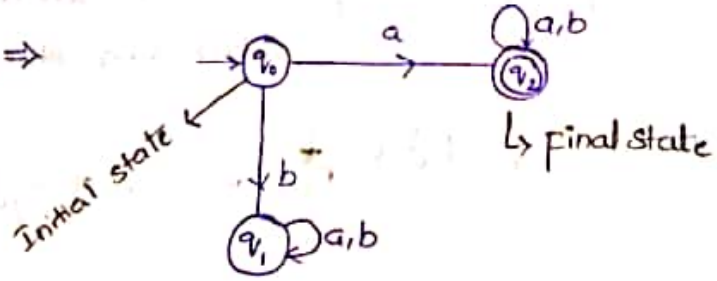


CONSTRUCTION OF FINITE AUTOMATA

① L_1 = {set of all strings which start with 'a'}

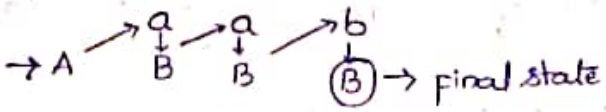
$= \{a, aa, ab, aaa \cdots\}$

⇒

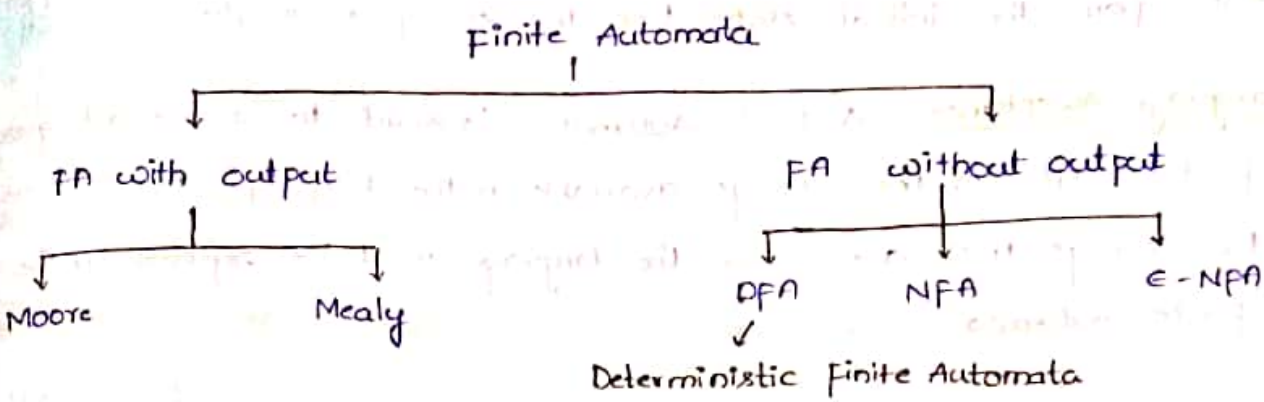


States = $\{q_0, q_1, q_2\}$

Now, Given string, = a a b

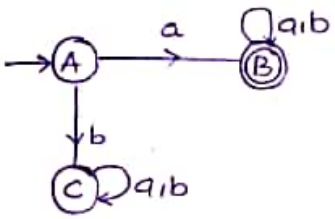


Classification of the Finite Automata.



DFA

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



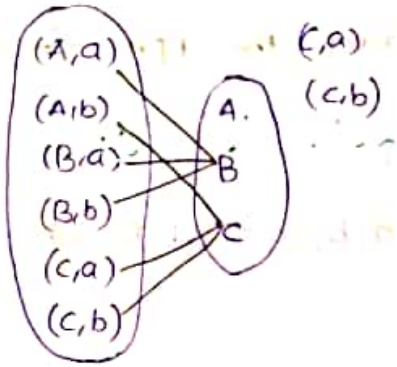
Q = {set of all states} = {A, B, C}

Σ = {set of Alphabets} = {a, b}

q_0 = {A}, f = {B}

Now, $\boxed{\delta = Q \times \Sigma \rightarrow Q}$

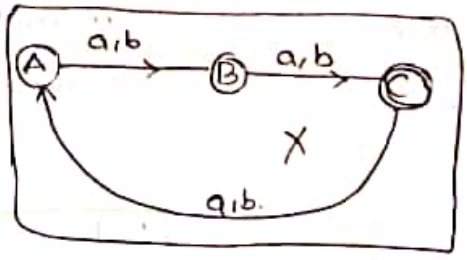
$\{A, B, C\} \times \{a, b\}$



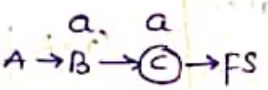
② Construct a DFA that accepts set of all strings over {a,b} of Length 2? $|w| = 2$.

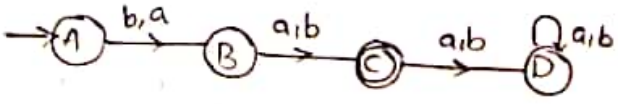
Sol $\Sigma = \{a, b\}$

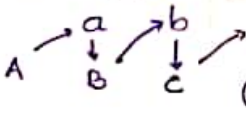
$L = \{aa, ab, ba, bb\}$



Given String







⇒ String Acceptance: Scan the entire string, if we reach a final state from the initial state then the string is accepted.

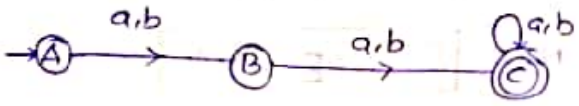
⇒ Language Acceptance: A finite Automata is said to accept a language if it accepts all the strings available in the language and similarly the strings that are not in the language must be rejected by the Finite Automata.

L-2 CONSTRUCTION OF MINIMAL DFA

construct a DFA for set of all strings over {a,b} such that length of the string is atleast 2? $|w| \geq 2$

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

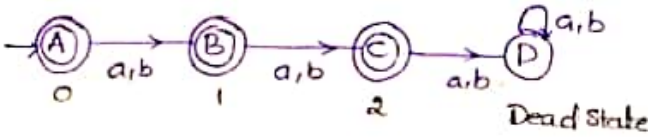
$L = \{aa, bb, aab, aaa, bbb, baa \ldots\}$



Now, construct the DFA for which $|w| \leq 2$

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

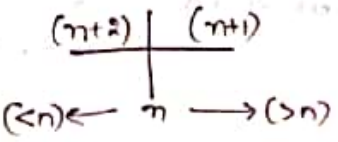
$L = \{\epsilon, a, b, aa, ab, ba, bb\}$



Now, from the above problems the conclusions that can be drawn are

$\|w\| = 2$	$\|w\| \geq 2$	$\|w\| \leq 2$
If $\|w\| = n$ then NO. of states $= (n+2)$	$\|w\| \geq n$ then no of states in DFA is $(n+1)$	$\|w\| \leq n$ the no. of states $= (n+2)$

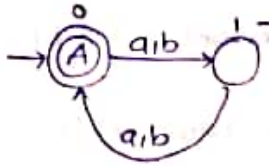
$\|w\| \leq n$	$(n+2)$ states
$\|w\| = n$	$(n+2)$ states
$\|w\| \geq n$	$(n+1)$ states



⑤ Construct the minimal DFA over {a,b} where $|w| \bmod 2 = 0$

The Input Alphabet set is $\Sigma = \{a,b\}$

L = {aa, ab, ba, bb} and 'ϵ'

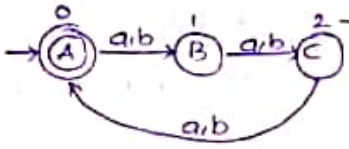


→ Remainders when divided by 2.

⑥ construct the minimal DFA over $w = \{a,b\}^*$ where $|w| \bmod 3 = 0 / |w| \cong 0 \bmod 3$

The Input Alphabet set is $\Sigma = \{a,b\}$

L = {aaa, bbb, abb, baa, aba, bab ...}



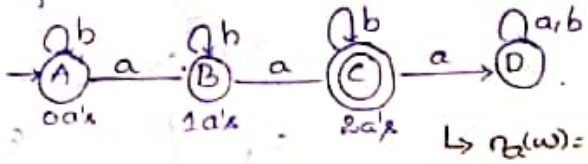
→ posible Remainders when a no is ÷ by 3

If $|w| \bmod n = 0$ then the DFA has 'n' states

⑦ CMDFA over {a,b} where no. of a's in $w = 2 \rightarrow \{n_a(w) = 2\}$

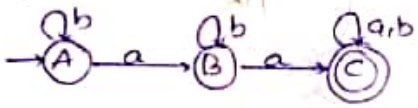
L = {aa, baa, aba, aab, aabbb ...}

The Input Alphabet set $\Sigma = \{a,b\}$



↳ $n_a(w) = 2$.

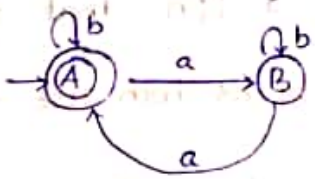
for $n_a(w) \geq 2$



⑧ CMDFA where $n_a(w) \bmod 2 = 0$ or $n_a(w) \cong 0 \bmod 2$.

L = {aa, aaaab, baa, aab, aba ...}

The Input Alphabet set is $\Sigma = \{a,b\}$



If $|w| \bmod n \cong K \bmod N$ then the DFA contain N states and each state represent the possible Remainders (0,1 -- N-1).

⑨ CMDFA over {a,b} where $n_a(w) \cong 0 \bmod 2$

$n_b(w) \cong 0 \bmod 2$

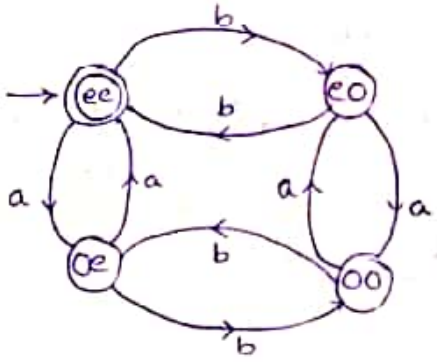
The Input Alphabet : {a,b}

L = {aa, bb, aabb, aaaabbbb, ... and ∈}

The entire Group of strings for the above language can be classified into 4 categories. They are

No of a's	No of b's
e	e → both the no of a's and no's of b's are even
e	o ⇒ even no of a's and odd no of b's
o	e ⇒ odd " " " " even " " "
o	o → " " " " odd " " "

∵ Therefore the DFA contains 4 states Representing each category



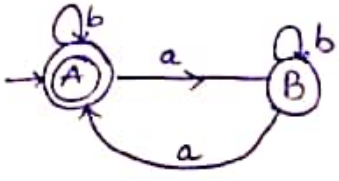
L-3 CROSS PRODUCT METHOD

⑩ construct a DFA that accepts strings over {a,b} where no of a's and no of b's both are even?

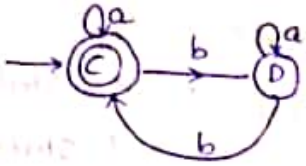
The Input Alphabet Σ = {a,b} w = (a,b)* $n_a(w) \cong 0 \bmod 2$

$n_b(w) \cong 0 \bmod 2$

Now, construct the DFA for counting a's

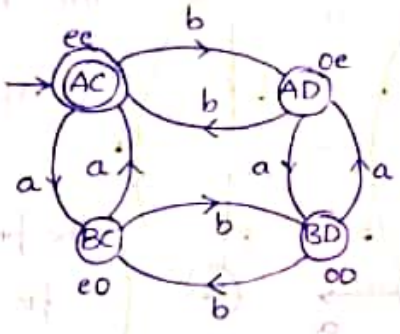


Similarly the DFA for counting the b's will be



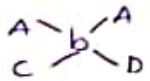
Now, the cross product of the two Automaton will be

$$\{A,B\} \times \{C,D\} = \{AC, AD, BC, BD\}$$



Now, $AC \xrightarrow{a} ? \Rightarrow$ 

$\Rightarrow AC \xrightarrow{a} \underline{BC}$

$AC \xrightarrow{b} ? \Rightarrow$ 

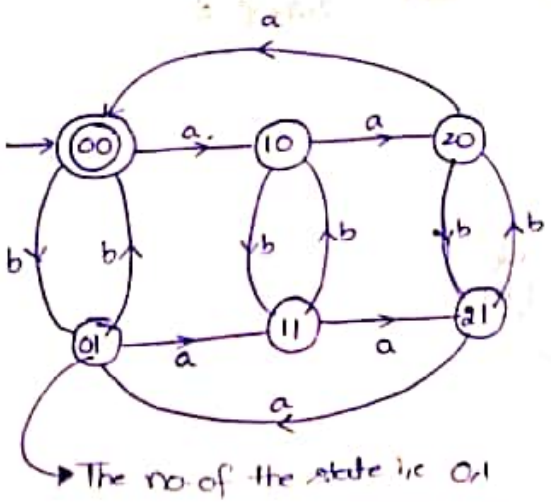
$\Rightarrow AC \xrightarrow{b} \underline{AD}$

If the 1st DFA has 'n' states and 2nd DFA has 'm' states then the cross product of DFA_1 and DFA_2 has $(n \times m)$ states.

L-4

(11) CMDFA over $\{a,b\}$ where $n_a(w) \cong 0 \bmod 3$ and $n_b(w) \cong 0 \bmod 2$.

The Input Alphabet $\Sigma = \{a,b\}$ $\omega = (a,b)^*$ $n_a(w) \cong 0 \bmod 3$ and $n_b(w) \cong 0 \bmod 2$



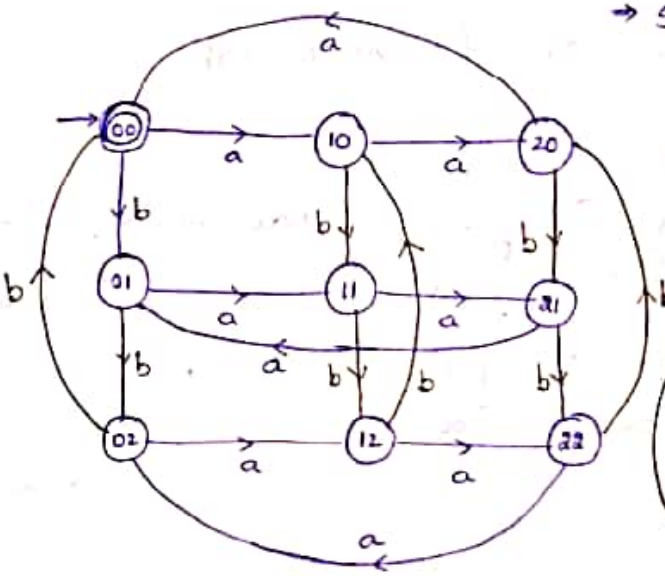
If $n_a(w) \bmod 3 \geq n_b(w) \bmod 2$ then final states are $\{10, 20, 21, 11\}$

If $n_a(w) \bmod 3 = n_b(w) \bmod 2$ $\{00, 11\}$ = final states.

The no of the state i.e 01 represent that to reach that state we should cross 0 a's and 1 b's.

(12) CMDFA over {a,b} where $n_a(w) \cong 0 \bmod 3$, $n_b(w) \cong 0 \bmod 3$ } 3×3 States = 9 States

The Input Alphabet set = {a,b}



→ If $n_a(w) \bmod 3 = 1$
$n_b(w) \bmod 3 = 2$
the final state is {12}

→ If $n_a(w) \bmod 3 >$ $n_b(w) \bmod 3$ the
the final state is/are
{10, 20, 21}
→ $(n_a(w) > n_b(w)) \bmod 3$

L-6

(13) CMDFA over {0,1} which when Interpreted as Binary number is ÷ by

The Input Alphabet set = {0,1}

1- Unary

2- Binary - 0,1

3- Ternary - 0,1,2

!

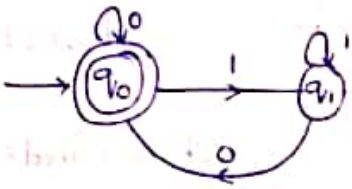
10- Decimal - 0,1--9

16- Hexadecimal - 0,1---15

Now, $(10)_2 = (1\times2)+0 = 2$

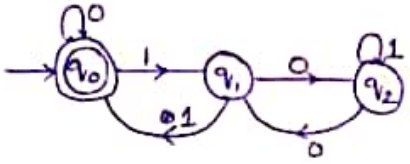
$(101)_2$ = $1\times2+0 = 2$
$2\times2+1 = 5$

Base
$1\times2+0=2$
$2\times2+1=5$



(14) CMDFA over {0,1} which interpreted as Binary number is ÷le by 3.

The Input Alphabet set = {0,1}



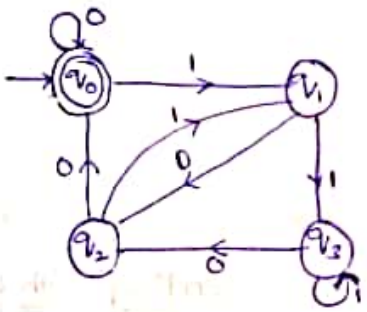
The Transition table is

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Short cut method

(15) CMDFA over {0,1} which when interpreted as Binary number is ÷le by 4

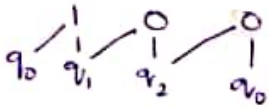
The Input Alphabet set = {0,1}, The Transition table is



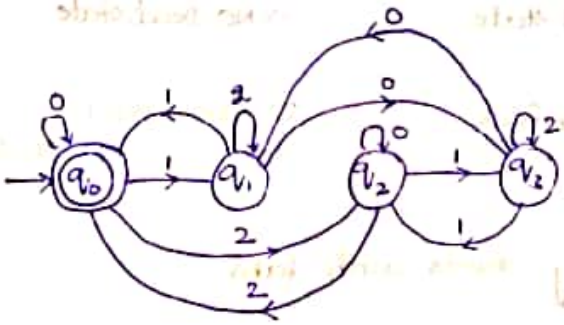
	0	1
*q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

⇒ Shortcut

100



Now if Σ = {0,1,2} (Ternary numbers) then, the Transition table is.



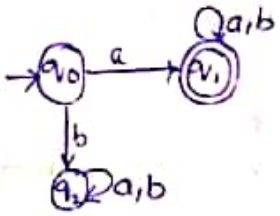
	0	1	2
*q_0	q_0	q_1	q_2
q_1	q_3	q_0	q_1
q_2	q_2	q_3	q_0
q_3	q_1	q_2	q_3

Short cut

L-9

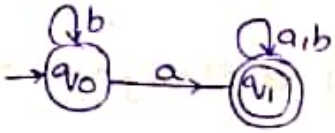
(16) CMDFA over $\Sigma = \{a, b\}$ and all the strings start with 'a'?

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



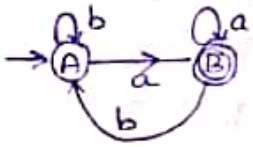
L-10

(17) CMDFA, $\Sigma = \{a, b\}$ where each string contains 'a'?



L-11

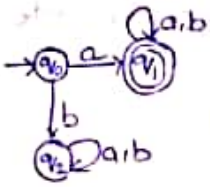
(18) CMDFA $\Sigma = \{a, b\}$ each string ending with 'a'?



L-12.

Now, comparing the above 3 DFA's we get

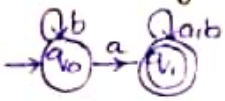
starts with 'a'



1> Dead state is present

2> No come Back

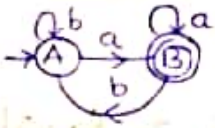
containing 'a'



1> NO Dead state

2> No come Back

ending with 'a'

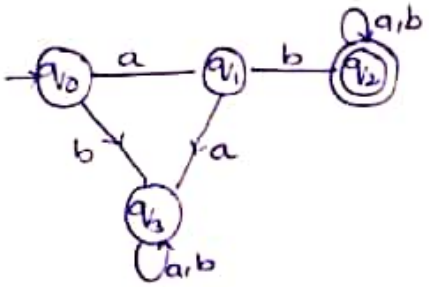


1> NO Dead state

2> Come Back

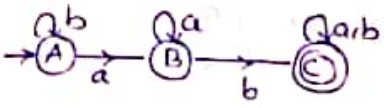
L-13

(19) CMDFA, $\Sigma = \{a, b\}$ and each string starts with $\{ab\}$



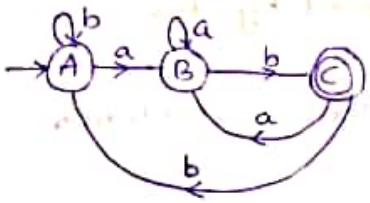
L-14

(20) CMDFA $\Sigma = \{a, b\}$, Each string containing {ab}



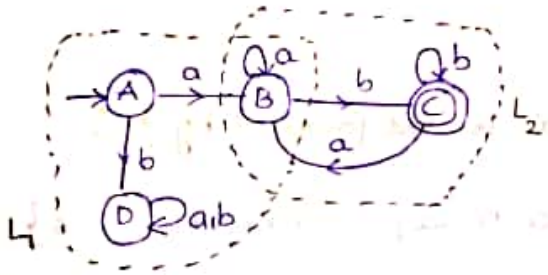
L-15

(21) CMDFA $\Sigma = \{a, b\}$ ending with {ab}

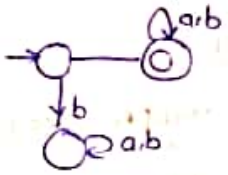


L-16

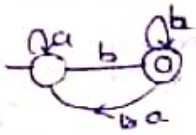
(22) CMDFA $\Sigma = \{a, b\}$ start with 'a' and end with 'b'?



L_1 = start with 'a' =

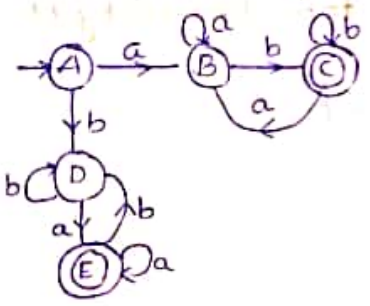


L_2 = end with b =



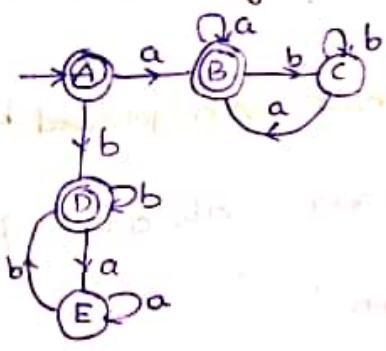
L-17

(23) CMDFA $\Sigma = \{a, b\}$, string start and end with different symbol?



L-18

(24) CMDFA $\Sigma = \{a, b\}$ string start and end with same symbol?



Two Languages are said to be compliment to each other when $L_2 = \Sigma^* - L_1$

↳ where L_1, L_2 are defined on same Σ!

COMPLEMENTATION OF DFA

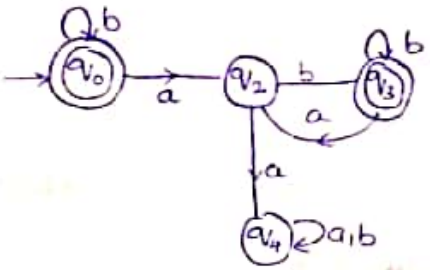
1) The complementation method is Applicable only for DFA

2) $(Q, \Sigma, \delta_0, q_0, f) \xrightarrow{comp} (Q, \Sigma, \delta, q_0, Q-F)$

L-20

(25) CMDFA over {a,b} where every 'a' is followed by 'b'?

Sol

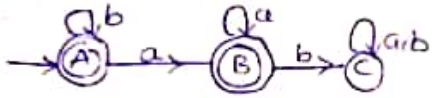


L = {ϵ, ab, abab, -- , b, bb, bbb -- }

L-21

(26) CMDFA over {a,b} where every 'a' is not followed by 'b'?

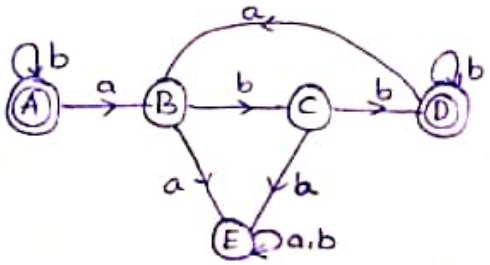
Language L = {a, aa, aaa, --- ba, baa, baaa, ... b, bb, bbb ---, ϵ}



L-22

(27) CMDFA over {a,b} where every 'a' has to be followed by 'bb'?

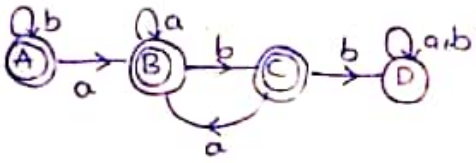
Language L = {ϵ, b, bbb --- abb, bbabb --- }



L-23

(28) CMDFA over {a,b} where every 'a' must not be followed by 'bb'?

Language L = {ϵ, b, bb, bbb --- , a, aa, aaa -- ab, aab -- }

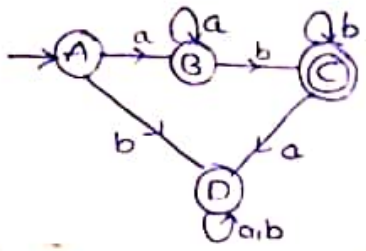


L-24

(29) CMDFA which accepts the Language $L = \{a^n b^m / n,m >= 1\}$

The Language $L = \{ab, aabb, aaabbb, aaaabbbb - - - -\}$

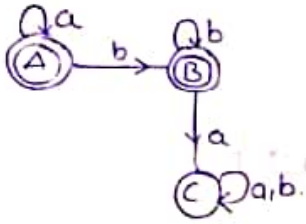
$abb, aabb, aaabb\}$



L-25

(30) CMDFA which accepts $L = \{a^n b^m / n,m >= 0\}$

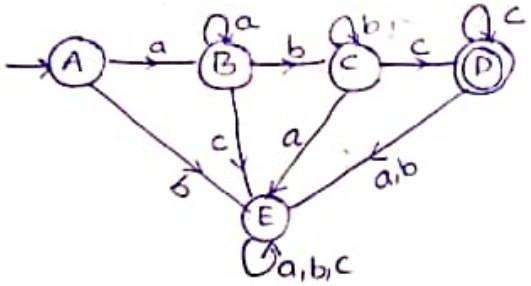
The Language $L = \{\epsilon, ab, aabb, aaabbb, abb, aa, a, aaa - -, b, bb, bbb - - -\}$



L-26

(31) CMDFA which accepts $L = \{a^n b^m c^l / n,m,l \geq 1\}$

The Language $L = \{abc, aabc, abbc, abcc - - -\}$

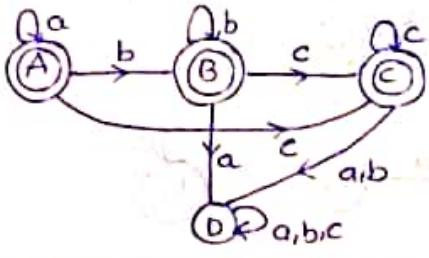


L-27

(32) CMDFA which accepts $L = \{a^n b^m c^l / n,m,l \geq 0\}$

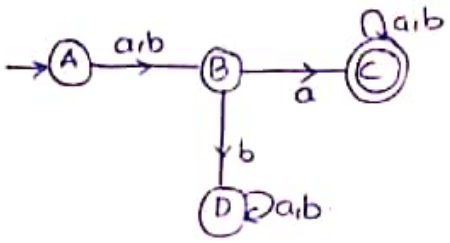
The Language $L = \{\epsilon, a, aa, aaa - - - b, bb, bbb - - - , c, cc, ccc - - - abc, aabbcc,$

$aabb, aacc - - -\}$



(33) CMDFA over {a,b} such that the 2nd symbol from the LHS is 'a'?

The Language L = {aaa, aa, ba, aab, ba----}



> If the nth symbol from the LHS is 'a' then the DFA contains (n+2) states

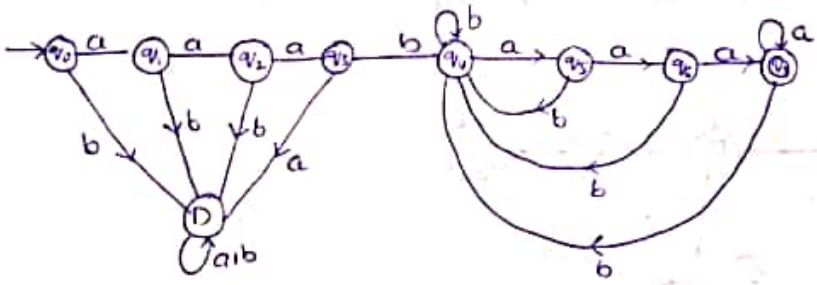
L-30

(34) CMDFA over {a,b} where strings are of the form a^3bwa^3 where w is any string over {a,b}

The Language $L = \{a^3bwa^3 / w \in (a,b)^*\}$

$L = \{a^3ba^3, a^3baa^3, a^3bba^3, a^3baaa^3 ---\}$

The min. length string = aaabaaa

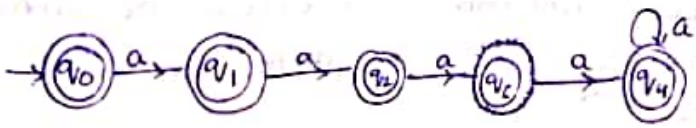


L-32

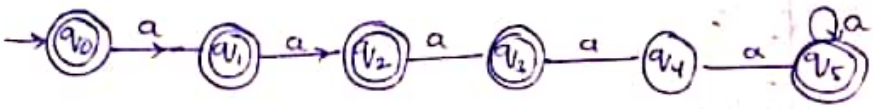
(35) CMDFA over {a} such that i) $\{a^n / n \geq 0, n \neq 3\}$

ii) $\{a^n / n >= 0, \quad , n \neq 4\}$

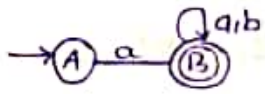
(1)

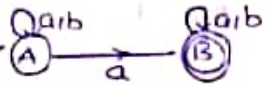


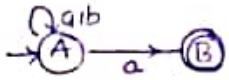
(2)

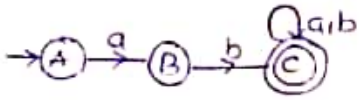


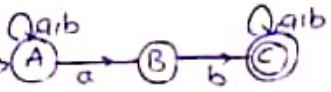
L-34 NFA

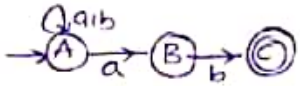
L_1 = {starts with 'a'} ⇒ 

L_2 = {containing a} ⇒ 

L_3 = {ends with a} ⇒ 

L_4 = {start with ab} ⇒ 

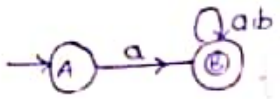
L_5 = {contains ab} ⇒ 

L_6 = {End with ab} ⇒ 

L-35 NFA - DFA CONVERTION

The powerfulness (Expressive power) of both NFA and DFA are same because every NFA can be converted to DFA. (NFA ≅ DFA)

L_1 = {starts with a} Σ = {a, b}

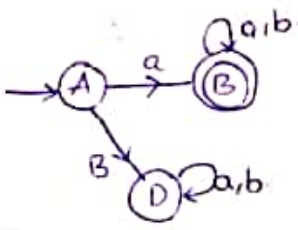
NFA for L_1 is 

The state Transition table for the above NFA is

	a	b
A	B	∅
B	B	B

The state Transition table for DFA will be

	a	b
→ A	B	D
*B	B	B
D	D	D

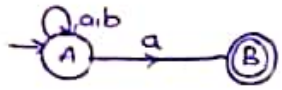


The transition function for NFA is

$$Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$$

Language L= {ending with a} $\Sigma = \{a, b\}$

NFA =



The TT will be

	a	b
A	{A,B}	{A}
B	{ϕ}	{ϕ}

DFA Transition table will be

	a	b
→ A	[AB]	[A]
[AB]	[AB]	[A]
[B]	D	D

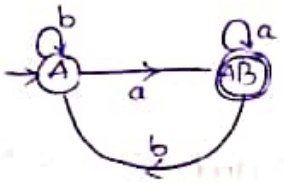
$[AB] \rightarrow a = (A \rightarrow a) \cup (B \rightarrow a)$

$= \{A, B\} \cup \{\phi\}$

$= \{A, B\}$

$[AB] \rightarrow b = (A \rightarrow b) \cup (B \rightarrow b)$

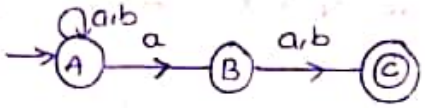
$= [A] \cup [\phi] \quad [A]$



L-37

L = {all strings where the 2nd symbol form RHS is a} $\Sigma = \{a, b\}$

NFA = L= {aa, ab, aaa, aab - - -}

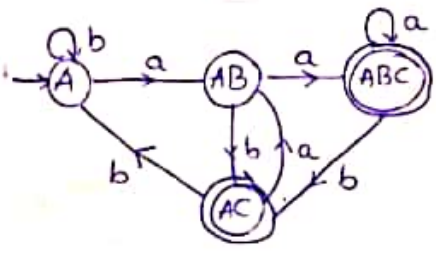


Transition table

	a	b
→A	{AB}	{A}
B	{C}	{C}
*C	ϕ	ϕ

The state Transition table for DFA will be

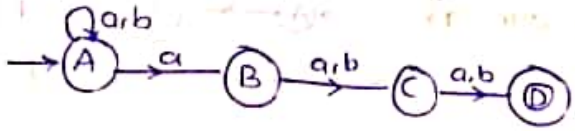
	a	b
→[A]	[AB]	[A]
[AB]	[ABC]	[AC]
* [ABC]	[ABC]	[AC]
* [AC]	[AB]	[A]
[B]	[C]	[C]



L-38

L = {All strings from which 3rd symbol from RHS is a}

NFA :



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

The transition table for DFA will be

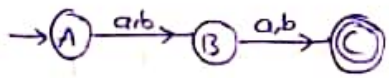
	a	b
→ [A]	[AB]	[A]
[AB]	[ABC] ✓	[AC] ✓
[ABC]	[ABCD] ✓	[ACD] ✓
[AC]	[ABD] ✓	[AD] ✓
* [ABCD]	[ABCD]	[ACD]
* [ACD]	[ABD]	[AD]
* [ABD]	[ABC]	[AC]
* [AD]	[AB]	[A]

Note : If the minimal NFA contains 'n' states then the ~~minimal~~ DFA contains '2^n' states (~~minimum~~ maximum)

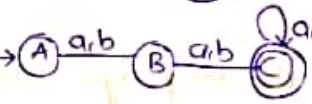
L-39

NFA for strings of length a. exactly 2 b> Almost 2 c> Atleast 2

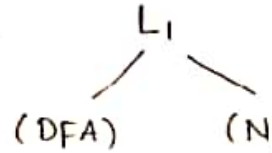
a> exactly 2 L = {aa, ab, ba, bb}



b> Atleast 2 L = {aaa, bbb, aa, bb, ba, bb···} ⇒



c> Almost 2 L = {ϵ, a, b, ab, ba, bb, aa} ⇒



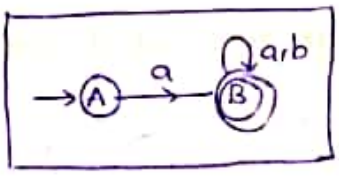
$n((DFA)) \geq n((NFA))$

[No of states in DFA] ≥ [No of states in NFA]

Note: The minimum no of states present in the finite Automata (NFA) which accepts set of all strings of length 'n' is 'n' (only for NFA)

L-40 COMPLEMENTATION OF NFA

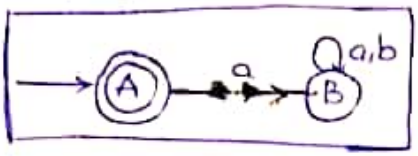
NFA over {a,b} L = { Starts with a }



= M

$L_1 = \{a, aa, ab\}$

Now, $\overline{M}$ =



$L_2 = \{\epsilon\}$

Σ^*

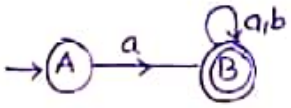
Start with 'a'	'ϵ' and start with 'b'
L_1	L_2

Note :

1> When we compliment the DFA the language will also get complimented

2> when we compliment the NFA the language may/ maynot get complimented.

Consider, the NFA



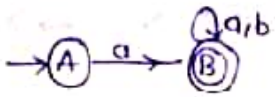
1> what is the compliment of language accepted by this NFA?

$L_1 = \{a, aa, ab, aaa \text{ - - -}\}$

$\overline{L_1} = \{\epsilon, b, bb, ba, bbb \text{ - -}\}$

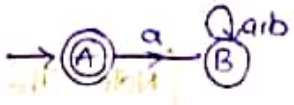
2) what is the language accepted by compliment of the NFA given above?

NFA M =



$L_1 = \{a, aa, ab, aaa \text{ - - -}\}$

$\overline{M}$ =



$L_2 = \{\epsilon\}$

L-41 MINIMISATION OF DFA

Two states 'p' and 'q' are Equivalent when

(p, q) are Equilant if → Input Alphabet

$\delta(p,w) \in F \Rightarrow \delta(q,w) \in F$

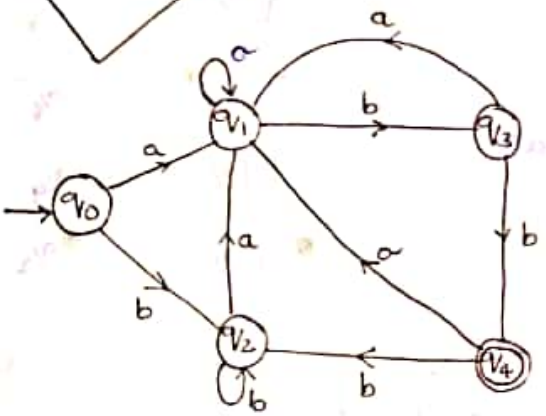
$\delta(p,w) \notin F \Rightarrow \delta(q,w) \notin F$

PARTITIONING METHOD

If $|w|=0 \Rightarrow$ '0' Equalent

$|w|=1 \Rightarrow$ '1' Equalent

$|w|=n \Rightarrow$ 'n' Equivalent



The Transition table is

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

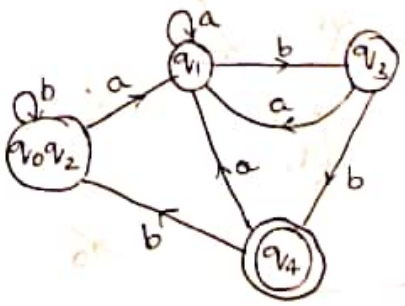
'0' Equivalent = $[q_0\ q_1\ q_2\ q_3]\ [q_4]$

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 = $[q_0\ q_2]\ [q_1]\ [q_3]\ [q_4]$

} Same, so stop Here



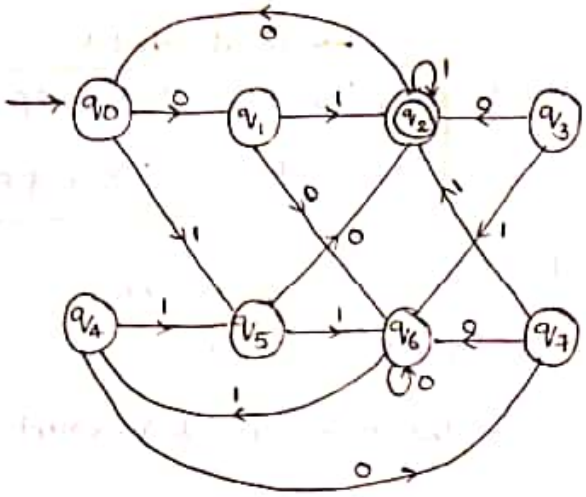
Note: The states that are not reachable from the finitial state must be Removed.

1> check whether they are in same group in previous Equalence

2> If they are not Equal, now check if they are going to same states or not

L-42

Minimise the following DFA



The Transition table will be

	0	1
→ q_0	q_1	q_5
q_1	q_6	*q_2
*q_2	q_0	*q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2*	q_6
q_6	q_6	q_4
q_7	q_6	*q_2

→ q_3 is not Reachable from Initial state

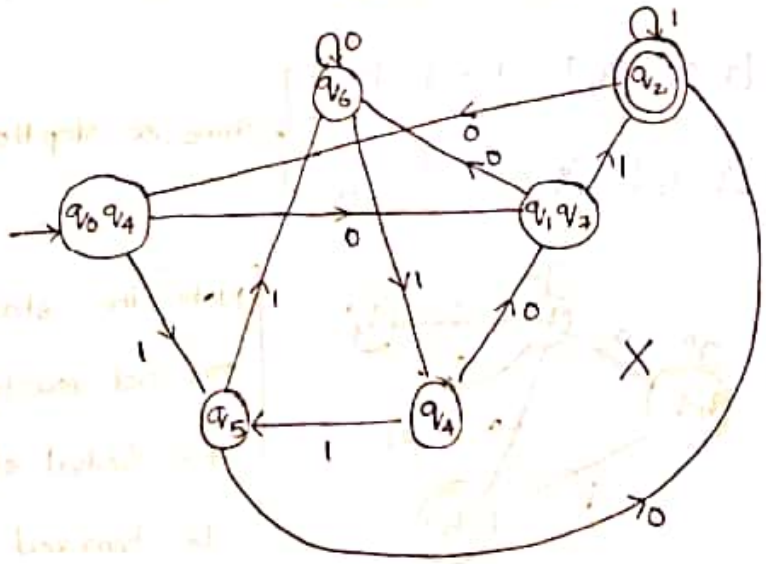
0 Equivalent states $[q_0\ q_1\ q_4\ q_5\ q_6\ q_7]$ $[q_2]$

1 Equivalent states $[q_0\ q_4\ q_6]$ $[q_1\ q_7]$ $[q_5]$ ~~$[q_4]$~~ $[q_2]$

2 Equivalent states $[q_0\ q_4]$ $[q_6]$ $[q_1\ q_7]$ $[q_5]$ ~~$[q_4]$~~ $[q_2]$

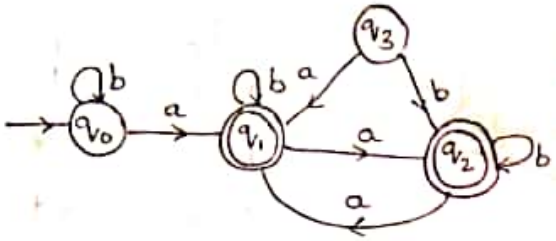
3 Equivalent states $[q_0\ q_4]$ $[q_6]$ $[q_1\ q_7]$ $[q_5]$ ~~$[q_4]$~~ $[q_2]$

} same so stop here



L-43 (GATE)

Minise the DFA

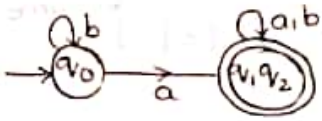


The Transition table is

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

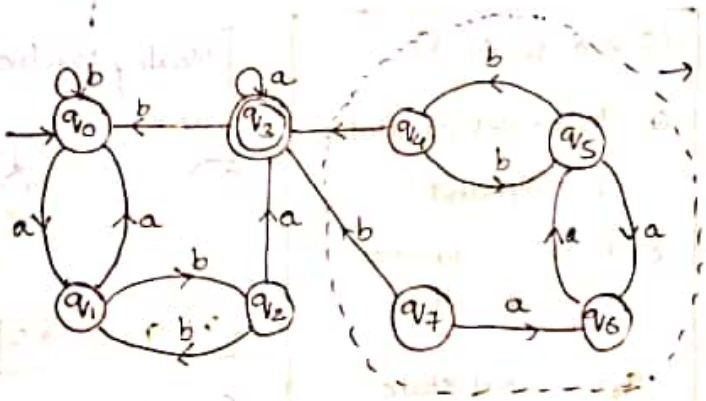
0 Equivalent states $[q_0]$ $[q_1\ q_2]$

1 Equivalent states $[q_0]$ $[q_1\ q_2]$ } same, stop here



L-44

Minimise the DFA



The Transition table is

	a	b
q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3*	q_1
* q_3	q_3*	q_0
q_4	q_3	q_5
X q_5	q_6	q_4
Not q_6	q_5	q_6
Reachable q_7	q_6	q_3

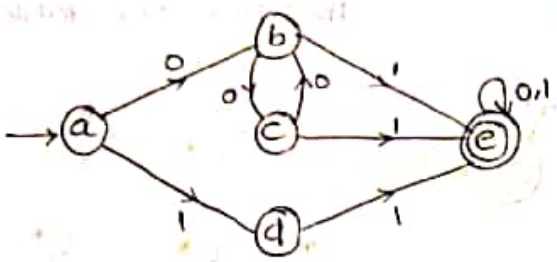
0 Equalent $[q_0\ q_1\ q_2]$ $[q_3]$

1 Equalent $[q_0\ q_1]$ $[q_2]$ $[q_3]$

2 Equalent $[q_0]$ $[q_1]$ $[q_2]$ $[q_3]$

L-45

Minimise the DFA



The Transition table is.

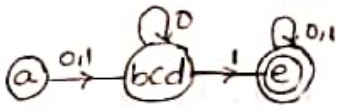
	0	1
→ a	b	d
b	c	e*
c	b	e*
d	c	e*
*e	e*	e*

0 EQuaLent states [a b c d] [e]

1 EQuaLent states [a] [b c d] [e]

2 EQuaLent states [a] [b c d] [e]

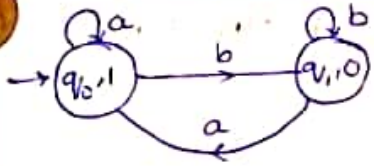
} same



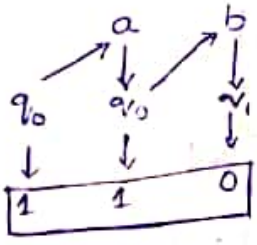
L-46 MOORE AND MEALEY MACHINES

Finite Automata with output

Moore Machine



$\lambda : Q \rightarrow \Delta$



If we give 'n' length Input then the output will be (n+1) length string

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

Q = Finite set of states

Σ = I/p Alphabet

δ = Transition function

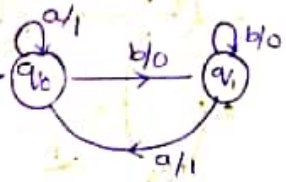
$Q \times \Sigma \rightarrow Q$

q_0 = Initial state

Δ = o/p Alphabet

λ = o/p function.

Mealey Machine



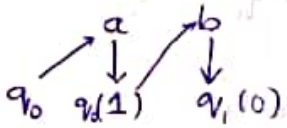
$\lambda : Q \times \Sigma \rightarrow Q$

$(q_0, a) \rightarrow 1$

$(q_0, b) \rightarrow 0$

$(q_0, a) \rightarrow 1$

$(q_1, b) \rightarrow 0$



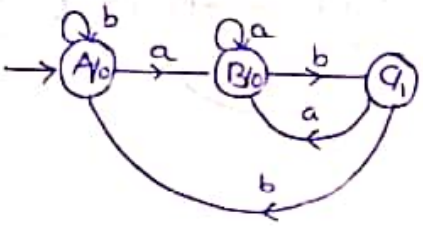
10

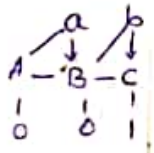
If we give 'n' length Input then output will be 'n' length String in Mealey Machine

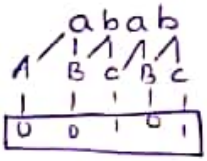
L-47

construct a moore machine that takes set of all strings over {a,b} as input and prints '1' as output for every occurance of 'ab' as substring.

$\Sigma = \{a,b\}$ $\Delta = \{0,1\}$ [∵ construct DFA ending with ab]



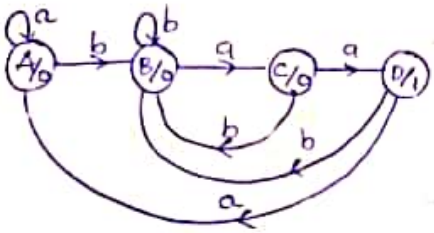


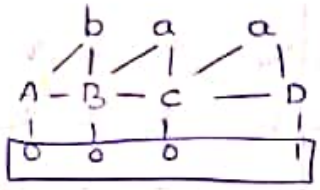


L-48

counting the occurrence of substring 'baa', construct Moore Machine?

$\Sigma = \{a,b\}$ $\Delta = \{0,1\}$ [∵ construct DFA ending with 'baa']



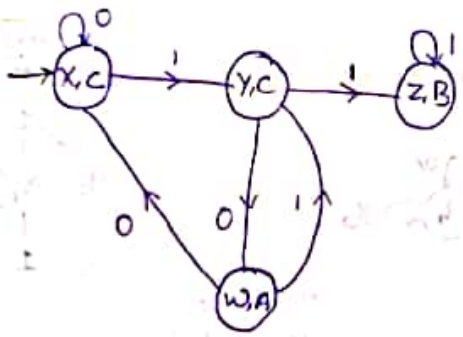


L-49

Construct a moore machine that takes set of all strings over {0,1} and produces 'A' as output if input ends with '10' or produces 'B' as output if i/p ends with '11' otherwise produces 'C'?

$\Sigma = \{0,1\}$ $\Delta = \{A,B,C\}$

10 - A
11 - B

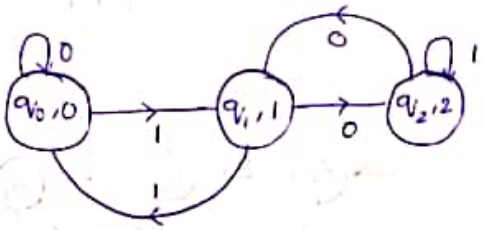


L-50

Moore Machine Residue modulo 'n'

construct a moore machine that takes binary no's as input and produces residue modulo '3' as output.

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



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

construct a moore machine that takes base 4 no's as input and produces residue modulo "5" as output?

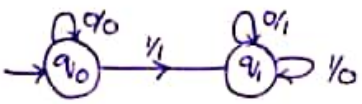
$\Sigma = \{0,1,2,3\}$ $\Delta = \{0,1,2,3,4\}$

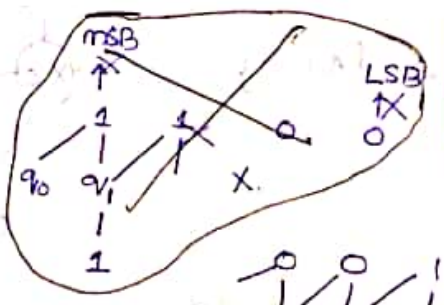
	0	1	2	3	Δ
q_0	q_0	q_1	q_2	q_3	0
q_1	q_4	q_0	q_1	q_2	1
q_2	q_3	q_4	q_0	q_1	2
q_3	q_2	q_3	q_4	q_0	3
q_4	q_1	q_2	q_3	q_4	4

L-51 MEALY MACHINE

construct a mealy machine that takes Binary number as i/p and produces 2's complement of that no as o/p. Assume the string is Read from Least Significant bit to most significant bit and end carry is discarded?

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





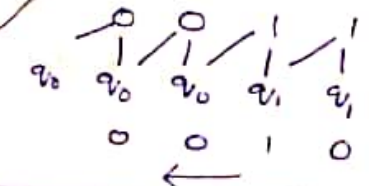
write same digits until 1st one and then compliment

(1100) ← 0100 same 1's comp + 1 = 2's comp

NO: 1100

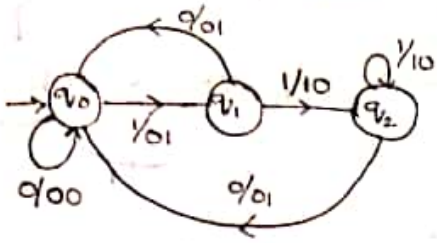
1's: 0011

2's = 0100 (1's +1)



L-52

What is the output produced by the following state machine (2.5)



a) 11 → 01

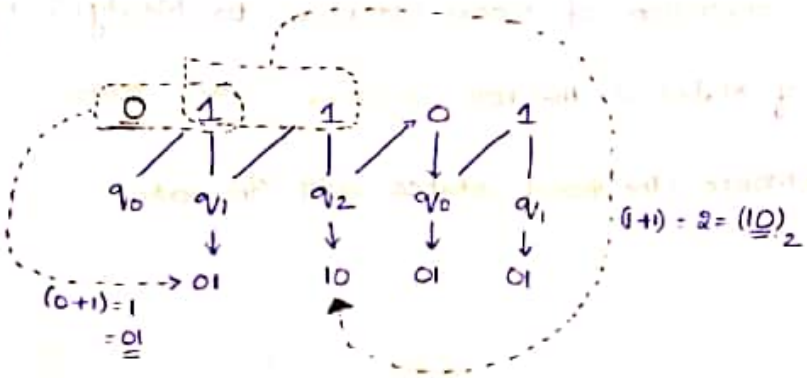
b) 10 → 00

c) Sum of present and previous bits

d) NOT

option a, b are false because if we give 11/10 as input the output must contain 4 bits. But they have given 2bits.

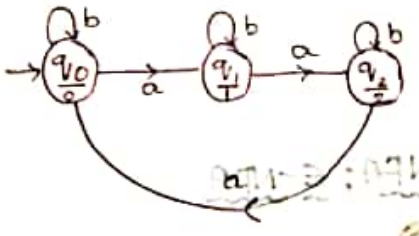
option c:



L-53 CONVERSION OF MOORE MACHINE TO MEALY MACHINE

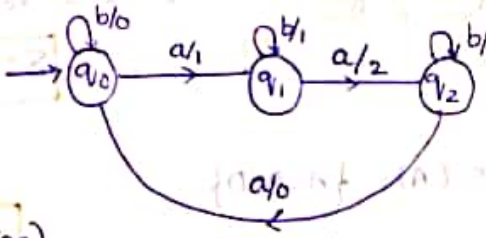
Moore Machine and Mealy Machine are equal in power

Convert the moore machine



⇒ Now, in the moore machine

(q_0, b) is going to q_0 and the output associated with q_0 is '0' ⇒ b/0

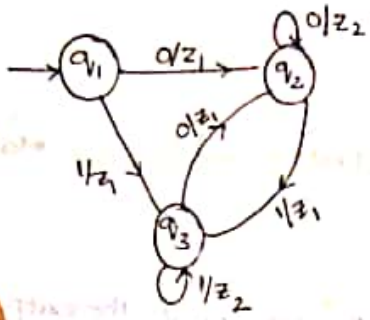


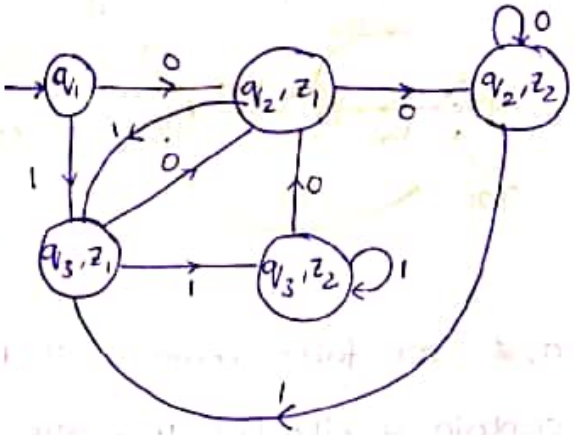
⇒ Now, q_0 on 'a' is going to q_1 and o/p associated at q_1 is 1 ∴ a/1 (in Mealy machine).

⇒ Mealy Machine

⇒ Similarly continue for other states too.

L-55 CONVERSION OF MEALY MACHINE TO MOORE MACHINE



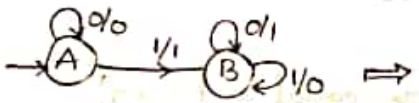


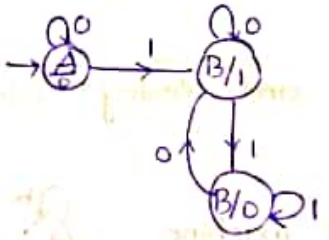
Note : 1. In the conversion of Moore Machine to Mealy Machine the No of states in the both machines are same

2. Mealy → Moore (The no of states will Increase)

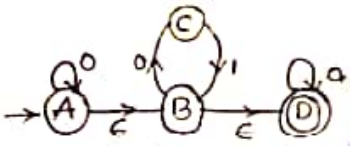
L-56

(2)





L-57 EPSILON NFA: ϵ-NFA



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

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

Now, ϵ-closure (A) = {A, B, D}

ϵ-closure (B) = {B, D}

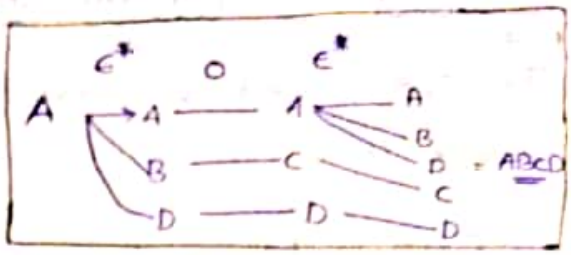
ϵ-closure (C) = {C}

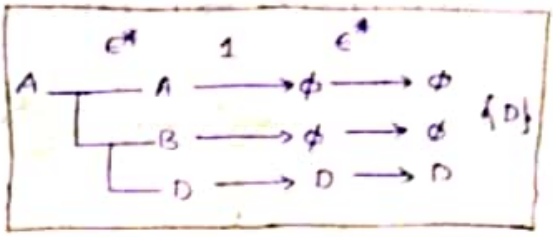
ϵ-closure (D) = {D}

ϵ-closure (A) = The states that are Reachable from 'A' on seeing Epsilon(ϵ)

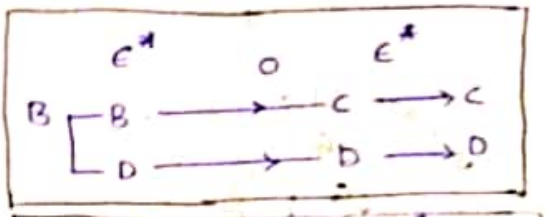
The state transition table

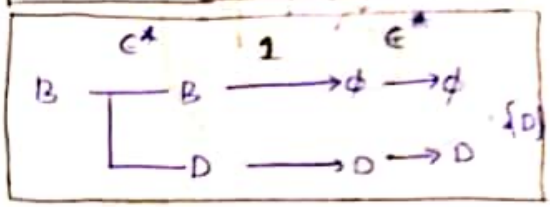
	0	1
→ A	{ABCD}	{D}
B	{CD}	{D}
C	{ϕ}	{B,D}
*D	{D}	{D}

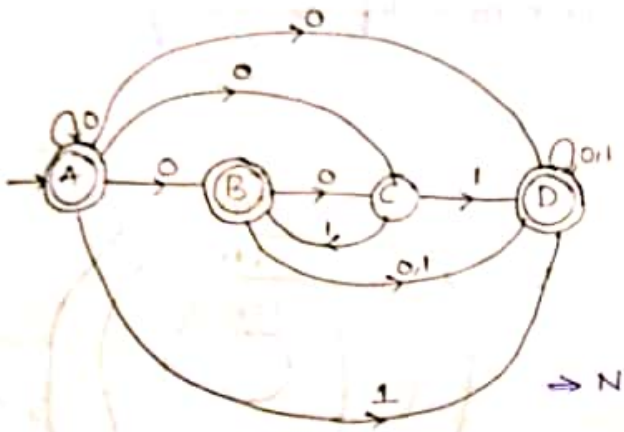




The No of states in the ϵ-NFA to NFA are Equal N(ϵ-NFA) = N(NFA)



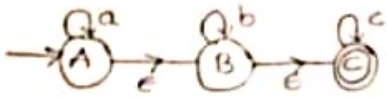




Conversion of ϵ-NFA to NFA

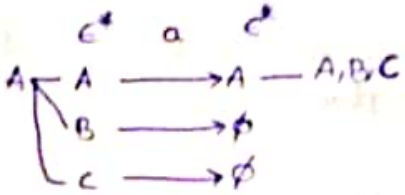
⇒ No of final states will increase from ϵ-NFA to NFA conversion. (The entry on which you can reach final states from epsilon transition are also final states)

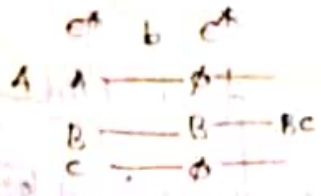
L-58



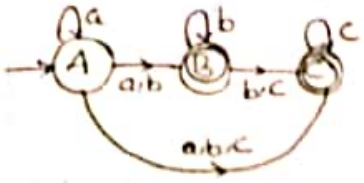
The state transition table is

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





(A→b) = {BC}

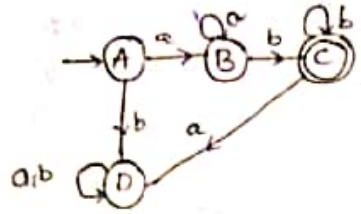


FAMILIES OF FORMAL LANGUAGES

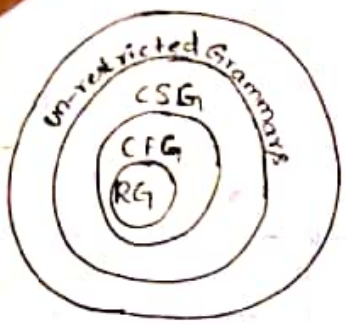
L-59

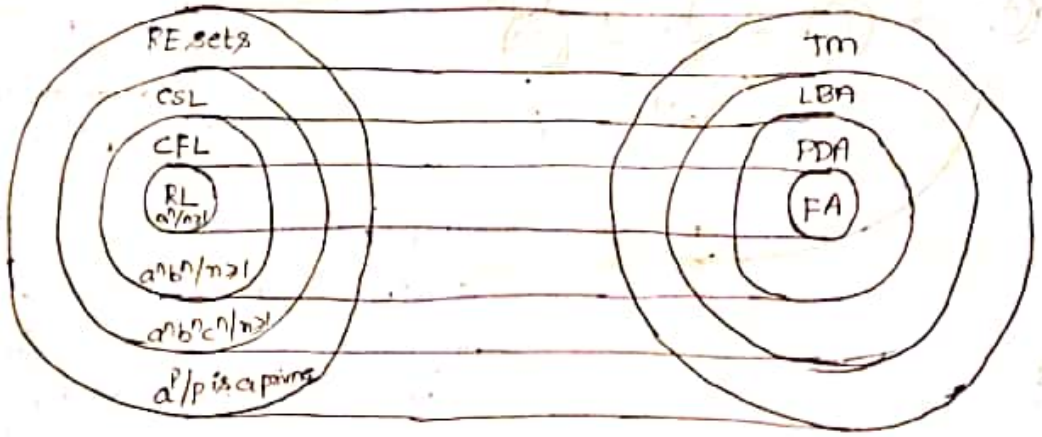
The DFA's cannot be constructed for all the Languages and one such example is

$L = \{a^n b^n / n \geq 1\} \Rightarrow L = \{ab, aabb, aaabbb - - - -\}$



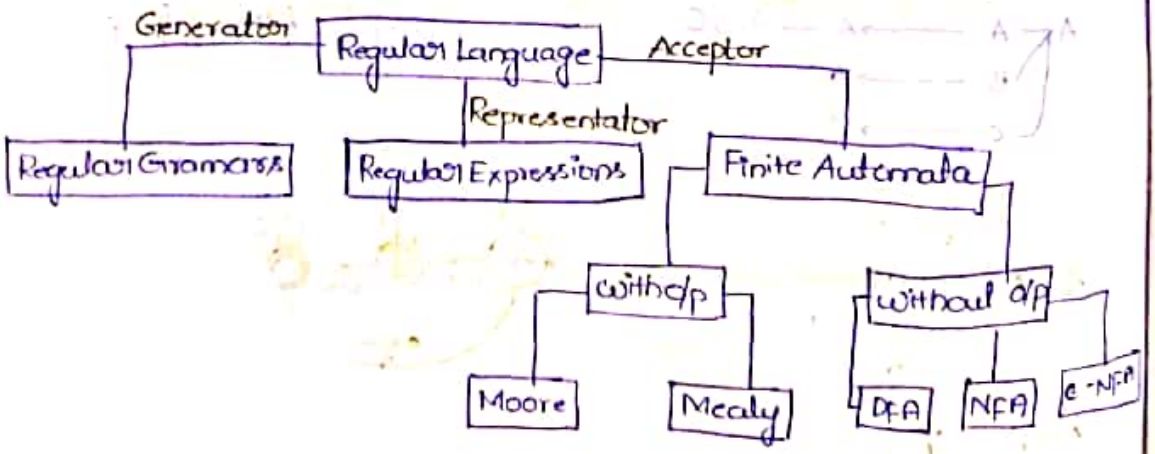
The Trap in this DFA is that the string "aab" will also be accepted, so we are unable to count the no's of 'a' occurred before 'b' so the memory required at every state will be Infinite. But FA cannot have Infinite memory





REGULAR EXPRESSIONS AND CONVERSIONS

L-60



Regular Expressions

The Regular Expression contains three operations

1) Union (+)

2) Concatenation (.)

3) Kleen closure (*)

a) $\phi = \{\}$, $\epsilon = \{\epsilon\}$, $a = \{a\}$ → primitive RE's.

(b) $r_1 + r_2$, $r_1 \cdot r_2$, r_1^*, r_2^* are also RE

Now, $a^* = \{\epsilon, a, aa, aaa \text{ ---} \}$

$a^+ = a\,a^*$ (or) $a^* . a = \{a, aa, aaa, aaaa \text{ --} \}$ (No Epsilon)

$(a+b)^* = \{\epsilon, a, b, aa, ab, ba, bb \text{ ---} \} = \{$set of all strings possible over $a,b\}$

L-61 $\Sigma = \{a, b\}$

① $L_1 = \{$set of all strings whose Length is exactly two$\}$

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

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

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

$= \{(a+b)(a+b)\}$ //.

② $L_1 = \{$set of all strings whose Length is atleast two$\}$

$= \{aa, ab, ba, bb, aaa, bbbb \text{ ---} \}$

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

$(a+b)(a+b)$ → All two Length strings; $(a+b)^*$ → All the strings over $\{a,b\}$

③ L_1 = {set of all strings whose Length is atmost 2}

$L_1 = \{\epsilon, a, b, ab, ba, bb, aa\}$

$L_1 = \{\epsilon + a + b + ab + ba + bb + aa\}$

$L_1 = \{(a+b+\epsilon)(a+b+\epsilon)\}$ //

④ L_1 = {even Length strings}

$= \{\epsilon, aa, ab, ba, bb \text{ --- }\}$

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

↳ Even Length strings repeated any no of times gives strings of even Length.

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

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

$= (a+b)^{2*}$ = even.

⑤ L_1 = {odd Length strings}

$= \{\epsilon, aaa, a, b, aab, baa, bbb \text{ --- }\}$

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

Analysis: $(a+b)^{2n+1}$

$= (a+b)^{2n}(a+b)$

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

⑥ L_1 = {Length is divisible by 3}

$L = \{$'0' length strings, 3 Ls, 6 Ls, 9 Ls $\text{ --- }\}$

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

Analysis: $(a+b)^{3n+2}$

$= (a+b)^{3n}(a+b)^2$

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

⑦ L = {Length $\cong$ 2 mod 3}

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

⑧ L = {exactly 2 a's} {atleast 2} {atmost 2}

$= \{b^*a\,b^*\,a\,b^*\}$ $\{b^*a\,b^*\,a(a+b)^*\}$ $\{b^*(\epsilon+a)\,b^*\,(\epsilon+a)\,b^*\}$

⑨ L: {a's are even} = {aa, aab, bbb, bbb ---}

$= (b^*ab^*ab^*)^* + (b^*)$ → The Language/RE with not generate the strings {b, bb, bbb ---} so we add 'b*' to generate the strings

⑩ L: {starts with 'a'} {Ends with a} {contains a}

- $\{a(a+b)^*\}$ $\{(a+b)^*a\}$ $\{(a+b)^*a(a+b)^*\}$

⑪ L: {start and end with diff symbols} {start and end with same symbols}

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

$\{b(a+b)^*a\}$

L: {ϵ, a, b, aa, bbb, aaa, aba ---}

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

L-62

⑫ L: {no 2a's should come together}

= {ϵ, b, bb, bbb --- a, ab, aba, abab, ababa, ba, bab, baba, babab, ----} = $\underbrace{(b+ab)^*}_{\text{set of all strings ending with 'b'}} + (b+ab)^*a$ → {set of all strings ending with a}

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

⑬ L: {NO 2a's and no 2b's come together}

= {ϵ, a, b, ab, ba, aba, bab ---}

Starts with	Ends with
{a, aba, abaaba ---} - a	a — $(ab)^*a$ (or) $a(ba)^*$
{ab, abab, ababab - a	b — $(ab)^*$ (or) $a(ba)^*b$
{ba, baba ----} - b	a — $(ba)^*$ (or) $b(ab)^*a$
{b, bab, babab ---} b	b — $(ba)^*b$ (or) $b(ab)^*$

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

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

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

Identities of Regular Expressions V.V.V. Imp.

① $\phi + R = R + \phi = R$

② $\phi \cdot R = R\phi = \phi$

③ $\epsilon R = R\epsilon = R$

④ $\epsilon^* = \epsilon$

⑤ $\phi^* = \epsilon$

⑥ $\epsilon + RR^* = RR^* + \epsilon = R^*$

⑦ $(a+b)^* = (a^* + b^*)^* = (a^* b^*)^*$

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

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

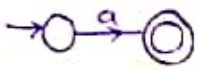
L-63

Regular Expression to finite Automata

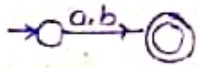
STATE CREATION METHOD

1> ϕ :

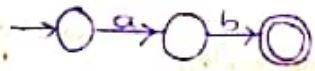
2> a :



3> a+b :

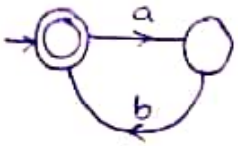


4> ab :

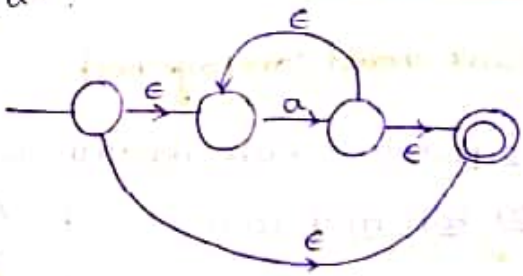


① ab^* :

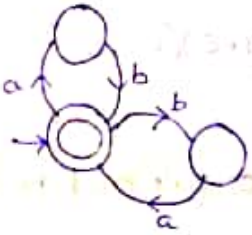
② $(ab)^*$:



⑤ a^* :



③ $(ab+ba)^*$:

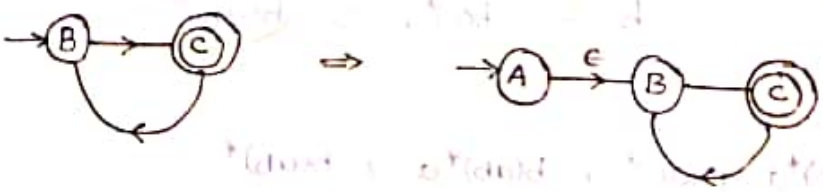


Finite Automata to Regular Expression conversion

STATE ELIMINATION METHOD

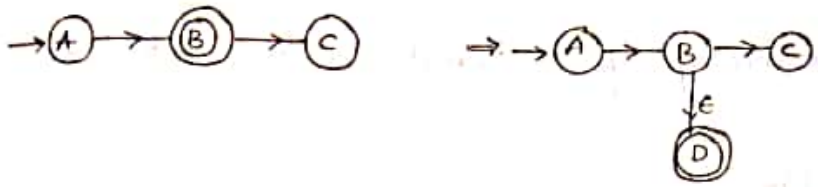
Rule 1

The initial stage should not have any incoming edge



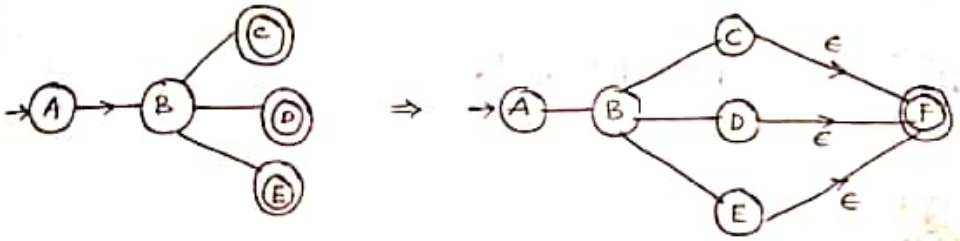
Rule: 2

The final state should not have any outgoing edge and there should be only one final state

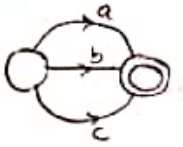


Rule -3

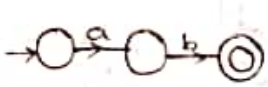
There must be only one final state and now eliminate the other states except the Initial and final states in any order you wish.



Examples

① 

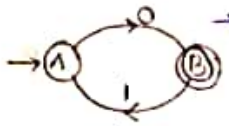
⇒(a+b+c)

② 

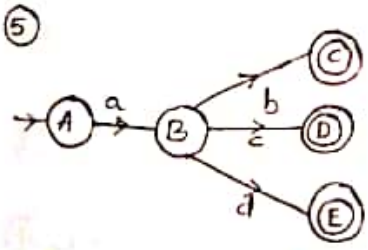
⇒ ab

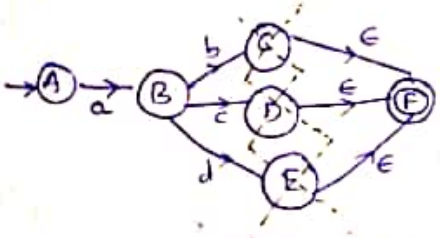
③ 

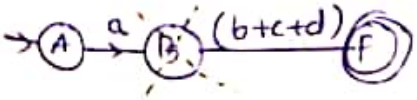
ab*c

④ 

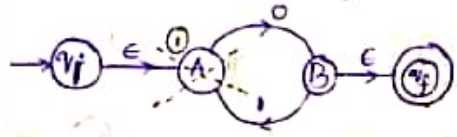
→ Here the Initial state is having incoming edge and final state has outgoing edge

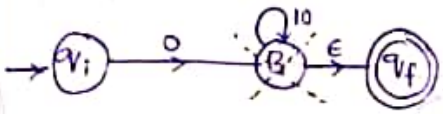
⑤ 

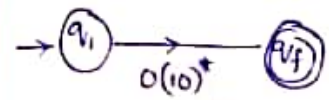




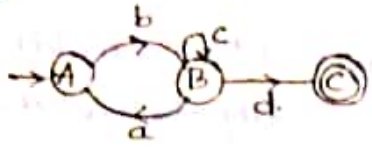
= a(b+c+d)



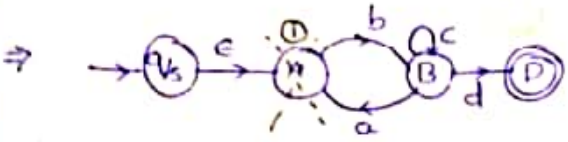


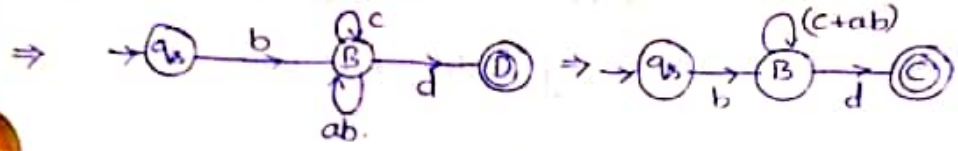


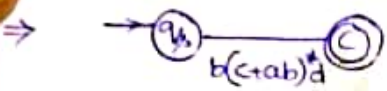
(6)



The initial state has incomming edge so create a new initial state

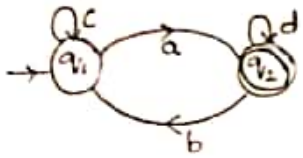




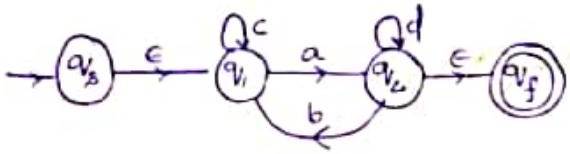


$\therefore$ RE = $b(c+ab)^*d$

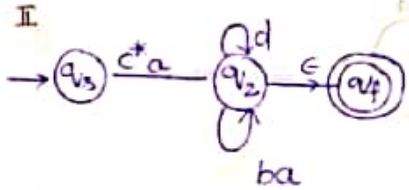
(7)



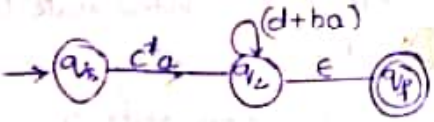
I



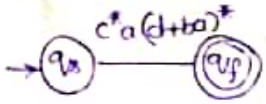
II



III

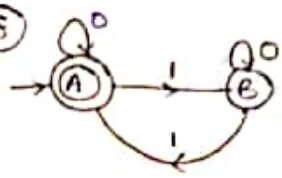


IV

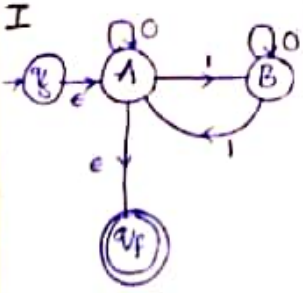


$\therefore$ RE = $c^*a(d+ba)^*$

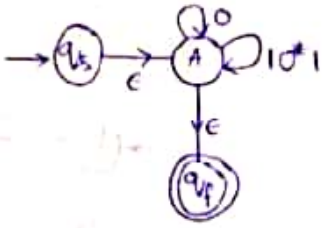
(8)



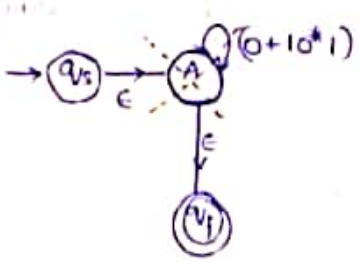
I



II



III



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

TESTING WHETHER A "LANG" IS "REGULAR" OR NOT

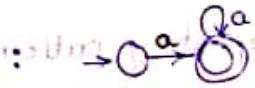
L-64

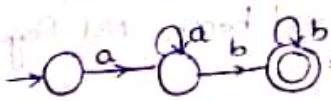
If the Language is finite then the language is Regular

⇒ Infinite Language may or maynot be Regular. And the Algorithm that is used to find whether the given language is Regular or not is called "PUMPING LEMMA"

⇒ Pumping Lemma states that if an Infinite language has to accepted by the Finite Automata then there has to be a loop inside the finite Automata, and we should find a repeating pattern in the Language which when kept on the loop generates all the remaining strings in the Language, If we do not find any such pattern then NO DFA can be constructed for that Language and the language is not Regular.

⇒ pumping Lemma is the Negative test which means pumping Lemma will say that a language is not Regular if we are not able to find such a pattern but pumping lemma doesnt say whether the language is Regular if you find such a pattern.

① $a^n / n \geq 1$:  ⇒ DFA possible = Regular Language

② $a^n b^m / n, m \geq 1$:  = Regular Language

③ $a^n b^n / n \leq 10^{10^{10}}$: The Language is finite (∵ value of n is bounded) so the Language is Regular

④ $a^n b^n / n \geq 1$: Not Regular because we need to count the Infinite occurrences of 'a's before 'b' but FA has finite memory so the language is not Regular.

⑤ $ww^R / |w| = 2, w \in (a,b)$: $\{aaaa, abba, baab, bbbb\}$ - Finite Language ⇒ Regular language

⑥ $ww^R / w \in \{a,b\}^*$ = {set of all palindromes} = The language is not Regular because the string 'w' can be of any length and that must be compared Against w^R and the word/string 'w' can be infinite Length, Hence FA has no capacity to store Infinite Length strings} ⇒ Not- Regular Language

⑦ $a^n b^m c^k / n,m,k \geq 1$ = Regular Language [No comparisions, a,b,c generated indepntly ↳ $aa^* bb^* cc^*$

⑧ $a^i b^{2j} / i,j \geq 1$ = Regular Language = $aa^* bb(bb)^*$

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

⑨ a^n / n is even = $\{a^0, a^2, a^4, a^6, a^8 \ldots\}$ = 0 2 4 6 8 = Ap ⇒ strings are in 'Ap' and the common difference can be kept in loop of DFA

a^n / n is odd = $\{a^1, a^3, a^5, a^7 \ldots\}$ = 1 3 5 7 = Regular Language

⑩ a^n / n is prime = $\{a^1, a^3, a^5, a^7, a^{11} \ldots\}$ = {Not in Ap ⇒ No Repeating pattern}

$a^{n^2} / n \geq 1$ = $\{a, a^4, a^9, a^{16}, a^{25} \ldots\}$ = {Not in Ap ⇒ Not RL}

$a^{2^n} / n \geq 1$ = $\{a^2, a^4, a^8, a^{16} \ldots\}$ = {Not in Ap ⇒ Not RL}

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

⑪ $a^i b^{j^2} / i,j \geq 1$ = Even the a,b are generated independently, due to j^2 there is not Repeating pattern in the Language and hence Not Regular

⑫ $a^i b^{2^n} / i,n \geq 1$ = a^i can be independently generated and '2^n' (2 to the power n) has no Repeating pattern.

⑬ $a^i b^p / i \geq 1$ = b^p cannot be generated and No Repeating pattern.

⑭ $\{w / n_a(w) = n_b(w)$

$n_a(w) \leq n_b(w)$

$n_a(w) \geq n_b(w)\}$ = Not Regular Language (∵ counting a's need Infinite Memory}

(15) $n_a(\omega) \bmod 3 \leq n_b(\omega) \bmod 3$ ⇒ Regular Language (DFA can be constructed for mod 3 counters)

(16) $a^n / n \geq 10$ ⇒ Regular Language (DFA can be constructed)

(17) $\omega w \omega^R / \omega \in (a,b)^*$ = Not Regular language

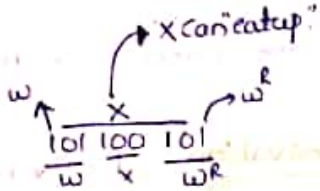
(18) $a^n b^n c^n / n \geq 1$ = Not Regular Language

(19) $a^n b^{n+m} c^m / n, m \geq 1$ = Not Regular Language

Imp VV Imp

(20) $\omega x \omega^R / \omega, x \in (0,1)^+$: Regular Language

$\omega = 101$
$x = 100$
$\omega^R = 101$



RE = $1(0+1)^*1 + 0(0+1)^*0$

(21) $\omega x \omega^R / \omega \in (0,1)^+$: Not Regular Language ('∵' x cannot go beyond '5' length)
$|x| = 5$

(22) $x \omega \omega^R y / x, y, \omega \in (0,1)^+$:

$x = 10$
$y = 11$
$\omega = 100$
$\omega^R = 001$

x	ω	ω^R	y
10	100	001	11

∴ Regular Language

RE: $(0+1)^* 00 (0+1)^*$
(or)
$(0+1)^* 11 (0+1)^*$

(23) $x \omega \omega^R / x, \omega \in (0,1)^+$: Not Regular Language

(24) $\omega \omega^R y / \omega, y \in (0,1)^+$: Not Regular Language

(25) Set of all strings having Equal number of '01' and '10' is Regular.

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

GRAMMARS

L-65

A Grammar is generally represented by (V, T, P, S)

- V = Vertices
- T = Terminals
- P = productions
- S = start symbol

$S \to aSB$
$S \to aB$
$B \to b$

$V = \{S, B\}$
$T = \{a, b\}$
$P = \{S \to aSB,\ S \to aB,\ B \to b\}$
$S = \{S\}$

⇒ The main intention of the grammar is to generate all the strings that are present in the language

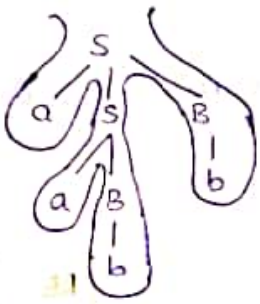
Derivation: Deriving the string from the grammar using the start symbol

$S \to aSB$
$S \to aB$
$B \to b$

Derivation of aabb

$S \to \underline{a}S\underline{B}$ → Sentential forms (or) Sequential forms.
$\Rightarrow aa\underline{B}\underline{B}$
$\Rightarrow aab\underline{B}$
$\Rightarrow aabb$

Left most derivation



CONSTRUCTION OF GRAMMARS

① $L = \{aa, ab, ba, bb\}$ $S \to aa/ab/ba/bb$

$(a+b)\cdot(a+b)$
A A

$S \to AA$
$A \to a/b$

② $L = \{a^n / n \geq 1\}$ $L = \{a, aa, aaa, aaaa \ldots\}$ $A \to aA/\epsilon$ (or) $S \to A$, $A \to aA/a$

③ $L = (a+b)^*$ $L = \{\epsilon, a, b, ab, ba, aa, bb \ldots\}$ $S \to aS/bS/\epsilon$

④ L = {set of all strings of length atleast 2}

= (a+b)(a+b)(a+b)*

S → AAB
A → a/b
B → aB/bB/ϵ

⑤ L = {Length atmost 2}

= (a+b+ϵ)(a+b+ϵ)
  A A

S → AA
A → a/b/ϵ

⑥ L = {start with 'a' end with 'b'}

= a(a+b)*b

S → aAb
A → aA/bA/ϵ

⑦ L = {start and end with diff symbols}

= a(a+b)*b + b(a+b)*a

S → aAb/bAa
A → aA/bA/ϵ

L = {same symbols}

S → aAa/bAb/a/b/ϵ
A → aA/bA/ϵ

⑧ L = {$a^n b^n / n \geq 1$}

S → aSb/ab

⑨ L = {ww^R} ∪ {waw^R} ∪ {wbw^R}

w ∈ (a+b)*

S → aSa/bSb/a/b/ϵ

⑩ L = {set of all strings of even length}

((a+b)(a+b))*
  A   A

S → BS/ϵ
B → AA
A → a/b

⑪ L = {$a^n b^m / n, m \geq 1$}

S → AB
A → aA/a
B → aB/b

L = {$a^n b^m c^m / n, m \geq 1$}

S → AB
A → aAb/ab
B → cB/c

⑫ L = {$a^n c^m b^n$} n, m ≥ 1

S → aSb/aAb
A → cA/c

PUSH DOWN AUTOMATA

L-68

Push Down Automata = Finite Automata + Stack

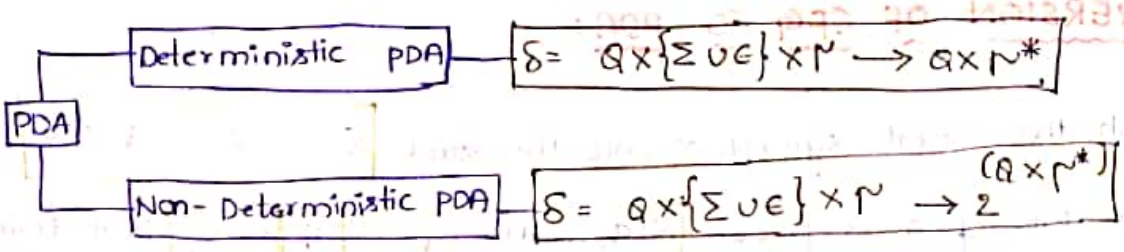
Push Down Automata = $(Q, \Sigma, \delta, q_0, Z_0, F, \Gamma)$

Q = finite set of states — Z_0 = Bottom of the Stack

Σ - Input symbol — F = set of final states

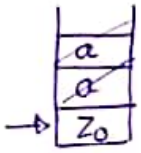
δ = Transition function — Γ = Stack Alphabet

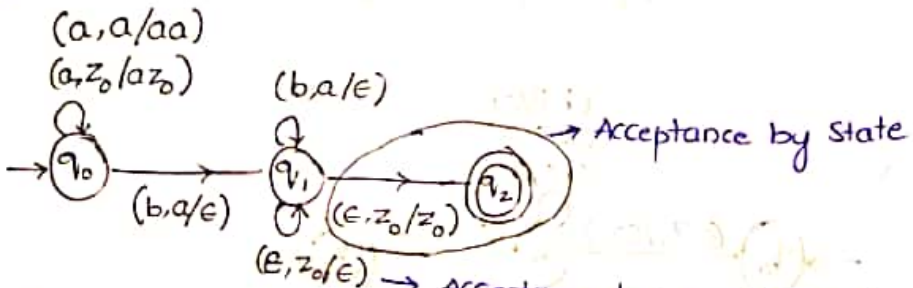
q_0 = Initial state



① $L = \{a^n b^n / n \geq 1\}$

Let the string be aabb





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

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

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

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

$\delta(q_1, \epsilon, Z_0) =$ (q_f, Z_0) (OR) (q_1, ϵ)

Acceptance by state — Acceptance by Empty stack

(13) $L = \{a^n b^n c^m d^m / n,m \geq 1\}$

$S \to AB$
$A \to aAb/ab$
$B \to cBd/cd$

(14) $L\{a^n b^n c^n \ n \geq 1\}$ $\quad S \to aSbc/abc$

$aabcbc$
$= a^n(bc)^n$ X

For this Language we cannot give such grammar & we give CSG for this language

(15) $L = \{a^n b^{2n} / n \geq 1\}$ $\quad S \to aSbb/abb$

(16) $L = \{a^n b^m c^m d^n / n,m \geq 1\}$

$S \to aSd/aAd$
$A \to bAc/bc$

(17) $L = \{a^n b^m c^n d^m / n,m \geq 1\} \Rightarrow$ Not possible to give CFG.

(18) $L = \{a^n a^m b^m c^n\}$

A

$S \to aSc/aAc$
$A \to aAb/ab$

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

$S \to AB$
$A \to aAb/ab$
$B \to bBc/bc$

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

$= \{a^n b^m c^m c^n / n,m \geq 1\}$

$S \to aSc/aAc$
$A \to bAc/bc$

L-66

CLASSIFICATION OF GRAMMARS

According to chomsky the Grammars are classified into 4 types.

1> Type '3'/Regular Grammars

2> Type '2'

3> Type '1'

4> Type '0'

Type-3:

If the Grammar has all the productions of the form

Right Linear Grammar (RLG)

$A \to \alpha B/\beta \quad (A,B) \in V \quad (\alpha,\beta) \in T^*$

$A \to B\alpha/\beta \quad (A,B) \in V \quad (\alpha,\beta) \in T^*$

Left Linear Grammar (LLG)

Ex:

RLG

$A \to aB/a$

$B \to aB/bB/a/b$

LLG

$A \to Ba/a$

$B \to Ba/Bb/a/b$

NOT T-3.

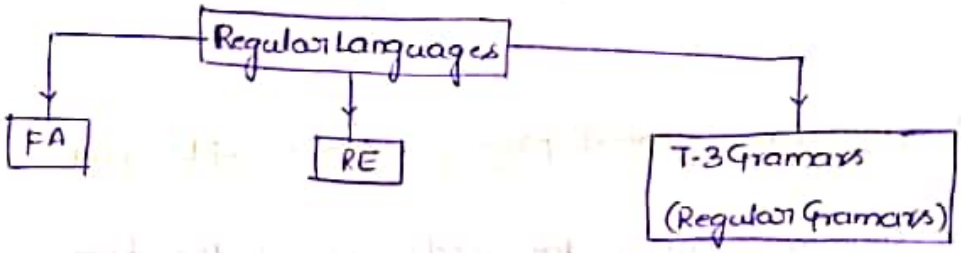
$A \to Ba/a$

$B \to aB/a$

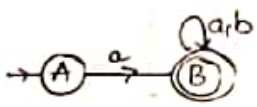
(41)

The Languages that are generated by Type-3 Grammar are always Regular Languages.

{T-3 Grammars should be either LLG or RLG but not both.}



CONVERSION OF FINITE AUTOMATA TO REGULAR GRAMMAR



⇒ The Initial state in the Automata is same as the Start state in the Grammar

$A \to aB$

$B \to aB/bB/\epsilon$ RLG

$L = \{a, aa, ab, abab \ldots\}$ start with 'A'

$\Downarrow R$

$A \to Ba$

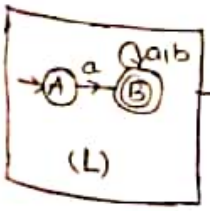
$B \to Ba/Bb/\epsilon$ LLG

$L = \{a, ba, baba \ldots\}$ Ending with 'a'

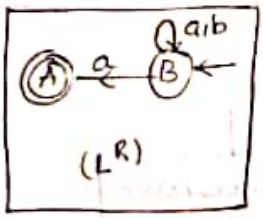
Finite Automata to LLG: $FA \xrightarrow{R} FA \longrightarrow RLG \xrightarrow{R} LLG$

$(L) \quad (L^R) \quad (L^R) \quad (L^R)^R = L$

⇒ Convert the above FA to LLG.



R



$B \to aB/bB/aA$

$A \to \epsilon$

(L^R)

$\to$

$B \to Ba/Bb/Aa$

$A \to \epsilon$

LLG

(L)

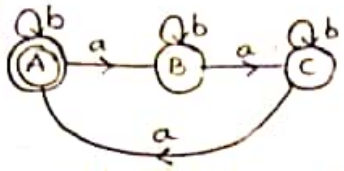
CONVERSION OF REGULAR GRAMMARS TO FINITE AUTOMATA

$A \to aB/bA/b$

$B \to aC/bB$

$C \to aA/bC/a$

RLG → FA



LLG → FA CONVERSION :

$LLG \xrightarrow{R} RLG \Rightarrow FA \xrightarrow{R} FA$

$(L) \quad (L^R) \quad (L^R) \quad (L^R)^R = L$

Type-2 Grammars / context Free Grammars - CFL - PDA

If the Grammar has all the productions of the form

$A \to \alpha \quad A \in V$

$\alpha \in (V \cup T)^*$

Ex: $A \to aAb/ab$

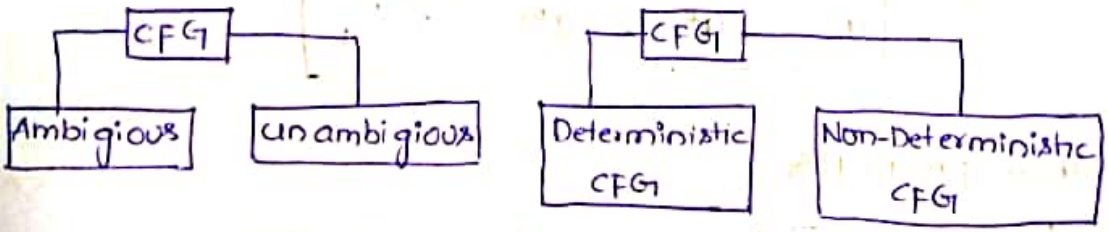
$A \Rightarrow aAb$

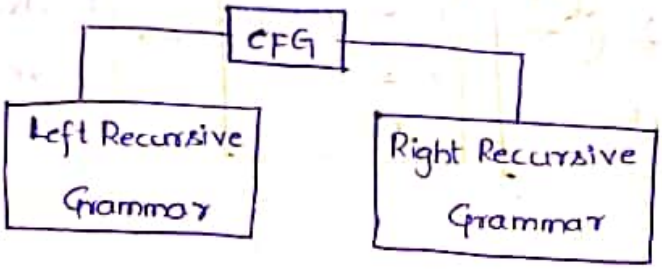
$\Rightarrow aabb$

⇒ The Grammar in the example is called context Free Grammar because we are replacing 'A' without any content (condition) that it must be preceded by 'a' likewise.

L-67

Type 2: context Free Grammars





ELIMINATION OF ∈- PRODUCTIONS

① $S \to aSb / aAb$
$A \to \epsilon$

$\Rightarrow$ $S \to asb/ab$

$S \to asb/aAb/ab$
↓
No production with A

$\Rightarrow$ $S \to asb|ab$

② $S \to AB$
$A \to aAA/\epsilon$
$B \to bBB/\epsilon$

Nullable states: $\{A, B, S\}$

$\Rightarrow$ Whenever the start state is nullable then the language contains ϵ and it must be present in the start state production.

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

③ $S \to AbaC$
$A \to BC$
$B \to B/\epsilon$
$C \to D/\epsilon$
$D \to d$

Nullable symbols: $\{B, C, A\}$

$S \to AbaC/Aba/ba/baC$
$A \to Bc/B/c$
$C \to D$
$D \to d$

(Dont add epsilon even though B,C are Nullable (∵ Uneed to add it only for starting symbol)).

ELIMINATION OF UNIT PRODUCTIONS

① $S \to Aa/B$
$B \to A/bb$
$A \to a/bc/B$

Step-1:

$S \to Aa/bb/a/bc$
$B \to bb/a/bc$
$A \to a/bc/bb$

$\Rightarrow S \to B \to bb$
$\Rightarrow S \to B \to A \to a$ / bc

$\Rightarrow B \to A \to a$ / bc
$\Rightarrow A \to B \to bb$
$A \to bb$

② $S \to AB$
$A \to a$
$B \to C/b$
$C \to D$
$D \to E$
$E \to a$

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

$S \to AB$
$A \to a$
$B \to b/a$

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

} Useless symbols Not Reachable from start state

ELIMINATION OF USELESS SYMBOLS

① $S \to AB/a$ $T = \{a, b\}$

$A \to BC/b$ $V = \{S, A, B, C\}$

$B \to AB/C$

$C \to aC/B$ ✗

Now, the useful symbols (which will generate some terminals) are $\{a, b, S, A\}$

⇒ B, C are not useful symbols (∵ I could not generate any string from B, C

$S \to a$

$A \to b$ } But here 'A' is not Reachable, therefore the minimised Grammar will be $S \to a$

② $S \to AB/AC$

$A \to aAb/bAa/a$

$B \to bbA/aaB/AB$

$C \to abCA/aDb$

$D \to bD/aC$ ✗

$T = \{a, b\}$

$V = \{S, A, B, C, D\}$

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

$\{B, S\}$

$S \to AB$

$A \to aAb/bAa/a$

$B \to bbA/aaB/AB$

③ $S \to ABC/BaB$

$A \to aA/BC/aaa$

$B \to bBb/a$

$C \to CA/AC$ ✗

$T = \{a, b\}$

$V = \{S, A, B, C\}$

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

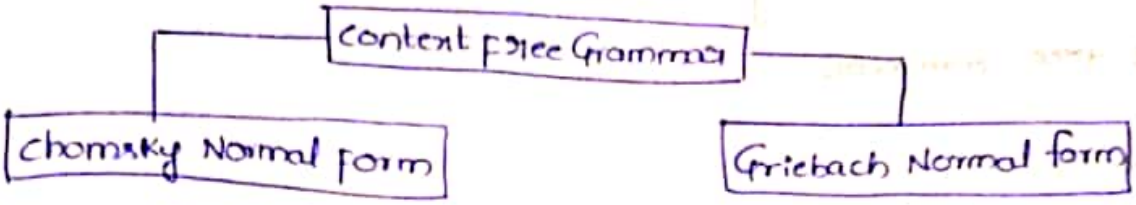
$S \to BaB$

$A \to aA/aaa$ ✗ → Not Reachable

$B \to bBb/a$

⇒ $S \to BaB$

$B \to bBb/a$



CNF:

If all the productions are of the form $A \to BC$ or $A \to a$ where A, B are variables and 'a' is a terminal.

Advantages:

1) Length of each production is Restricted

2) Derivation tree (or) parse tree obtained from CNF is always Binary tree

3) The no of steps required to derive a string of length $|w|$ is $(2|w|-1)$

4) It is easy to apply CYK membership Algorithm. $O(n^3)$ time

① convert the following into CNF?

$S \to bA / aB$

$A \to bAA / aS / a$

$B \to aBB / bS / b$

Let $\{ N_a \to a, N_b \to b \}$ then

$S \to N_b A / N_a B$

$A \to N_b(AA) / N_a S / a$

$B \to N_a(BB) / N_b S / b$

$N_a \to a$

$N_b \to b$

$\Rightarrow$

$S \to N_b A / N_a B$

$A \to N_b T / N_a S / a$

$B \to N_a P / N_b S / b$

$N_a \to a$

$N_b \to b$

$T \to AA$

$P \to BB$

CYK ALGORITHM (MEMBERSHIP ALGORITHM)

1) Check whether the string "baaba" is a valid member of the following context free Grammar?

$S \rightarrow AB/BC$

$A \rightarrow BA/a$

$B \rightarrow CC/b$

$C \rightarrow AB/a$

Sol: The CYK Algorithm is used to check the string belong to the language generated by the Grammar or not.

⇒ The CYK Algorithm can be applicable for only CNF form Grammars

Given Grammar $S \rightarrow AB/BC$

$A \rightarrow BA/a$

$B \rightarrow CC/b$

$C \rightarrow AB/a$

Given string b|a|a|b|a

1 2 3 4 5

	5	4	3	2	1
1	A, S, C	∅	∅	A, S	B
2	S, A, C	B	B	A, C	
3	B	S, C	A, C		
4	A, S	B			
5	A, C				

Cell no (1,1):

Now the (1,1) represent the variable 'b' in the required string

[1 to 1]

⇒ Now check what are the variables that are required to generate 'b'

⇒ B ✓

So the value of Cell (1,1) = B

Cell no (2,2):

The value of (2,2) in the given string is 'a' (2 to 2) ↪ start with '2' and end with '2'.

Now see which variable derives 'a'

= {A, C}

Cell no (3,3):

The Alphabet in the (3,3) is 'a' so {A, C} will generate 'a'

= {A, C}

Cell no: (4,4)

(4,4) = b ⇒ Generated by 'B'

Cell no (5,5) :-

(5,5) = 'a' = {A, C} will generate 'a'.

Cell no (1,2): $\begin{matrix}1&2&3&4&5\\b&a&a&b&a\end{matrix}$

(1,2) means start with 1 and end with 2 ⇒ "ba"

(12) = (11) × (22) [cross product]

= {B} × {A, C} = {BA, BC} = {A, S} → Generated by

Generated by

Cell no (2,3):

(2,3) = (22) × (33)

= {A, C} × {A, C}

= {AA, AC, CA, CC} = {B}

X X X B

↪ Not derived by any of the symbols.

Cell no (3,4):

(33) × (44) = {A, C} × {B}

= {AB, CB}

SC X

(4,5) = (44) × (55)

= {B} × {A, C}

= {BA, BC}

= {A, S} = {A, S}

(1,3) = b a a | b a = (baa)
1 2 3 | 4 5

(1) | (2 3)
(1 2) | (3)

(1,3) = (11)(23) (or) (12)(33)

= {B} × {B} : {A,S} × {A,C}

= {BB} : = {AA, AC, SA, SC}

= {ϕ} : = {x x x x} = {ϕ}

(2,4): (2 2) × (3,4) (or) (2 3) × (4 4)

= {A,S} × {S C} (or) {B} × {B}

= {AS, AC, SS, SC} = {BB}
x x x x = {ϕ}

= {ϕ}

Similarly continue for remaining cells

Now, (1,5) = baaba

(14) = baab

(1,5) = { (11) (2 5) = {A,S}
(12) (3 5) = {S C}
(13) (4 5) = {ϕ}
(14) (5 5) = {ϕ}
= {S, AC}

(14) = { (11) (2 4) = BBx
(12) (3 4) = AS, AC, SS, SC x
(13) (4 4) = {ϕ}

(2,5) = 2 3 4 5

(2,5) = { (22) (3 5) = AB, CB → {S, C}
(23) (4 5) = BA, BS = {A}
(24) (45) = BA, Bc = {A, S}

If the cell (last cell) (1,5) contains the starting symbol 'S' we can generate the string from the given Grammar.

Now, what are the various substrings of the given string that can be generated from the Grammar?

Now check where the starting symbols are present in the table i,e it is present at cells (1,2) (3,4) (4,5) (2,5) (1,5)

baaba
1 2 3 4 5

{ba} {ab} {ba} {aaba} ← Given string = {ba, ab, aaba}.

5) GREIBACH NORMAL FORM (GNF)

If all the productions of the Grammar are of the form $A \to a\alpha$ where $\alpha \in V^*$ then it is called GNF

Ex: $A \to aBCDE$

$A \to a$

Advantages

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

2) GNF is useful inorder to convert a CFG to "push down Automata".

CONVERSION OF CFG TO PDA:

1) Push the start symbol 'A' onto the stack $\delta(q_0, \epsilon, Z_0) = (q_1, AZ_0)$

2) push RHS of 'A' as follows $\delta(q_1, a, A) = (q_1, \alpha)$ if $A \to a\alpha$ is in Gramar

$\delta(q_1, b, A) = (q_1, \beta)$ if $A \to b\beta$ is in Gramar

3) Add final state with

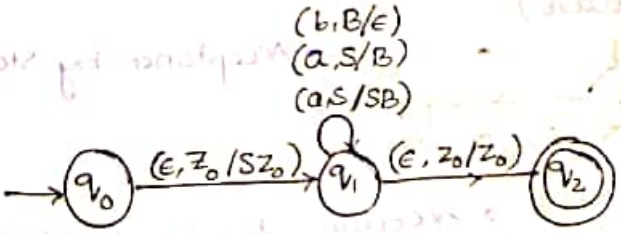
$$\delta(q_1, \epsilon, Z_0) = (q_f, \epsilon)$$

① $S \to aSb/ab$ → Not in GNF

$S \to aSB/aB$

$B \to b$

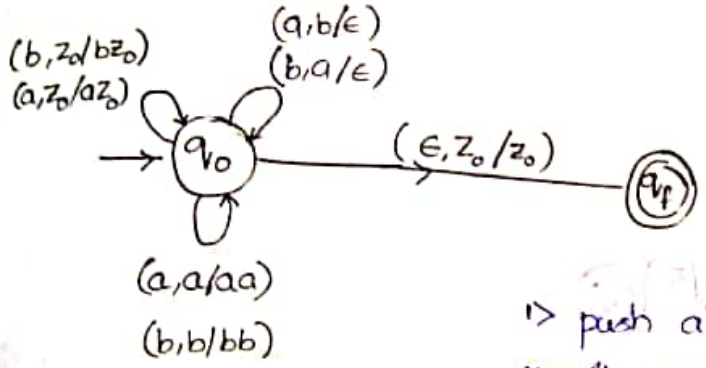
In GNF

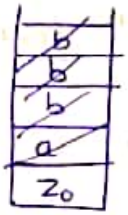


PDA → Acceptance by State
PDA → Acceptance by Empty stack

② $w \in (a,b)^* / n_a(w) = n_b(w)$

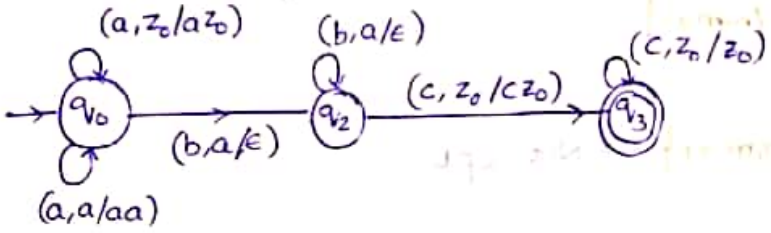
Let the string be abbbaa





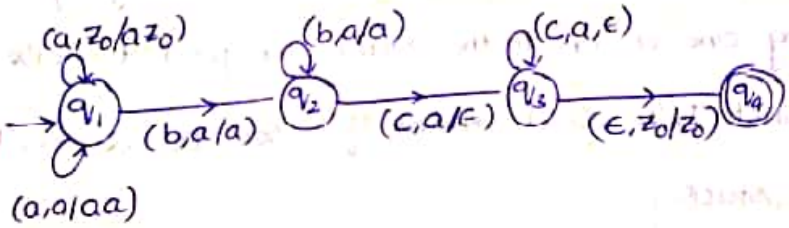
1> push a's onto the stack.
2> If you see b pop one 'a' & vice versa.
3> If your Alphabet is 'b' & Bottom of Stack = Z0
⇒ push 'b' on stack.

③ $L = \{a^n b^n c^m / n,m \geq 1\}$

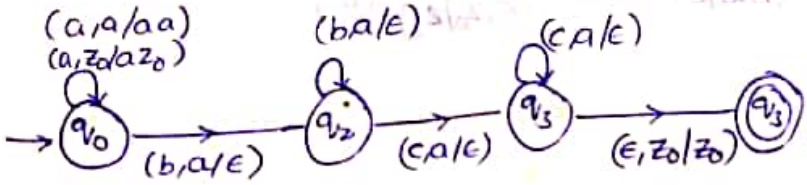


L-69

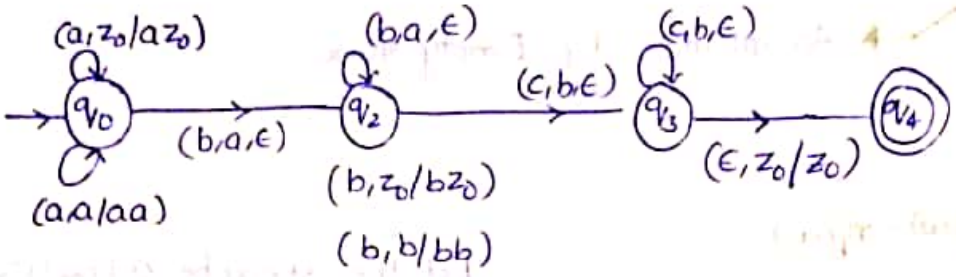
④ $L = \{a^n b^m c^n / n,m \geq 1\}$



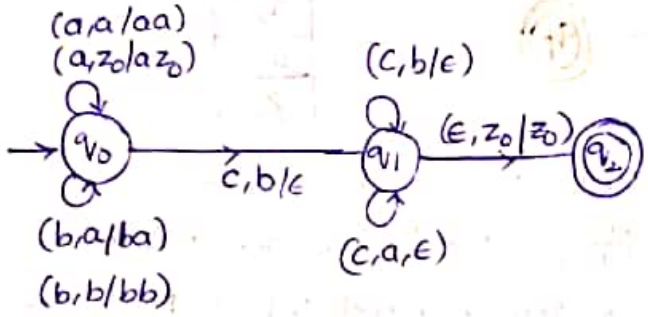
⑤ $L = \{a^{m+n} b^m c^n\} (m,n \geq 1)$



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



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



⑧ $L = \{a^n b^n c^m d^m / n,m \geq 1\}$ ⇒ context free Languages

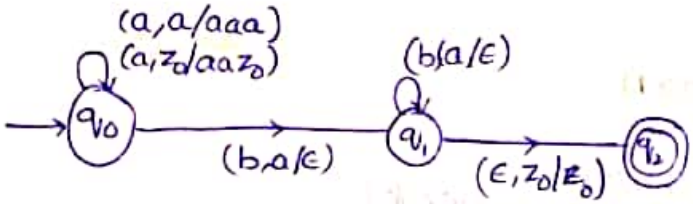
$L = \{a^n b^m c^m d^n / n,m \geq 1\}$

⑨ $L = \{a^n b^m c^n d^m / n,m \geq 1\}$ ⇒ Not CFL.

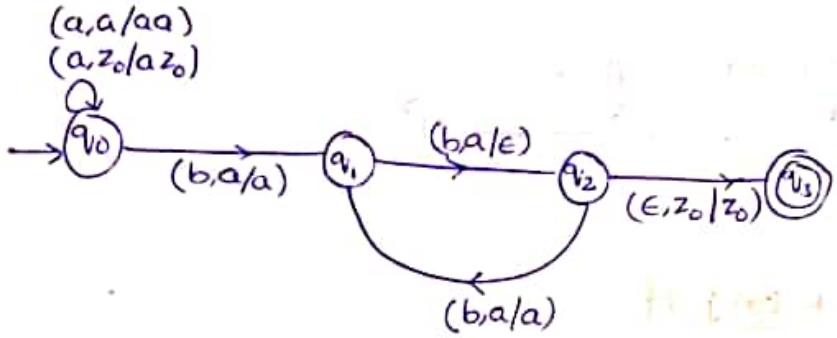
⑩ $L = \{a^n b^{2n} / n \geq 1\}$

The Language accepted is $L = \{abb, aabbbb, aaabbbbbb \text{---}\}$

⇒ The solution to construct PDA for the above language is instead of pushing one 'a' on to the stack we push two a's and pop them against 'b's which means if i see a 'a' i will push 2a's onto the stack.

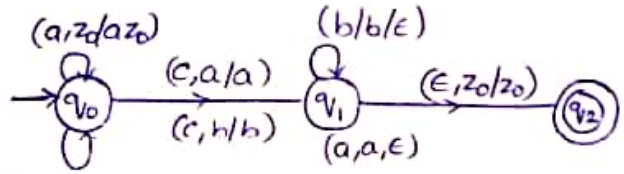


⇒ The other solution for the above problem is for every two b's we should pop 1 a'. which means we push all the a's and for pushing 'b' on to the stack we pop 1a' against the occurrence of 2 b's



(11) $L = \{a^n b^n c^n / n \geq 1\}$ ⇒ Not Regular CFL

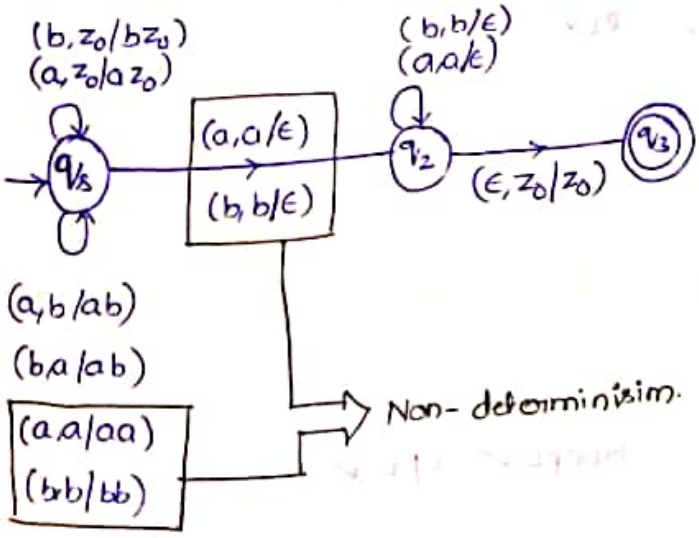
(12) $L = \{\omega c \omega^R / \omega \in (a,b)^+\}$



(b,z0/bz0)
(a,a/aa)
(a,b/ab)
(b,a/ba)
(b,b/bb)

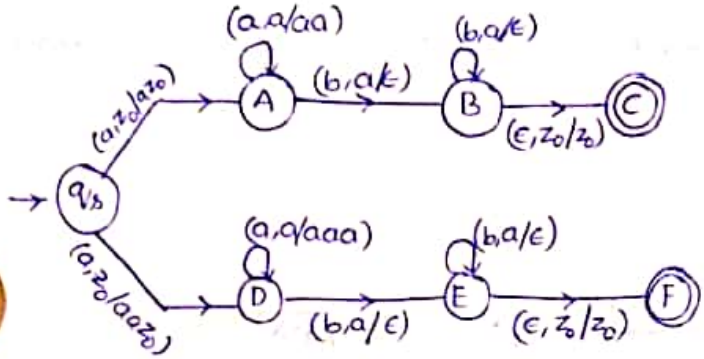
L-70

$L = \{\omega\omega^R / \omega \in (a+b)^+\}$ ⇒ NO "DPDA" for this Language we have to construct "NDPDA"



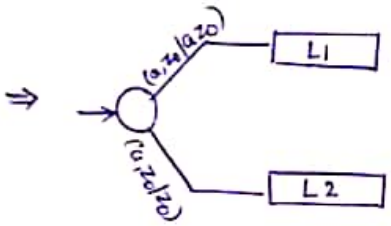
NDPDA is not as powerful as DPDA

$L = \{a^n b^n / n \geq 1\} \cup \{a^n b^{2n} / n \geq 1\}$



③ $L = \{a^i b^j c^k d^l / i = k \text{ (or) } j = l\}$

$= \{a^m b^j c^m d^l\} \cup \{a^i b^m c^k d^m\}$



L-71

① $a^{m+n} b^n c^m / n,m \geq 1$ ⇒ RL ✗ CFL ✓ (DCFL)

$a^m b^{m+n} c^n / n,m \geq 1$

② $a^m b^n c^{m+n}$ = RL ✗ CFL ✓

③ $a^m b^m c^n d^n$ = RL ✗ CFL ✓

④ $a^m b^n c^m d^n / m,n \geq 1$ = RL ✗ CFL ✗

⑤ $a^m b^n c^n d^m / m,n \geq 1$ = CFL ✓ RL ✗

⑥ $a^m b^i c^m d^k / m,n \geq 1$ = DCFL ✓ RL ✗

⑦ $a^m b^n / m > n$ = DCFL ✓

⑧ $a^n b^{2n} / n \geq 1$ = DCFL ✓

⑨ $a^n b^{n^2} / n \geq 1$ = RL ✗ CFL ✗

⑩ $a^n b^{2n} / n \geq 1$ = RL ✗ CFL ✗

⑪ $ww^R / w \in (a,b)^+$ = RL ✗ DCFL ✗ NDCFL ✓ CFL ✓

⑫ $ww / w \in (a,b)^+$ = RL ✗ ✗CFL

13) $a^n b^n c^m / n > m$ = XRL XCFL

14) $a^n b^n c^n d^n / n \leq 10^{10}$ = RL✓ DCFL✓ CFL✓

15) $a^n b^{2n} c^{3n} / n \geq 1$ = RLX CFLX

16) $xcy / x, y \in (0,1)^+$ = RL✓ DCFL✓ CFL✓

17) $xx^R / x \in (a,b)^*, |x| = 1$ = RL✓ CFL✓

18) $wwww^R / w \in (a,b)^*$ = RLX CFLX

19) $a^n b^{3n} / n \geq 1$ = CFLX

20) $a^m b^n / m \neq n$ = CFL✓ DCFL✓

21) $a^m b^n / m = 2n+1$ = DCFL✓ CFL✓

22) $a^i b^j / i \neq 2j+1$ = DCFL✓

23) $a^{n^2} / n \geq 1$ = CFLX

24) $a^{2^n} / n \geq 1$ = XCFL

25) $a^{n!} / n \geq 1$ = XCFL

26) $a^m /$ m is prime = XCFL

27) a^k / k is even = RL, CFL, DCFL

28) $a^i b^j c^k / i > j > k$ = XCFL

29) $a^i b^j c^k / j = i + k$ = CFL✓, DCFL✓

30) $a^i b^j c^k d^l / i = k$ or $j = l$ = CFL✓

31) $a^i b^j c^k d^l / i = k$ and $j = l$ = CFLX

32) $a^m b^l c^k d^n / m, l, k, n \geq 1$ = RL✓ CFL✓

33) $a^n b^{4m} / n, m \geq 1$ = RL✓ CFL✓

34) $\{a^3, a^8, a^{13} \ldots\}$ = RL✓ CFL✓

35) $\{a^{2n+1} / n \geq 1\}$ = RL✓ CFL✓

36) $\{a^{n^n} / n \geq 1\}$ = RLX CFLX

37) $\{w / w \in \{a,b\}^*, |w| \geq 100\}$ = RL✓ CFL✓

38) $\{w / w \in \{a,b,c\}^*, \{n_a(w) = n_b(w) = n_c(w)\}$ = $\{a^n b^n c^n / n \geq 1\}$ XCFL

39) $\{w / w \in \{a,b\}\ \{n_a(w) \geq n_b(w) + 1\}$ = CFL✓

(3) TURING MACHINE AS TRANSDUCER OF ONE'S COMPLIMENT

⇒ TM has the capacity to Read from the tape and also write on the tape

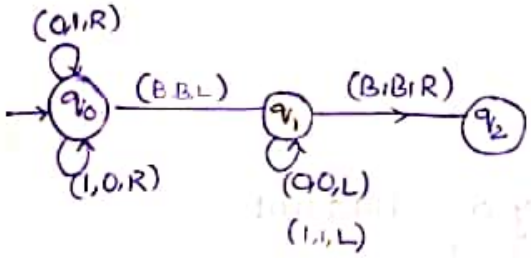
TM to find ones compliment of a Binary number

⇒ Here the TM is designed in a such a way that it need not accept anything so there will not be final state

⇒ If i see a '0' mark it as '1' and move 'R'

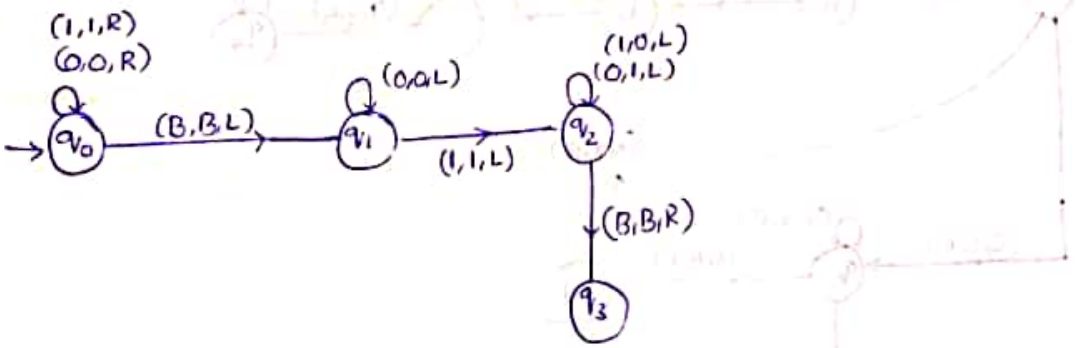
⇒ If i see a '1' mark it as '0' and move 'R'

TRANSDUCER = CONVERTER



(4) TURING MACHINE AS TRANSDUCER OF TWO'S COMPLIMENT

The Given string BBB 0111 000 BBB



⇒ start from the starting point of Input symbol and move 'R' till the end

⇒ Now start moving Left without changing anything untill you see the 1st '1' (one)

⇒ Now, the Remaining bits to the Left of bits should be complemented.

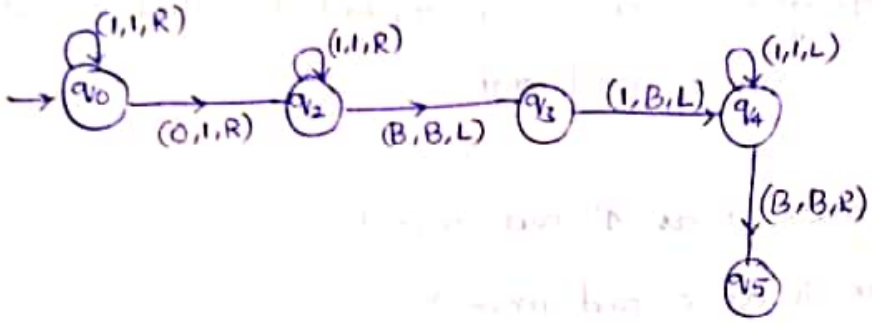
(5) TM AS AN ADDER (OF UNARY NUMBERS)

(3) = $(111)_1$ ⇒ |x| = 3

(4) = $(1111)_1$ ⇒ |y| = 4

and both x, y are seperated by '0'

(1110 1111)



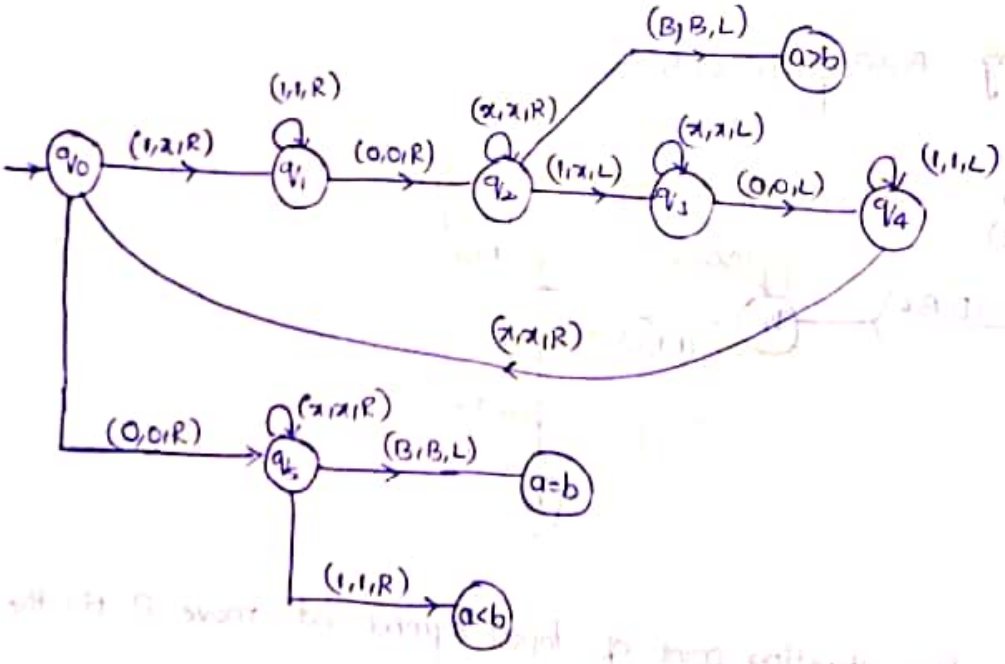
(6) TM AS A COMPARATOR

a = 4 = $(1111)_1$

b = 5 = $(11111)_1$

} a, b are seperated by 0 11110 11111

a = b a < b a > b



TM can perform any mathematical function

THE STANDARD TURING MACHINE

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

Q = Set of states

Σ = Input Alphabet

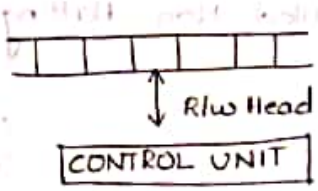
Γ = Tape Alphabet

δ = Transition function

q_0 = Initial state ($q_0 \in Q$)

B = is used to Represent Blank ($B \in \Gamma$)

F = Set of final states. ($F \subseteq Q$)



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

Summary

1) Tape is unbounded, so any number of Left and Right moves are possible
2) It is deterministic, i,e atmost one move for each configuration.
3) NO special i/p or o/p file.

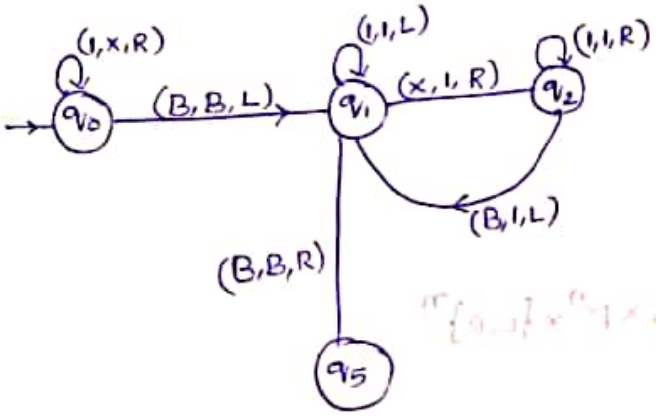
8) TM AS A COPIER

Copier means the Turing machine should produce the copy of the given input. If the input is B11B then the TM should produce BB1111BB

→ Make all the 1's to 'x' and move 'Right'

→ And now if you see a blank move left (1 position) and convert the x to '1'

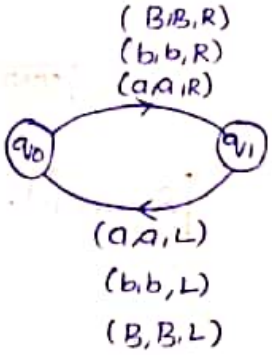
→ Move Right and convert the next Blank to '1' ---



9)

NON-HALTING TURING MACHINE

There are some turing machines that will never halt for some inputs they are called Non-Halting turing machines.



10)

TURING THESIS

Any computation that can be carried out by mechanical means can be performed by some Turing machine.

⇒ Anything that can be done by existing digital computers can also be done by TM

⇒ No one has yet been able to suggest a problem, solvable by what we intuitively considering an Algorithm, for which TM program cannot be written.

⇒ Alternative models have been proposed for mechanical computation, but none of them are more powerful than the Turing machine model.

11)

MODIFICATIONS TO STANDARD TURING MACHINE

1) TM with stay option

2) TM with semi infinite tape

3) Offline TM

4) Multitape TM $\quad \delta: Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{L,R\}^n$

5) Jumping TM

6) Non-Erasing TM

7) Always writing TM

6) Multidimensional TM $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D\}$

9) Multihead TM

10) Automata with a Queue

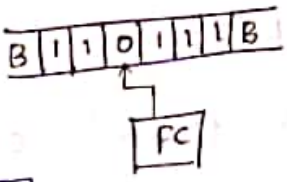
11) TM with only 3 states

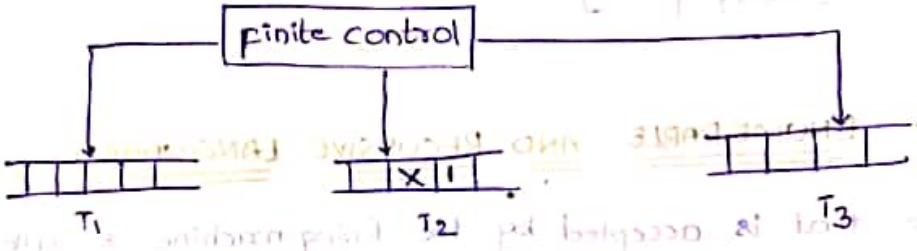
12) Multitape TM with stay option and atmost 2 states

13) Non-deterministic TM $\delta: Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$

14) A NPDA with 2 independent stacks $\delta: Q \times (\Sigma \cup \epsilon) \times \Gamma \times \Gamma \to 2^{Q \times \Gamma^* \times \Gamma^*}$

12) UNIVERSAL TM AND ENCODING TM





(Rep of TM) (I/p) (Internal state)

$\delta(q_1, 1) = (q_2, X, R)$

$\delta(q_2, 1) = (q_3, Y, R)$

$Q = (q_1, q_2, q_3, q_4 \ldots)$

$\Gamma = (a_1, a_2, a_3, a_4 \ldots)$

$q_1 = 1$ $a_1 = 1$

$q_2 = 11$ $a_2 = 11$ '0' seperate

$q_3 = 111$ $a_3 = 111$

$q_4 = 1111$ $a_4 = 1111$

$R = 11$ $L = 1$.

$\delta(q_1, a_1) = (q_2, a_2, R)$

010 10 11 0 11 0 110

→ Every "TM" can be represented as the string of 0's and 1's.

13) LINEAR BOUNDED AUTOMATA

⇒ Non-deterministic TM + Tape (stack) = PDA

⇒ NTM + Tape (Finite size) = Finite Automata

⇒ LBA = TM + Tape (Input size) [a a b b]

FA < PDA < LBA < TM ⇒ powers.

(DFA = NFA) (DPDA < NPDA) (DTM = NDTM)

DLBA NDLBA

The some of the Languages that are accepted by the LBA are

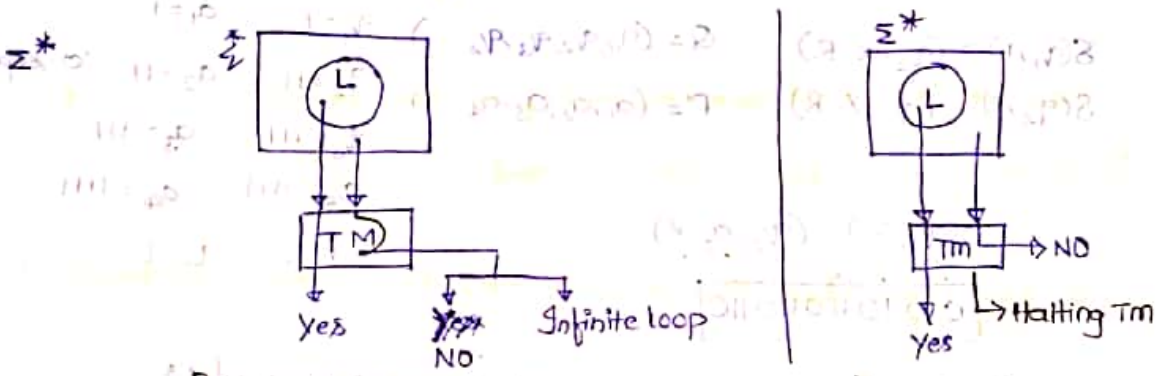
1) $L = \{a^n b^n c^n : n \geq 1\}$
2) $L = \{a^{n!}, n \geq 0\}$
3) $L = \{a^n : n = m^2, m \geq 1\}$
4) $L = \{a^n : n$ is prime$\}$
5) $L = \{a^n / n$ is not prime$\}$
6) * $L = \{ww / w \in (a,b)^+\}$
7) $L = \{w^n : w \in (a+b)^+, n \geq 1\}$
8) $L = \{wwww^R : w \in \{a,b\}^+\}$

} Context Sensitive Languages

"LBA is a Halting TM"

RECURSIVELY ENUMERABLE AND RECURSIVE LANGUAGES.

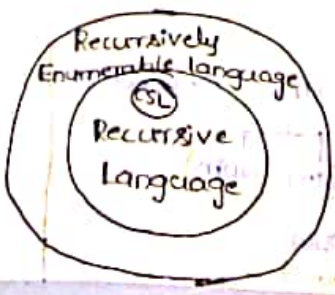
The Language that is accepted by the Turing machine is called Recursively enumerable languages

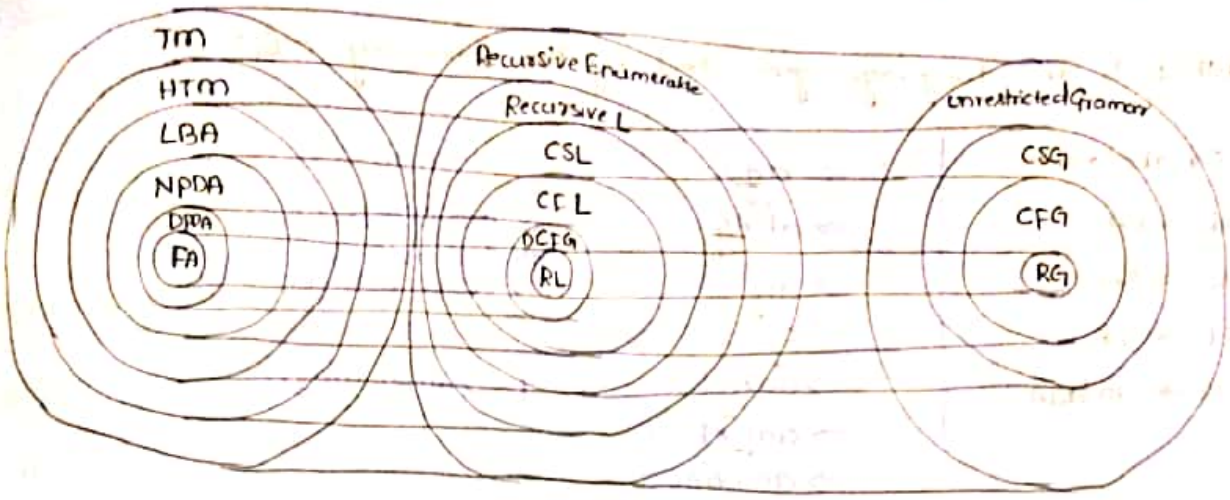


Recursively Enumerable Languages | Recursive Language

⇒ If there exist a turing machine for a language then the language is called Recursively Enumerable language.

⇒ If there exits a Halting TM for a language then the language is called Recursive Language





UNRESTRICTED GRAMARS

⇒ The Grammars corresponding to the TM are called Unrestricted Gramars.

⇒ A Gramar is called unrestricted if all the productions are of the form $u \to v$ where $u \in (V \cup T)^+$ and $v \in (V \cup T)^*$

① what language does the following unrestricted Gramar derive?

$S \to S_1 B$

$S_1 \to a S_1 b$

$bB \to bbbB$

$aS_1 b \to aa$

$B \to \lambda$

Sol: $S \Rightarrow S_1 B$

$\Rightarrow a S_1 b B$

$\Rightarrow aa B$

$\Rightarrow aa$

$L = \{a^{n+1} b^{n+K} / n \geq 1, K = -1, 1, 3, 5 \ldots\}$

$S \Rightarrow S_1 B$

$\Rightarrow a S_1 bB$

$\Rightarrow a S_1 bbbB$

$\Rightarrow aa bbbB$

CONTEXT SENITIVE GRAMMAR - EXAMPLE 1

A Grammar is said to be Context sensitive if all the productions are of form $\alpha \to \beta$ where $(\alpha, \beta) \in (V \cup T)^+$ and $|\alpha| \leq |\beta|$

What is the language generated by the following CSG?

$S \to abc / aAbc$

$Ab \to bA$

$Ac \to Bbcc$

$bB \to Bb$

$aB \to aa / aaA$

$S \Rightarrow aAbc \Rightarrow aabbcc$

$\Rightarrow abAc \Rightarrow aabbcc$

$\Rightarrow abBbcc \Rightarrow aaBbbcc$

$\Rightarrow aBbbcc \Rightarrow aaabbbccc$

$\Rightarrow aaAbbcc$

$\Rightarrow aabAbcc$

$\Rightarrow aabbAcc$

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

EXAMPLE-3 [Not in GATE]

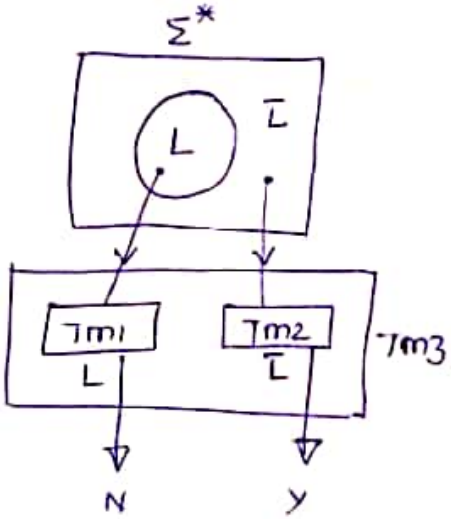
THEOREM ON RECURSIVE AND RE LANGUAGES

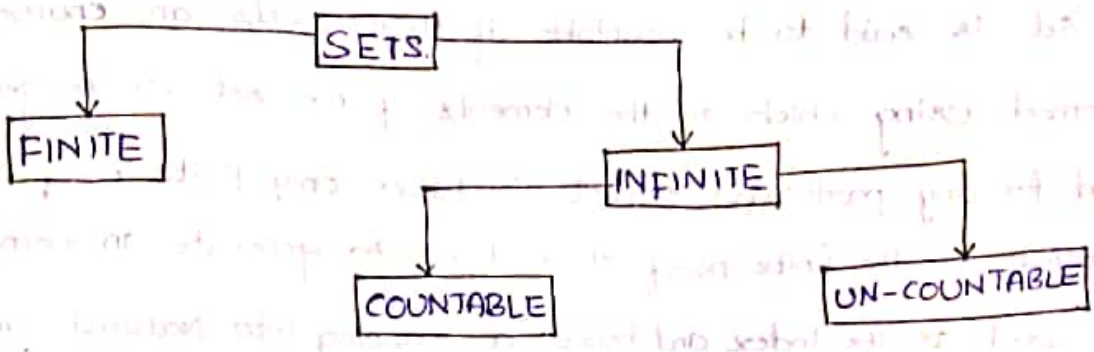
Recursively Enumerable

If a language 'L' and its complement $\bar{L}$ are both RE, then both the languages are Recursive. If 'L' is recursive, then $\bar{L}$ is also recursive and consequently both are Recursively enumerable.

⇒ There is no membership Algorithm for RE Languages.

⇒ Membership Algorithm exists for the Recursive Languages





COUNTABLE

A set is 'S' is countable if the elements of the set can be put in one to one correspondence with the set of natural numbers

UNCOUNTABLE

A set is uncountable if it is infinite and not countable.

E = {0, 2, 4, 6, 8, 10, 12 - - - } $\boxed{2i}$
N = {1, 2, 3, 4, 5, 6 - - - - } $\boxed{i+1}$

one to one correspondence is there so countable

O {1, 3, 5, 7, 9, 11 - - - } $\boxed{2i+1}$
N = {1, 2, 3, 4, 5, 6 - - - - } $\boxed{i+1}$

one-one correspondence is there so countable.

Real No's {0 —— 1}

↳ Infinite No's are present and could eatup all the Natural No's set **so set of Real numbers is uncountable**

→ There are some languages for which Turing machines cannot be constructed. ⇒ There are some languages which are not Recursively Enumerable

ALTERNATIVE DEFINITION OF COUNTABILITY

A Set is said to be countable if there exists an enumeration method using which all the elements of the set can be generated and for any particular element, it takes only finite no of steps to generate it. The finite no of steps taken to generate an element can be used as its index and hence a mapping into Natural numbers.

Set of all Even numbers = for (i=0 to ∞)
{
generate('2i')
}

0, 2, 4, 6, 8 - --
↓ ↓ ↓ ↓ ↓
1 2 3 4 5 → index
step steps

Set of all odd numbers = for (i=0 to ∞)
{
generate (2i+1)
}

{ 1, 3, 5, 7, 9, 11 ---
↓ ↓ ↓ ↓ ↓ ↓
1 2 3 4 5 6 → index

Set of all Real numbers = No Eumeration method so Not countable

SET OF ALL QUOTIENTS ARE COUNTABLE

Quotients of P/q where $\{(p,q) \in z^+\}$

S = { 1/1, 1/2, 2/1, 1/3, 1/4, 3/4 ----}

= { 1/1, 1/2, 1/3, 1/4, 1/5, 1/6 --- } X 2/1 will not be generated so this Enumeration method is wrong.

so follow this enumeration method, of (P/q)

1> find the sum of (P+q) and go in increasing order of the sum

sum = ② ③ ④ ⑤

{ 1/1, 3/4 } { 1/2, 2/1 } { 1/3, 2/2, 3/1 } { 1/4, 2/3, 3/2, 4/1 } ---- }

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪

so Every 'p/q' can be generated after finite no of steps and therefore the set of all Quotients are countable"

SET OF ALL STRINGS OVER ANY FINITE ALPHABET ARE COUNTABLE

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

$\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb \text{ ---}\}$

Now the Enumeration method that must be followed is

1) First Generate all the strings of length '0'(ϵ) and now generate all the strings of length 1 {a,b} and in the set of length{1,-}follow the Lexicographic order.

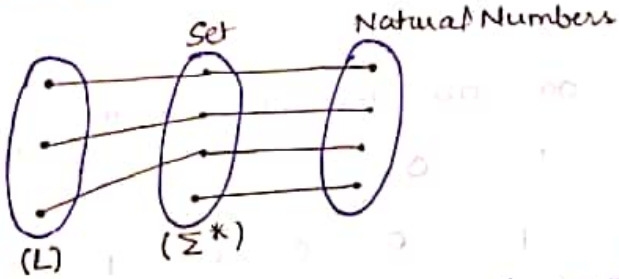
$S = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab \text{ ----}\}$

Steps = {1 2 3 4 ------}

∵ SET OF ALL POSSIBLE STRINGS OVER ANY FINITE ALPHABET SET ARE COUNTABLE, i.e. Σ^* IS COUNTABLE

Now, Another important property is.

Every subset of countable set is either finite or Countable



Now we know that Σ^* is countable and $L \subseteq \Sigma^*$ therefore

"Every Language is countable" → NOT ALL LANGUAGES

→ GIVEN A LANGUAGE ITS STRING CAN BE PUT IN ONE-ONE CORRESPONDANCE OF NATURAL NO'S.

SET OF ALL TURING MACHINES ARE COUNTABLE

$\Sigma = \{0,1\}$ (1) $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11 - - -\}$ = countable

(2) Every TM can be encoded as a strings of 0's and 1's

(3) Set of all TM's (S) $\{S \subseteq \Sigma^*\}$

(4) Every subset of countable set is finite or countable

∴ **SET OF ALL TM'S ARE COUNTABLE**

IMPLICATIONS OF THE FACT THAT THE TURING MACHINES ARE COUNTA

TM's are countable

⇒ Recursively Enumerable Languages are Countable

⇒ Recursive Languages are countable

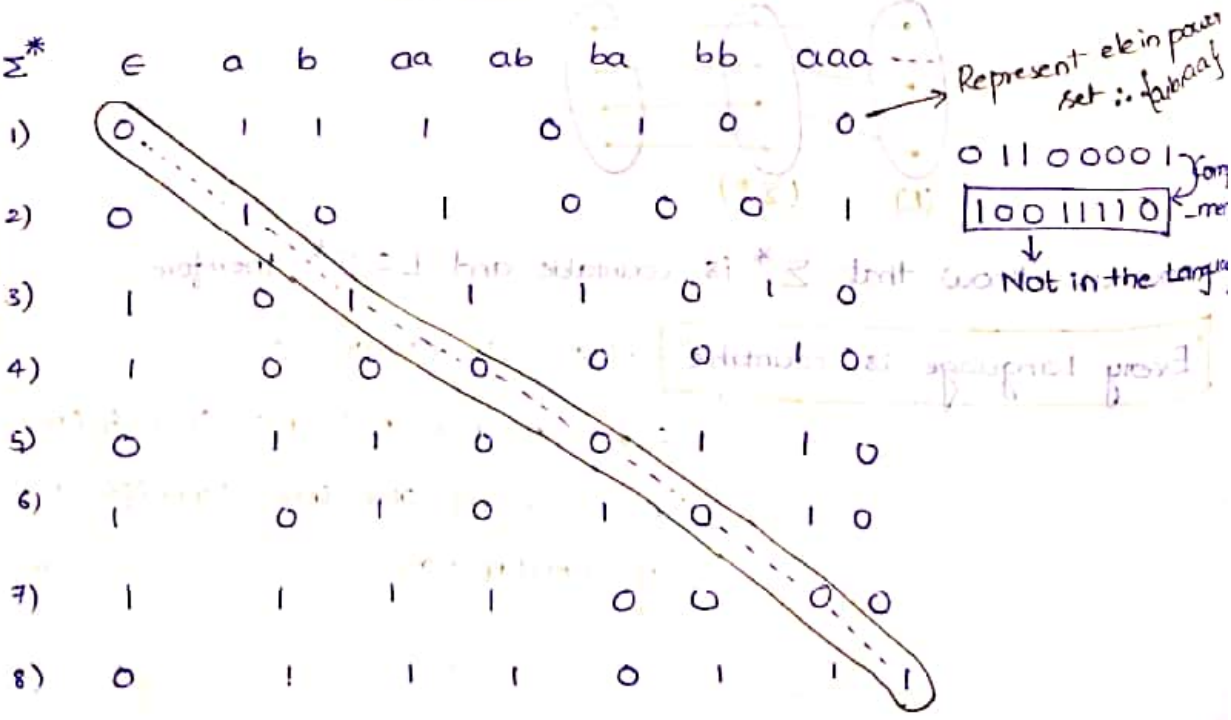
⇒ CSL are Countable ⇒ LBA are Countable

⇒ CFL are countable ⇒ PDA are Countable

⇒ RL are countable ⇒ Finite Automata are countable

DIAGNOLIZATION METHOD TO PROVE SET OF REL. LANGUAGES ARE UNCOUNTABLE

$\Sigma = \{a, b\}$ Σ^* is countable, 2^{Σ^*} is uncountable



∴ Set of Languages possible are uncountable

∴ All TM's are countable but all the Languages possible are uncountable

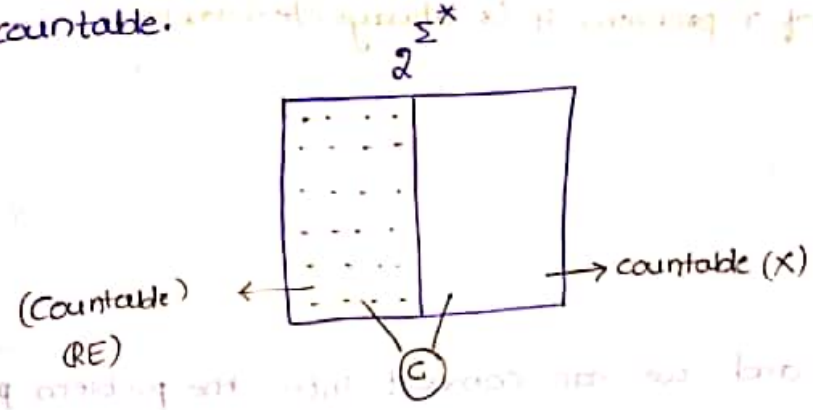
∴ There exist some languages which a Turing machine cannot Accept.

SOME PROBLEMS ON COUNTABILITY

* 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 no of countable sets is countable

⇒ The set of all languages that are not recursively Enumerable is uncountable.

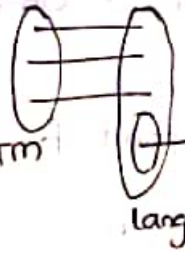


→ If 'S' is countably infinite then (2^S) is uncountable → powerset {set of all subsets possible}

put $S = \Sigma^*$ ⇒ If Σ^* is countably infinite then 2^{Σ^*} (set of all languages) are uncountable (uncountable infinite).

Now, TM- are countably infinite but all languages are uncountably infinite

∴ There are some languages for which TM does not exist.



Computability and Decidability

function

$f(n) = (n^2+1)$

Algo (or) TM (Halts ✓ Tape)

Problem

A stmnt whose ans is either { True or false then it is decidable

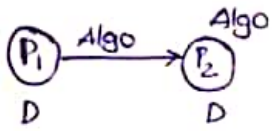
Ex: If 'n' is a prime

D = set of all Natural no's

Ex: A given Grammar 'G' is Ambigious ⇒ Undecidable

Domain = {Set of all CFG's}

Given an Instance of a problem, it is always decidable



Given an problem 'P1' and we can convert into the problem P2 using an Algorithm and the problem P2 can be solved using another Algorithm, which means that there exists an Algorithm for problem 'P1'. The conversion should be in such a way that the solution to 'P2' should also be a solution to P1. which means if the sol to problem P2 is True then the solution to P1 should also be True. And the procedure of convertion is called "Reducability". If you find out that if 'P2' is decidable then 'P1' is also decidable. But before it is known that 'P1' is un-decidable then the problem 'P2' is undecidable

⇒ The starting point for the problem of decidability is the halting problem.

TM- HALTING PROBLEM

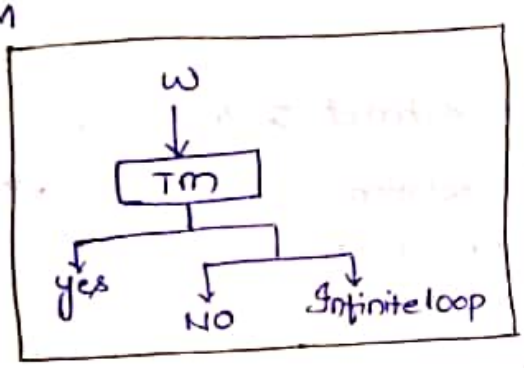
Given the description of a TM 'M' and an input 'w' does 'm' ~~started with 'w'~~, does 'm' when started with 'w' as its input eventually Halts.

Theorem

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

Recursively Enumerable

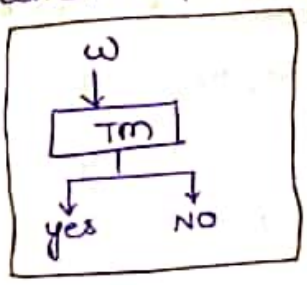
TM



⇒ No membership Algorithm

Recursive language

TM which halts.



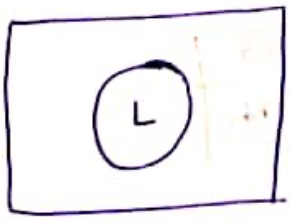
⇒ Membership Algo exists

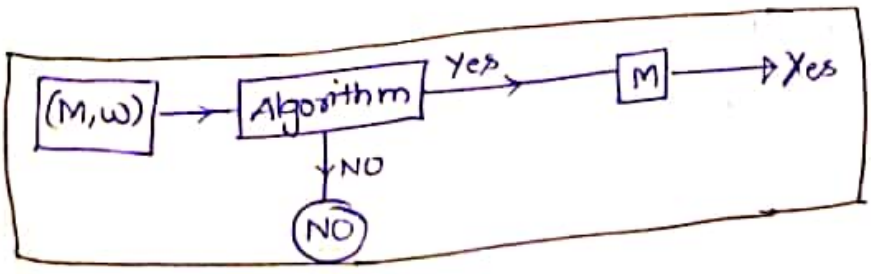
Consider a Language 'L' which is Recursively Enumerable (No halting TM exits)

Σ^*

L = RE

TM = m





∴ This is the membership Algorithm we got for the "REL" but it will contradict the statement that 'REL do not have Membership Algo'.

SOME UNDECIDABLE PROBLEMS BASED ON TM HALTING PROBLEM

1) The state entry problem is, given a TM, a state ($q \in Q$) and $w \in \Sigma^+$, decide whether or not the state 'q' is ever entered when 'M' is applied to 'w'. This is undecidable.

2) Given a TM M, whether or not M halts if started with a blank space. This is undecidable

3) Almost any problem related to RE languages is undecidable

POST CORRESPONDENCE PROBLEM

Given two sequences of 'n' strings on some Alphabet Σ, say $A = w_1 w_2 \cdots w_n$ and $B = v_1 v_2 \cdots v_n$, we say there exists a PC-solution for pair (A,B) if there is a non empty sequence of integers i,j,k such that

$$w_i w_j \cdots w_k = v_i v_j \cdots v_k$$

PC-problem is to device an Algorithm that will tell us for any (A,B), whether or not there exist a PC-solution

Ex:

A =

w_1	w_2	w_3
a	ab	bba

B =

V1	V2	V3
baa	aa	bb

The post correspondence solution is (3,2,3,1)

$w_3 w_2 w_3 w_1 =$ bbaabbbaa

$v_3 v_2 v_3 v_1 =$ bbaabbbaa

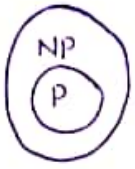
COMPLEXITY CLASSES

P-CLASS

The set of all languages that are accepted by some deterministic TM in polynomial time

NP-CLASS

The set of all languages accepted by Non-deterministic TM in polynomial time is called Np-class.



NDTM $\longrightarrow$ DTM

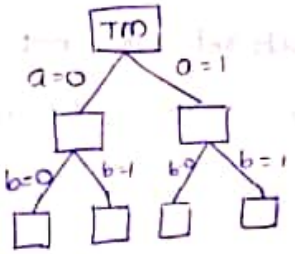
$O(n^k) \longrightarrow O(2^n)$

$(a \vee b \vee c)$

$2 \times 2 \times 2$

$= (2^n)$

$: O(2^n)$



DECIDABILITY TABLE

PROBLEM	RL	DCFL	CFL	CSL	Recursive language	REL
Does $w \in L$? (Membership Algo)	D	D	D	D	D	UD
Is $L = \phi$? (Emptiness problem)	D	D	D	UD	UD	UD
$L = \Sigma^*$ (completeness problem)	D	UD	UD	UD	UD	UD
Is $L_1 = L_2$ (Equality problem)	D	UD	UD	UD	UD	UD
Is $L_1 \subseteq L_2$ (Subset problem)	D	UD	UD	UD	UD	UD
Is $L_1 \cap L_2 = \phi$	D	UD	UD	UD	UD	UD
Is L finite or not? (Finiteness)	D	D	D	UD	UD	UD
Is complement of L a language of same type or not?	D	D	D	UD	UD	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

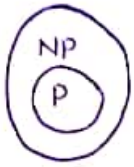
COMPLEXITY CLASSES

P-CLASS

The set of all languages that are accepted by some deterministic TM in polynomial time

NP-CLASS

The set of all languages accepted by Non-deterministic TM in polynomial time is called Np-class.



NDTM $\longrightarrow$ DTM

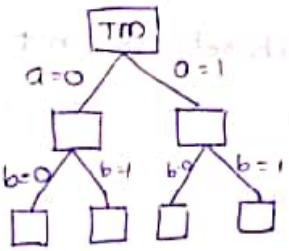
$O(n^k) \longrightarrow O(2^n)$

$(a \vee b \vee c)$

$2 \times 2 \times 2$

$= (2^n)$

$: O(2^n)$



DECIDABILITY TABLE

PROBLEM	RL	DCFL	CFL	CSL	Recursive language	REL
Does $w \in L$? (Membership Algo)	D	D	D	D	D	UD
IS $L = \phi$? (Emptiness problem)	D	D	D	UD	UD	UD
$L = \Sigma^*$ (completeness problem)	D	UD	UD	UD	UD	UD
IS $L_1 = L_2$ (Equality problem)	D	UD	UD	UD	UD	UD
Is $L_1 \subseteq L_2$ (Subset problem)	D	UD	UD	UD	UD	UD
Is $L_1 \cap L_2 = \phi$	D	UD	UD	UD	UD	UD
Is 'L' finite or not? (Finiteness)	D	D	D	UD	UD	UD
Is complement of 'L' a language of same type or not?	D	D	D	UD	UD	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

CFL ⇒ closed under union, concatenation, closure

union	concatenation	closure
L_1 L_2	L_1 L_2	L_1 L_1^*
$S_1 \to$ $S_2 \to$	$S_1 \to$ $S_2 \to$	$S_1 \to$ $S \to SS \mid \epsilon$
CFL { $S \to S_1 \mid S_2$, $S_1 \to$, $S_2 \to$	$S \to S_1 \cdot S_2$, $S_1 \to$, $S_2 \to$ } CFL	CFL

Not closed under intersection → L_1 L_2

$L_1 = \{a^n b^n c^m / m,n \geq 0\}$

$L_2 = \{a^m b^n c^n / n,m \geq 0\}$

$L_1 \cap L_2 = \{a^n b^n c^n / n \geq 0\}$ — Not CFL.

Not closed under complimentation = $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

If we assume L_1 = CFL then $\overline{L_1}$ = CFL
If we assume L_2 = CFL then $\overline{L_2}$ = CFL
} then $\overline{L_1} \cup \overline{L_2}$ should be Context-free and $\overline{\overline{L_1} \cup \overline{L_2}}$ should be contextfree which means $L_1 \cap L_2$ should be CFL because but we have seen that $L_1 \cap L_2$ is not CFL. so our assumption is wrong

That, the complimentation of CFL is Context-free

CFL is not closed under complimentation.

15.

L-5

⇒ L_1/L_2 means right quotient of L_1 with L_2.

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

$L_1 = \{01, 001, 101, 0001, 1101\}$

$L_2 = \{01\}$

$\frac{01}{01}$	$\frac{001}{01}$	$\frac{101}{01}$	$\frac{0001}{01}$	$\frac{1101}{01}$
$= \epsilon$	0	1	00	11

$L_1/L_2 = \{\epsilon, 0, 1, 00, 11\} \Rightarrow \{\epsilon, 0, 1, 00, 11\}$.

Now, $L_1 = 10^*1$

$L_2 = 0^*1$

$L_1/L_2 = \{\frac{11}{1}, \frac{101}{01}, \frac{1001}{001}, \frac{10001}{0001}, \dots\}$

$L_1/L_2 = \{1, 10, 100, 1000, \dots\}$

$L_1/L_2 = 10^*$

$\frac{11}{01} = \phi$, $\frac{101}{11} = \phi$ → 11 don't match with 01

→ Some of the decidable problems of the context free languages are Emptiness, finiteness, Membership

1> Membership- CYK Algorithm

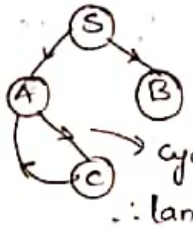
2> Emptiness — If you find starting symbol is useless then the lang is Empty.

3> Finiteness — Take Grammar 'G' and apply simplification and draw dependency Graph, If Graph has cycle then the lang is Infinite.

Say, the simplified Grammar is

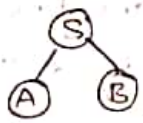
$S \to AB$
$A \to aC \mid a$
$C \to aA \mid b$
$B \to a$

Dependency Graph



∴ lang is Infinite

$S \to AB$
$A \to a$
$B \to b$



↳ No cycle ⇒ the language is finite

PROPERITIES OF REGULAR LANGUAGES

CLOSED UNDER UNION, INTERSECTION, CONCATENATION, COMPLEMENTATION, KLEN CLOSURE

1> $L_1 \cup L_2$ ⇒ L_1 has Regular expression R_1

L_2 has Regular expression R_2

∴ $L_1 \cup L_2 = [R_1 + R_2]$ = RE (Regular language)

2> $L_1 \cap L_2$ ⇒ $L_1 \to R_1$, $L_2 \to R_2$

$L_1 \cap L_2 = [R_1 \cdot R_2]$ = Regular Language

3. $L_1 \cdot L_2$ =

→ Some of the decidable problems of the context free languages are Emptiness, finiteness, Membership

1> Membership- CYK Algorithm

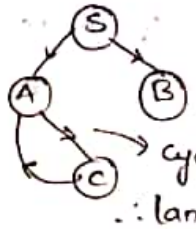
2> Emptiness — If you find starting symbol is useless then the lang is Empty.

3> Finiteness — Take Grammar 'G' and apply simplification and draw dependency Graph, If Graph has cycle then the lang is Infinite.

Say, the simplified Grammar is

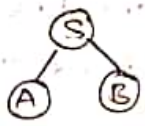
$S \to AB$
$A \to aC \mid a$
$C \to aA \mid b$
$B \to a$

Dependency Graph



∴ lang is Infinite

$S \to AB$
$A \to a$
$B \to b$



↳ No cycle ⇒ the language is finite

PROPERITIES OF REGULAR LANGUAGES

CLOSED UNDER UNION, INTERSECTION, CONCATENATION, COMPLEMENTATION, KLEN CLOSURE

1> $L_1 \cup L_2$ ⇒ L_1 has Regular expression R_1

L_2 has Regular expression R_2

∴ $L_1 \cup L_2 = [R_1 + R_2]$ = RE (Regular language)

2> $L_1 \cap L_2$ ⇒ $L_1 \to R_1$ $L_1 \cap L_2 = [R_1 \cdot R_2]$ = Regular Language

$L_2 \to R_2$

3. $L_1 \cdot L_2$ =

4 L^* $\Rightarrow$ R_1 is the RE for L_1

R_1^* is the RE of L_1^* $\therefore$ R_1^* is Regular Expression

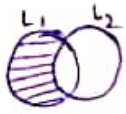
$\therefore$ Regular languages are closed under closure

5 $\overline{L_1} = \Sigma^* - L_1$ $\Rightarrow$ Language L_1 has DFA [Given L_1 is Regular]

$\Rightarrow$ Complement the DFA then we get DFA accepting $\overline{L_1}$

$\Rightarrow$ Since L_1 has DFA it is Regular

6 closed under difference and Reversal



$L_1 - L_2$ $\Rightarrow$ $L_1 - L_2$ can be written as $L_1 - L_2 = L_1 \cap \overline{L_2}$

$= RL \cap RL$

$= RL$

L_1 is regular language $\Rightarrow$ L_1^R should be Regular

$\therefore \Rightarrow L_1 \rightarrow$ has DFA

$\Rightarrow$ Reverse the DFA then we get DFA accepting Reverse of the language

7 closed under Homomorphisim and Inverse Homomorphisim

↳ function $h: \Sigma \rightarrow \Gamma^*$ ↳ $h^{-1}(w) = \{x / h(x) \in L\}$

$\Sigma = \{a, b\}$ $\Gamma = \{0, 1, 2\}$

$h(a) = 01$, Let $L = a^*b$

$h(b) = 112$ $= (01)^* 112$ $\Rightarrow$ Regular Expression

$\therefore$ closed homomorposim

$\Sigma = \{a, b\}$ $\Gamma = \{0, 1, 2\}$

Let $h_1 = \{ababa\}$ $\underline{ab}\,\underline{ab}\,\underline{a}$ $\quad \underline{a}\,\underline{b}\,\underline{a}\,\underline{b}\,\underline{a}$

$1\ 1\ 0$ $\quad 1$

$h(0) = a$

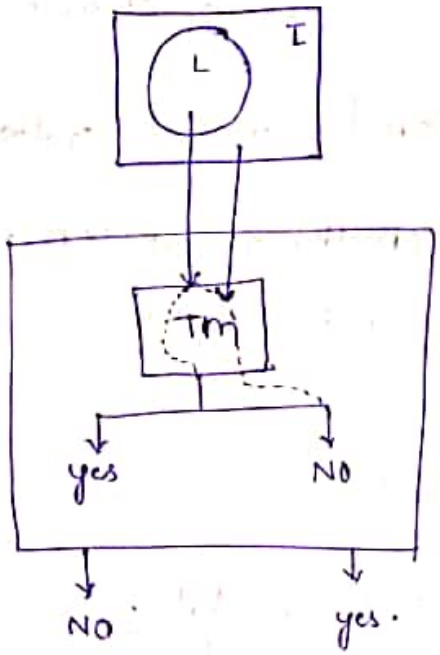
$h(1) = ab$ $\quad h^{-1}(L_1) = \{110, 022, 102\}$

$h(2) = ba$

8. NOT CLOSED UNDER INFINITE UNION.

→ The compliment of Recursive language is Recursive

Say there is a Recursive Language then it means there is a HTM



TM ⇒ This TM just works in opposite way. which means if the inner TM says NO (the string is not present in L) then the outer TM says yes the string is present in L̄ and viceversa.

∴ we got a halting TM that halts for L̄ also. ∴ THE COMPLEMENT OF RECURSIVE LANGUAGE IS RECURSIVE.