16

Theory of Computation

Sep

Marks 8 to 13

- Finite Automata
- Regular Expressions & Regular language
- grammer
- Push down Automata
- Turing Machine
- undecidability

- Reference books:

  Finite Automata.

→ By
- Ullman
- John.C. martin
- michael sipres
- Peter linz

- Finite Automata
  - DFA & NFA
  - Equivalence b/w 2 automata
  - Minimization of DFA
  - Elimination of null moves
  - Construction of DFA's
  - Construction of NFA.
  - mealy & moore machine Conversion & design
  - A language is Regular or not?

$a^m / m \geq 0$ will always be whole no. or only +ve Integer.

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

$\Rightarrow \Sigma^* = \{a, b\}^*$

↑ if you apply Klnee closure then you need to apply also in $\{a, b\}^*$

Language : Set of String may be finite or infinite

$L = \{\omega_1, \omega_2, \omega_3 \ldots\ldots \text{-- finite/infinite}\}$

String : ab, abb, ba, baa

- Input Alphabet
- there is no prescribed define" for i/p Alphabet but it must have finite no. of element

ex: $\Sigma = \{a, b, c\}$

$|\Sigma|$ = The cardinality of Σ

i.e The no. of element in Σ

$|\Sigma| = 3$

- String
- A String is any finite sequence of or Combination over I/p Alphabet

- language

A language is set of string may be finite or infinite

Ex:

$L = \{\omega_1, \omega_2, \omega_3 \ldots$ --- finite/infinite$\}$

- Operations on strings

1. Concatenation

Let $u = ab$ $v = abb$

$u.v = ababb$

$|u| = 2$

$|v| = 3$

$|u.v| = |ababb|$

$= 5$

$= |u| + |v|$

$v.u = abbab$

$u.v \neq v.u$ } except uniary operatn

i.e

$u = bbb$ $u = bb$

$uu = uv$

$|uu| = |vu|$

not always.

and sometime on few example

$u = ab$ $v = abab$

$u.v = v.u$

- only
- we say that $uv = v.u$ only when it is true for all cases.

- Concatenation is not closed under commutative
- concatenation is closed under associative

i.e $(u \cdot v) \cdot w = u \cdot (v \cdot w)$

- **Length of a string**

- It is the no. of symbols in the string

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

- cardinality of Σ, i.e $|\Sigma| = 3$

- the string of length zero in 'ε', $|\epsilon| = 0$ ⟶ the no. of string of length '0' is $|\Sigma|^0$ i.e $3^0 = 1$

- the string of length 1 are a, b, c ⟶ the no. of string of length '1' is $|\Sigma|^1$ i.e $3^1 = 3$

- the string of length 2 are

aa, ab, ac
ba, bb, be
ca, cb, cc

⟶ the no. of string of length '2' is $|\Sigma|^2$; $3^2 = 9$

no generalⁿ

$1 + 1 3 + 1 3$

3 possibility

The no. of string of length 'n' is $|\Sigma|^n$

no. of input [ ] length with the no. of possibility

- **reverse of a string (Transpose)**

Let $u = abb$ ⟶ $u^R = bba$

$v = ab$ ⟶ v^R or $v^T = ba$

$uv = abbab$

$(uv)^T = babba$

$v^T u^T = babba$

$= (uv)^T$

$v^T v^T = bbaba$

$vu = abab b$

$(vu)^T = bbaba$

$(uv)^T = v^T u^T$

$(uvw)^T = w^T v^T u^T$

- Power of a string

$\omega^0 = \epsilon$ [$\omega^0 \neq \Lambda$]

$\omega^1 = \omega$

$\omega^2 = \omega \cdot \omega$

$\omega^3 = \omega \cdot \omega^2 = \omega^2 \cdot \omega = \omega \cdot \omega \cdot \omega$

$\omega^{m+n} = \omega^m \cdot \omega^n$

let $\omega = ab$

$\omega^2 = (ab)^2 = ab \cdot ab$

$(ab)^2 \neq a^2b^2$

& $(ab^2) = (ba)^2$

- **Kleen's closure** $[\epsilon \notin \Sigma]$ i.e $\Sigma \neq \{\epsilon, a, b\}$ ✗

$\Sigma = \{a, b\}$ ✓

$\Sigma = \{0, 1\}$ ✓

Let Σ is the input alphabet

i.e $\Sigma = \{a, b, c\}$

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

$= \{\epsilon, a, b, c, aa, ab, ac, ba, \ldots\}$

positive closure

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots$

$= \{a, b, c, aa, ab, ac, ba \ldots\}$

$\Sigma^+ = \Sigma^* - \{\epsilon\}$ always true

- **prefix of a string**

▸ Taking zero or more symbols from left to right

Let $\omega =$ abbab

Prefixes(ω) = $\{\epsilon, a, ab, abb, abba, abbab\}$

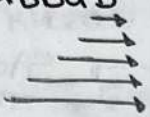
→ proper prefix we don't include these

→ If the length of the str in n ⇒ the no of prefixes will be

n+1

- Suffix of a String : Taking zero or more symbol from right to left in forward direction

$\omega = abbab$



Suffix will be n+1

Suffix $(\omega) = \{\epsilon, b, ab, bab, bbab, abbab\}$

In proper suffix we don't include these

// Size of a proper suffix is exactly to the size of the element. "n" (excluding ω)

Proper Suffix & Proper prefix length $= |\omega|$

$Prefix(\omega) \cap suffix(\omega) \supseteq \{\epsilon, \omega\}$

Some time you may get other than ϵ & ω as common part.

- Substring:

Taking zero or more Continous part of the String

Let $\omega = abbab$

Substring $(\omega) = \{\epsilon, a, b, ab, bb, ba, abb, bba, bab, abba, bbab, abbab\}$

$Prefix(\omega) \cup suffix(\omega) \subseteq Substring(\omega)$

- Language :

$L = \{\epsilon, a^2, a^4, a^6, a^8, a^{10} \ldots\ldots\}$ → Listing

↳ L is the set of all string of even no. of a's → Statement

$L = \{a^{2n} / n \geq 0\}$ → set builder form

these are informal way of representation of a language

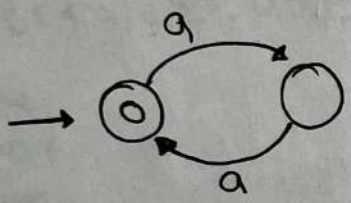
$a^0 = \epsilon$

$a^{2\cdot 1} = a^2$

$a^{2\cdot 2} = a^4$

$\vdots$

$L = (aa)^*$ → Regular Expression



→ machine

$\{S \rightarrow aSa/\epsilon\}$ → Grammer

Formal ways of representation of a language.

$L = \{\epsilon\}$ is the language having null string, here $|L| = 1$

$L = \{\}$ is the language with no string ; i.e $L = \phi$, $|L| = 0$

$\phi = \{\} \Rightarrow \phi^* = \{\}$

$\{\epsilon\}^* = \{\epsilon\}$

$\epsilon \cdot \epsilon = \epsilon$

$\epsilon \cdot \{a,b\} = \{a,b\}$

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

$\{\} \cdot \epsilon \cdot \{\} = \{\}$

ϕ with anything $= \phi$

means no string accepted i.e no final state.

means there is a string of length 0 & it is accepted (there is a final state)

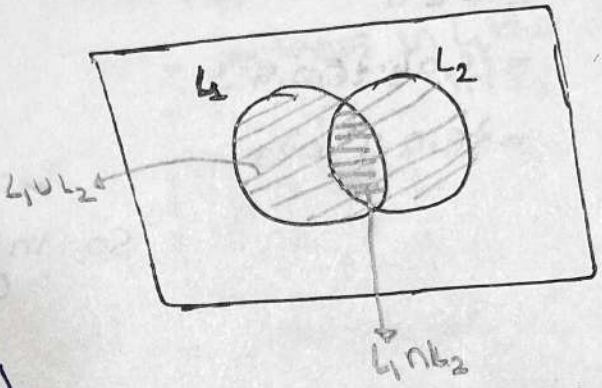
It is but we don't say that.

$\phi^0 = \epsilon$

$\phi^2 = \phi$

- Operation on language
  - union, Intersection, Concatenation,
  - Reversal, Kleen's closure, positive closure
  - Complement, Exclusive OR (⊕OR), Exclusive NOR (⊕NOR), Set difference
  - Implication, NAND, NOR

$$|L_1 \cup L_2| = |L_1| + |L_2| - |L_1 \cap L_2|$$



// there is no formula for $L_1 \cap L_2$

1. $L_1 \cdot L_2 \Rightarrow L_1 = \{a, b\}$ & $L_2 = \{aa, ab, b\}$

   then $L_1 \cdot L_2 = \{aaa, aab, ab, baa, bab, bb\}$

   $$|L_1 \cdot L_2| \leq |L_1| \cdot |L_2|$$

2. $\epsilon \cdot x = x$ $[\epsilon \cdot \epsilon = \epsilon]$

   i.e $L_1 = \{\epsilon, a, bb\}$ & $L_2 = \{a, aa\}$

   $L_1 \cdot L_2 = \{a, aa, aaa, bba, bbaa\}$

   $$|L_1 \cdot L_2| \leq |L_1| * |L_2|$$

3. $L = \{abb, babb, abab\}$

   $L^R = \{bba, bbab, baba\}$

4. Kleen's closure / star closure.

   $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \ldots$

   $L^0 = \{\epsilon\}$, $L^1 = L$, $L^2 = L \cdot L, \ldots$

   positive closure

   $L^+ = L^1 \cup L^2 \cup L^3 \ldots$

   $L^* \neq L^+ - \{\epsilon\}$ ✗ not true

let $L = \{\epsilon, a\}$

$L^0 = \{\epsilon\}$, $L^1 = L$;

$L^2 = L \cdot L$

$= \{\epsilon, a\} \cdot \{\epsilon, a\}$

$= \{\epsilon, a, a^2\}$

$L^3 = L \cdot L^2$

$= \{\epsilon, a\} \cdot \{\epsilon, a, a^2\}$

$= \{\epsilon, a, a^2, a^3\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \ldots$

$= \{\epsilon, a, a^2, a^3, a^4, \ldots\}$

$L^+ = L^1 \cup L^2 \cup L^3 \ldots$

$= \{\epsilon, a\} \cup \{\epsilon, a, a^2\} \cup \{\epsilon, a, a^2, a^3\} \ldots$

$= \{\epsilon, a, a^2, a^3, a^4 \ldots\}$

So, in these case $L^+ = L^*$ → if L contain ϵ

i.e $\{\epsilon, a\} = L$

if $L = \{a\}$; $L^2 = L \cdot L$; $L^3 = L \cdot L^3$

$\{a\} \cdot \{a\}$ $\Rightarrow \{a\} \cdot \{a^2\}$

$L^0 = \{\epsilon\}$ $= \{a^2\}$ $\Rightarrow \{a^3\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \ldots$ & $L^+ = L^1 \cup L^2 \cup L^3 \cup L^4 \ldots$

$= \{\epsilon, a, a^2, a^3, a^4 \ldots\}$ $\{a, a^2, a^3, a^4, \ldots\}$

So, in these case $L^+ = L^* - \{\epsilon\}$

→ if L doesn't Contain 'ϵ'

i.e $\{a\} = L$

- Complement of a language.

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

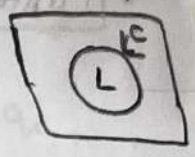
$\Sigma^* = \{\varepsilon, a, b, a^2, ab, \ldots\}$

L is defined over Σ

$\bar{L}$ or $L^c = \Sigma^* - L$

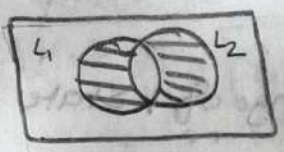
$\bar{L} = \Sigma^* - L$

So, $\boxed{L + L^c = \Sigma^*}$



- Exclusive OR (XOR)

$L_1 \oplus L_2$ is belongs to L_1 (or) L_2 but not both



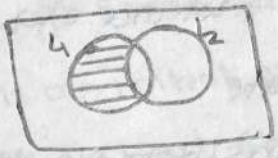
So, $L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$

$= (L_1 \cap \bar{L_2}) \cup (L_2 \cap \bar{L_1})$

$\boxed{|L_1 \oplus L_2| = |L_1| + |L_2| - 2|L_1 \cap L_2|}$

- Set difference

$L_1 - L_2$ in those in "L_1" but not in "L_2"



$L_1 - L_2 = L_1 \cap \bar{L_2}$

- NAND:

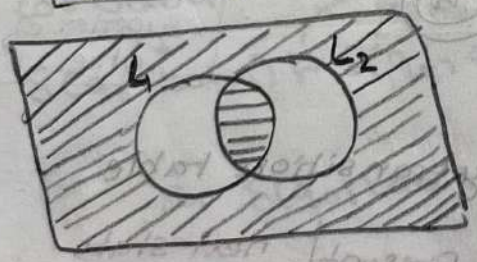
$\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$

- NOR:

$\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$

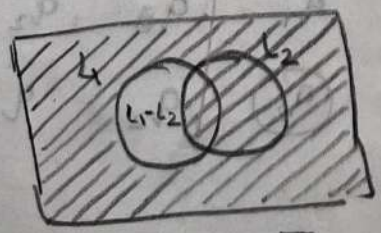
- Exclusive NOR (XNOR)

$\boxed{L_1 \longleftrightarrow L_2 = (L_1 \cap L_2) \cup (\bar{L_1} \cap \bar{L_2})}$



- Implication:

$\boxed{L_1 \longrightarrow L_2 \quad \bar{L_1} \cup L_2}$



$L_1 - L_2 = L_1 \cap \bar{L_2}$

- **Finite Automata.** It is a 5-tuple

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

- Q: Finite non-empty (set of states) — Can't be empty
- Σ: Finite set of input
- δ: Transition Functn, $\delta : Q \times \Sigma \to Q$
- q_0: Initial State, $q_0 \in Q$
- F: Set of final State, $F \subseteq Q$ [F can be empty] — may be empty

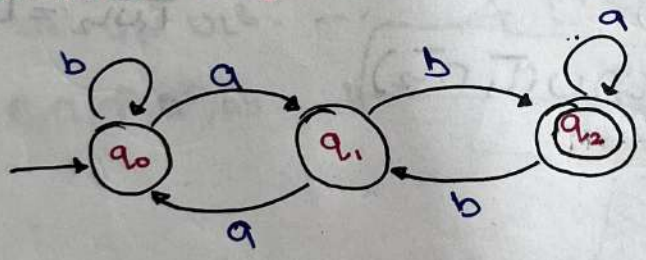
- At any instance of a time, the automata will be in only one state
- the transition function describe the change of a state during the transition
- Input signal scan one symbol at a time. doesn't scan more than 1 symbol at any point of time.

FA is by default NFA

NFA & DFA have equal powers no one is superior of other

- Representation of finite automata

By Transition Diagram :-



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

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

q_0 is the initial state

q_2 is the final state

By transition table

Present State	next state a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
((q_2))	q_2	q_1

By 5-Tuple

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

(Q, Σ, q_0, F)

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

$\delta(q_1, b) = q_2$ & $\delta(q_2, a) = q_2$ & $\delta(q_2, b) = q_1$

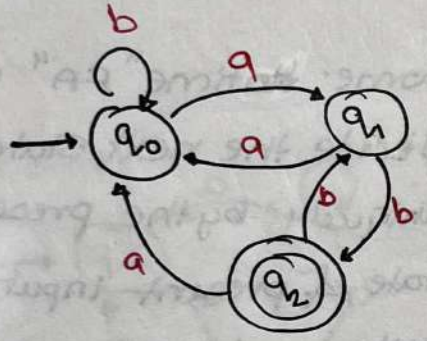
- Acceptance By finite automata (FA):

A String $\omega \in \Sigma^*$ is said to be accepted by FA if it start at initial state & terminates at some final state.

i.e $\boxed{\delta(q_0, \omega) = q_f}$ where q_0 is an initial state. & q_f is an final state

(ω = String)

Ex:



$\omega = baba$

① $\delta(q_0, baba) \vdash \delta(q_0, aba) \vdash \delta(q_1, ba) \vdash \delta(q_2, a) \vdash \delta(q_0, \epsilon)$ (next transition)

So, $\delta(q_0, baba) \vdash^* q_0 \notin F \Rightarrow$ baba not accepted

② $\delta(q_0, abbb) \Rightarrow q_2 \in F \Rightarrow$ abbb is accepted.

- Non Deterministic finite State Machine (NDFSM / NDPA / NFA)

when it is not mention anywhere NFA (min state automata) to regular expression

- It is a Stuple

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

q_0 → Initial state, $q_0 \in Q$

F → set of final state, $F \subseteq Q$

Q → finite non empty set of states

Σ → finite set of I/ps

δ → Transition functn

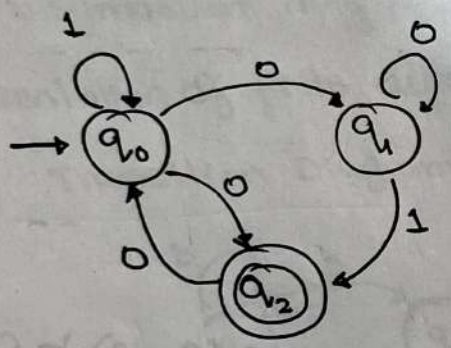
$\delta : Q \times \Sigma \longrightarrow 2^Q$

2^Q → The power set of Q

i.e the set of all subset of Q.

For an I/p if there are 2 possibility or more or no possbility then its NFA

ex:



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

$2^Q = \{\emptyset, q_0, q_1, q_2, \{q_0, q_1\}, \{q_1, q_2\}, \{q_0, q_2\}, \{q_0, q_1, q_2\}\}$

- Some time "FA" can't decide the next State uniquely by the present State & present Input Such machine are called NFA

gmp

$L(\bar{M}) = \overline{L(M)}$ → is true in DFA but not in NFA

$L(\bar{M})$ → the machine is getting Complemented So, final become non final & non-final become final

$\overline{L(M)}$ → whatever langⁿ it was accepting it's getting complemented

Complementary machine accept Complementary language

- Deterministic finite state Machine (DFSM/DFA)

It is a 5-tuple $M=(Q, \Sigma, \delta, q_0, F)$

- Q: finite non-empty set of state
- Σ: finite set of i/ps
- δ: Transition function $\delta: Q \times \Sigma \longrightarrow Q$
- q_0: initial state, $q_0 \in Q$
- F: set of final states, $F \subseteq Q$

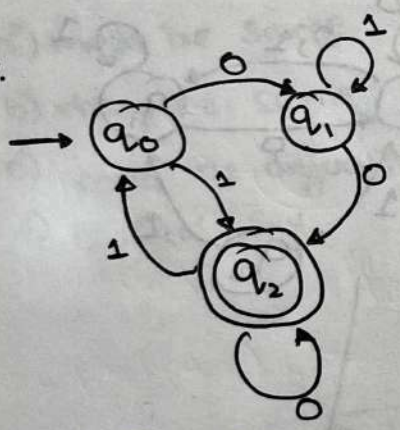
- A "FA" is said to be "DFA" if it satisfies the following two condition

Condition

1. Every state must be defined for every input.
2. At every state for every input the o/p state must be single

// every FA is a NFA & DFA

ex:



it is a DFA

note

:. Every DFA is also NFA but every NFA is not a DFA

- For every NFA there exists an equivalent DFA

i.e NFA ≃ DFA

- In DFA table if the table contain empty slot so it will go to Dummy/Dead/Trap state that you can create

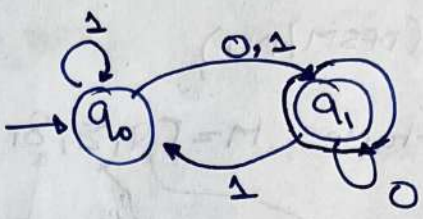
↳ if $F=\phi$ then $L(M)=\phi$ (in both nfa & DFA) but not true for vice versa

↳ if $F=Q$ then $L(M)=\Sigma^*$ (in dfa)

note

- There is a unique minimal DFA for every regular language
- 2-way DFA (2-DFA): it can go back & forth in the word during the computation.
- It behave same as DFA (in terms of power)

Q] construct a DFA equivalent to the given NFA



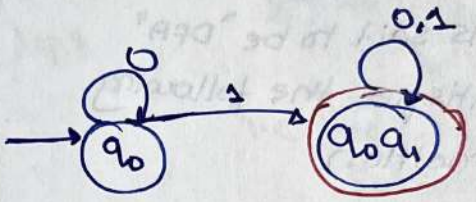
Present State	next state 0	1
→ q_0	q_0	q_0, q_1
(q_1)	q_1	q_0

[NFA]

Convert NFA to DFA

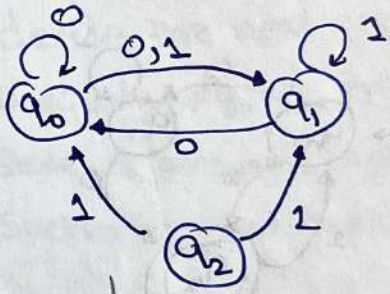
for a given NFA this is the DFA

Present State	next state 0	1
→ q_0	q_0	q_0q_1
- (q_0q_1)	q_0q_1	q_0q_1



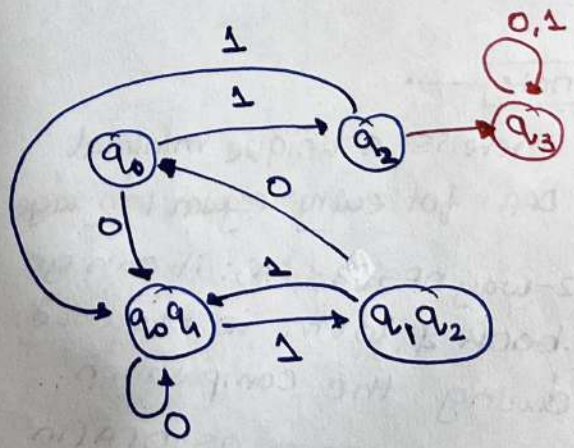
Q]

Present State	next state 0	1
→ q_0	q_0q_1	q_2
q_1	q_0	q_1
(q_2)	-	q_0q_1



[NFA]

Present State	next state 0	1
→ q_0	q_0q_1	q_2
q_0q_1	q_0q_1	q_1q_2
(q_2)	q_3	q_0q_1
(q_1q_2)	q_0	q_0q_1
[Dead/Trap Dummy] → q_3	q_3	q_3



Q] the no. of states of a DFA equivalent to NFA of 'n' states is

(a) always more (>n)
(b) always less (<n)
(c) always has 2^n state
(d) Sometime has less no. of state

minimum DFA | DFA | NFA
2 | 2
4 ≃ 6 ≃ 5

DFA	NFA
n	n
>n	
<n	
2^n (max)	

Note

- the max no. of states of a DFA equivalent to NFA of n state is 2^n

Q] the acceptability of NFA & DFA for any language is

a) must be same
b) may be same
c) must be different
d) Can't say

	DFA	NFA
L_1	✓	✓
L_2	✓	✗
L_3	✗	✗
L_4	✗	

every DFA = NFA

can create an equivalent DFA

Q] The max. no. of final state of DFA equivalent to NFA of 10 State out which 3 are the final states is

a) 2^3
b) 7×2^3
c) 3×2^7
d) 7×2^7

NFA ≃ DFA
10 state → 2^{10} max (F+NF)
(3F + 7NF)
7NF → 2^7 max NF (non final)

F + NF = 2^{10}
$F + 2^7 = 2^{10}$
$F = 2^{10} - 2^7$
$= 2^7 (2^3 - 1)$
$\Rightarrow 7 * 2^7$

10 states

$q_1, q_2, q_3 \ldots q_4, q_5 \cdots q_{10}$

3F — 7NF

$\Downarrow$ $\Downarrow$

$(2^3 - 1)F$ — $(2^7)NF$

7F

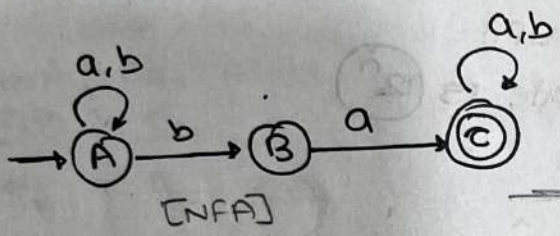
7×2^7 Max final State.

ϕ X

q_1,
q_2,
q_3,
q_1q_2,
q_1q_3,
q_2q_3,
$q_1q_2q_3$

7F

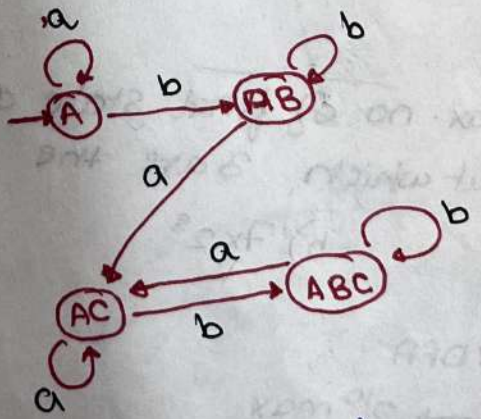
Pg. 27
Q 89.



[NFA]

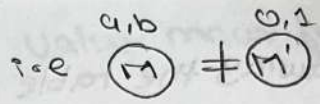
Present State	next state a	b
→ A	A	A,B
B	C	—
(C)	C	C

Present State	next State a	b
→ A	A	A,B
AB	AC	AB
(AC)	AC	ABC
(ABC)	AC	ABC



[equivalent DFA]

- Equivalence b/w two automatas:

i.e 

- 2 automata are said to be equal if they accept same set of string

i.e $M \equiv M'$ $\quad M \neq M'$

ω_1 ✓ ✓

ω_2 ✓ ✓

ω_3 × ×

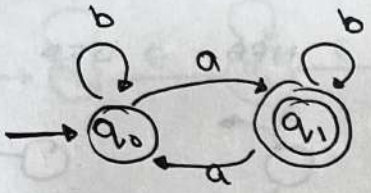
ω_4 × ×

ω_5 ✓ ×

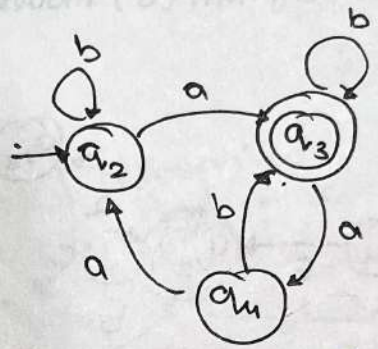
▸ no. of state b/w 2 can differ. [need not be same]

- 2 automata are said to be equal if they accept same set of string.
- Here the two automata must be define over same i/p alphabet

ex:



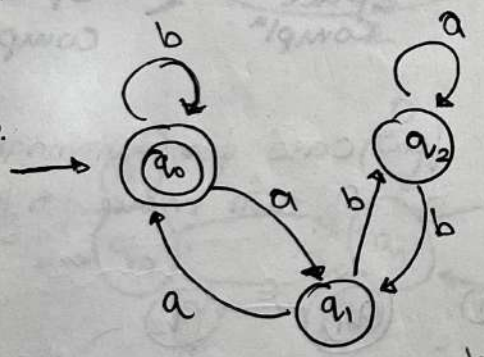
≠ not equal



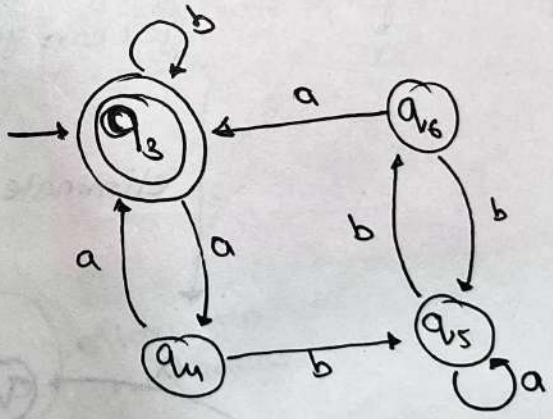
✓ aaa

aaa ×

eg:



both are equal =



(q,q')	a (q_a,q_a')	b (q_b,q_b')
→ (q_0,q_3)	(q_1,q_4)	(q_0,q_3)
(q_1,q_4)	(q_0,q_3)	(q_2,q_5)
(q_2,q_5)	(q_2,q_5)	(q_1,q_6)
(q_1,q_6)	(q_0,q_3)	(q_2,q_5)

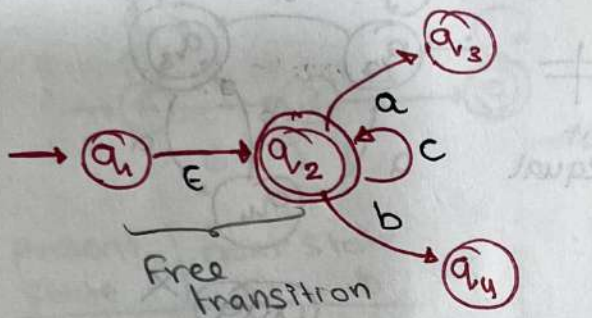
(Final, final) } then continue
(non final, non final) }

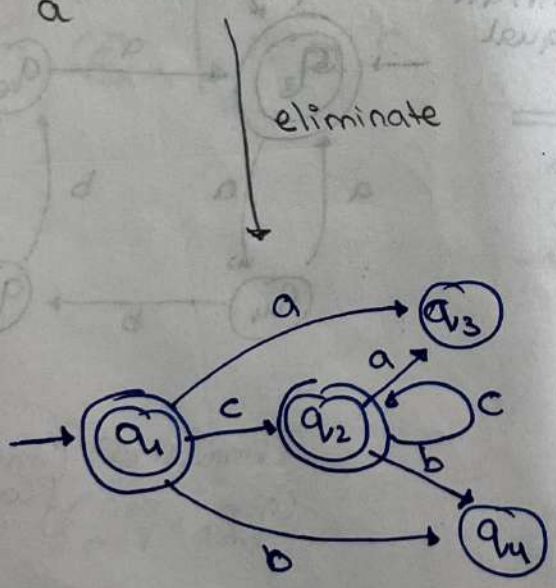
(Final, non final) } then stop, & we can say that these both are not equal.

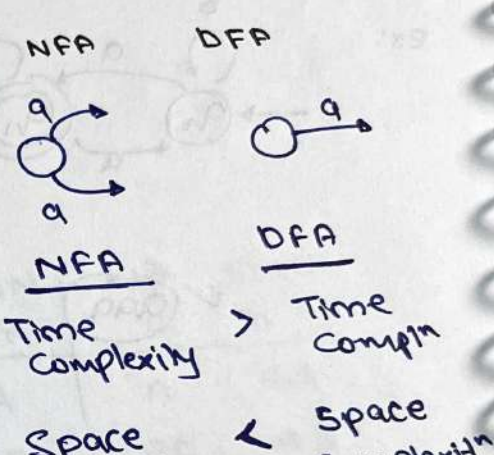
(NF, F) }

- while Drawing the table if you get a state of (F,F) or (NF,NF) then continue or if you get (NF,F) or (F,NF) then stop and now you can say that both are not equal.
- continue until you find one pair of (NF,F) or (F,NF) if you don't find then it is equal.

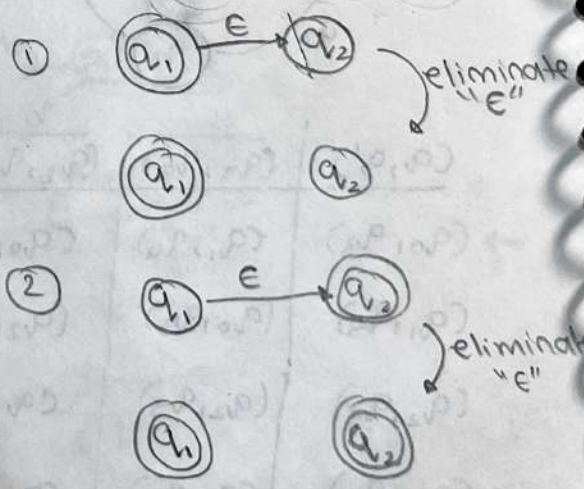
- Elimination of null (ϵ) moves:-







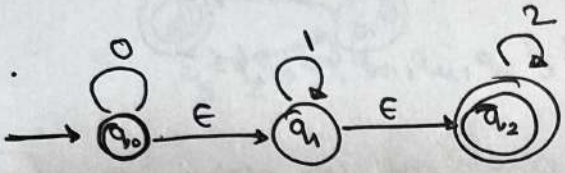
few case for elimination of null move for final state



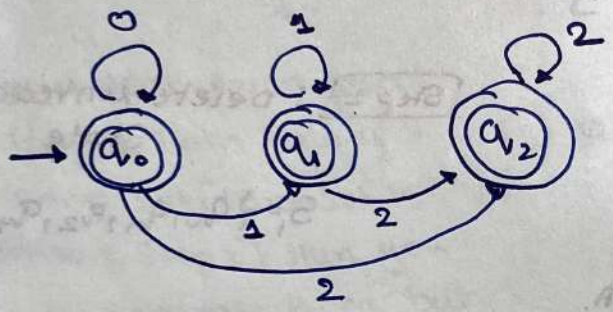
- Suppose we want to delete a null value move from q_1 to q_2 then proceed as follows

1. Identify all the transactions that are defined at Q_2 State
2. Duplicate all the transaction at Q_1
3. If Q_2 is final then make Q_1 also as final.

ex:

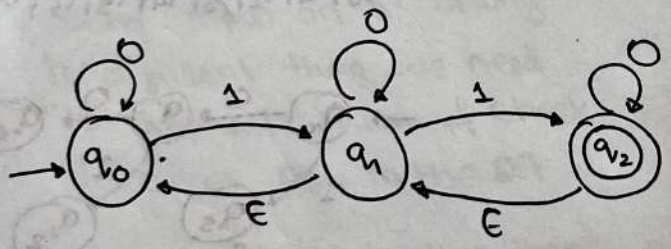


eliminated null move

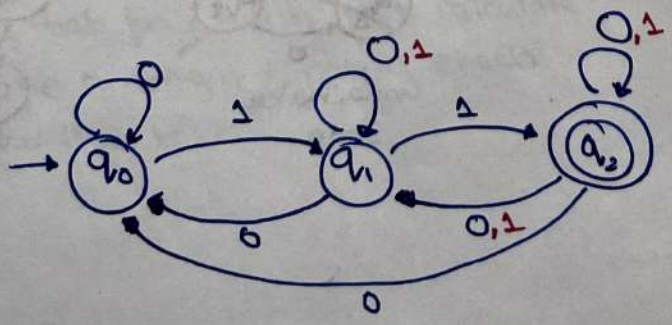


- for a given language
  - ↳ DFA NOT Unique
  - ↳ NFA Not unique
  - ↳ minimal DFA is unique
  - ↳ minimal NFA is not unique
  - ↳ no. of State in min. NFA is unique.

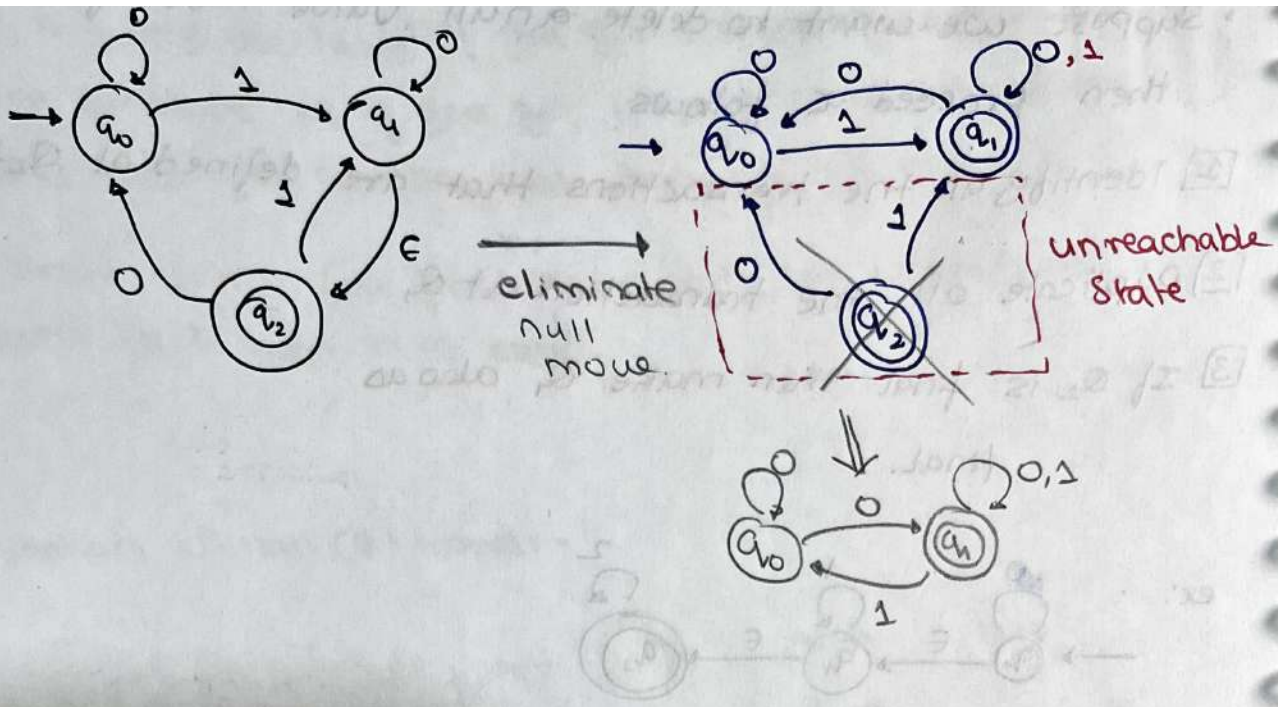
ex:



eliminated null move



ex:



- Minization of a DFA:
- It is the process of reducing no. of state if possible
  - min. state is 1 not 0

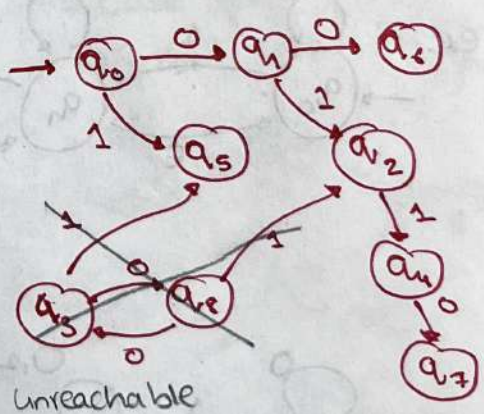
ex: Construct a minimal DFA equivalent to the given DFA.

Present State	next state 0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_2	q_4
q_3	q_8	q_5
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_0
q_7	q_6	q_2
q_8	q_3	q_2

Step 1 : Delete unreachable State

$S_1 = \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$

i.e $q_0, q_1, q_2, \cancel{q_3}, q_4, q_5, q_6, q_7, \cancel{q_8}$



Step 2 : seperate final & non final state

$S_2 = \{q_2\}, \{q_0, q_1, q_4, q_5, q_6, q_7\}$

Step 3 : seperate those state that are directly going to the final state [for these check the next state ; if the final state is present in the next state that means it is directly connected].

$S_3 = \{q_2\}\{q_0, q_4, q_6\}\{q_1, q_5, q_7\}$

Step 4 : In each set for each pair of state check whether they are 0-equivalent and also 1-equivalent or not?

- 0-equivalent mean for a "0" i/p if both belong to same set [these set is from the step 3] then you can't separate them but if both input on "0" belong to different then we need to separate them & check for other pair in the set
- also if one pair is together then no need to check each element of that pair with just calculate one among the pair & check for the rest.

after doing step (4) we will again check for step 4.

(1) if in a case if there are 2 set suppose $\{q_0\ q_1\}$ $\{q_2\ q_3\ q_4\}$. if q_0 belong to other set for any value then q_1 should also belong to that set for that value.

∴ for a given input on any set all the state of that set should belong to same set for a particular input (it is not necessary that all state should belong to same state of other state).

$S_4 = \{q_2\}\{q_0, q_4\}\{q_6\}\{q_1 q_7\}\{q_5\}$

q_0 & q_4 are 0 equivalent & also 1-equivalent

q_0 & q_6 are not 0-closure

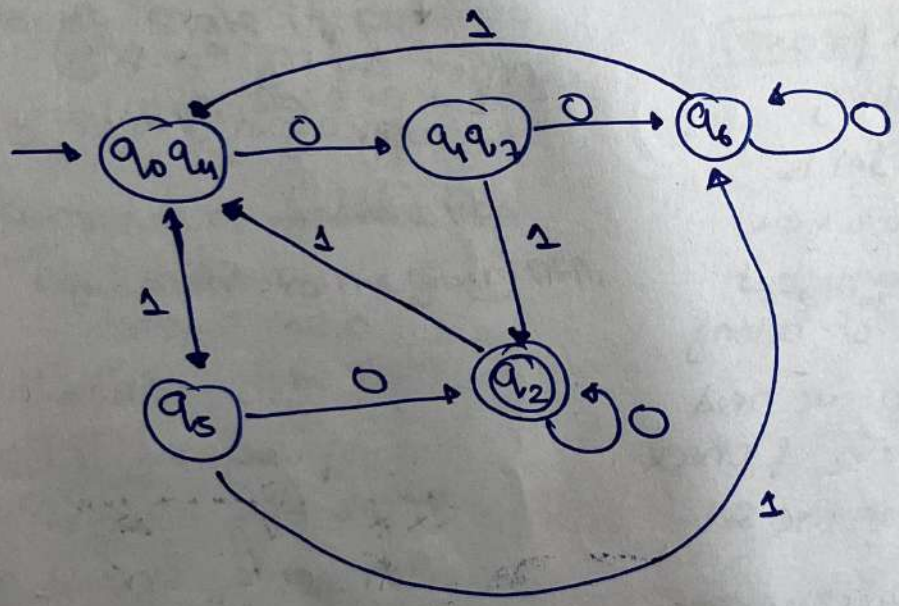
q_1 & q_7 are not 0-closure

q_1 & q_7 are 0-equivalent & 1-equivalent

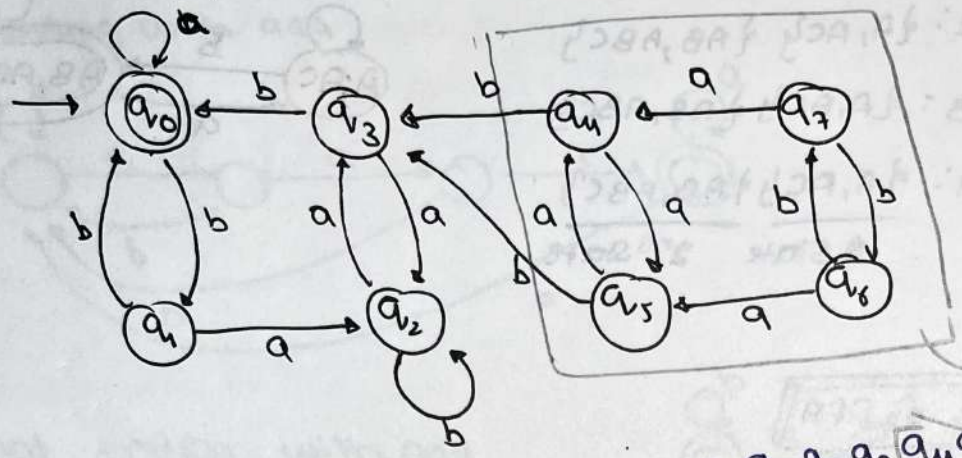
$S_7 = \{q_2\}\{q_0, q_4\}\{q_6\}\{q_1, q_7\}, \{q_5\}$

q_0 & q_4 are 0-equivalent &

q_1 & q_7 are



8] the no. of state in the minimal DFA equivalent to given DFA



unreachable state

Present State	next state a	b
q_0		
q_1		
q_2		
q_3		
q_4		
q_5		
q_6		
q_7		

don't build the table Waste of time

$S_1 = q_0 q_1 q_2 q_3 q_4 q_5 q_6 q_7$

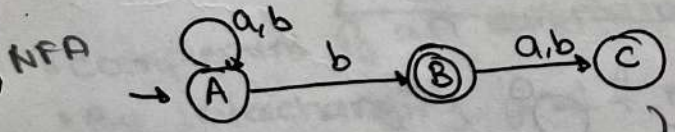
$S_2 = \{q_0\}\{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

$S_3 = \{q_0\}\{q_1, q_3\}\{q_2, q_4, q_5, q_6, q_7\}$

$S_4 = \{q_0\}\{q_1\}\{q_3\}$

$S_4 = \{q_0\}\{q_1 q_3\}\{q_2\}$

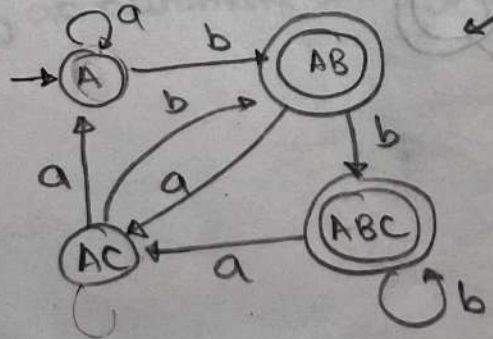
188] consider a NFA



	a	b
→ A	A	AB
B	C	C
C	–	–

↓ NFA to DFA

to DFA



	a	b
→ A	A	AB
AB	AC	ABC
AC	A	AB
ABC	AC	ABC

DFA

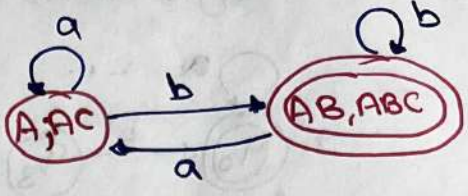
S_1: {A, AB, AC, ABC}

S_2: {A, AC} {AB, ABC}

S_3: {A, AC} {AB, ABC}

S_4: {A, AC} {AB, ABC}

1 State 2nd State



Construct^n of DFA

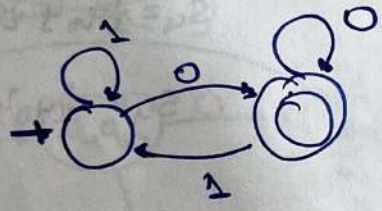
Q] Construct a minimal DFA over {0,1} that accept all

(a) String ending with '0'

xxxxx — 0 ✓

— 1 ✗

00
0010
110
⋮



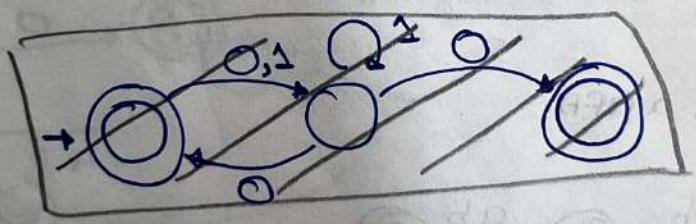
note

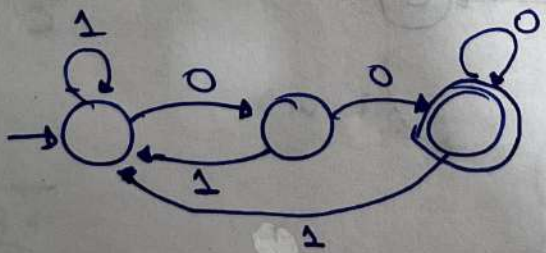
- always 1st find Smallest string
- the length of the Smallest string +1 state will be there

if n = smallest string

then no. of State = n+1

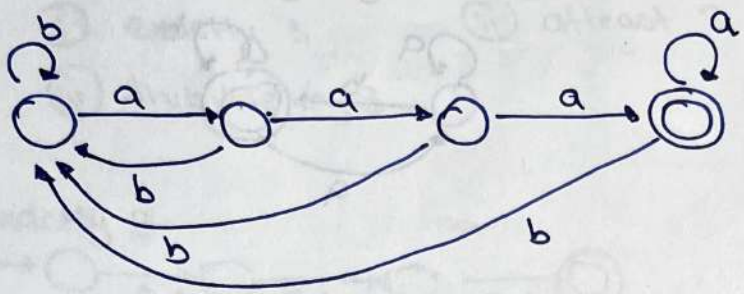
(b) if String ending with "00" ?



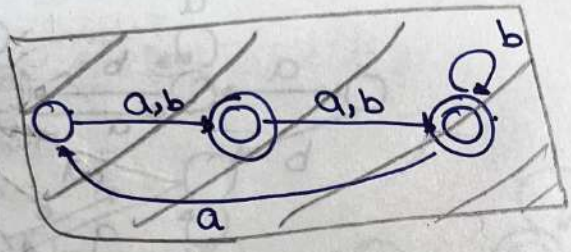


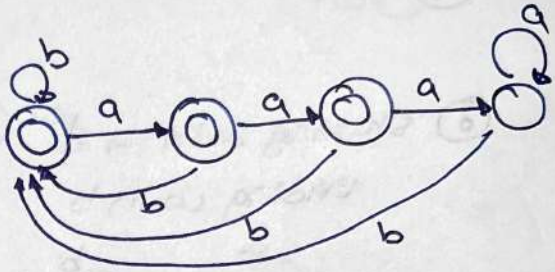
Q] construct the minimal DFA over {a,b} that accept string

(i) ending with aaa



(ii) not ending with aaa





↓

these is compl^n of "aaa" automata means

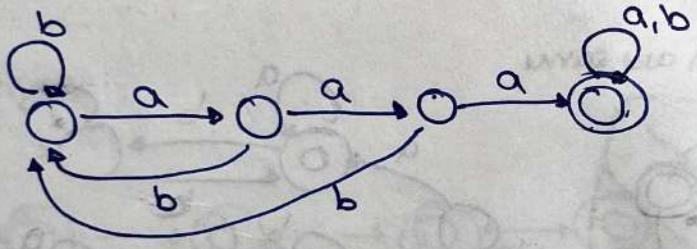
$\Sigma = \{a,b\}$

Σ^*

$L_1 =$ ___aaa

$L_2 = \Sigma^* - L_1$

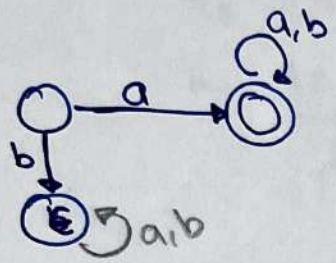
(iii) having substring aaa



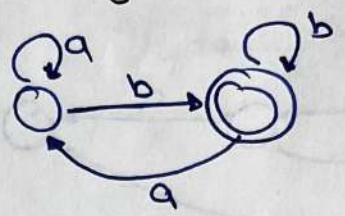
- complement of an automata.
- By interchanging final & non-final state the complement of an automata will be formed

Q] Construct a minimal DFA over {a,b} that accepts all string.

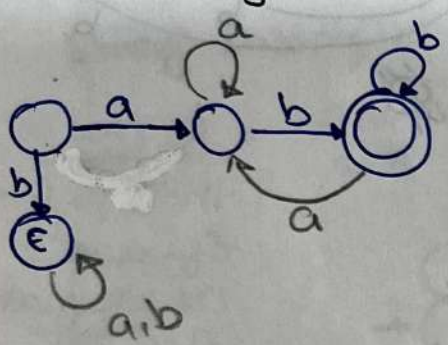
(i) Starting with a



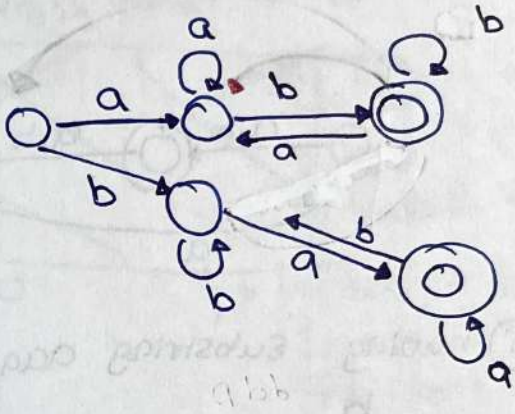
(ii) ending with b



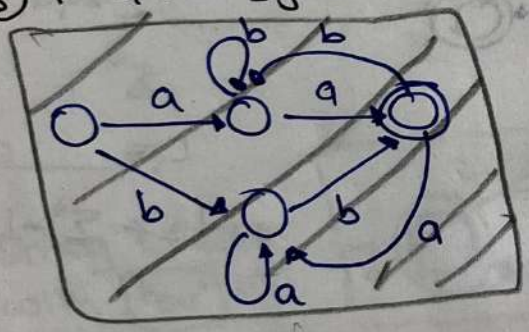
(3) Starting with a & ending with b

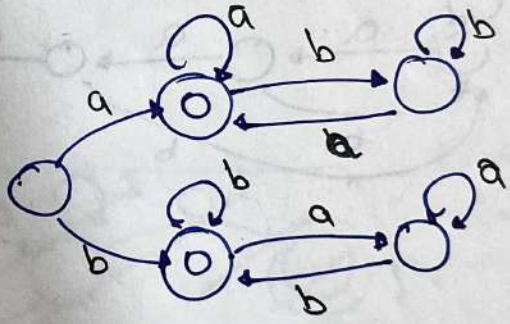


(4) having 1st & last symbol are different

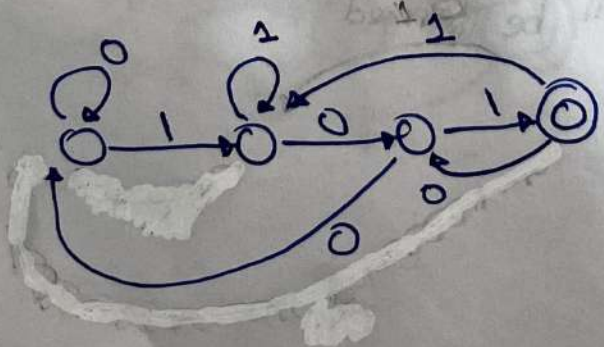


(5) 1st & last symbol are same





Q] Construct a minimal DFA over {0,1} that accept all String ending with "101"

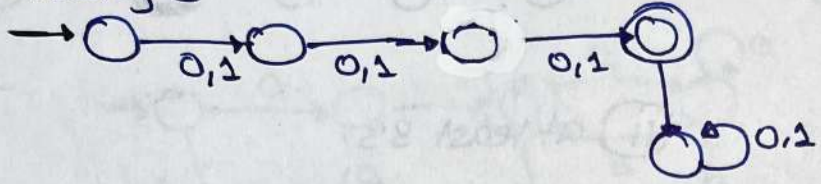


Q] Construct a minimal DFA over {0,1} that accept all string having length

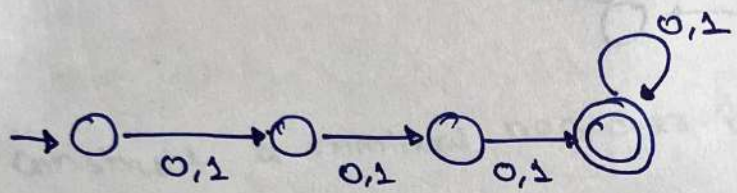
(i) exactly 3 (ii) atleast 3 (iii) atmost 3

(iv) divisible by 3

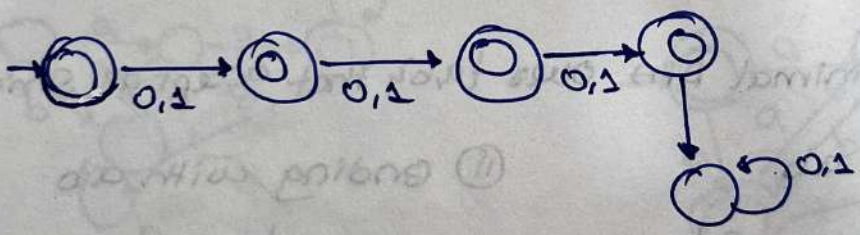
(i) exactly 3



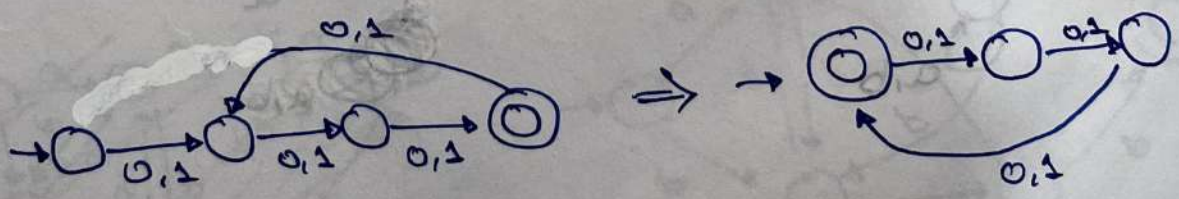
(ii) atleast 3



(iii) atmost 3



(iv) divisible by 3

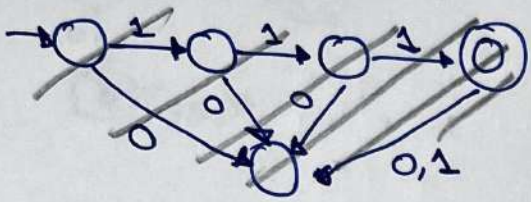


3,6,9,12---

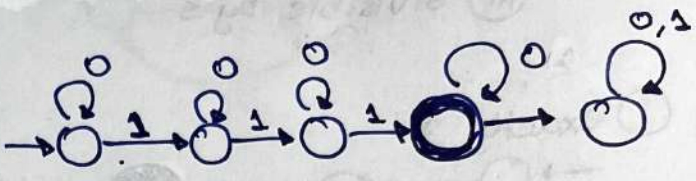
0,3,6,9,12---

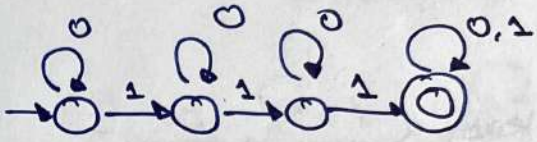
Q] Construct a minimal DFA over {0,1} that accepts all string no. of 1's.

(i) exactly 3's

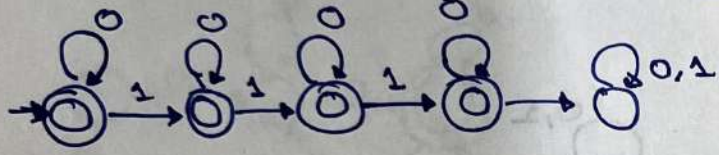


(ii) atleast 3's

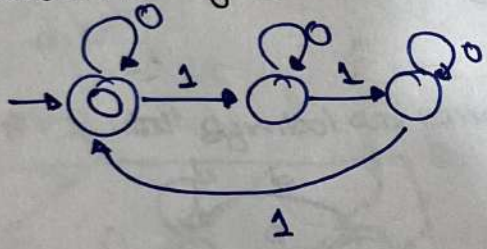




(iii) atmost 3

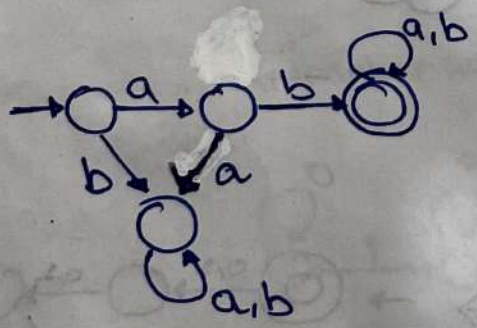


(iv) dividisble by 3

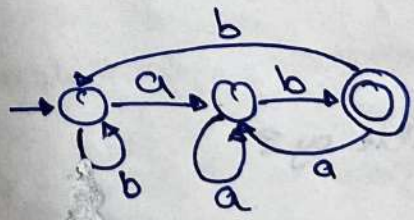


Q] Construct a minimal DFA over {a,b} that accept all string

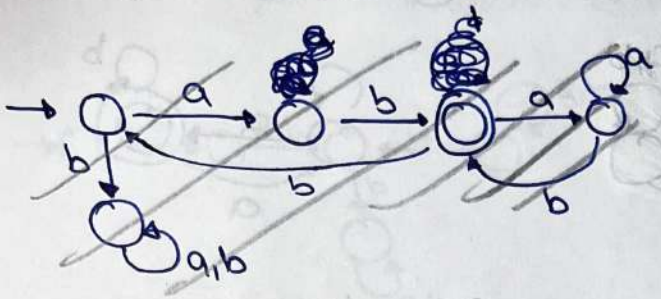
(i) Start with ab



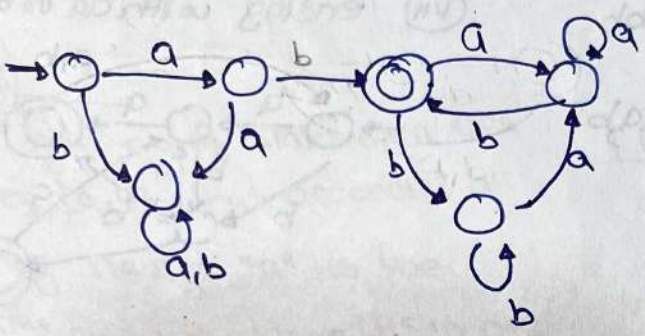
(ii) Ending with ab



(ii) Starting & ending with ab

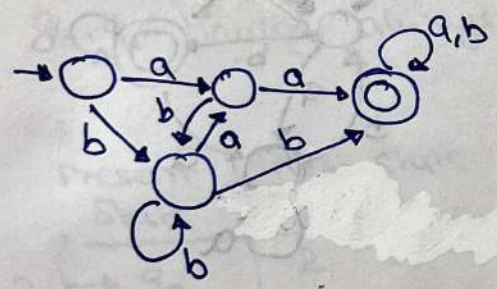


abbab
ababab
abaab
abbbab

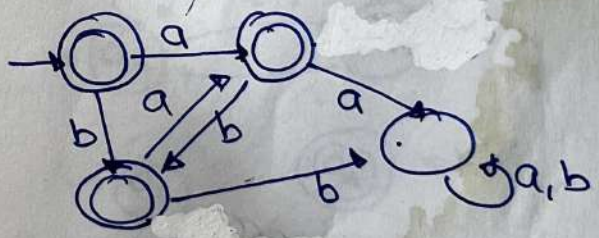


Q/ Construct a minimal DFA over {a,b} that accepts all string

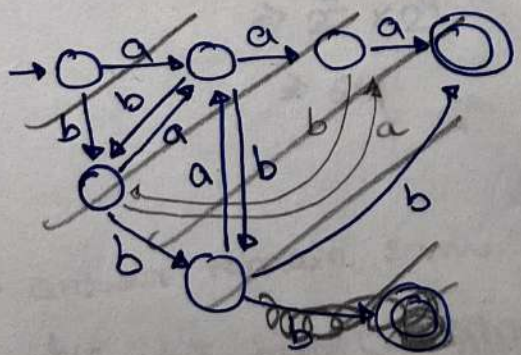
(i) aa or bb as substring.

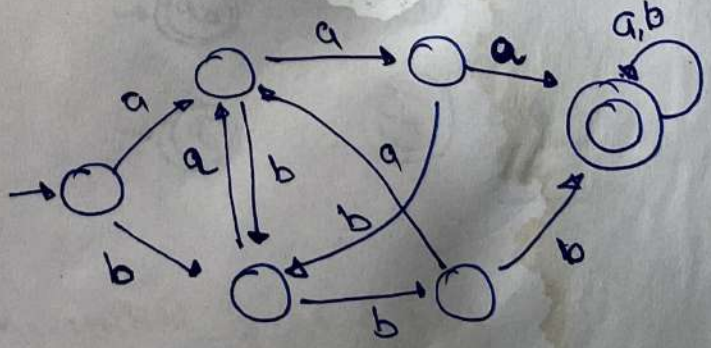


(ii) no two consecutive Symbol are same

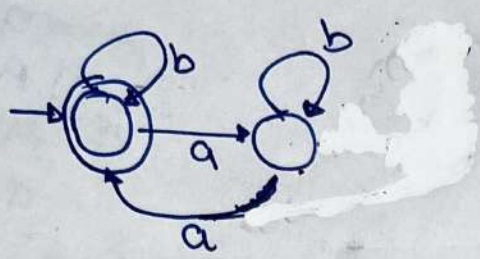


(iii) aaa or bbb as substring

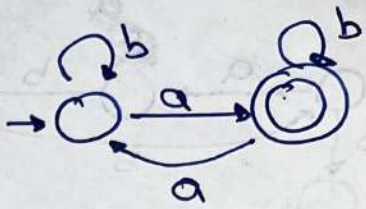




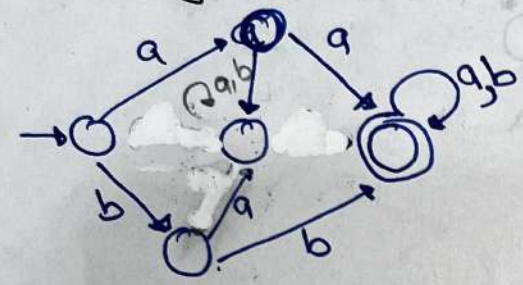
(iv) even no. of a's.



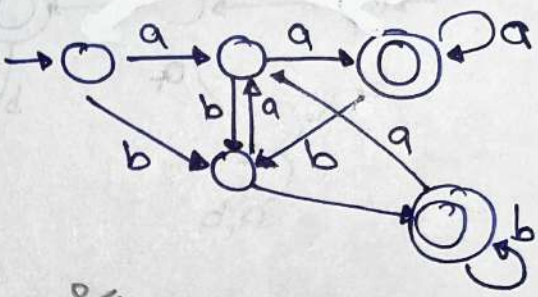
(v) odd no. of a's.



(vi) Starting with aa or bb



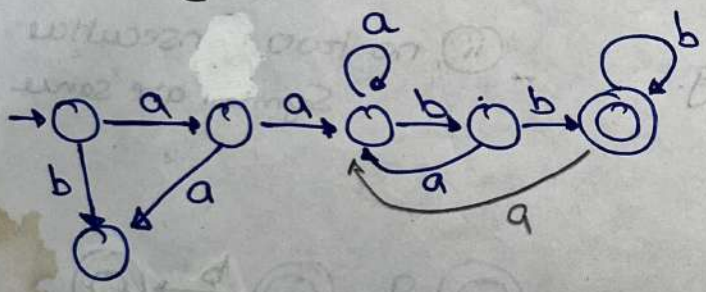
(vii) ending with aa or bb

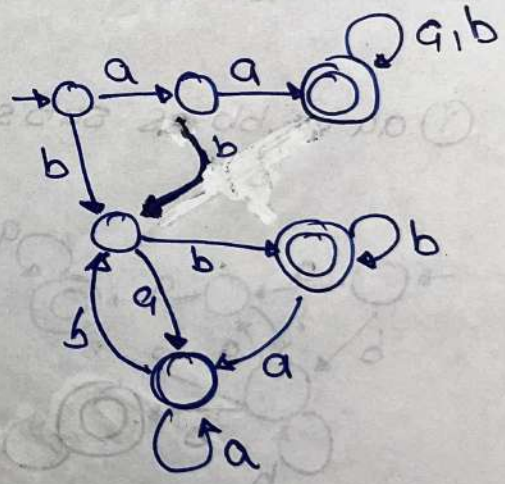


~~Imp~~

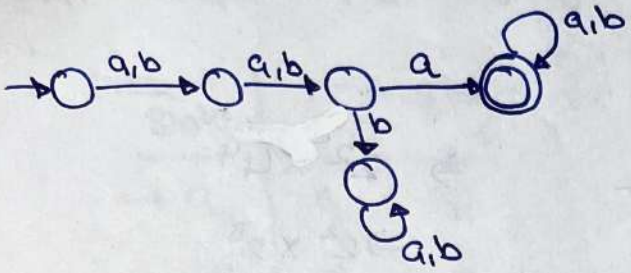
(xi) Starting with aa or ending with bb

(viii) Starting with aa & ending with bb





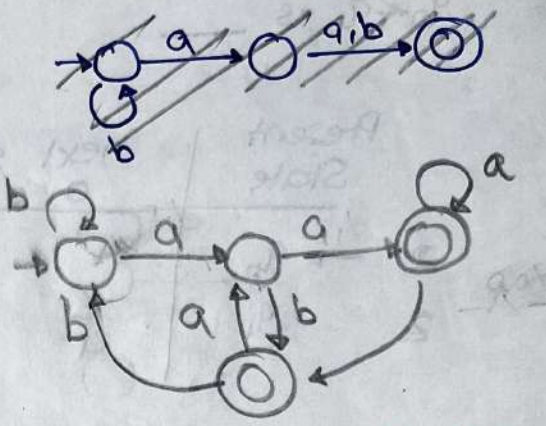
① "a" as 3rd symbol from LHS



Note

the min no. of states of a DFA over {a,b} that accpected all string having "a" as the "nth" symbol from LHS in n+2

② "a" as 2nd symbol from RHS



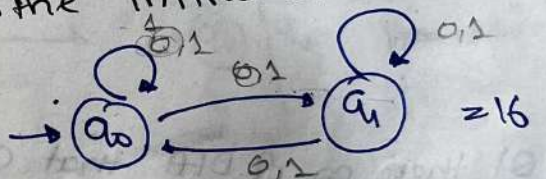
the min no. of states of a DFA over {a,b} that accepts all string having "a" as the nth symbol from RHS in 2^n.

Q] the no. of DFA that can be drawn over {0,1} with 2 state q_0, q_1 having always q_0 as the initial state is.

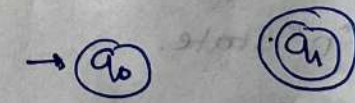
Present State	next state 0	1
2 → q_0	2	2
2 → q_1	2	2

$\Rightarrow 2^2 \times 2^4$

$\Rightarrow 64$



= 16



= 16

= 16

= 16

64

∴ answer remain same if the initial is not given bcz it's just changing symbol

Q] the no. of FA that can be drawn over {0,1} with 2 State q_0 & q_1 is ___

Present State	next State 0	next State 1
2 → q_0	4	4
2 q_1	4	4

$\Rightarrow 2^2 \times 4^4$

$\Rightarrow 2^2 \times 2^8$

$\Rightarrow 2^{10}$

$\Rightarrow 1024$//

Q] the no. of DFA that can be drawn over {0,1} with 3 states is

Present State	next State 0	next State 1
2 → q_0	3	3
q_1	3	3
q_2	3	3

$\Rightarrow 2^3 \times 3^6$

$\Rightarrow 5832$//

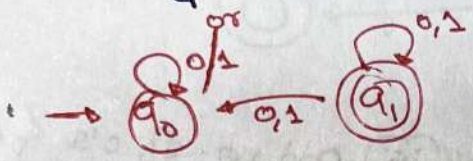
Q] the no. of DFA that can be drawn over {1,---m} with "n" State.

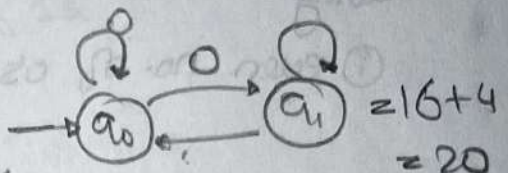
	1	2	3	----	m
→ q_1	n	n	n	----	n
q_2	n	n	n	----	n
q_3	n	n	n	----	n
q_4					
⋮					
q_n	n	n	n	...	n
2^n	n^{m*n}				

Q) the no of DFA that can be drown one {0,1} with 2state that accepts empty language.

Present State	next State 0	1
→ q_0	q_0	q_0
(q_1)	2	2

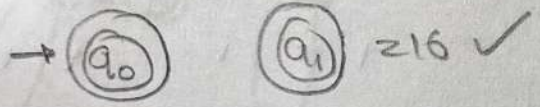
4



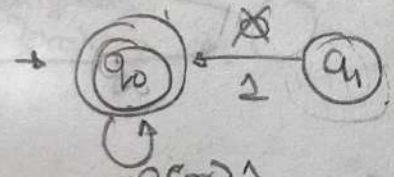


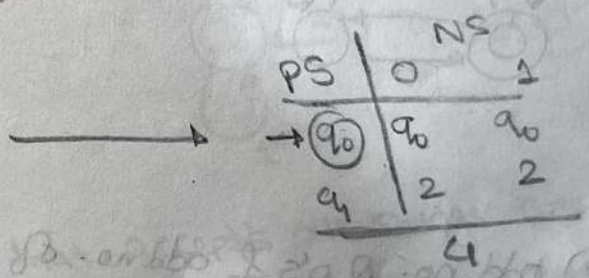
=16+4
=20

Q) the no. of DFA that can drown with 2state that accepts all strings over {0,1} is.



=16 ✓





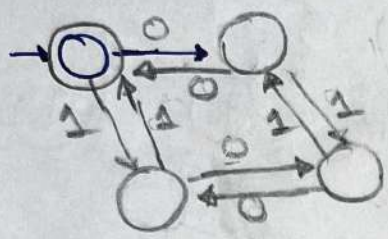
PS	NS 0	1
→ (q_0)	q_0	q_0
q_1	2	2

4

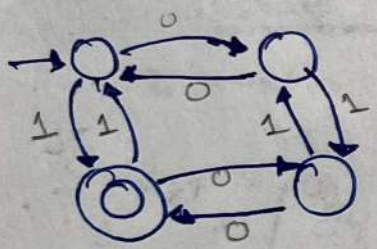
⇒ 16+4
= 20

Q) Construct a minimal DFA over {0,1} that accepts all string having

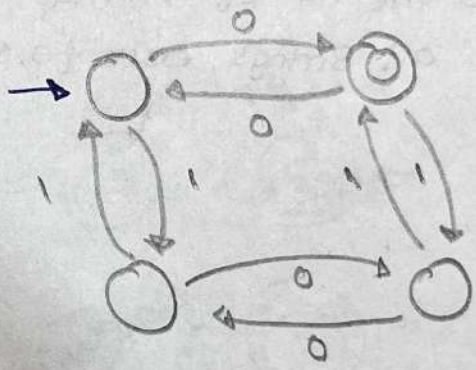
(i) even no. of 0's & even no. of 1's.



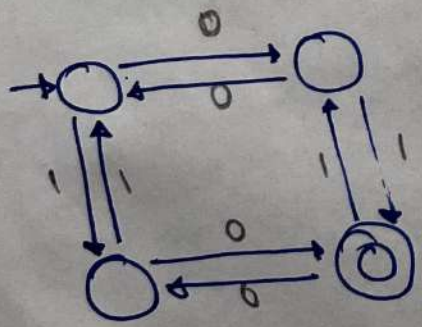
(ii) even no. of 0's & odd no. of 1's.



(iii) odd no. of 0's & even no. of ones.

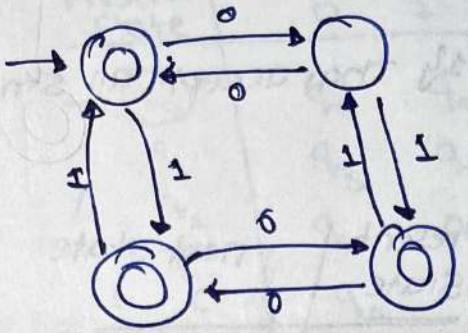


(iv) odd no. of 0's & odd no. of 1's.



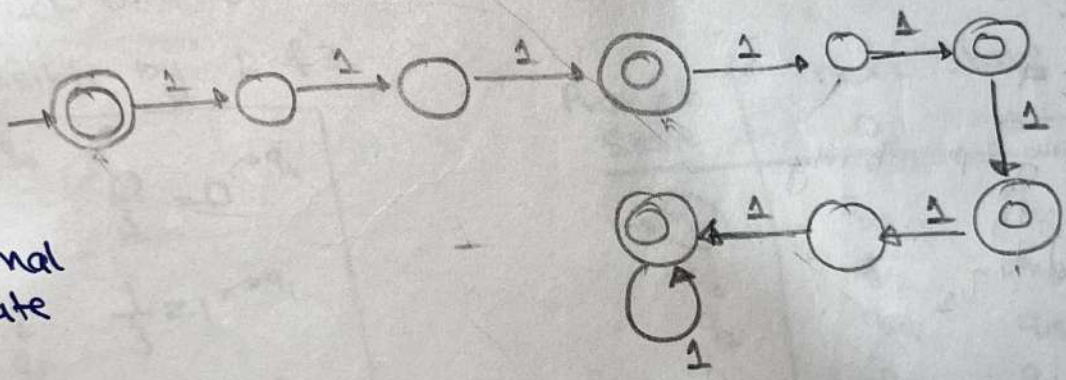
(v) even no. of 0's or odd no. of 1's

↳ odd no. of 0's & even no. of 1's.



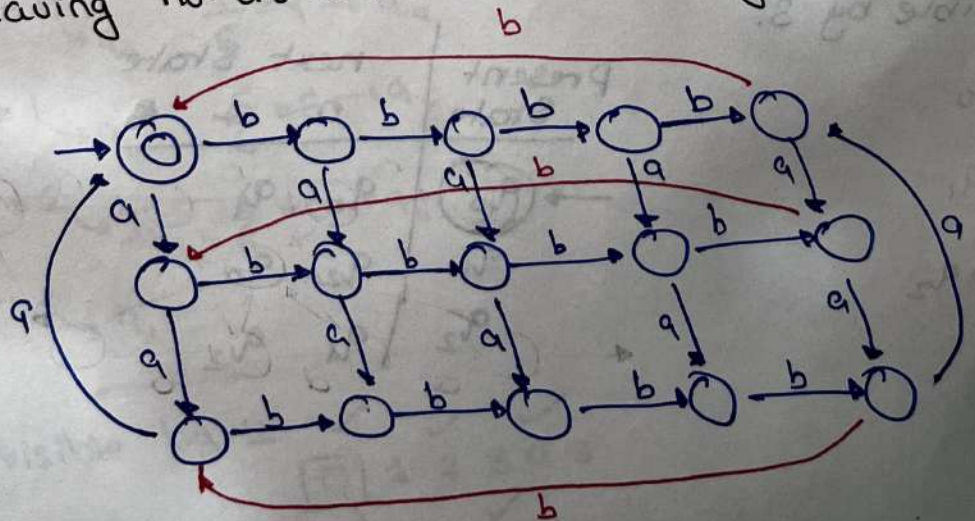
Q] construct a minimal DFA that accept all string over {111, 11111}

$\Sigma = \{1\}$,



3 minimal state

Q] Construct a minimal DFA that accepts all strings one {a,b} having no. a's & b's divisible by 3 & 5 respectively.



note: the min. no of state of DFA over {a,b} that accepts string in which no. of a's & b's are divisible by m & n respectively is $\boxed{m*n}$

Q] Construct a DFA over {0,1} that accept all string which are divisible by 2.

$\frac{0}{2} = 0 \rightarrow q_0$

$\frac{1}{2} = 1 \rightarrow q_1$

$\frac{2}{2} = 0$

$\frac{3}{2} = 1$

$\frac{4}{2} = 0$

$\frac{5}{2} = 1$

only 2 possibility

{0,1}

q_0 q_1

Present State	next state 0	1
→ q_0	q_0	q_1
q_1	q_0	q_1

Q] construct a DFA over {0,1} that accept all string which are divisible by 3.

$\frac{0}{3} = 0 \rightarrow q_0$

$\frac{1}{3} = 1 \rightarrow q_1$

$\frac{2}{3} = 2 \rightarrow q_2$

$\frac{3}{3} = 0$

⋮

Present State	next State 0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Q] Construct a DFA that accepts all base 3 no. which are divisible by 4

$\frac{0}{4} = 0 \rightarrow q_0$

$\frac{1}{4} = 1 \rightarrow q_1$

$\frac{2}{4} = 2 \rightarrow q_2$

$\frac{3}{4} = 3 \rightarrow q_3$

$\vdots$

Present State	next State 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_4
q_3	q_0	q_1	q_2

minimize DFA

$\Rightarrow$

$S_1 = \{q_0\}, \{q_1, q_2, q_3\}$

$S_2 = \{q_0\}, \{q_1, q_2\}, \{q_3\}$

$S_4 = \{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}$

Q] construct DFA over $\{0,1\}$ that accepts all strings

① divisible by 2 & 3.

$\frac{0}{2} = 0 \rightarrow q_0$ $\quad$ $\frac{0}{3} = 0 \rightarrow q_0$

$\frac{1}{2} = 1 \rightarrow q_1$ $\quad$ $\frac{1}{3} = 1 \rightarrow q_1$

$\frac{2}{2} = 0 \rightarrow q_1$ $\quad$ $\frac{2}{3} = 2 \rightarrow q_2$

$\frac{0}{6} = 0 \rightarrow q_0$ $\quad$ $\frac{4}{6} = 4 \rightarrow q_4$

$\frac{1}{6} = 1 \rightarrow q_1$ $\quad$ $\frac{5}{6} = 5 \rightarrow q_5$

$\frac{2}{6} = 2 \rightarrow q_2$ $\quad$ $\frac{6}{6} = 0 \rightarrow q_0$

$\frac{3}{6} = 3 \rightarrow q_3$

Present State	next state 0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

divisible by 2 ✓ ✓ ✓

Q 1 2 3 4 5

divisible by 3 ✓ ✓

Common for both

DFA minimization

divisible by 2 and 3.

Present State	next state 0	1
→ q_0	q_0	q_4
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

$S_1 = \{q_0\}\{q_1, q_2, q_3, q_4, q_5\}$

$S_2 = \{q_0\}\{q_3\}\{q_1, q_2, q_3, q_4\}$

$S_3 = \{q_0\}\{q_3\}, \{q_1 q_4\}$
$\{q_2 q_5\}$

q_1 & q_2 not 1-equivalent

q_1 & q_4

(ii) divisible by 2 or 3.

Present State	next state 0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

(iii) divisible by 2 but not by 3

Present State	next state 0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

(iv) divisible by 3 but not by 2

Present State	next state 0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

by2 ✓ ✓ ✓
~~0~~ 1 ~~2~~ 3 ~~4~~ 5
by3 ✓ ✓

(v) neither divisible by 2 nor by 3

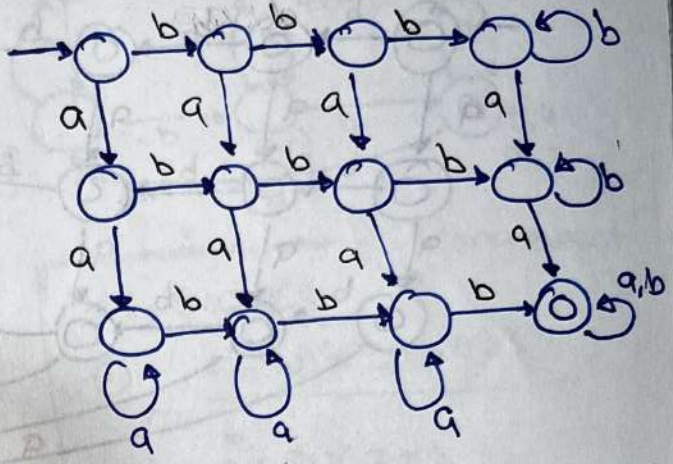
Present State	next state 0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

by2 ✓ ✓ ✓
~~0~~ 1 ~~2~~ 3 ~~4~~ 5
by3 ✓ ✓

Q] Construct a minimal DFA over {a,b} that accept all string have

(i) at least 2 a's & atleast 3 b's.

Present State	next State a	b

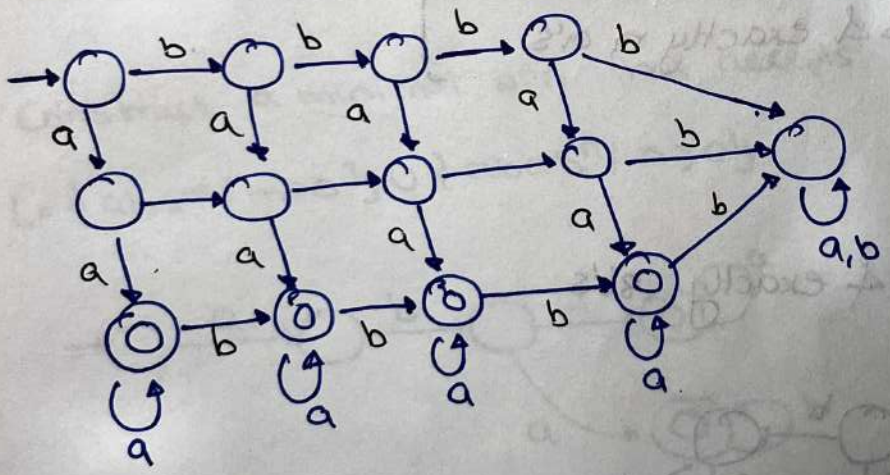


note:

- (2+1)*(3+1)

the min no. of states of a DFA over {a,b} that accepts all string having atleast m a's and at least n b's is (m+1)*(n+1).

(ii) atleast 2 a's & atmost 3 b's.



No. of a ≥ 2 & no of b ≤ 3

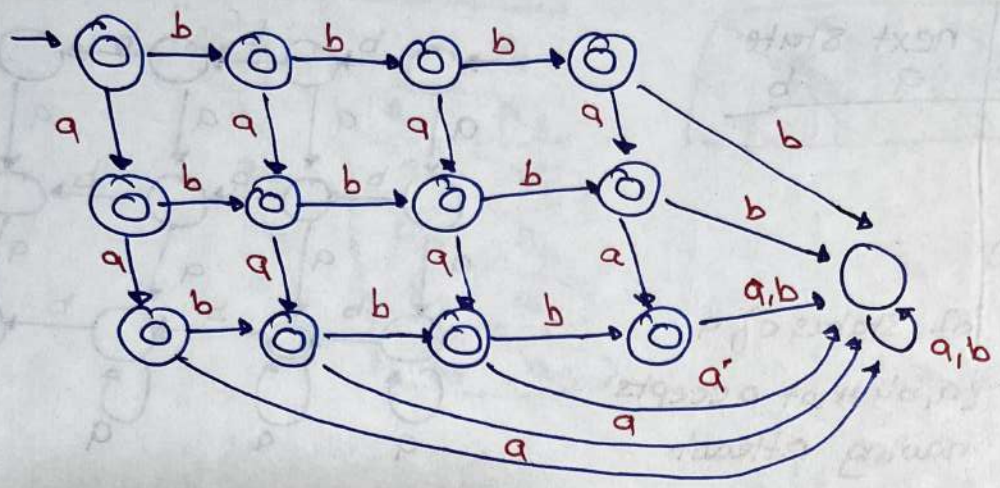
note:-

- the min no. of State of a DFA over {a,b} that accept all string having atleast m a's & atmost n b's is

(m+1)*(n+1)+1

(iii) atmost 2 a's & at most 3b's.

no. of $a \leq 2$ & $N_b(\omega) \leq 3$

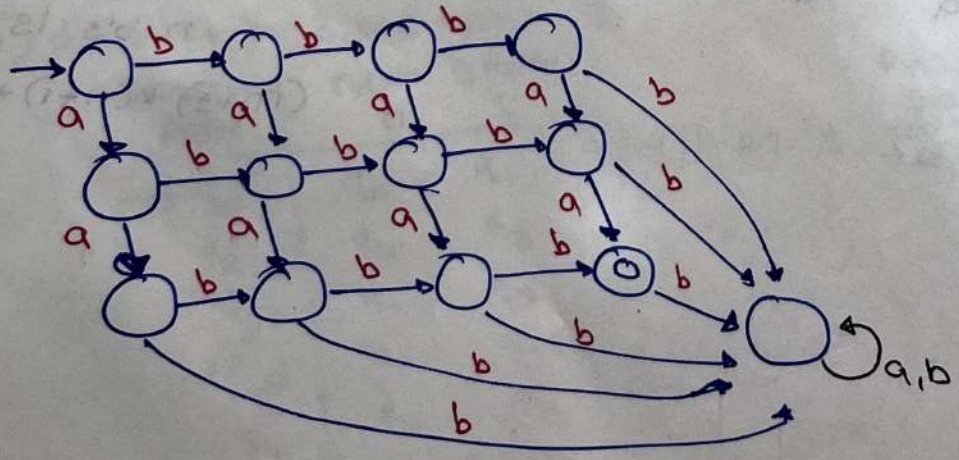


note. thamin no. of state of a DFA over {a,b} that accepts all substring having

- atmost m a's & atmost n a's
- exactly m a's & exactly n a's

$(m+1) * (n+1) + 1$

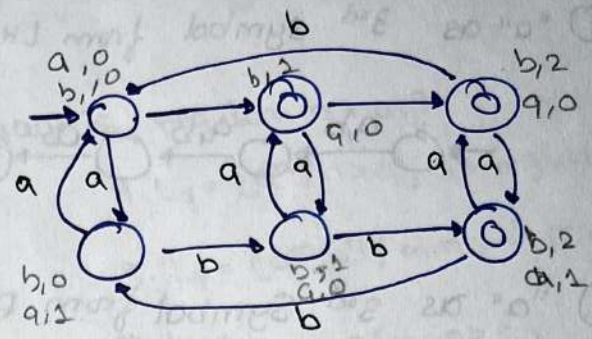
(iv) exactly 2 a's & exactly 3b's.



(1) $N_b(w) \bmod 3 \times N_a(w) \bmod 2$

$$\frac{N_b(w)}{3} = 0,1,2$$

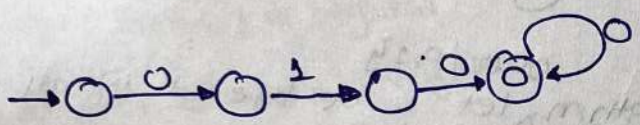
$$\frac{N_a(w)}{2} = 0,1$$



Construction of NFA

Q] Construct of minimal NFA that accept $L = \{010^n / n>0\}$

$L = \{010, 0100, 01000 \cdots\}$



∴ suppose — congruent

① $n_a(w) \cong 1 \bmod 3$

or

$n_a(w) \,\%\, 3 = 1$

② $n_a(w) \equiv k \bmod m$

or we can write this as

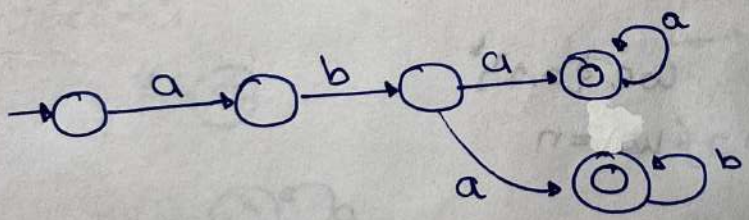
$n_a(w) \,\%\, m = k$

↓

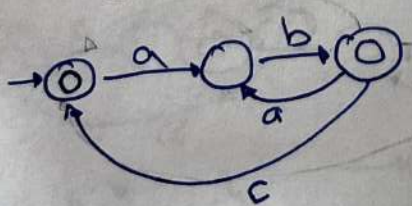
It will contain m state & the (k+1)th state will be final

Q] Construct a minimal NFA that accepts

$L = \{aba^m / m>0\} \cup \{abab^n / n \geq 0\}$

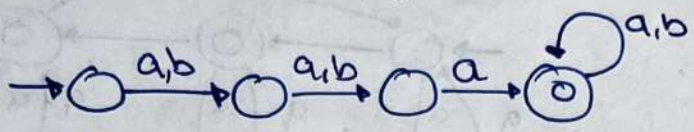


Q] construct a minimal NFA that accept all string over $\{ab, abc\}$

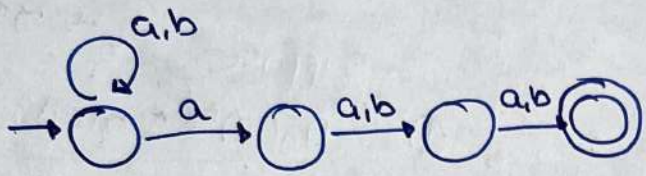


Q) Construct a minimal NFA that accepts all string having. over {a,b}

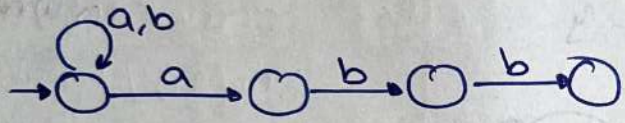
(i) "a" as 3rd symbol from LHS



(ii) "a" as 3rd symbol from RHS

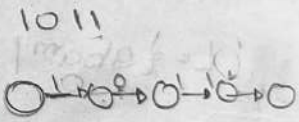


(iii) String ending with abb



Q) let w a string of length n, let L be the set of all Substring of w. The min. no. of state of a NFA that accepts L is. over {0,1}

(a) n
(b) n+1
(c) n+2

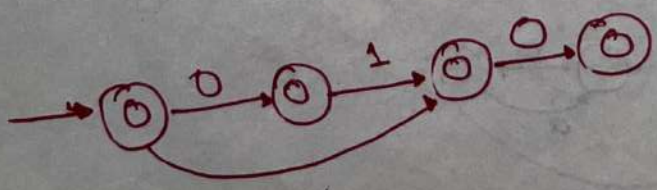


$w \in \{0,1\}^*$

$|w| = n$

let $|w| = 3$, $w = 010$

$L = \{\epsilon, 0, 1, 01, 10, 010\}$



Note:

- a language is said to be regular if it accepted by a finite automata (FA).

Q) Check whether the fall language is regular or not?

1. $L_1 = \phi$ is regular
2. $L_2 = \{\epsilon\}$ is regular
3. $L_3 = \{a\}$ is regular
4. $L_4 = \{a^n / n \geq 0\}$ is regular
5. $L_5 = \{a^{2n} / n \geq 0\}$ is regular
6. $L_6 = \{a^{3n} / n > 0\}$ is regular
7. $L_7 = \{a^n / n > 0\}$ is regular
8. $L_8 = \{a^m b^n / m, n \geq 0\}$ is regular
9. $L_9 = \{a^m b^n / m, n > 0\}$ is regular.
10. $L_{10} = \{a^n b^{2n} / m, n \geq 0\}$ is regular
11. $L_{11} = \{a^m b^{2n} / m, n > 0\}$ is regular

no comparsion required

$L_7 =$
12. $\{a^{2m} b^{3n} / m, n \geq 0\}$ is regular

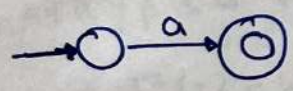
$L_8 =$
13. $\{a^n b^n / n \leq 3\}$ is regular

$L_9 =$
14. $\{a^n b^n / n \leq 10000\}$ is regular

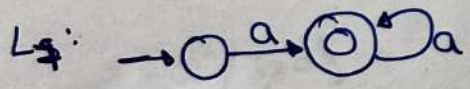
15. $L_{10} = \{a^n b^n / n \geq 0\}$ not regular } comparsion

L_1:

L_3:



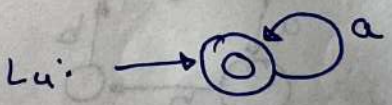
L_4:



$\Rightarrow \{a, a^2, a^3 \ldots\}$

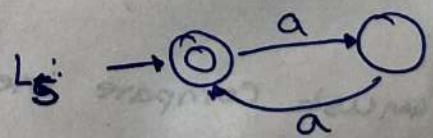
L_2:

L_4:



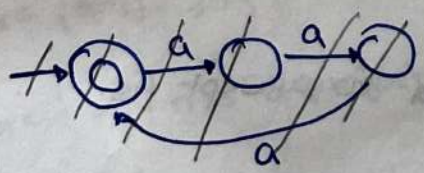
$\Rightarrow \{\epsilon, a, a^2, a^3 \ldots\}$

L_5:

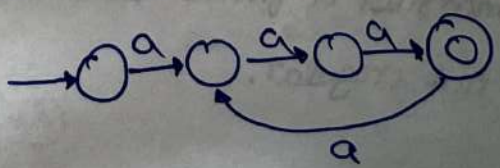


$\{\epsilon, a^2, a^4, a^6, a^8 \ldots\}$

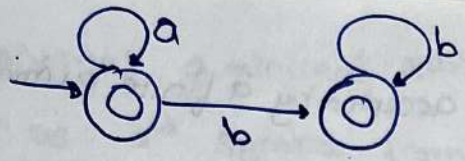
L_6:



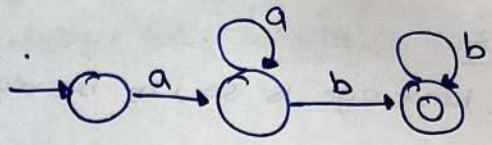
$\{\epsilon, a^3, a^6, a^9 \ldots\}$



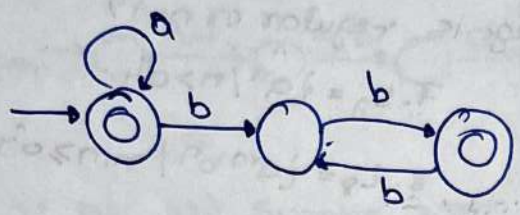
L_8:



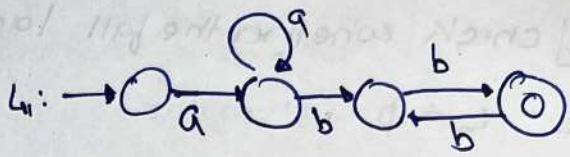
L_9:



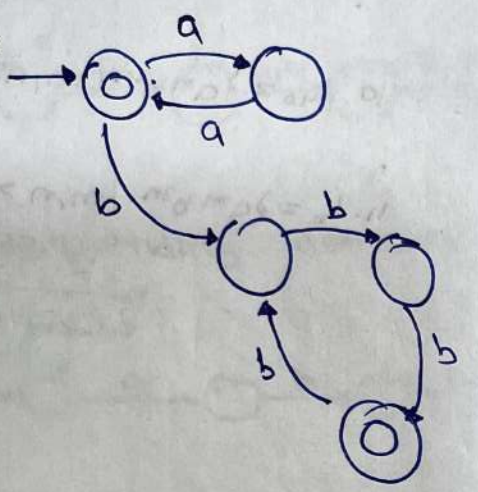
L_{10}:



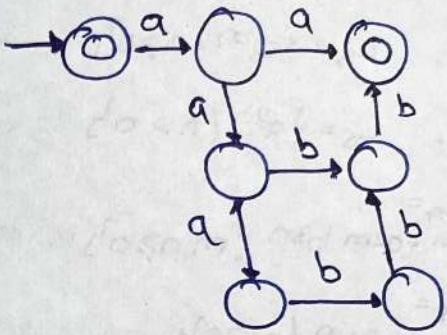
L_{11}:



L_{12}:

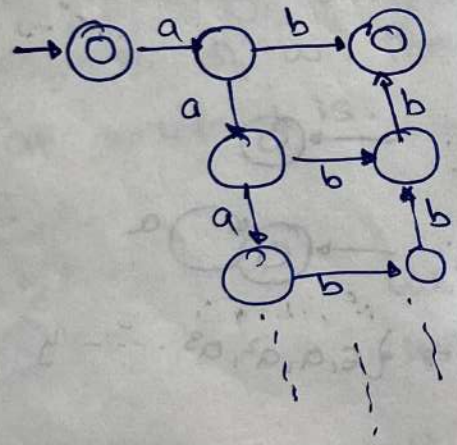


L_{13}:



// Finite Automata doesn't have memory for infinite. It only have finite memory

L_{14}:



▸ In L_{15} we must compare the no. of a's & b's for that a memory is required, but a finite automata doesn't have a memory

∴ we can't construct a finite automata to accept L_5.

Hence L_5 is not regular.

no Relation — finite / infinite — Regular

Relation — finite — Regular

Relation — Infinite — not regular.

$L_{16} = \{a^l b^m c^n / l,m,n \geq 0\}$ is regular

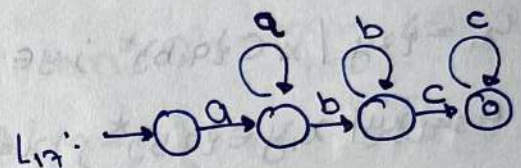
$L_{17} = \{a^l b^m c^n / l,m,n > 0\}$ is regular

$L_{18} = \{a^l b^{2m} c^{3n} / l,m,n \geq 0\}$ is regular

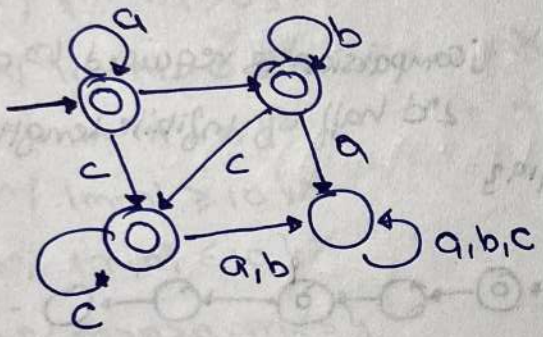
$L_{19} = \{a^m b^n / m > n > 0\}$ not regular

$L_{20} = \{a^m b^n / m \neq n\}$ not regular

L_{17}:



L_{16}:



✓ → regular

✗ → not regular

$L_{21} = \{a^p / p$ is a prime no. $; p \leq 20\}$ ✓

$L_{22} = \{a^p / p$ is a prime no.$\}$ ✗

$L_{23} = \{a^{i^2} / i \leq 5\}$ ✓ ---- $L_{23} = \{\epsilon, a, a^4, a^9, a^{16}, a^{25}\}$

$L_{24} = \{a^{i^2} / i \geq 0\}$ ✗

$L_{25} = \{a^n / n \leq 10\}$ ✓

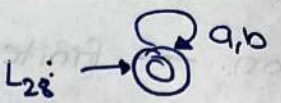
$L_{26} = \{a^n / n \geq 0\}$ ✗

$L_{27} = \{a^n / n$ is a prefect no.$\}$ ✗

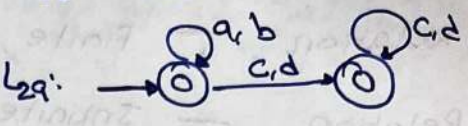
↳ ex: $6 = \{1 \times 6, 2 \times 3\}$

ex: $28 = \{1 \times 28, 2 \times 14, 4 \times 7\}$

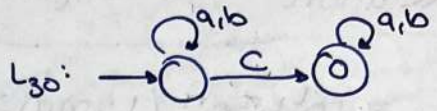
$L_{28} = \{xy \mid x,y \in \{a,b\}^*\}$ ✓



$L_{29} = \{xy \mid x \in \{a,b\}^* ; y \in \{c,d\}^*\}$ ✓



$L_{30} = \{xcy \mid xy \in \{a,b\}^*\}$ ✓



$L_{31} = \{xcy \mid x \in \{0,1\}^* ; y \in \{2,3\}^*\}$ ✓

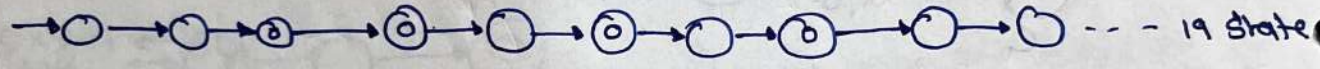
$L_{32} = \{xy \mid xy \in \{a,b\}^* , |x| = |y|\}$ ✓

$L_{33} = \{xcy \mid xy \in \{a,b\}^* ; |x| = |y|\}$ ✗

$L_{34} = \{xy \mid x \in \{a,b\}^* ; y \in \{c,d\}^* , |x| = |y|\}$ ✗

$L_{35} = \{xy \mid x,y \in \{a,b\}^* ; x = y\}$ ✗ ; comparsion is required to check 1st & 2nd half of Infinite length.

L_{21}: $\{a^2, a^3, a^5, a^7, a^{11}, a^{13}, a^{17}, a^{19}\}$



L_{32}: even length of string

$|x| = |y| = 2$

$x = ab \quad y = bb$

$xy = abbb$ ✓

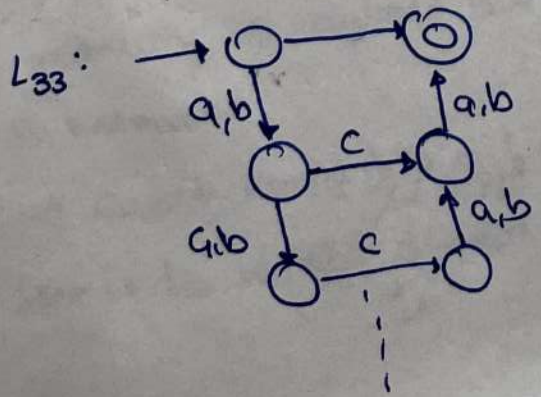
$\epsilon \cdot \epsilon = 0$

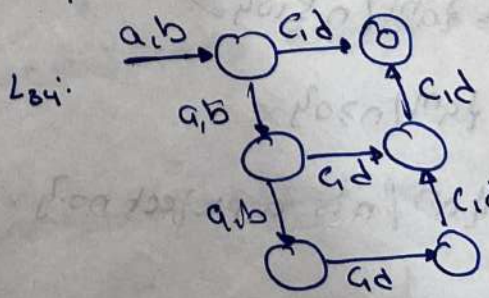
$x \mid x = 2$

$xx \mid xx = 4$

$xxx \mid xxx = 6$

$xxxx \mid xxxx = 8$





$L_{36} = \{wcw^T / w \in \{a,b\}, |w|=5\}$ ✓ — $\{wcw^T$ — w^T should be reverse of w for that comparsion is required but its of finite length

$L_{37} = \{wcw^T / w \in \{a,b\}^*\}$ ×

$L_{38} = \{ww^T / w \in \{a,b\}^*, |w| \leq 10\}$ ✓

$L_{39} = \{ww^T / w \in \{a,b\}^*\}$ ×

$L_{40} = \{wcw / w \in \{a,b\}^*, |w|=10\}$ ✓

$L_{41} = \{wcw / w \in \{a,b\}^*\}$ ×

$L_{42} = \{ww / w \in \{a,b\}^*, |w| \leq 10\}$ ✓ — $\{ww$ should be same. so comparision required but finite length

$L_{43} = \{ww / w \in \{a,b\}^*\}$ ×

$L_{44} = \{a^m b^n / |m-n| \geq 0\}$ ✓ ---- $\{a^m b^n / m,n \geq 0\}$ → for any value of m there exist an n which is ≥ 0

$L_{45} = \{a^m b^n / |m-n| > 0\}$ × → $\{a^m b^n / m \neq n\}$

$L_{46} = \{a^m b^n / |m-n| = 0\}$ × → $\{a^m b^n / m = n\}$

$L_{47} = \{a^m b^n / |m-n| \geq 10\}$ ×

$L_{48} = \{a^m b^n / |m-n| \leq 10\}$ ×

$L_{49} = \{a^m b^n / m > n > 0; m \leq 5\}$ ✓

$L_{50} = \{a^m b^n / m > n > 0; n \leq 3\}$ ✓ → Comparisions are finite but string are infinite possible

$L_{51} = \{a^m b^n / m = 2n, n \leq 100\}$ ✓

$L_{52} = \{a^m b^n / 100 \leq n \leq 1000000\}$ ✓

✓ — regular
× — not regular

L_{42}: $\underline{\varepsilon} = 0$

$\underline{a/b} = 2^1$

Same

$\underline{a/b}\ \underline{a/b} = 2^2$

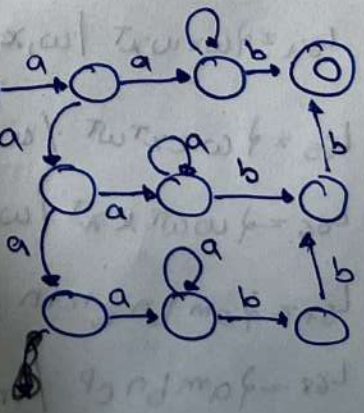
$\underline{a/b}\ \underline{a/b}\ \underline{a/b}\ \underline{1}\ \underline{1}\ \underline{1} = 2^3$

$2^0 + 2^1 + 2^2 + \cdots 2^{10}$

$\Rightarrow \frac{1(2^{11}-1)}{2-1}$

$\Rightarrow 2047$

L_{50}:



L_{53}: $\{a^m b^n / m+n = 10\}$ ✓

L_{54}: $\{a^m b^n / m * n = 64\}$ ✓

L_{55}: $\{a^m b^n / m - n = 10\}$ ✗

L_{56}: $\{a^m b^n / m \div n = 10\}$ ✗

L_{57} = $\{\omega \in \{a,b\}^* / N_a(\omega) = N_b(\omega)\}$; $|\omega| = 100\}$ ✓

L_{58} = $\{\omega \in \{a,b\}^* / N_a(\omega) = N_b(\omega)$ ✗

" $N_a(\omega) \leq N_b(\omega)$ ✗

" $N_a(\omega) \geq N_b(\omega)$ ✗

✓ – regular
✗ – not regular

mod funct^n provide finiteness to the string

L_{59} = $\{\omega \in \{a,b\}^* / N_a(\omega) \bmod 5 = N_b(\omega) \bmod 3\}$ ✓

L_{60} = $\{\omega \in \{a,b\}^* / N_a(\omega) \bmod 5 = 0 \text{ & } N_b(\omega) \bmod 7 = 0\}$ ✓

Imp

L_{61} = $\{\omega x \omega^r / \omega, x \in \{a,b\}^*\}$ ✓

L_{62} = $\{\omega x \omega . / \omega, x \in \{a,b\}^*\}$ ✓

we are always taking it as 'ε' & only counting x

L_{63} = $\{\omega x \omega . / \omega, x \in \{a,b\}^+\}$ ✗

L_{64} = $\{\omega x \omega x^T / \omega, x \in \{a,b\}^*\}$ ✗

L_{65} = $\{\omega x x^T \omega^T . / \omega, x \in \{a,b\}^*\}$ ✗

L_{66} = $\{\omega \omega^r x x^T / \omega, x \in \{a,b\}^*\}$ ✗

L_{67} = $\{a^m b^n c^{m+n} / m,n \geq 0\}$ ✗

L_{68} = $\{a^m b^n c^p / m > n > p > 0;\ p \leq 10\}$ ✗

L_{69} = $\{a^m b^n c^p / m > n > p > 0;\ n \leq 10\}$ ✓

L_{70} = $\{\omega x \omega^T / \omega, x \in \{a,b\}^*\}$ ✓

L_{71} = $\{\omega (\omega^R)^* / \omega \in \{a,b\}^*\}$ ✓

L_{72} = $\{\omega (\omega)^* / \omega \in \{a,b\}^*\}$ ✓

L_{73} = $\{a^n b^n c^n / n \leq 100\}$ ✓

bro

L_{74}: $\{a^n b^n c^n \mid n \geq 0\}$ ✗

$L_{75} = \{a^n b^{n^2} \mid n \geq 0\}$ ✗

$L_{76} = \{\omega c^m \omega^T \mid \omega \in \{a,b,c\}, m \geq 0\}$ ✓ → finite comparsion

$L_{77} = \{\omega c^m \omega \mid \omega \in \{a,b,c\}, m \geq 0\}$ ✓

$L_{78} = \{\omega c^m \omega^T \mid \omega \in \{a,b,c\}^*, m \geq 0\}$ ✗ → infinite comparsion

$L_{75} = \{\omega c^m \omega \mid \omega \in \{a,b,c\}^*, m \geq 0\}$ ✗

$\omega = a$
$\omega = b$
$\omega = \epsilon$

you can draw finite automata for these (different)

L_{70}: $\omega x \omega^T$ ($\epsilon x \epsilon$)

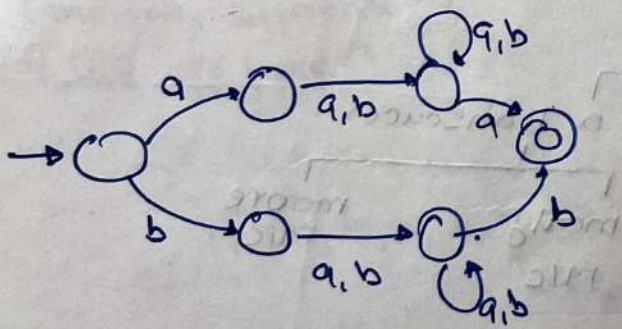
abbxx ——— bab
ω x ω^T

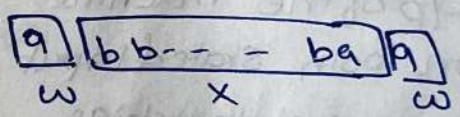
we will just make

$\Rightarrow$ ϵ abb - - - bab ϵ
ω x ω^T

L_{61}: a ——— a
b ——— b

$\omega x \omega^T \mid \omega, x \in \{a,b\}^+$



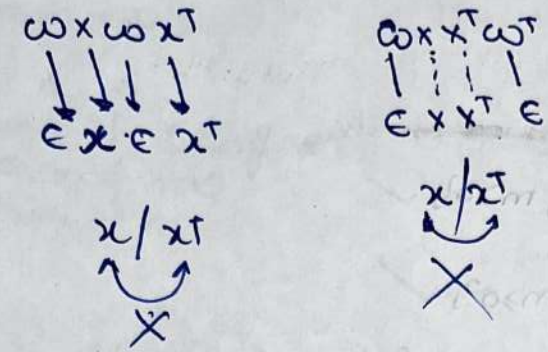


baa ——— baa ✗
ω x ω^T

L_{63}: abbb - - - bbbb

we need to compare
& then strings are infinite

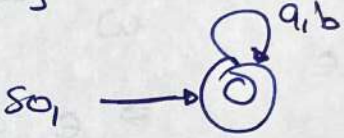
L_{64} to L_{66} :



L_{71} : $a^* = \{\epsilon, a, a^2, a^3, a^4 \ldots\}$

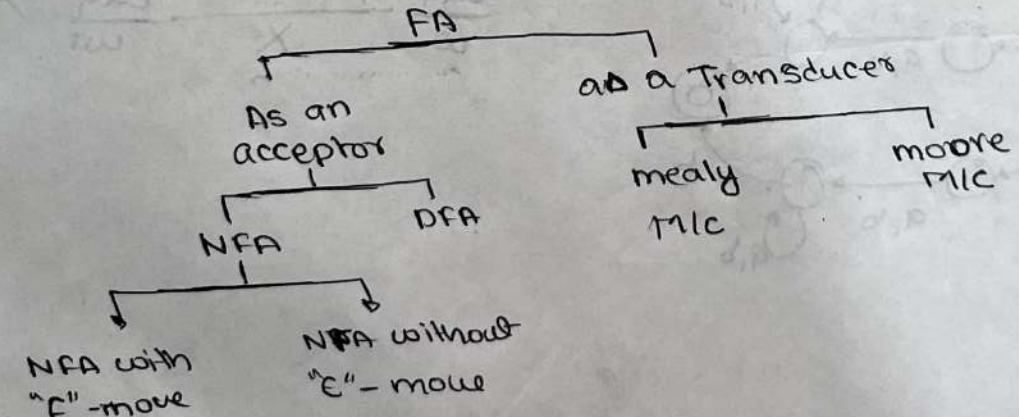
$\omega(\omega^T)^* = \omega \{\epsilon, \omega^T, \omega^T\omega^T, \omega^T\omega^T\omega^T \ldots\}$

$= \{\omega, \omega\omega^T, \omega\omega^T\omega^T \ldots\}$



Mealy Machine

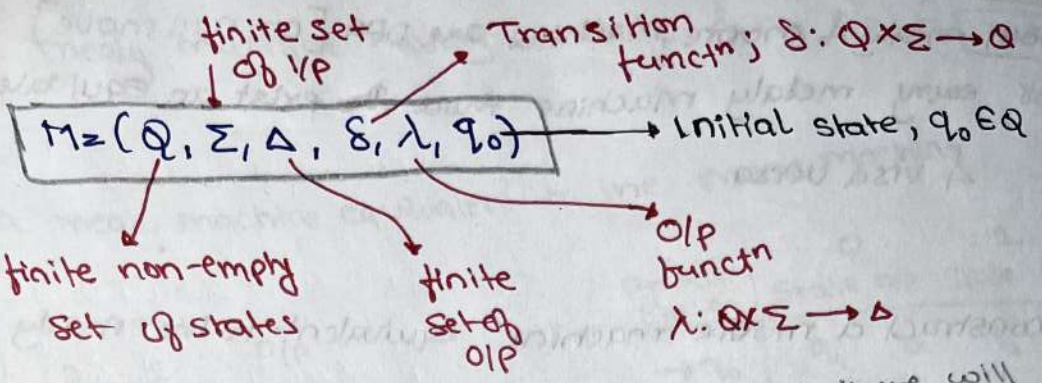
- If the o/p of the machine depend on both state & input is called as mealy machine



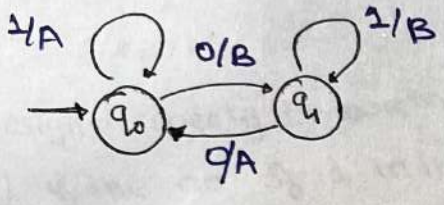
without "ε"-move : $\delta : Q \times \Sigma \longrightarrow 2^Q$

with "ε"- move : $\delta : Q \times (\Sigma \cup \{\epsilon\}) \longrightarrow 2^Q$

6-tuple:



if the directed edge have both input & output



every input there will O/P column

Present State	0 State	0 O/P	1 State	1 O/P
→ q_0	q_1	B	q_0	A
q_1	q_0	A	q_1	B

$q_0 q_1 q_1 q_0$
I/P = 0 1 0
O/P = B B A

note

Moore Machine

- If the O/P of the machine depend on state only then it is called as moore machine

[the only difference by moore & mealy is the O/P functn]

$q_0 q_1 q_1 q_0$
I/P = 0 1 0
O/P = A B B A

$M: (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
↳ O/P functn; $\lambda: Q \to \Delta$

Present State	next state 0	next state 1	O/P
→ q_0	q_1	q_0	A
q_1	q_0	q_1	B

Note

- Both mealy & moore machine are DFA [no null move]
- for every mealy machine there exist an equivalent machine & visa versa

Q] construct a moore machine equivalent to the mealy machine

Present State	0		1	
	State	O/P	State	O/P
→ q_0	q_1	A	q_2	B
q_1	q_0	A	q_1	B
q_2	q_2	A	q_0	A

2
O/P: A&B m
q_0, q_1, q_2 n
3
⇒ m x n

- Draw the Corresponding O/P

q_0	A, A	have same O/P [q_0]
q_1	A, B	q_{1A} & q_{1B}
q_2	A, B	q_{2A} & q_{2B}

those with different O/P, spilt them into those many states

as q_{1A} is a split version of q_1 so we will bring $q_{1A} \to A$ with associativity

Present State	next state 0	next state 1	O/P
→ q_0	q_{1A}	q_{2B}	A
q_{1A}	q_0	q_{1B}	A
q_{1B}	q_0	q_{1B}	B
q_{2A}	q_{2A}	q_0	A
q_{2B}	q_{2A}	q_0	B

Present State	0 State	0 O/P	1 State	1 O/P
→ q_0	q_{1A}	A	q_{2B}	B
q_{1A}	q_0	A	q_{1B}	B
q_{1B}	q_0	A	q_{1B}	B
q_{2A}	q_{2A}	A	q_0	A
q_{2B}	q_{2A}	A	q_0	A

Converted to moore machine

O/p a particular state will be same every where & that O/p will be plan in moore machine O/p

Note: the max no. of state of a moore machine equivalent to mealy machine having n state & m o/p is m*n

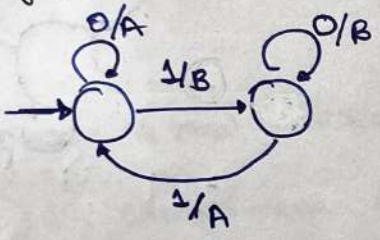
Q] construct a mealy machine equivalent to the moore machine

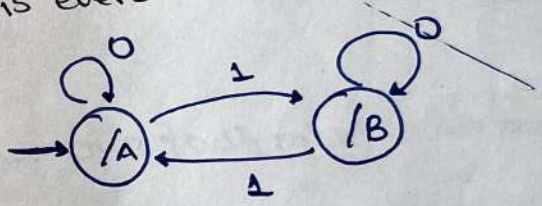
Present State	next State 0	next State 1	o/p
→ q_0	q_1	q_2	A
q_1	q_0	q_1	B
q_2	q_2	q_0	C

Present State	0 State	0 o/p	1 State	1 o/p
→ q_0	q_1	B	q_2	C
q_1	q_0	A	q_1	B
q_2	q_2	C	q_0	A

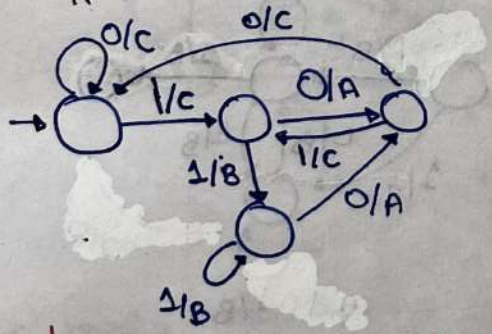
• Design a mealy & moore machine over {0,1} that produce o/p - A if the no. of 1 in the string is even otherwise o/p - B

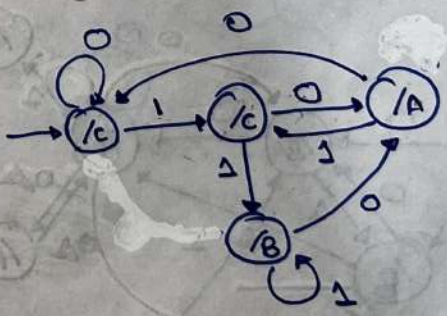
(A)



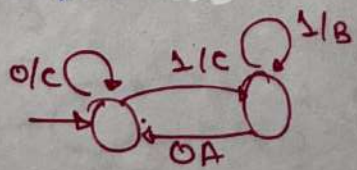


(B) Design mealy & moore machine over {0,1} for the following process if the i/p string end with 10 then o/p A, end with 11 then o/p - B otherwise o/p C

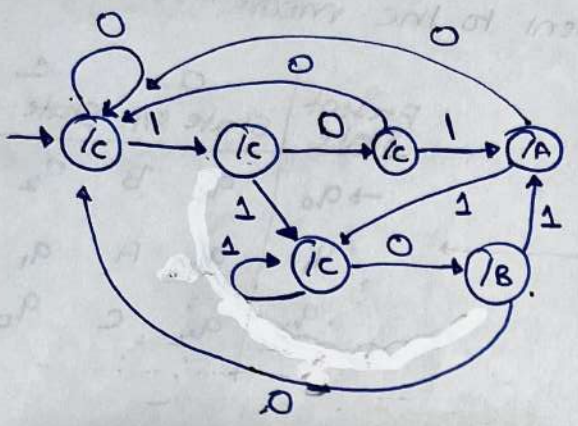


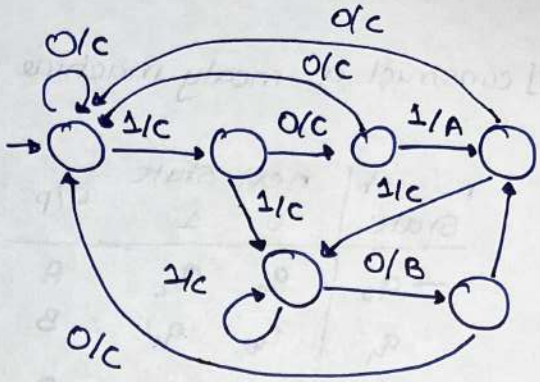


Can be drawn with 2 state

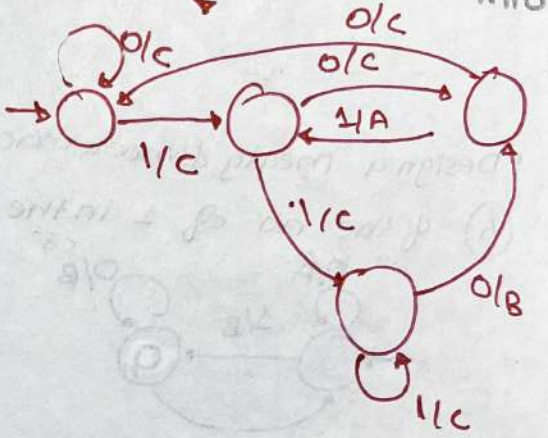


(c) If I/p string end with 101 then O/p-A
& if end with 110 then O/p-B
Otherwise O/p-C

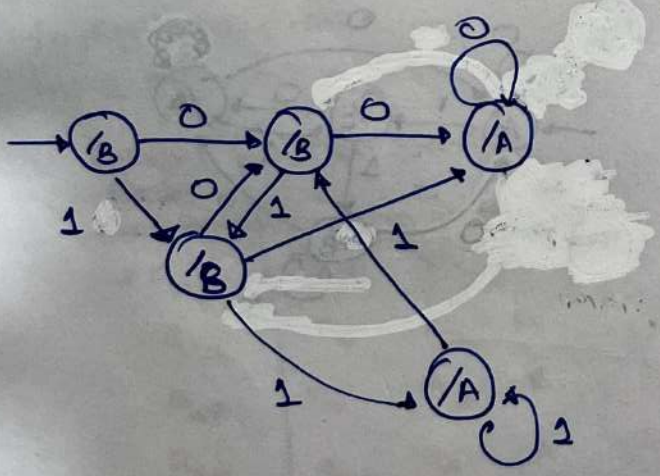


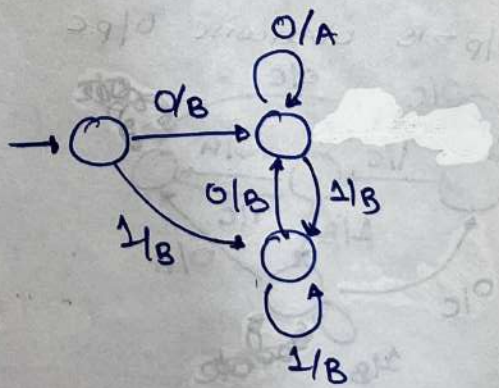


can be converted into

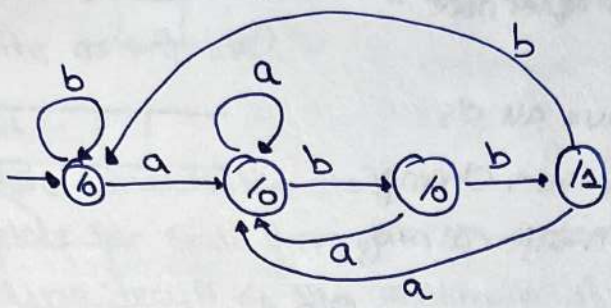


(d) If I/p str end with
either 00 or 11 then O/p A
Otherwise - O/p: B

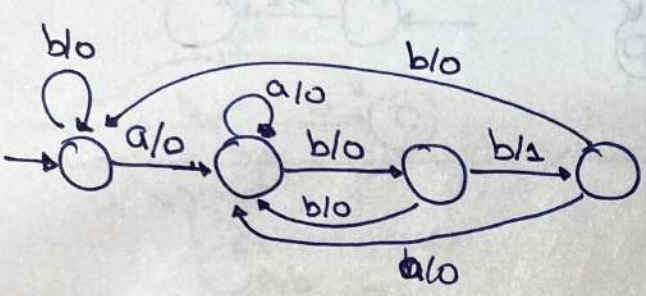




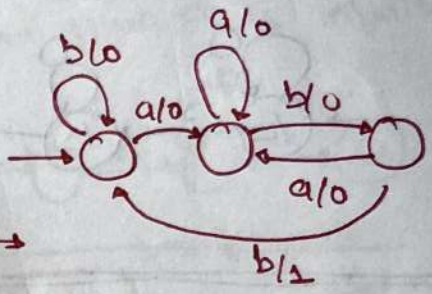
④ Design a mealy & moore machine over {a,b} which can produce o/p - 1 for every occurance of the substring abb in the given i/p string.



i.e Moore machine



can be converted into



i.e mealy machine

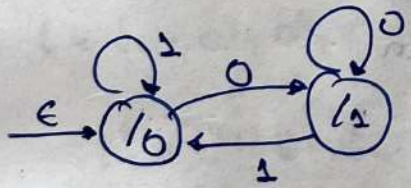
⑤ Design a Mealy & Moore Machines over {0,1} which can produce

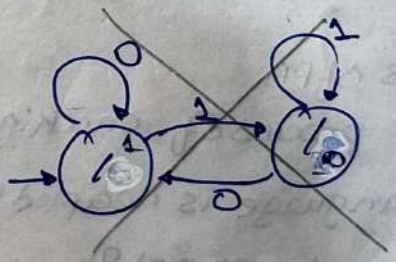
① 1's complement of the i/p str

② 2's complement of the i/p str.

∴ if i/p start with "0" then send it to "1" of i/p 1 state

moore machine





i/p = ~~0XX~~

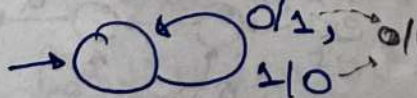
o/p = 100

0100

But i/p: 0XX

o/p: 1100 X

mealy machine

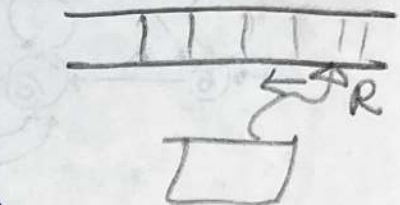


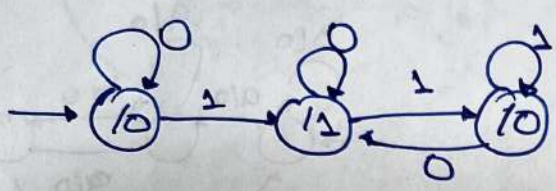
2's Complement

I/P: 1001100 → only Change in 1st complement

O/P: 0110100

from RHS leave all 0s & one 1's & then change after that in alternatively





X

Membership problem of FA

- It is a process of checking wheather a given string $w \in \Sigma^*$ is belong to the language accepted by the given "FA"

i.e checking wheather "w" is accepted by the given FA or not.

$\delta(q_0, w) = q_f$ where $q_f \in F$

Emptiness of FA

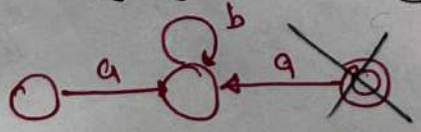
- It is the process of checking wheather the language is accepted by the given FA is empty or not?

procedure

→ 1. delete unreachable state

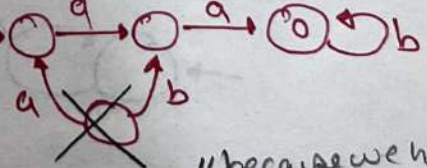
2. In the result of the automata if there is atleast one final state then the language accepted by the given "FA" is non-empty

ex:



$L = \phi$

ex



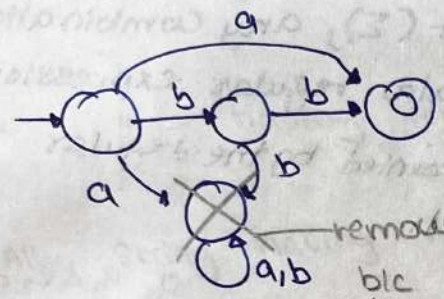
$L \neq \phi$ // because we have reachable final state

Finiteness of FA

- It is the process of checking whether the language accepted by given FA is finite or infinite?

- procedure
1. Delete unreachable state
2. Delete the state from which we can't reach any of the final state
3. In the result of the automata if there is atleast one self loop or a cycle then the language accept by the given FA is infinite.

i.e

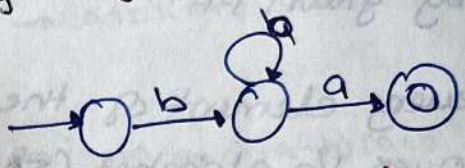


bic these state can't reach to final state

L is finite

$L = \{a, bb\}$

i.e



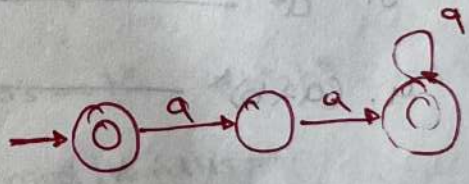
L is infinite

$L = \{aa, aba, ab^3a, \dots\}$

W.B

(191) $L = \{(aa)^* / p$ is a prime no.$\}$

$L = \{aP / P$ is a prime no.$\}$



$L = \{a^2, a^3, a^5, a^7, a^{11} \dots\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \dots$

$L^0 = \{\epsilon\}$, $L^1 = L$ & $L^3 = L \cdot L \cdot L$

$L^2 = L \cdot L$

$= \{a^2, a^3, a^5, a^7, a^{11} \dots\}\{a^2, a^3, a^5, a^7, a^{11} \dots\}$

$\Rightarrow \{a^4, a^5, a^6, a^8, a^9, a^{10}, \dots$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \dots$

$= \{\epsilon, a^2, a^3, a^4, a^5, a^6, a^7 \dots\}$

Regular Expression

- Regular Expression
- FA ⇄ RE
- Myhill nerode theorem
- pumping lemma & pumping length
- Properties of Regular language

- Regular Expression

→ It is an Algebraic representation of set of all string accept by given FA

→ every element of the I/p alphabet (Σ), any combination over I/p alphabet (Σ^*), ϵ & ϕ are also regular expression

Regular set: the set of all string represented by the regular expression is called as regular set.

If R_1 & R_2 are regular then

$R_1 + R_2$

$R_1 R_2$

R_1^+

are also regular

RE	Regular set
1. a+b	{a,b}
2. ab	{ab}
3. a*	{ε, a, a², a³, ---}
4. (a+b)*	{a,b}*

{ε, a, a², a³, a⁴ --
b, b², b³ --
ab, a²b² --- }

5. {aa, bb, ab} ⟶ aa + bb + ab

6. {ε, ab, ba} ⟶ ε + ab + ba

7. {a, a², a³ ---} ⟶ {a}⁺ ⟶ or aa* or a*a

8. {a², a⁴, a⁶, a⁸ --} ⟶ {a²}⁺
or a²(aa)*
or (aa)*a²

By taking common

a{ε, a, a², a³, ---}

⇒ aa*

Q] Construct a regular expression for the following language over $\{a, b\}$

(i) All string starting with a. ⟶ $a(a+b)^*$

(ii) All string ending with b ⟶ $(a+b)^* b$

(iii) All string starting with a & ending with b ⟶ $a(a+b)^* b$

(iv) " " having length exactly 3 → $(a+b)(a+b)(a+b)$

(v) " " having length atleast 3 → $(a+b)(a+b)(a+b)(a+b)^*$

(vi) All string having length atmost 3 → $(a+b+\epsilon)(a+b+\epsilon)(a+b+\epsilon)$

(vii) All string having length divisible by 3 → $((a+b)(a+b)(a+b))^*$

(viii) " " " " exactly 2's → $b^*ab^*ab^*$

(ix) " " " " atleast 2a's → $(a+b)^*a(a+b)^*a(a+b)^*$

(x) All string having length atmost 2a's → $b^* + b^*ab^* + b^*ab^*ab^*$
or $b^*(\epsilon+a)b^*(\epsilon+a)b^*$

(xi) All string having 'a' as the 3rd symbol from LHS → $(a+b)(a+b)a(a+b)^*$

(xii) " " " " " " " from RHS
→ $(a+b)^*a(a+b)(a+b)$

(xiii) All string having even no. of a's → $(b^*ab^*ab^*)^*b^*$ or $(b+ab^*a)^*$

(xiv) " " " odd no. of a's → $b^*(ab^*ab^*)^*ab^*$ or $(b+ab^*a)^*ab^*$

(xv) " " either starting with aa or bb → $(aa+bb)(a+b)^*$

(xvi) All String having substring either aa or bb

→ $(a+b)^*(aa+bb)(a+b)^*$

(xvii) " " having equal no. of a's & b's.

→ not regular

(xviii) " " $N_a(w) > N_b(w)$

→ not regular

∴ Regular expression only for Regular language

IDENTITIES

1. $\phi + R = R = R + \phi$ (any Regular expression)
2. $\phi R = \phi = R\phi$
3. $\epsilon R = R = R\epsilon$
4. $\epsilon + R = R + \epsilon$
5. $\epsilon^* = \epsilon$ & $\phi^* = \epsilon$
6. $R + R = R$
7. $R^* R^* = R^*$
8. $RR^* = R^+ = R^*R$
9. $(R^*)^* = R^*$
10. $\epsilon + RR^* = R^* = \epsilon + R^*R$
11. $R(QP)^* = (PQ)^*P$
12. $(P^* + Q^*)^* = (P+Q)^* = (P^*Q^*)^*$
13. $(P+Q)R = PR + QR$
14. $\{\phi\}^* = \{\epsilon\}$

ARDEN'S theorem's

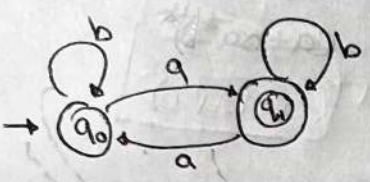
- if P & Q are 2 RE's → Regular Expression or Regular language. and

If $P \neq \epsilon$ then the following eqn in R,

$R = Q + RP \Rightarrow$ i.e $R = QP^*$

has unique & is same solution

- Construction of Regular expression for given 'FA'



$q_0 = \epsilon + q_0 b + q_1 a$

have 3 edge incoming in q_0

2 incoming edge to q_1

$q_1 = q_0 a + q_1 b$

$R = Q + RP$

$[R = Q + RP]$

$[R = QP^*]$

$\Rightarrow q_1 = q_0 ab^*$ (1)

Using eqn (1)

$q_0 = \epsilon + q_0 b + q_1 a$

$q_0 = \epsilon + q_0 b + q_0 ab^* a$

$q_0 = \epsilon + q_0 (b + ab^* a)$

$q_0 = \epsilon (b + ab^* a)^*$

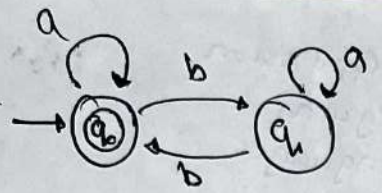
$q_0 = (b + ab^* a)^*$

String that can be formed in q_0 state

$q_1 = q_0 ab^*$

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

ex:



$q_0 = \epsilon + q_0a + q_1b$ & $q_1 = q_0b + q_1a$

(R) (Q) (R) (P)

using eq^n (1)

$\Rightarrow q_0 = \epsilon + q_0a + q_0ba^*b$

$\Rightarrow \boxed{q_1 = q_0ba^*}$ — (1)

$\Rightarrow q_0 = \epsilon + q_0(a + ba^*b)$

(R) (Q) (R) (P)

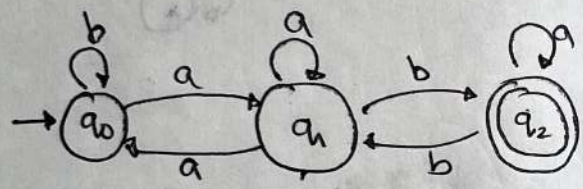
$\Rightarrow q_0 = \epsilon(a+ba^*b)^*$

ba^*

then $q_1 = (a+ba^*b)^*ba^*$

$q_0 = (a+ba^*b)^*$

ex:



$q_0 = \epsilon + q_0b + q_1a$ & $q_1 = q_0a + q_1a + q_2b$ & $q_2 = q_1b + q_2a$

(R) (Q) (R) (P)

$\Rightarrow q_0 = \epsilon + q_0b + q_0a(a+ba^*b)^*$ $\Rightarrow q_1 = q_0a + q_1a + q_1ba^*b$

$\Rightarrow \boxed{q_2 = q_1ba^*}$ — (1)

$q_0 = \epsilon + q_0(b + a(a+ba^*b)^*)$ $\Rightarrow q_1 = q_0a + q_1(a+ba^*b)$

(R) (Q) (R) (P) (R) (Q) (R) (P)

$q_0 = \epsilon(b+a(a+ba^*b)^*)^*$ $\Rightarrow \boxed{q_1 = q_0a(a+ba^*b)^*}$ — (2)

$\boxed{q_0 = (b+a(a+ba^*b)^*)^*}$ — (3)

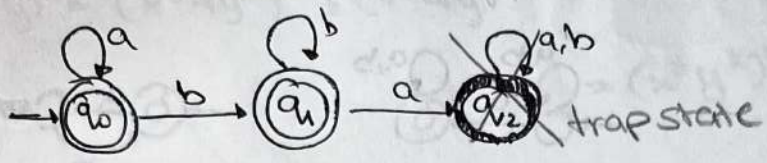
$\Rightarrow \boxed{q_1 = (b+a(a+ba^*b)^*)^*a(a+ba^*b)^*}$

value of q_1

$\boxed{q_2 = (b+a(a+ba^*b)^*)^*a(a+ba^*b)^*ba^*}$

String that can be generate by q_2

ex:



- If there is more than one final state then ans is sum of all final state.

$q_0 = \underset{R}{\epsilon} + \underset{Q}{q_0}\underset{R}{a}\underset{P}{}$ & $q_1 = \underset{R}{q_0}\underset{Q}{b} + \underset{R}{q_1}\underset{P}{b}$ & $q_2 = q_1 a + q_2(a+b)$

$q_0 = \epsilon a^*$ — (1)

$\Rightarrow q_1 = q_0 b + q_1 a(a+b)^* b$

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

$q_1 = q_0 b b^*$

using eqn (1)

$q_1 = a^* b b^*$

$q_1 = q_0 b (a(a+b)^* b)^*$

$q_2 = [a^* b (a(a+b)^* b)^* a (a+b)^*]$

$\Rightarrow q_1 = a^* b (a(a+b)^* b)^*$

$\Rightarrow q_0 + q_1 = a^* + a^* b b^*$

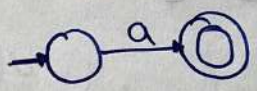
$\Rightarrow a^*(\epsilon + bb^*)$

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

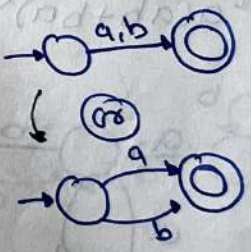
- Construction of a Diagram for RE → Regular Expression.

ϵ:

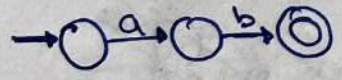
a:



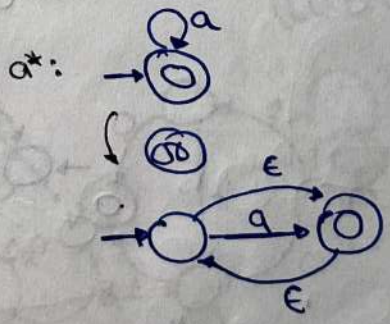
a+b:



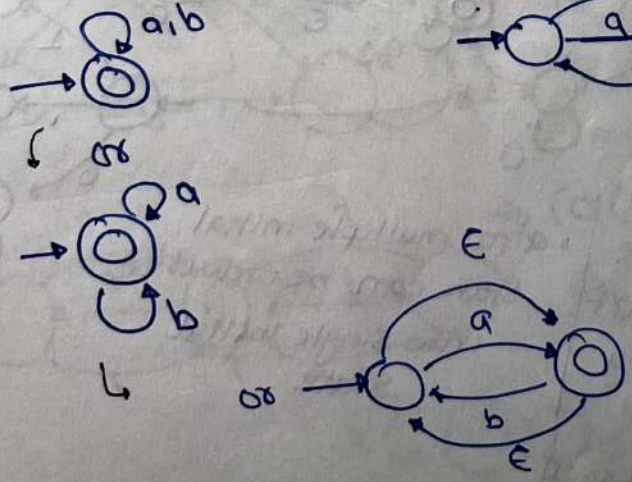
ab:



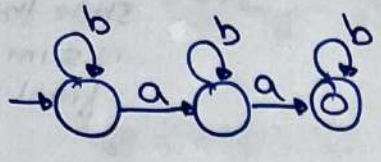
a^*:



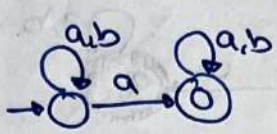
$(a+b)^*$:



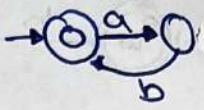
ex: 1. $b^*ab^*ab^*$



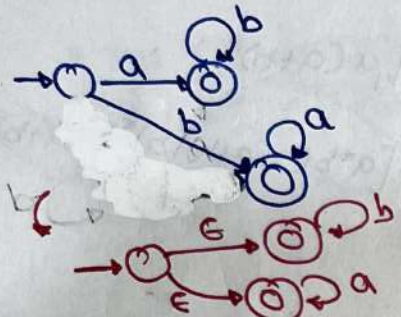
2. $(a+b)^*a(a+b)^*$



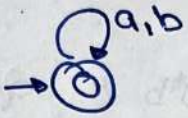
3. $\epsilon+ab$



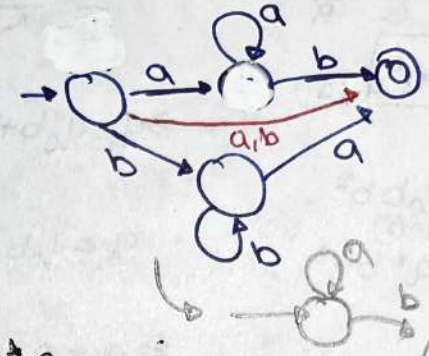
4. $ab^*+b^*a^*$



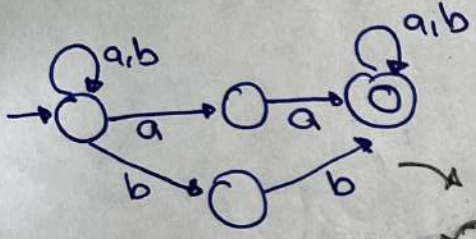
5. a^*+b^*



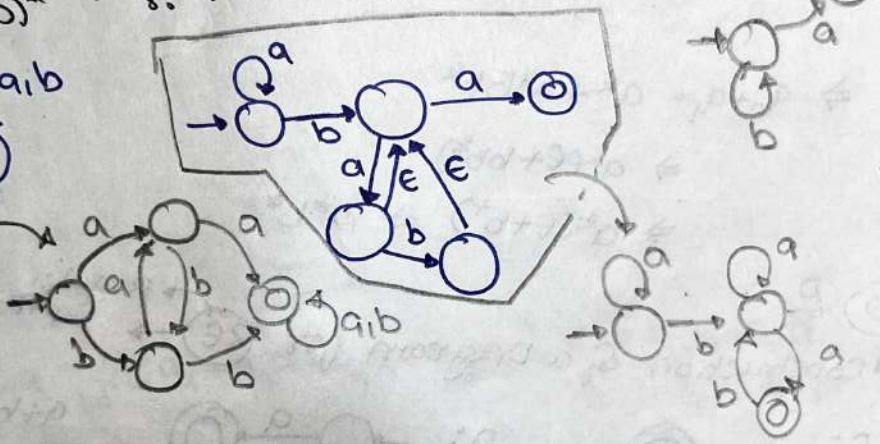
6. a^*b+b^*a



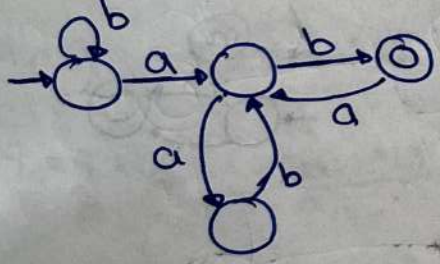
7. $(a+b)^*(aa+bb)(a+b)^*$



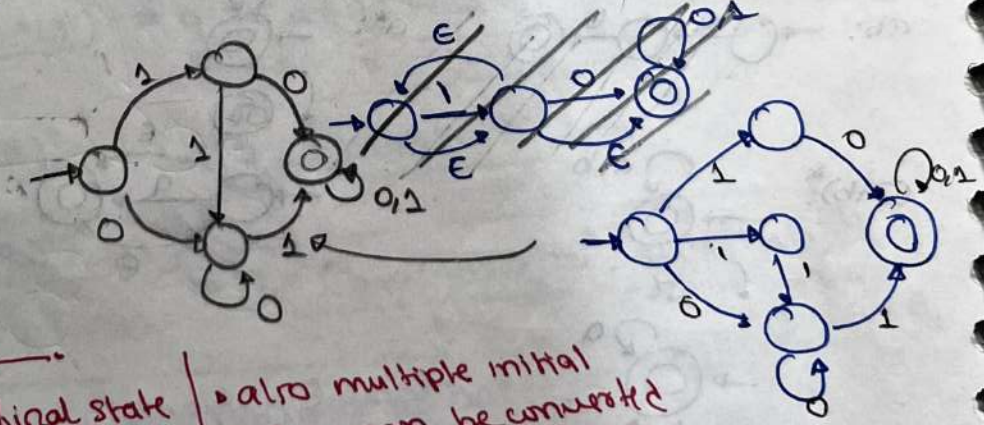
8. $a^*b(ab+a)^*a$



9. $b^*a(ab+ba)^*b$



10. $[10+(11+0)0^*1](0+1)^*$



Note

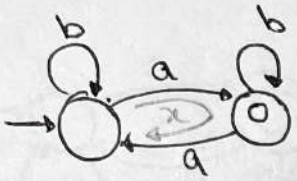
- If we have multiple final state in NFA then we can convert it into single final state by just adding "ε" Epsilon. But you can't convert a DFA as it doesn't contain "ε"
- also multiple initial state can be converted into single initial state

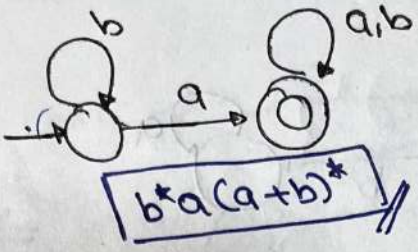
• $(x+y)^* = (x^*+y^*)^* = (x^*+y^*)^* = (x+y^*y^*) = (x+y)^* \quad x^*y^*$

$= (x^*y^*)^* = (y^*x^*)^*$

• $(x+y)^* \neq (x^*y^*)^*$

ex:





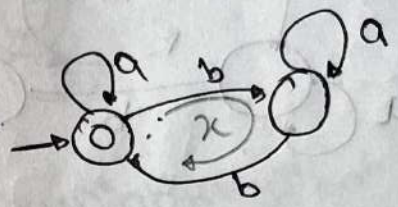
~~b^*~~ab^*ab^*

If there is a possibility to come back to b^* again then we don't write it as b^* instead

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

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

ex:-

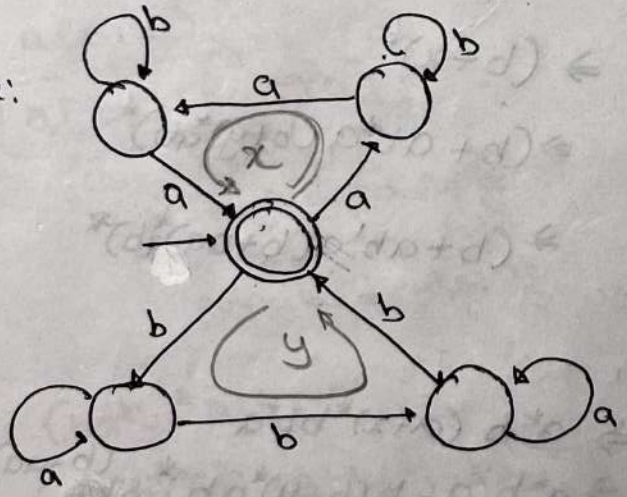


$(a+x)^*$

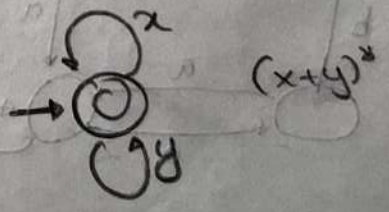
$(a+ba^*b)^*$

// we are finding for final state

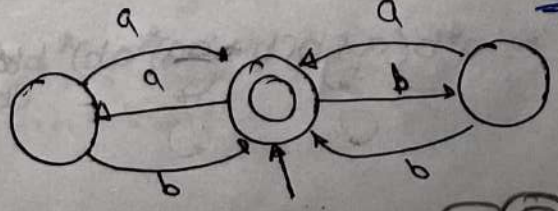
ex:



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

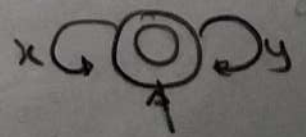


ex:



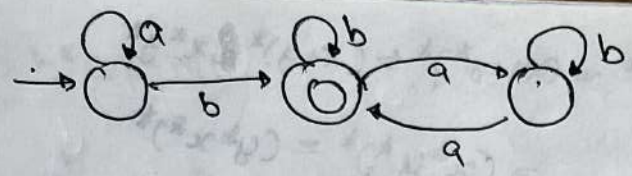
$\Rightarrow (a(a+b)+b(a+b))^*$

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



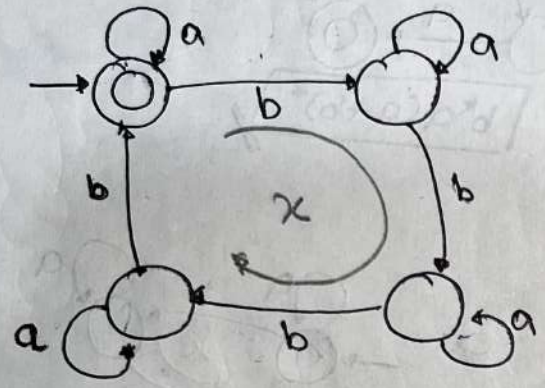
$\Rightarrow (x+y)^*$

ex:



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

ex:

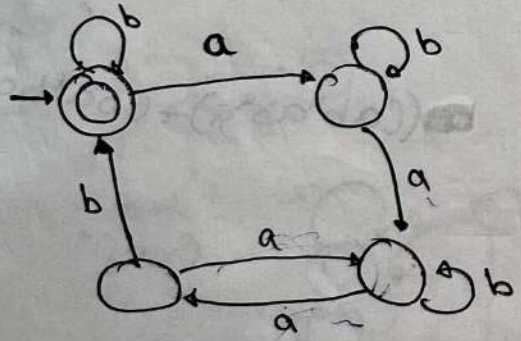


$\Rightarrow (a+x)^*$

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

$\Rightarrow$

ex:

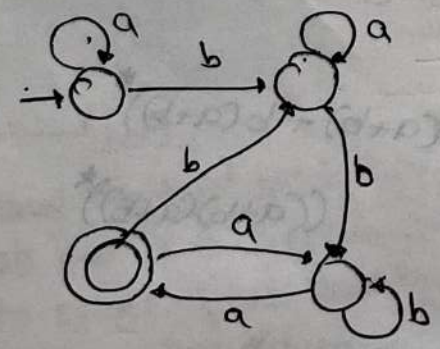


$\Rightarrow (b+x)^*$

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

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

ex:

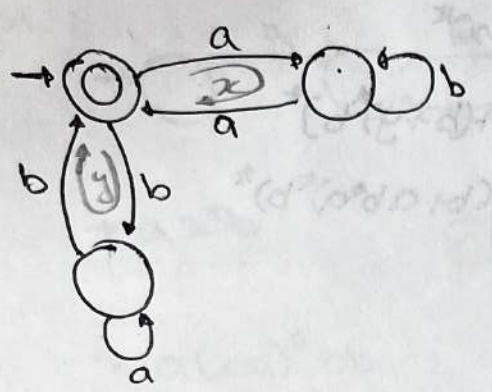


$\Rightarrow a^*b\ (a+x)^*bb^*a$

$\Rightarrow a^*b(a+b(b+y)^*ab)^*b^{(b+aa)^*a}$

$\Rightarrow a^*b(a+b(b+aa)^*ab)^*b^{(b+aa)^*a}$

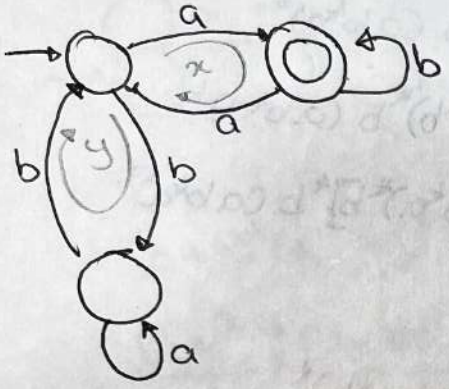
ex:



$\Rightarrow (x+y)^*$

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

ex



$\rightarrow (x+y)^*ab^*$

$\rightarrow (ab^*a+ba^*b)^*ab^*$

OR

1st go to final state

$\rightarrow a(b+x)^*$

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

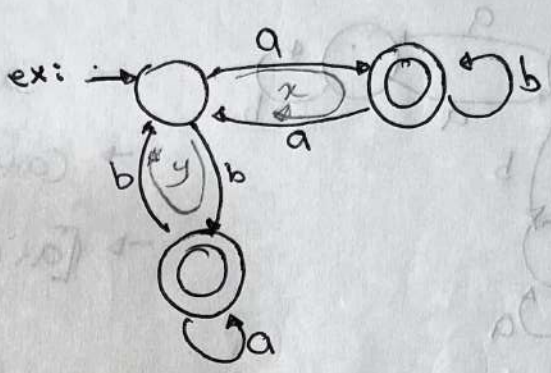
can accept bba
so there is a possiblity
from initial state befor
going to final state

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

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

we did this b/c final state is
not initial state

ex:

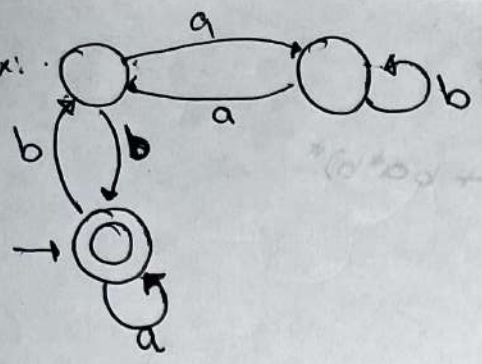


$\rightarrow (x+y)^*ab^*+(x+y)^*ba^*$

$\rightarrow (ab^*a+ba^*b)ab^*+(ab^*a+ba^*b)ba^*$

$\rightarrow [(ab^*a+ba^*b)(ab^*+ba^*)]$

ex:

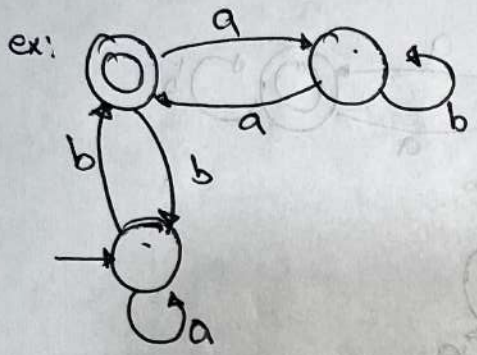


→ $(a+x)^*$

→ $(a+(b+y)^*b)^*$

→ $(a+(b+ab^*a)^*b)^*$

ex:



→ $(a+x)^*b\,(ab^*a)^*$

→ $(a+(b+y)^*b)^*b\,(ab^*a)^*$

→ $[a+(b+ab^*a)^*b]^*b\,(ab^*a)^*$

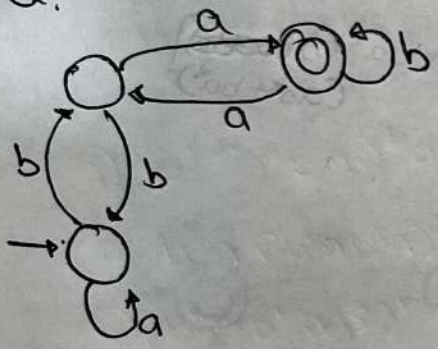
OR

1st going no final state

→ $a^*b\,(x+y)^*$

→ $a^*b\,(ab^*a+ba^*b)^*$

ex:



→ $(a+x)^*\,b\,(ab^*a)^*ab^*$

→ $(a+(b+y)^*b)^*\,b(ab^*a)^*ab^*$

→ $(a+(b+ab^*a)^*b)^*$

OR

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

→ $a^*bcba^*b)^*a(b+x)^*$

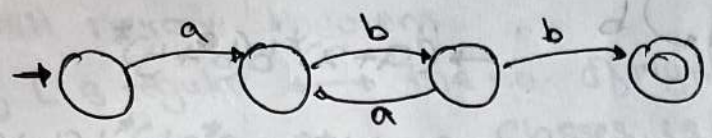
→ $a^*b(ba^*b)^*a[b+a(ba^*b)^*a]^*$

OR

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

$a^*b[ab^*a+ba^*b]^*ab^*$

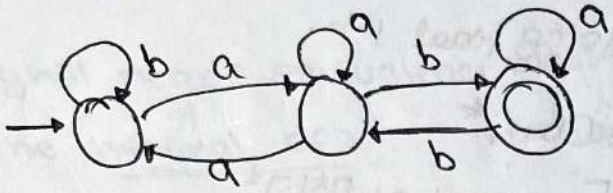
ex:



→ ax^*b OR $ab(ab)^*b$

→ $a(ba)^*bb$

ex:

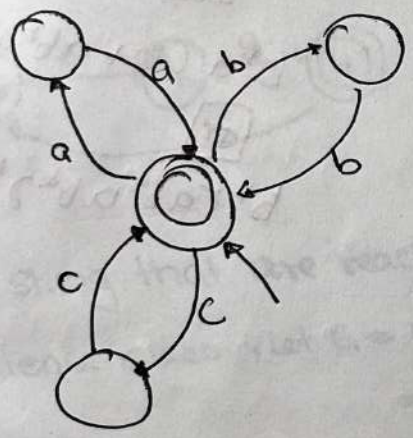


→ $(b+x)^*(a+y)^*ba^*$

→ $(b+x)^*(a+ba^*b)^*ba^*$

→ $[b+a(a+ba^*b)^*a]^*a(a+ba^*b)^*ba^*$

ex:

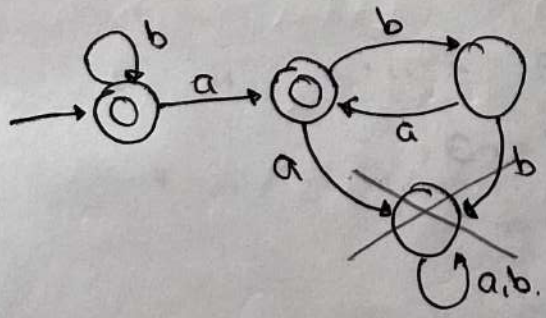


→ $(x+y+z)^*$

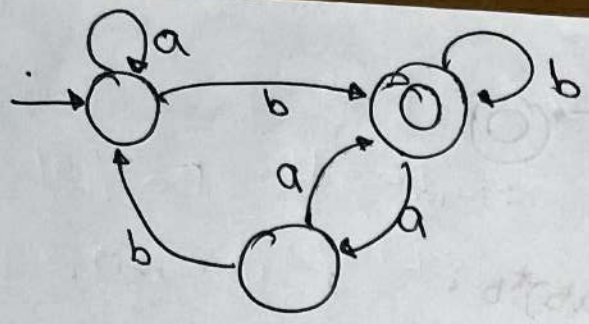
→ $(aa+bb+cc)^*$

→ $b^*a(ba)^*+b^*$

ex:



ex.



→ $(a+x)^*b(b+y)^*$

→ $(a+b\ (b+y)^*ab)^*b(b+aa)^*$

⇒ $(a+b(b+aa)^*ab)^*b(b+aa)^*$

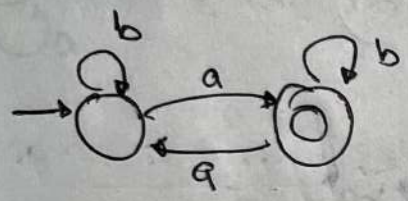
or

↳ going final 1st

→ $a^*b(b+x)^*$

→ $a^*b[b+aa+ab a^*b]^*$

ex.



OR

→ $(b+x)^*ab^*$

→ $(b+ab^*a)^*ab^*$

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

or

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

or

$b^*(ab^*ab^*)^*ab^*$

× a) $b^*(a^*ba)^*ab^*$? aaba

× b) $b^*a\cdot(b^*ab^*a)^*$ ab

× c) $(b^*ab^*ab^*)^*ab^*$ ba

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

- Myhill nerode theorem
  - If L is regular ⟷ the no. of Myhill nerode equivalence classes is finite (include the dead state).
  - If L is not regular ⟷ The no. of Myhill Nerode equivalence classes is infinite

Myhill nerode equivalence classes means the no. of state in the minimal DFA (we don't talk about NFA)

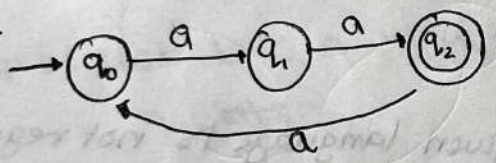
Equivalence relation over Σ^*:-

Two string x & y are said to be equal iff

$\delta^*(q_0, x) = \delta^*(q_0, y)$ where q_0 is initial state.

i.e All those string that are reaching the same state are same equivalence class.

ex:



- All the string that are reaching to the state q_0 are one equivalence class, let $\epsilon_1 = RE(q_0)$

  $= (aaa)^*$ i.e $\epsilon_1 = \{\epsilon, a^3, a^6, a^9 \ldots\}$

- " " " " " " State q_1 are " "

  " " , $\epsilon_2 = RE(q_1)$

  $= a(aaa)^*$ $\epsilon_2 = \{a, a^4, a^7, a^{10} \ldots\}$

- " " " " " State q_2 $= \epsilon_3 = \{a^2, a^5, a^8, a^{11} \ldots\}$

  $\epsilon_3 = RE(q_2)$

  $= aa(aaa)^*$

- properties of Myhill Nerode equivalence class.

(i) $|E_i| > 0 \quad \forall i = 1, 2, 3 \ldots$

(ii) $E_i \cap E_j = \phi$ if $i \neq j$

(iii) $\cap E_i = \phi$

(iv) $\cup E_i = \Sigma^*$

- Application of Myhill nerode theorem
- to prove that a language is regular
- To minimize the no. of state of a DFA

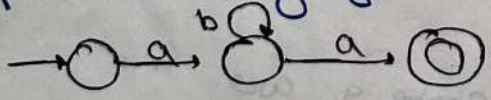
Pumping lemma (PL) [only for language not for grammer or automata

PL Test → Pass → Regular/not Regular

↓ Fails

not Regular.

- PL is used for proving a given language is not regular
- PL works on Pigeon hole principle
- PL use the proof by contradiction, i.e it is used to prove a language is not regular.



$L = \{aa, aba, ab^2a, \ldots\}$

- we can't use PL to prove that a language is regular
- PL gives a necessary condition but not the sufficient condition

not Imp

Pumping lemma statement: $\{L = w_1, w_2, w_3 \cdots w_n \cdots\}$

Step 1: Let L is regular and let M is a FA accepting L with "n" no of states

Step 2: Let $w \in L$, $|w| > n$ & decompose w as xyz such that

$|xy| \leq n$; $|y| \geq 1$

Step 3: $w_i = x(y)^i z$ also in L, $\forall i = 0, 1, 2 \cdots$

Find an i-value such that $w_i \notin L$

This contradicts the stmt $\underline{1}$

$\Rightarrow$ L is not regular.

▸ PT $L = \{a^n b^n / n > 0\}$ is not regular

$= \{ab, a^2b^2, a^3b^3, \cdots\}$

Step 1: Let L is a regular & M is a FA accepting L with n states.

Step 2: Let $w = aabb \in L$

$w = \underset{x}{a}\,\underset{y}{(ab)}\,\underset{z}{b}$

Step 3: $w_2 = xy^2z$

$= a(ab)^2 b$

$= aababb \notin L$

$\Rightarrow$ L is not regular//

Pumping length

$L = \{w_1, w_2, w_3, w_4 \ldots\}$ (each w_i is a string)

If "L" is a regular language then there exists +ve integer K such that for any string $w \in L$; $|w| \geq K$ will have some string (w') such that by pumping that substring any no. of time thus formed all string also be in L. there $1 \leq |w'| \leq K$ Any no. greater than or equal to MPL is called as pumping length (PL)

{pumping length can't be zero i.e $PL \geq 1$ & $MPL \geq 1$}

// what every you pump that all string by pumping will appear in "L" — min pumping length

$L = 01^*0$

$= \{00, 010, 0110, 01110 \ldots\}$

Let $|w| = 1$

Let $w = (0)^*$

$= \{\epsilon, 0, 00, 000 \ldots\} \notin L$

Let $w = 00$

$= (0)^*0$

$= \{0, 00, 000 \ldots\} \notin L$

PL can't be 2

Let $w = (1)^*$

$= \{\epsilon, 1, 11, 111 \ldots\} \notin L$

$\Rightarrow$ PL can't be 1

Let $w = 0\underline{1}0$

$= 0(1)^*0 \cdot \Rightarrow \{00, 010, 0110 \ldots\} \in L$

$\therefore MPL = |010| \Rightarrow PL \geq MPL$

$= 3$ $\quad PL = \{3, 4, 5 \ldots\}$

- properties

a(bb)*

- MPL ≥ 1
- MPL ≤ n [if the no. of states in the minimal DFA is n]
- MPL ≤ n-1 [if the DFA has dead state]
- MPL > $|w_{min}|$ [where w_{min} is the smallest string in the infinite lang.]
- MPL = $|w_{max}|+1$ [where w_{max} is the largest string in the finite language]
- If $L = \phi \Rightarrow$ MPL = 1
- If $L = \{\epsilon\} \Rightarrow$ MPL = 1
- the string with length is equal to MPL need not be in the language.
- if $L = L_1 \cup L_2 \Rightarrow MPL(L) = \max[MPL(L_1), MPL(L_2)]$
  - out of L_1 & L_2 if one is subset of the other than MPL (L) is the MPL of superset language
  - By taking $L_1 \cup L_2$ if we get a new language MPL (L) is MPL of new language

ex:- $L = \{01, 010, 1011, 0100\}$

MPL = $|w_{max}|+1$

= $|1011|+1$

= 4+1

= 5

ex: $L = 1001^*$

= {100, 1001, 10011, ----}

MPL = $|w_{min}|+1$

there is not true always (which case?)

⇒ |100|+1

⇒ 3+1

⇒ 4

Let. w = 1001

= 100(1)*

⇒ {100, 1001, 10011---} ∈ L

MPL = |w|

⇒ |1001| ⇒ 4

Q) regular expression → aaaa* .

$L = \{a^n / n \geq 3\} \cup \{b^n / n \geq 5\}$ → bbbbbb*

↓ $|\omega_{min}|+1$ ↓ $|\omega_{min}|+1$

$\Rightarrow |aaa|+1$ $|bbbbbb|+1$

$\rightarrow 4$ 6

$\Rightarrow MPL(L) = \max(4,6)$

$= 6$ //

Q) $L = \{aa\} \cup \{aaaa^*\}$ → infinite length

finite length ⇒

$|\omega_{min}|+1$ $|\omega_{min}|+1$

↓ ↓

2+1 3+1

↓ ↓

3 4

⇒ $MPL = \max(3,4)$ ~~✗~~

→ $L = \{aa\} \cup \{aaaa^*\}$

↓ ↓

aa $\{a^3, a^4, a^5 \ldots\}$

⇒ aaa* $\{a^2, a^3, a^4, \ldots\}$

$MPL(L) = |aa|+1$

$= 3$ //

Q] $L = \{aaa\} \cup \{aa^*\}$ aaa is subset of aa^*.

a^3 ⊂ $\{a, a^2, a^3, \ldots\}$

$\Rightarrow L = \{a, a^2, a^3 \ldots\}$; aa^*

MPL = |a| + 1

⇒ 2//

Q] $L = (01)^*$ ∴ MPL = 2

$L = \{\epsilon, 01, 0101, \ldots\}$

MPL(L) = |ε| + 1

= 0 + 1

= 1

|ω| = 1 not possible

let $\omega = (0)^*$

$= \{\epsilon, 0, 00, 000 \ldots\} \notin L$

let $\omega = (1)^*$

$= \{\epsilon, 1, 11, 111 \ldots\} \notin L$

let ω = 01

$= (01)^*$

$= \{\epsilon, 01, 0101 \ldots\} \in L$

∴ MPL = |ω| = |01|

= 2//

Q] $L = 0^*1^*$

$L = \{\epsilon, 0, 1, 01, 11, 001, 100, 011 \ldots\}$

L can be written as

$0^* + 1^* + 0^+1^+$

$\omega = (0)^+$

$= \{\epsilon, 0, 0^2, 0^3 \ldots\} \in L$

MPL = |01|

⇒ 1//

Q] $L = 001 + 10^*$

|001| + 1 . |11| + 1

⇒ 4 ⇒ 2

MPL(L) = max(4, 2)

⇒ 4// not merging so take the max among two.

Q] $L = (a+b)^* ab (a+b)^* + ba^*$

these both are complement of each other.

a^* ×
b^* ×
b^*a^* ×

can be generate

these can generate these

$L = (a+b)^*$ new language

$MPL(\in L) = |\in| + 1$

$\Rightarrow 0+1$

$\Rightarrow 1$ //

Q] $L = a(a+b)^* + b(a+b)^* + \in$

$\Rightarrow (a+b)^*$

$MPL = |w_{min}| + 1$

$= |\in| + 1$

$= 1$ //

• Properties of Regular language

1. Regular language are closed under

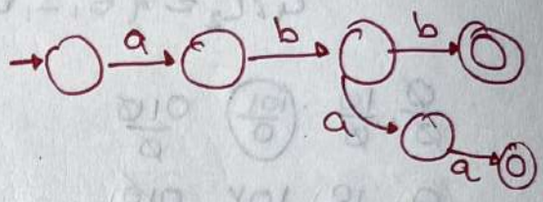
- union
- Intersection
- Concatenation
- Reversal
- Kleen's closure/ star closure (L^*)
- positive closure (L^+)
- Complement ($\bar{L}$)
- Set difference
- Quotient (Right & left) L_2 / L_1 & $L_1 \backslash L_2$

If L_1 & L_2 are regular

- $L_1 \cup L_2$ is also regular
- $L_1 \cap L_2$ is also regular
- $L_1 \cdot L_2$ is also regular

If L is Regular ⟶ L^R is also regular

Let $L = \{abb, abaa\}$



$L^R = \{bba, baba\}$

note

- complement of a L would be inter-changeng final & nonfinal State
- where as reversal of L would be changing / inter changig initial & final state & by changig the arrow. we will get reversal.

If L_1 & L_2 are regular

- $L_1 - L_2$ is also regular

$L_1 - L_2 = L_1 \cap \bar{L}_2$

$\Rightarrow$ Regular $\cap$ $\overline{\text{Regular}}$

$\Rightarrow$ Regular $\cap$ Regular

$\Rightarrow$ Regular.

- L_1 / L_2 & $L_1 \backslash L_2$ are also regular

* Right Quotient :- such that L_1 suffix match with L_2 completely & there the remaining is the result.

$$L_1/L_2 = \{x \mid xy \in L_1 \& y \in L_2\}$$

$$L_1/L_2 \Rightarrow \frac{xy}{y} = x$$

ex: let $L_1 = \{0, 10, 101, 010\}$

$L_2 = \{0, 1, 11, 00, 01, 110\}$

$L_1/L_2 = \{\epsilon, 1, 01, 10\}$

$L_1/L_2 = \{\epsilon, 10, 0, 01\}$

$\frac{\emptyset}{\emptyset}$ $\frac{10}{\emptyset}$ $\frac{101}{0}$ $\frac{010}{\emptyset}$

$\frac{0}{1}$ $\frac{10}{1}$ $\frac{10x}{x}$ $\frac{010}{1}$

$\frac{0}{11}$ $\frac{10}{11}$ $\frac{10x}{1x}$ $\frac{010}{11}$

$\frac{\emptyset}{00}$ $\frac{10}{00}$ $\frac{101}{00}$ $\frac{010}{00}$

$\frac{\emptyset}{\emptyset}$ $\frac{10}{0}$ $\frac{101}{0}$ $\frac{010}{0}$

$\frac{0}{1}$ $\frac{10}{x}$ $\frac{101}{x}$ $\frac{010}{1}$

$\frac{\emptyset}{01}$ $\frac{10}{01}$ $\frac{101}{01}$ $\frac{0x0}{0x}$

$\frac{\text{Finite}}{\text{Infinite}} = \text{Finite}$ & $\frac{\text{Finite}}{\text{Finite}} = \text{Finite}$

Right Quotient L_1/L_2

$\frac{\text{Infinite}}{\text{finite}} = \text{Infinite}$ finite

$\frac{\text{Infinite}}{\text{Infinite}} = \text{Infinite}$ finite

we can get both finite or Infinite

ex:

(i) $\frac{a^*}{a} \Rightarrow \frac{(\epsilon + a + aa + aaa \cdots)}{a} \Rightarrow a^*$

(ii) $\frac{a^+}{a} \Rightarrow \frac{(a + aa + aaa + \cdots)}{a} \Rightarrow a^*$

(iii) $\frac{ab^*}{a} \Rightarrow \frac{(a + ab + abb \cdots)}{a} = \{\epsilon\}$

(iv) $\frac{ab^*}{b^*} \Rightarrow \frac{(a + ab + abb + abb \cdots)}{(\epsilon + b + bb + bbb \cdots)} \Rightarrow \{a, ab, abb \cdots\}$

$\Rightarrow ab^*$

(v) $\frac{ab^*}{ba^*} \Rightarrow \frac{(a + ab + abb + abb + \cdots)}{(b + ba + baa + baaa \cdots)} = \{a, ab, abb \cdots\}$

$\{ab^*\}$

(iv) $\frac{ab^*}{bat} \Rightarrow \frac{(ab+abb+abbb\cdots)}{(ba+baa+baaa+\cdots)} \Rightarrow \{\epsilon\}$

Q] If $L_1 = \{ab a^m / m>0\}$ & $L_2 = \{a^n / n \geq 0\}$ then $L_1 \backslash L_2$ is

$\Rightarrow \frac{L_1}{L_2} = \frac{(aba+abaa+abaaa\cdots)}{(\epsilon+a+aa+aaa+\cdots)} \Rightarrow \{ba+baa+baaa\cdots\}$

$\Rightarrow ba^*$

$\Rightarrow \{\underbrace{\frac{aba}{\epsilon} + \frac{abaa}{\epsilon} + \cdots}_{} + ba+baa+baaa\cdots\}$

$\Rightarrow \{ab^i a / i>0\} \cup \{ba^i / i>0\}$

$\Rightarrow \{a^i ba^j / i=0,1 ; j>0\}$

- Regular language are closed under
  - Exclusive OR (XOR) ⊕
  - Exclusive NOR (XNOR) [↔]
  - Implication [→]
  - NAND $[\overline{L_1 \cap L_2}]$
  - NOR $[\overline{L_1 \cup L_2}]$
  - Homomorphism (HOL)
  - ε-free Homomorphism
  - Substitution
  - ε-free substitution
  - Inverse Homomorphism

if L_1 & L_2 are regular $\xrightarrow{XOR}$ [Symmetric difference]

→ $L_1 \oplus L_2$ is also regular

$L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$

$= (L_1 \cap \bar{L}_2) \cup (L_2 \cap \bar{L}_1)$

$=$ reg. $\cup$ reg.

$=$ regular.

• Homomorphism:- It is a process of replacing an I/p symbol, by a string of another I/p alphabet.

Let Σ & Γ are I/p alphabet then homomorphism 'h' is defined as $h: \Sigma \rightarrow \Gamma^*$

these will be given in ques^n

Ex: Let $\Sigma = \{a,b\}$ & $\Gamma = \{0,1,2\}$ & $h(a) = 00$, $h(b) = 12$ then

if $w \in \Sigma^*$, $w = aba \rightarrow h(w) = h(a)\,h(b)\,h(a)$

$= 00\,12\,00$

Let $L = (a+b)^* \rightarrow h(L) = (00+12)^*$

ex: $\Sigma = \{a,b\}$ & $\Gamma = \{0,1,2\}$ & $h(a) = \epsilon$, $h(b) = \epsilon$

if $L = (ab)^* \Rightarrow h(L) = (\epsilon\epsilon)^*$

$= \{\epsilon\}$

* If L is finite → h(L) is also finite
* If L is infinite → h(L) may be finite/ infinite

• ϵ-free homorphism

In these every I/p symbol must be replaced by any string other than ϵ, $h: \Sigma \rightarrow \Gamma^+$

■ Substitution: It is a process of replacing an I/p symbol by a language of another input alphabet but that type of language must be of some type

Let $\Sigma = \{a,b\}$, $\Gamma = \{0,1,2\}$, $s(a) = 0^*$, $s(b) = (12)^*$

Substitution value should also be regular

let $L = (a+b)^* \Rightarrow s(L) = (0^* + (12)^*)^*$

$\Rightarrow (0+12)^*$

- ε-free substitution:
  In there every i/p symbol must be replaced by any language other than ε ; $h: \Sigma \to \Gamma^+$

- Inverse Homomorphism:

  if $h: \Sigma \to \Gamma^* \longrightarrow h^{-1}: \Gamma^* \to \Sigma$

let $\Sigma = \{a,b\}$, $\Gamma = \{0,1,2\}$, $h(a)=10$, $h(b)=20$, then if $L=(10+20)^*$ then $h^{-1}(L)=?$

$L=(10+20)^*$

$h^{-1}(L) = (a+b)^*$

Q] let $\Sigma = \{0,1,2\}$ & $\Gamma = \{a,b\}$; $h: \Sigma \to \Gamma^*$

if $h(0)=a$ & $h(1)=ab$, $h(2)=ba$ then

(i) if $L=\{ababa\} \Rightarrow h^{-1}(L)$?

(ii) if $L=a(ba)^* \longrightarrow h^{-1}(L)$?

(i) $h^{-1}(L) = (110)$
or (022)
or (102)
ab a ba

(ii) $h^{-1}(L) =$
if $L=a(ba)^*$
$\Rightarrow \{a, aba, ababa \ldots\}$
$\Rightarrow a\{\epsilon, ba, baba \ldots\}$
$\Rightarrow \{a, aba, ababa \ldots\}$

$h^{-1}(L) = \{0, 10, 02, 110, 102, 022, 1110, 1102 \ldots\}$

$\Rightarrow 1^*02^*$

Q] Let $\Sigma = \{0,1\}$ & $\Gamma = \{a,b\}$

& $h(0) = aa$, $h(1) = aba$

if $L = (ab+ba)^* a$ then $h^{-1}(L)$?

note
- if L is finite → $h^{-1}(L)$ may be finite/infinite

⇒ ~~$L = \{a, aba, baa, abaaba, abbaa, aabaa, \dots\}$~~

⇒ $L = \{a, aba, baa, abbaa, abaaba, abbaaaba, abbaba \dots\}$

$h^{-1}(L) = \{1\}$

- $h[h^{-1}(L)] \neq L$
  - need not be equal to L

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

$h^{-1}(L) = 1$

$h[h^{-1}(L)] = h(1)$

$= aba$

$\neq L$

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

& $h(0) = a$ & $h(1) = a$

If $L = 0^* 1^*$ then $h(L)$?

⇒ $L = 0^* 1^*$

$h(L) = a^* a^*$

$= a^*$

$h^{-1}(a) = 0$ & $h^{-1}(a) = 1$

Here $L = 0^*, 1^*$

$h^{-1}[h(L)] = h^{-1}[a^*]$

$= \{\epsilon, 0, 1, 00, 11, 000, 111, 01, \dots\}$

$= (0+1)^*$

$\neq L$

∴ $h^{-1}[h(L)]$ & $h[h^{-1}(L)]$ need not be equal to L.

3. Regular language are closed under

- Prefix / Init
- Suffix
- min
- max
- Half
- ALT
- Cycle

ex:

L	Init(L)	Suffix L.
a^*	$\{a^*\}$	$\{a^*\}$
a^+	a^*	a^*
$(ab)^*$	$\{ab\}^* + \{ab\}^*a$	$\{ab\}^* + b\{ab\}^*$
a^*b^*	a^*b^*	a^*b^*
$(a+b)^*$	$(a+b)^*$	$(a+b)^*$
$a^n b^n / n>0$	$\{a^m b^n / m \geq n \geq 0\}$	$\{a^m b^n / n \geq m \geq 0\}$

$\Rightarrow (ab)^* = \{\epsilon, ab, abab, ababab \cdots\}$

prefix $= \{\epsilon, a, ab, aba, abab, ababa \cdots\}$

suffix $= \{\epsilon, b, ab, bab, abab, babab \cdots\}$

$\Rightarrow a^+ = \{a, aa, aaa \cdots\}$

$\Rightarrow \{\epsilon, a, aa, aaa \cdots\}$

$\Rightarrow a^*b^* = \{\epsilon, a, b, aab, abb \cdots\}$

prefix $= \{\epsilon, a, b, aa, aab, ab, abb \cdots\}$

Suffix $= \{\epsilon, a, b, ab, aab, bb, abb \cdots\}$

$\Rightarrow (a+b)^* \rightarrow \{\epsilon, a, b, ab, ba, aab \cdots\}$

$\Rightarrow (a^n b^n / n>0)$

$\hookrightarrow L = \{ab, aabb, aaabbb \cdots\}$

Prefix $= \{\epsilon, a, ab, a^2, a^2b, a^2b^2, a^3, a^3b, a^3b^2 \cdots\}$

$L = \{w_1, w_2, w_3 \cdots\}$ aab

$\{\epsilon, a, aa \notin L\}$

$\min(L) = \{w_3\}$

↳ if any proper prefix of (w_3) is not present in L

$\text{Min}(L) = \{w \mid w \in L$, no proper prefix (w) is in $L\}$

Ex1 - $L = \{a^n \mid n \geq 3\}$

$= \{aaa, aaaa, aaaaa \cdots\}$

Prefix $= \{\epsilon, a, aa, aaa, aaaa \cdots\}$ (aaa ✗, aaaa ✗)

So, $\min(L) = a^3$

$L = \{w_1, w_2, w_3 \cdots\}$

$\max(L) = \{w_3\}$

$\text{Max}(L) = \{w \mid w \in L$, for no string x other than ϵ, $wx \in L\}$

In other word the string should not be prefix of any other string

Ex:- $L = \{a^n \mid n \geq 3\}$

$= \{aaa, aaaa \cdots aaaaa \cdots\}$

$\text{Max}(L) = \phi$

↓

no max exist.

$L = \{ab, aab, abb, babb, aabb\}$

$max(L) = \{abb, babb, aabb\}$

- $Half(L) = \{$ The set of first halves of all even length string of $L\}$

  Ex: $L = \{ab, aab, abba, babba, bbbaa, baba\}$

  $Half(L) = \{a, ab, ba\}$

- $ALT(w, x)$:

  Let $w = w_1 w_2 w_3 \cdots w_n$ & $x = x_1 x_2 x_3 \cdots x_n$ are 2 string of same length

  $ALT(w, x) = w_1 x_1 w_2 x_2 w_3 x_3 \cdots w_n x_n$ i.e taking alternative symbols of w & x starting with first element of w.

  $ALT(L_1, L_2) = ALT(w, x)$ where $w \in L_1$, $x \in L_2$ & $|w| = |x|$

- $Cycle(L) = \{w |$ we can write was yx if $xy \in L\}$

  ex. $L = \{ab, aab, babb\}$

  $\frac{ab}{x}\frac{}{y} \rightarrow ab$

  $\frac{}{x}\frac{ab}{y} \rightarrow ab$

  $\frac{a}{x}\frac{b}{y} \rightarrow ba$

  $Cycle(L) = \{ab, ba, aab, aba, baa, abbb, bbba, bbab\}$

4. Regular language are closed under

- Finite union
- Finite intersection
- Finite set difference

If L is regular of finite set then its subset will also be regular. but its superset will not be regular.

5. every finite language is regular

6. Regular language are not closed under

- Subset
- Superset
- Infinite union
- Infinite intersection
- Infinite set difference

} No language are closed under these operation.

∴ we say closed if it is closed in all case

ex: $L = \{a^m b^n / m, n > 0\}$ is regular.

$L' = \{ab, a^2b^2, a^3b^3 \ldots\ldots\}$

$= \{a^i b^i / i > 0\}$ is not regular

$L' \subset L$

ex: $L = \{a^n b^n / n \leq 100\}$ is regular

$L' = \{a^i b^i / i \geq 0\}$ is not regular.

$L \subset L'$

ex: $L_1 \cup L_2 \cup L_3 \cup \ldots\ldots\ldots \infty = \{a^n b^n / n > 0\}$ is not regular

Infinite union

ex: $L_1 = \{a^n b^n / n \leq 3\}$ is regular

$L_2 = \{a^n b^n / n \leq 5\}$

$L_3 = \{a^n b^n / n \leq 7\}$

$L_{50} = \{a^i b^n / n \leq 101\}$

finite no. of union.

So, $L_1 \cup L_2 \cup L_3 \cup \ldots\ldots\ldots \cup L_{50} = \{a^n b^n / n \leq 101\}$ is regular

ex: $L_1 \cap L_2 \cap L_3 \cdots \infty$ $\longrightarrow$ $A \cap B = \overline{\overline{A} \cup \overline{B}}$

$\Rightarrow \overline{\overline{L_1} \cup \overline{L_2} \cup \overline{L_3} \cdots \infty}$

$\Rightarrow \overline{\overline{reg} \cup \overline{reg} \cup \cdots \infty}$

$\Rightarrow \overline{reg \cup reg \cup \cdots \infty}$

$\Rightarrow \overline{\text{Not regular}}$

$\Rightarrow$ not regular.

Converse of properties

- If L is regular $\longleftrightarrow$ L^R is regular
- If L is regular $\longleftrightarrow$ $\overline{L}$ is regular

- If $L_1 \cup L_2$ is Regular $\longrightarrow$ L_1 & L_2 need not be regular

ex: Let, $L_1 = \{a^n b^n / n > 0\}$ is not regular

$L_2 = (a+b)^*$ is regular

$L_1 \cup L_2 = \{a^n b^n\} \cup (a+b)^*$

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

ex: $L_1 = \{a^n b^n / n \geq 0\}$ is not regular

$L_2 = \{a^m b^n / m \neq n\}$ is not regular

$L_1 \cup L_2 = \{a^i b^j / i, j \geq 0\}$ is regular

- If $L_1 \cap L_2$ is Regular $\longrightarrow$ L_1 & L_2 need not be regular

ex. Let $L_1 = \{a^n b^n / n \geq 0\}$ is not regular

$L_2 = \phi$ is regular

$L_1 \cap L_2 = \phi$ is regular

Let $L_1 = \{a^n b^n / n \geq 0\}$ is not regular.

$L_2 = \{a^{2n} b^n / n \geq 0\}$ is not regular

$L_1 \cap L_2 = \{\epsilon\}$ is regular

- If $L_1 \cdot L_2$ is regular $\longrightarrow$ L_1 & L_2 need not be regular

Let $L_1 = \{a^n b^n / n \geq 0\}$ is not regular

$L_2 = \phi$ is regular

$L_1 \cdot L_2 = \{a^n b^n\} \cdot \phi$

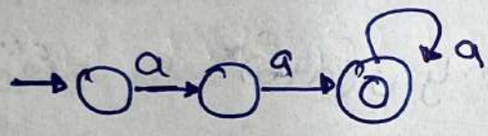
$= \phi$ is regular

Let $L_1 = \{a^p / p$ is a prime no.$\}$ is not regular

$L_2 = L_1$ is not regular

$L_1 \cdot L_2 = \{a^2, a^3, a^5, a^7 \ldots\} \cdot \{\epsilon, a, a^4, a^6, a^8 \ldots\}$

$= \{a^2, a^3, a^4, a^5 \ldots\}$



$L_1 \cdot L_2$ is regular

- If L^* is regular $\longrightarrow$ L need not be regular

Let $L = \{a^p / p$ is a prime no.$\}$ is not regular

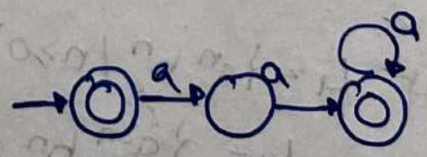
$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup L^4 \ldots$

$L^2 = L \cdot L = \{a^2, a^3, a^5, a^7 \ldots\} \cdot \{a^2, a^3, a^5 \ldots\}$

$= \{a^4, a^5, a^6, a^7, a^8, a^9, a^{10}, \ldots\}$ $\Rightarrow L^0 = \{\epsilon\}$

$L^* = L^0 \cup L^1 \cup L^1 \cup L^2 \ldots$

$= \{\epsilon, a^2, a^3, a^4, a^5 \ldots\}$



$\therefore$ L^* is regular

Grammer

- Grammer → Language
- Language → grammer
- Simplification of grammer
- Type of language
  - CNF
  - GNF

$L = \{\omega_1, \omega_2, \omega_3 \dots \omega_n \dots\}$ ← string

may be finite/infinite

- A grammer is a set of rules used for deriving the string of a language

$V \cap T = \phi$

$G = (V, T, P, S)$

- V: finite set of terminals [can be empty]; but still if not still we can generate "ε"
- T: general small letter
- P: Set of production, $\alpha \to \beta$
  - It must contain one symbol from (V) or "α" must have atleast one non-terminal
- S: Start symbol, S∈V
- V: general capital letter otherwise it will be mentioned in question
- T: finite non empty Set of non-terminal/ Variable

Ex:

$G = (\underline{V}, \underline{T}, \underline{P}, \underline{S})$

$G = (\{S, A, B\}, \{a, b\}, P, S)$

$P = \{ S \to AB,\ Aa \to BB,\ BbS \to ab,\ ABa \to bSabA,$

$A \to aa,\ B \to \epsilon,\ ab \to SaA \}$

- $A \to aa$: these is production
- $ab \to SaA$: there should be atleast one non-terminal else it is not a production.

L(G) [The Language generate by the grammer G]:

L(G) is the set of terminal string that are derived from the start Symbol.

$L(G) = \{ \omega \in T^* / S \overset{*}{\underset{G}{\Rightarrow}} \omega \}$

* → 0 or more production Rule through which you can generate "ω"

i.e $G = (\{S, A, B\}, \{a, b\}, \{S \to aA, A \to aA/\epsilon, B \to bB/b\}, S)$

$S \to aA$
$\to a\epsilon$
$\to a$

$S \to aA$
$\to aaA$
$\to aa\epsilon$
$\to aa$

$S \to aA$
$\to aaA$
$\to aaaA$
$\to aaa\epsilon$
$\to aaa$ ..... .

$\therefore L(G) = \{a, a^2, a^3, a^4 \ldots\} \Rightarrow \{a^n / n > 0\}$

$B \to b$
$B \to bB$
$B \to bb$
$B \to bB$
$B \to bbB$
$B \to bbb$

✗ these not b/c the terminal b or bb or bbb b- - - are not starting from ~~starting~~ starting symbol.

1. If $G = (\{S\}, \{a\}, \{S \to aS/\epsilon\}, S)$ find $L(G)$

$S \to \epsilon$
$S \to aS$
$S \to a\epsilon$
$S \to aS$
$\to aaS$
$\to aa\epsilon$
$\to aa$

So, we can write
$[\epsilon, a, aa, aaa, \ldots]$
$\{a^n / n \geq 0\}$

2. if $G = (\{S\}, \{a\}, \{S \to asa/\epsilon\}, S)$ find $L(G)$

$S \to \epsilon$
$S \to asa$
$\to a\epsilon a$
$\to aa$
$S \to asa$
$\to aasaa$
$\to aaaaa$

So, $\{a^n a^n / n \geq 0\}$

3. If $G(\{S\}, \{a,b\}; S \to aSb/\epsilon, S) \to \{a^n b^n / n \geq 0\}$

4. If $G = (\{S\}, \{a,b\}, S \to aS/bS/a/b, S)$ find $L(G)$

$S \to a$
$S \to b$

$S \to aS$
$\to aa$ or $\to ab$

$S \to bS$
$\to ba$ or bb
⋮

$\therefore L(G) = \{w \in \{a,b\}^+\}$

5. If $G = (\{S\}, \{a\}, S \to SS, S)$ Find $L(G) \to \phi$

6. If $G = (\{S,A,B\}, \{a,b\}, S \to AB, A \to aA/a, B \to bB/b, S)$ Find $L(G)$

$S \to AB$
$S \to ab$

$S \to AB$
$\to aAB$
$\to aab$

$S \to AB$
$\to abB$
$\to abb$
⋮

$\{a^i b^j / i,j > 0\}$

7. If $G = (\{S,A\}, \{a,b,c\}, S \to abS/A, A \to cA/\epsilon, S)$ Find $L(G)$

$S \to abS/Sc/\epsilon$

$S \to A$
$\to \epsilon$

$S \to A$
$\to CA$
$\to C\epsilon$
$\to C$

$S \to abS$
$\to ab\epsilon$
$\to ab$
⋮

$\Rightarrow (ab)^* c^*$ or $\{(ab)^m c^n / m,n \geq 0\}$

$G = \{S \rightarrow abS/cS/\epsilon\}$

$S \rightarrow \epsilon$

$S \rightarrow cS$

$S \rightarrow c\epsilon$

$\rightarrow c$

$S \rightarrow abS$

$\rightarrow abS$

$\rightarrow ab\epsilon$

$\rightarrow ab$

$\vdots$

$(ab+c)^*$

- Construction of a grammer from Language -

Q] Construct a grammer which can generate all palindrome over $\{a,b\}$

$\{S \rightarrow \epsilon/a/b/asa/bsb\}$

i.e $S \rightarrow a\underline{S}a$

$\rightarrow aa\underline{S}aa$

$\rightarrow aabSbaa$

$\vdots$

$\therefore G = (\{S\}, \{a,b\}, \{S \rightarrow asa/bsb/a/b/\epsilon\}, S)$

$G = \{S \rightarrow asa/bsb/\epsilon\}$

$\downarrow$

even length palindrome

$G = \{S \rightarrow asa/bsb/a/b\}$

$\downarrow$

odd length palindromes

Q] construct the grammer for the following language

1. $L = \{wcw^T / w \in \{a,b\}^*\}$

$G = \{S \rightarrow aSa/bSb/c\}$

2. $L = \{a^n b^n / n > 0\} \rightarrow G = \{S \rightarrow aSb/ab\}$

3. $L = \{a^n b^{2n} / n \geq 0\} \rightarrow G = \{S \rightarrow aSbb/\epsilon\}$

4. $L = \{a^{2n} b^{3n} / n \geq 0\} \rightarrow G = \{S \rightarrow aaSbbb/\epsilon\}$

5. $L = \{a^m b^n / m, n \geq 0\} \rightarrow G = \{S \rightarrow aSb / \overset{aS}{a} / \overset{Sb}{b} / \epsilon\}$

6. $L = \{a^m b^n / m > n > 0\} \rightarrow G = \{S \rightarrow aab / asb / as\}$

7. $L = \{a^n b^{n+3} / n \geq 0\} \rightarrow G = \{$ $A \rightarrow aAb / bbb\}$

ϵ
abbbb
aabbbbb

8. $L = \{a^m b^n c^{m+n} / m, n \geq 0\} \rightarrow$ $\{S \rightarrow aSc / bAc / \epsilon, A \rightarrow bAc / \epsilon\}$

9. $L = \{a^m b^{m+n} c^n / m, n \geq 0\} \rightarrow G = \{S \rightarrow AB$
$A \rightarrow aAb / ab,$
$B \rightarrow bBc / bc\}$

aasbAbA

$\underbrace{a^m b^m}_{A} \; \underbrace{b^n c^n}_{B} / m, n > 0$

10. $L_{10} = \{w \in \{a,b\}^* / N_a(w) = N_b(w)\} \rightarrow \{S \rightarrow \epsilon / asb / bsa / ss\}$

$\rightarrow G'_{10} = \{S \rightarrow aSbS / bSaS / \epsilon\}$

$\rightarrow G''_{10} = \{S \rightarrow SaSb / SbSa / \epsilon\}$

11. $\{w \in \{a,b\}^* / N_a(w) = 2N_b(w)\}$

$G_{11} = \{S \rightarrow aSaSb / aSbSa / bSaSa / SS / \epsilon\}$

$\rightarrow G'_{11} = \{S \rightarrow aSaSbS / aSbSaS / bSaSaS / \epsilon\}$ same way different approach

$\rightarrow G''_{11} \rightarrow \{S \rightarrow SaSaSb / SaSbSa / SbSaSa / \epsilon\}$

12. $\{\underbrace{a^n b^n}_{A} \; \underbrace{c^r}_{B} / n \geq 0, r > 0\}$

$G_{12} = \{S \rightarrow AB,$
$A \rightarrow aAb / \epsilon$
$B \rightarrow cB / c\}$

(OR) $G_{12} = \{S \rightarrow AC / Sc$
$A \rightarrow aAb / \epsilon\}$

L_R = {All integer upto 498} → G = {S → A BB/ CDB/ 49D → 490-498.
0-399 400-489

A → 0/1/2/3

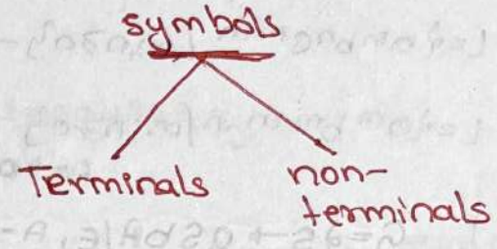
B → 0/1/2/3/4/5/6/7/8/9

D → 0/1/3/4/5/6/7/8}

- Simplification of the grammer:

A grammer may have

- useless symbols
- useless production
- Null (ϵ) production
- Unit production



Useful non terminal :-

'A' :- $L(G) = \{\omega_1, \omega_2, \omega_3 \ldots \omega_n \ldots\}$

- A non-terminal 'A' is said to be useful if it occur in atleast one derivation of a terminal string which is derived from the start symbol.

using "A" if you can produce atleast one string.

Useless nonterminal

useful terminal: A terminal 'a' is said to be useful if it is present in atleast one terminal string which is derived from the start symbol.

Useless production: A production is useless if it contain any useless non terminals.

A is useless

then S → aAb ✗

A → b ✗

ex: $G=(\{S, A, B\}, \{a, b\}, P, S)$

$P = \{S \to aA,\ A \to aA/\epsilon,\ B \to bB/b\}$

B is unreachable

$\therefore$ B is useless

[so b, BB are useless]

$B \to b$ $B \to bB$

Here b is also useless.

$S \to aA$	$S \to aA$
$\to a\epsilon$	$\to aaA$
$\to a$	$\to aa\epsilon$
	$\to aa$

So, $G' = (\{S, A\}, \{a\}, \{S \to aA, A \to aA/\epsilon\}, S)$.

ex: $G = (\{S, A, B\}, \{a, b\}, \{S \to AB/b, A \to aA, B \to bB/BS\}, S)$

$S \to AB/b, A \to aA$

$G' = (\{S, A, B\}, \{a, b\}, \{$, $B \to bB/BS\}$

B can not be terminated

$\Rightarrow S \to AB,\ B \to bB$ & $B \to BS$ are useless

$G' = (\{S, A\}, \{a, b\}, \{S \to b, A \to aA/a\}, S)$

$S \to b$

Here A is useless $\Rightarrow A \to aA/a$ are useless

& "a" is also useless

$\therefore G'' = (\{S\}, \{b\}, \{S \to b\}, S)$.

- Null (ϵ) production:

  Any production of the form $A \to \epsilon$ is called as null production

This null production can be eliminated after adding new production obtained by substituting "ϵ" wherever 'A' is on the RHS of the arrow

Ex: $G = \{S \to AaB,\ A \to aBa/b,\ B \to Ba/\epsilon\}$

$B \to \epsilon$ is null production

$S \to AaB$
$\to AaE$
$S \to Aa$

$A \to aBa$
$\to aEa$
$A \to aa$

$B \to Ba$
$\to Ea$
$B \to a$

$\Rightarrow G' = \{S \to AaB/Aa,\ A \to aBa/b/aa;\ B \to Ba/a\}$ //

Q] The no. of production in the grammer after eliminating ϵ-product for the full grammar

$G = \{S \to aAbB/a,\ A \to aA/\epsilon/Bb,\ B \to bBb/\epsilon\}$

$B \to b\ b/bBb$

$S \to aAbB$
$S \to a$
$S \to aAb$

$A \to aA$
$A \to \epsilon$
$A \to Bb$
$A \to b$

↓

$S \to aAbB$
$S \to a$
$S \to aAb$
$S \to a\ bB$
$S \to a\ b$

$A \to a$
$A \to Bb$
$A \to b$
$A \to aA$

$B \to bb$
$B \to bBb$

Q] Eliminate null production

$G = \{S \to as/\epsilon\}$ Here we can't eliminate $S \to \epsilon$ production as it is also an valid string if we eliminate then language will be change

$L(G) = \{a^n / n \geq 0\}$

$G' = \{S \to as/a\} \Rightarrow L(G') = \{a^n / n > 0\}$

~~$S \to as$~~
~~$\to a\epsilon$~~
~~$\to a$~~

- Unit production

▸ Any production of the form $A \rightarrow B$ where $A, B \in V$ is called unit production. this unit production can be eliminated after adding all B's production to A.

ex: $G = \{ S \rightarrow aSb \mid AB, A \rightarrow as \mid bB \mid a$
$B \rightarrow aBb \mid b \mid A \}$

$A \rightarrow a$ ✗
$A \rightarrow B$ ✓ (B should be non-terminal)

$\Rightarrow G = \{ S \rightarrow aSb \mid AB, A \rightarrow as \mid bB \mid a, B \rightarrow aBb \mid b \mid as \mid bB \mid a \}$

A production will be substituted to B. to eliminate the unit production (B → A)

- **Type of language:**

- **Type-0 production:** (RE language)

Every production is type-0 without any restriction.

Any production is of the form $\alpha \rightarrow \beta$; $\alpha, \beta \in (V \cup T)^*$

ex: $as \rightarrow b$ ✓
$SbaA \rightarrow \epsilon$ ✓
$A \rightarrow aba$ ✓
$Aa \rightarrow b$ ✓
$AB \rightarrow a$ ✓
$ab \rightarrow AaB$ ✗

- **Type-0 grammer:**

grammer is set of all production

▸ If every production of the grammer is type-0 then it is type-0 grammer (or) unrestricted grammer.

Type-0 language:

the language derived from type-0 grammer is called as type-0 language (or) Recursively Enumerable language (REL).

Type 1 production (CSL)

- Any production of the form $\alpha \to \beta$ such that $|\alpha| \leq |\beta|$ where where $\alpha, \beta \in (V \cup T)^*$

$aS \to AB$ (2 ≤ 2) ✓

$aS \to b$ (2 ≰ 1) ✗

Ex: $AaB \to baSb$ (3 < 4) ✓

$Sa \to b$ (2 ≰ 1) ✗

$A \to \epsilon$ (1 ≰ 0) ✗

$aAb \to ABA$ (3 = 3) ✓

$S \to \epsilon$ (1 ≰ 0) ✓ Still b/c iff S is a start symbol

$A \to abSa$ (1 < 4) ✓

- Type-1 grammer:-
  - If every production of the grammar is type-1 then its is type-1 grammer (or) context sensitive grammer (CSG) if $S \to \epsilon$ is present in the grammer then A is type-1 grammer iff S is the start symbol & "S" is no where on the RHS of any production

ex: $G_2 = \{S \to aAb, aS \to bB, AaB \to bSaA, A \to a\gamma\}$ is a CSG (1 ≤ 3, 2 ≤ 2, 3 ≤ 4, 1 ≤ 1)

$G_2 = \{S \to aBAb/\epsilon, aA \to bBb, B \to bA, B \to a\gamma\}$ is a CSG

$G_2 = \{S \to AaB/\epsilon, A \to aS/b, BB \to aab\gamma\}$ is not a CSG

Here $S \to \epsilon$ is present & 'S' is on RHS → not CSG

- Type-1 language

the language derived from type-1 grammer is called as type-1 language (or) context sensitive language

type-1 grammer (or) CSG (or) non-contracting grammer

(or) Length Increasing grammer (or) context-dependent grammers.

- **Type-2 production:** (CFL)
  - any production of the form $A \to \alpha$ where $A \in V, \alpha \in (V \cup T)^*$ is called as type-2 production

(A should be single.)

ex: $S \to aAbS$, $S \to abb$, $A \to BAS$, $B \to \epsilon$, $B \to BSab$

2 A's: $AA \to abSb$ ✗

$a \to asb$ ✗ it is not at all production

- **Type-2 grammer:**
  - If every production of the grammar is $A \to \alpha$ type-2 then it is type-2 grammer (or) context free grammer (CFG)

Type-2 language:

- The language derived from type-2 grammar is called as type-2 language or context free language (CFL)

Linear grammer (is a subset of CFG)

↳ it is in the form of $A \to \alpha$ but (α) should be almost one non-terminal

- If every production of the grammar is of the form $A \to \alpha$ where $A \in V$; $\alpha \in (V \cup T)^*$ & α must have almost one non-terminal

ex: $S \to aSaA$ ✗

$S \to aAa$ ✓

$S \to aa$ ✓

$A \to a$ ✓

$A \to \epsilon$ ✓

$A \to Bab$ ✓

$A \to BaBb$ ✗

- every Linear grammer is a CFG but not every CFG is not Linear grammar

- Linear Language
  - the language derived from linear grammer is called as linear language

Left linear grammer:

- If every production of the grammer is of the form

$A \rightarrow Bw$ or $A \rightarrow w$ where $A,B \in V$; $w \in T^*$

$G = \{S \rightarrow asb / aA, A \rightarrow b\}$ in linear grammar but not left linear grammer

$G = \{S \rightarrow Aa / sba, A \rightarrow a/\epsilon\}$ in left linear grammar.

Right linear grammer:

- If every production of the grammer is of the form

$A \rightarrow wB$ or $A \rightarrow w$; where $A, B \in V$; $w \in T^*$

$G = \{S \rightarrow asb / Ab, Aa\}$ is linear grammar but not right linear grammar.

$G = \{S \rightarrow abA / ab, A \rightarrow bS / a\}$ is right linear grammar

$G = \{S \rightarrow Aab / bb, A \rightarrow aa.S / \epsilon\}$ is just linear grammar

• Type-3 production:- (Regular)

▸ Any production of the form. $A \to wB$. or $A \to Bw$. or $A \to w$

where $A, B \in V$; $w \in T^*$.

Ex: $S \to aS \mid bbA$, $A \to Ba \mid b$, $B \to aAy$ is not regular grammer but linear grammer.

(aS: Right linear; Ba: left linear; aAy: can be of any no.)

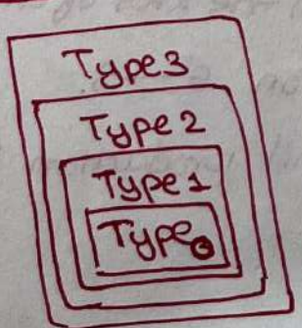
• Type-3. grammar:

A. grammar is type-3 or Regular grammar if it is either left linear only or Right linear only but not both.

Language
↓
grammar
↓
Production

Type-0	Type-1	Type-2	Type-3
no restriction	$\alpha \to \beta$	$A \to \alpha$	$A \to wB/w$ (or) $A \to Bw/w$
$\alpha \to \beta$	$\lvert\alpha\rvert \leq \lvert\beta\rvert$	$A \in V$, $\alpha \in (V \cup T^*)$	
unrestricted grammar	CSG	CFG	Regular grammar
REL	CSL	CFL	Regular Language

note.



Q the following grammer is

T-2 T-1 T-1 T-2 ✗ T-2 (Type-3) T-3

$G = \{S \to aAS, aA \to bb, AbS \to BaB, B \to aBb, A \to abS, B \to \epsilon\}$

$A \to \alpha$ $2 \leq 2$ $3 \leq 3$ $A \to \alpha$ $A \to wB$ $A \to w$ $|| \neq 0$

(a) Type-0 (b) Type-1 (c) Type-2 (d) Type-3

due to $AbS \to BaB$ & $B \to \epsilon$

$AbS \to aS$

$3 \not\leq 1$

$G = \{S \to baB / Ab, A \to as / \epsilon, B \to b\}$ is CFG

note

$B \to \epsilon$ is Type-3 but not Type-1

Normal form

- Chomsky normal form (CNF)
- Greibach normal form (GNF)

} the are well structured grammar

1 Chomsky normal form (CNF):

- If every production of the CFG is of the form $A \to BC$ or $A \to a$. where $A, B, C \in V$; $a \in T$ is said to be in CNF

(should be single A; only 2 non-terminal should be there; only on terminal)

Here $S \to \epsilon$ is allowed iff "S" is a start symbol & S is no where on the R.H.S of any production.

Ex: $G = \{S \to AS / BB / a, A \to BA / b, B \to a\}$ is in CNF

$G = \{S \to \underset{\times}{SAS} / \underset{\times}{aS} / \underset{\times}{Sb} / \underset{\times}{ab}, A \to \underset{\times}{bAb} / \underset{\times}{B}, B \to a / \underset{\times}{\epsilon}\}$ is not CNF

Note

- Every CFG can be converted into CNF.
- if $S \to \epsilon$ in the grammar & if S is in the RHS of any production then add the augmented production $S' \to S$.
- then eliminate null production & unit production if any
- then convert it into $A \to BC$ or $A \to a$

Ex: $G = \{S \to \underbrace{aA}_{C}\underbrace{bB}_{D} / bb,\ A \to bBas/ab,\ B \to bbB/a\}$ is not in CNF

$G' = \{S \to CD/FF,\ A \to DG/EF,\ B \to FD/a,$
$C \to aA,\ G \to ES$
$E \to a$
$F \to b\}$ is in CNF.

ex: $G = \{S \to as/\epsilon\}$ is not in CNF

we can't eliminate ϵ as it is coming in S (start symbol) so it's a valid string.

So, we add augmented production.

$S' \to S$ such that S' as new start symbol.

$G' = \{S' \to S,\ S \to as/\epsilon\}$

$S' \to S$ is a unit production

$G'' = \{S' \to as/\epsilon,\ S \to as/\epsilon\}$

$S' \to \epsilon$ is valid

but $S \to \epsilon$ is not allowed

$S' \to aS$ $\quad S \to aS$
$\to aE$ $\quad \to aE$
$S' \to a$ $\quad S \to a$

after eliminating $S \to \epsilon$ production.

- $G'' = \{S' \to as/\epsilon/a,$
  $S \to as/a\}$ is not in CNF

$G''' = \{S' \to AS/a/\epsilon,\ S \to AS/a$

$A \to a\}$
is in CNF

Properties of CNF

- If a grammar is in CNF then to derive any string of length 'n' it takes exactly $2n-1$ steps

- Every derivation tree in CNF is a binary tree

- If the grammar is in CNF and if the length of the string ω is in then the minimum height (h_{min}) of the derivation tree is

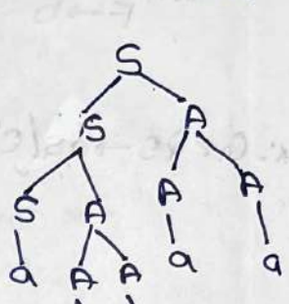
$$h_{min} \geq (\log_2 n + 1)$$

& maximum height (h_{max}) is 'n'

$G = \{S \to SA/a, A \to AA/b\}$ is in CNF

S
SA
SAA
aAA
$aAAA$
$abAA$
$abbA$
$abbb$

$2*4-1 = 7$



$|\omega| = n$

$\omega = abbb$

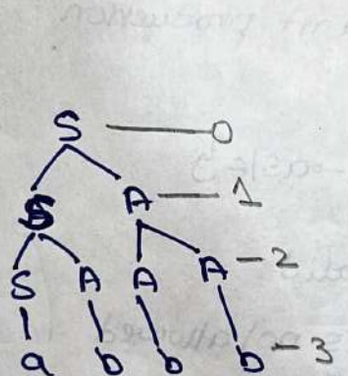
$|\omega| = 4$

$h_{min} \geq (\log_2 n + 1)$

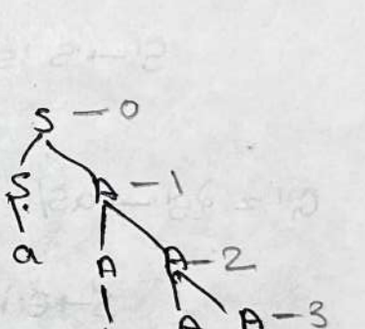
$\geq \log_2 4 + 1$

$\geq 2 + 1$

≥ 3



balanced BST



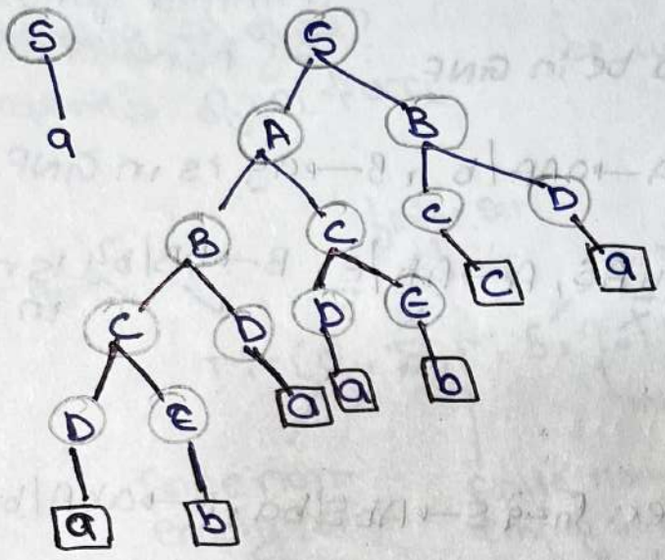
unbalance BST

∴ for min. the the derivation tree must be balance & for max. the derivation tree must be unbalance.

- If the rank of every non-terminal in CNF is finite the language derived from the grammer is also finite
- If the rank of any non-terminal in CNF is infinite & if that non-terminal is useful one then the language derived from the grammer is infinite.

- The rank of a non-terminal is the longest path in the derivation tree from that non-terminal.

Ex: $G = \{S \to AB/a, A \to BC/a, B \to CD, C \to DE/c, D \to a, E \to b\}$



Rank(S) = 5
Rank(A) = 3
Rank(B) = 3
Rank(C) = 2
Rank(D) = 1
Rank(E) = 1

Here L(G) is finite.

- A language is infinite if it's Rank is infinite & you can eliminate it as it is a usefull symbol.

- If $G = (V, T, P, S)$ is a CFG with no unit production & no null production & the length of the longest production on RHS is $|K'|$ then if the grammar converted into CNF $G' = (V', T, P', S)$ then the max no. of production in P' is

$$|P'| \leq [|P| * (K-1) + |T|]$$

ex: $K = 7$ & $|V| = 5$, $|T| = 3$, $|P| = 20$

then,

$$\leq [20 * [7-1] + 3]$$

$$\leq 123$$

2. Greibach Normal Form (GNF):

- If every production of the form $A \rightarrow a\alpha$ or $A \rightarrow a$

  where $A \in V$,

  $a \in T$

  $\alpha \in (V)^*$ is said to be in GNF

  (1st symbol should be terminal followed by non terminal's)

ex: $G = \{S \rightarrow aSA / bABS / a,\ A \rightarrow aAA / b,\ B \rightarrow a\}$ is in GNF

$G = \{S \rightarrow \underset{\times}{A}BS / a\underset{\times}{S}\underline{b} / \underline{bb}\underset{\times}{A}S,\ A \rightarrow \underset{\times}{A}\underline{b} / \underset{\times}{\epsilon},\ B \rightarrow bB / b\}$ is not in GNF

Note

- every CFG can be convert into GNF for that initial we need to eliminate null & unit production if any

---

- If G is in GNF then to derive any string of length "n" it takes exactly "n" Step

ex: $G = \{S \rightarrow AbS / ba,\ A \rightarrow abA / ba\}$ is not in GNF but a CFG

$G' = \{S \rightarrow ABS / bC,\ A \rightarrow aBA / bC,\ B \rightarrow b,\ C \rightarrow a\}$ is not in GNF

$G'' = \{S \rightarrow aBABS / bCBS / bC,\ A \rightarrow aBA / bC,\ B \rightarrow b,\ C \rightarrow a\}$ is in GNF

ex: $\underline{S}$

$bC\underline{B}S$

$baB\underline{S}$

$babS$

$babbC$

$babbba$

Push down Automata (PDA) → have storage space

- PDA, NPDA
- Design of PDA's
- Checking CFL or not?
- Properties of CFL & DCFL

▸ PDA is used to accept CFL, in PDA a stack / pushdown store (PDS) will be used as a memory. which can be used to push or pop the symbols.

7-type

finite set of i/p

Transition funct^n, $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*$

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

- Finite non-empty set of states
- Finite non-empty set of stack / pushdown symbols.
- initial state, $q_0 \in Q$
- Initial stack symbol, $z_0 \in \Gamma$
- set of final states, $F \subseteq Q$

▸ the element of transition funct^n are of the form

$\delta: Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$

$\delta(q, a, z) = (q', \alpha)$

∴ Scan only one i/p at a time

▸ after completion of a transition it may or may not change state

$\delta(q, a, z) = (q', \alpha)$ where $q, q' \in Q$, $a \in \Sigma$, $z \in \Gamma$, $\alpha \in \Gamma^*$

- In PDA it is not necessary to define for every move
- will impact the the stack symbol

▸ In every transition the PDA take the present state, present i/p symbol & the top most symbol from the stack. after completion of each transition the PDA may or may not change the state & the top most symbol in the stack may be deleted or kept as it is or one or more symbol can be added on top of the stack by deleting or keep the top symbol in the stack.

- But the symbols which are below the top most symbols will not be distrubed

- Some time a PDA can make a transition without taking a I/p symbol. Such move are called "Null" moves.

  [i.e $\delta(q, \epsilon, z)$]

$\delta(q_1, \epsilon, q) = q_2, \epsilon)$ when the machine is in q_1 & top symbol is a then make a "ϵ" move
↓
don't take a I/p symbol but still make a transition.

- Instantaneous description (ID): ↳ tell present state of a machine

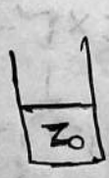
- An ID is $\delta(q, w, \alpha)$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$

  Ex: $\delta(q_1, bbaaba, z_2 z_1 z_0)$

- Initial ID
  - If $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is a PDA

    then the Initial ID is

    $\delta(q_0, w, z_0)$ where $w \in \Sigma^*$ (initial stack symbol)



- Acceptance by a PDA.
  - a PDA can accept a string in two ways
    - By final state
    - By empty stack

If $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is a PDA, then the set of all strings accepts by PDA by final state are defined as

$L = \{w \in \Sigma^* / \delta(q_0, w, z_0) \vdash^* (q_f, \epsilon, \alpha)$ where $q_f \in F; \alpha \in \Gamma^*\}$

- String
- Initial stack symbol
- one or more transition
- Should be at final state
- stack can be anything
- String should be empty

- **By Empty Stack.**

If $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \phi)$ is a PDA, then set of all string accepted by PDA by empty stack are defined as

$$L = \{w \in \Sigma^* / \delta(q_0, w, Z_0) \vdash^* (q', \epsilon, \epsilon) \text{ where } q' \in Q\}$$

- q' → can be at any state
- ϵ → string should be empty
- ϵ → stack should be empty

Deterministic PDA (DPDA) :- A PDA is said to be deterministic if it satisfies the following 2 condition

(Should have atmost 1 transition)

(i) Every $\delta(q, a, Z)$ is either empty or has a single element.

(ii) If $\delta(q, \epsilon, Z) \neq \phi \implies \delta(q, a, Z) \neq \phi, \forall a \in \Sigma$

(For these combination of state & stack symbol if you did ϵ move then → then for that combination of state & stack symbol you should not make any transition.)

ex:

$\delta(q_0, aabb, Z Z_0)$ $\quad \delta(q_0, a, Z) = (q_1, Z)$

$\vdash \delta(q_1, abb, Z Z_0)$ $\quad \delta(q_1, \epsilon, Z) = (q_2, \epsilon)$

$\vdash \delta(q_2, abb, Z_0)$ $\quad \delta(q_1, \epsilon, Z_1) = \phi$

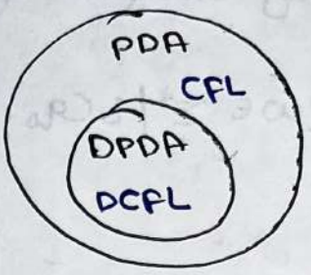
Non-Deterministic PDA (NPDA) : If atleast one move have more than one elements. then it is an NPDA

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longrightarrow 2^{Q \times \Gamma^*} \text{ (i.e finite subset of } Q \times \Gamma^*)$$

ex: $\delta(q_0, a, Z) = \{(q_1, \epsilon), (q_0, aZ_0) \ldots\}$

Note

- DPDA accepts DCFL
- NPDA accept CFL
- By Default PDA means NPDA
- PDA accept CFL
- Every DPDA is also NPDA but every NPDA is not a DPDA
- NPDA can't be converted to DPDA [construct]



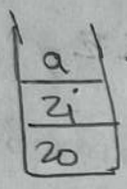
Design a PDA for the following language $L=\{a^n b^n / n>0\}$

$L=\{a^n b^n / n>0\}$

PDA doesn't count it will just store [it have memory]

str = aabb

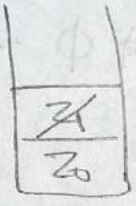
ex $\delta:(q_0,a,z_1)=(q_0,az_1)$



str = aabb

means pushing a over z_1.

ex.



$\delta:(q_0,a,z_1)=(q_0,\epsilon)$

remove the top stack symbol.

$\delta(q_0,a,z_0)=(q_0,az_0)$

$\delta(q_0,a,a)=(q_0,aa)$ — push a over previous element

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

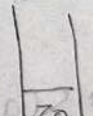
$\delta(q_1,b,a)=(q_1,\epsilon)$ — eliminate the top symbols.

$\delta(q_1,\epsilon,z_0)=(q_1,\epsilon)$

---

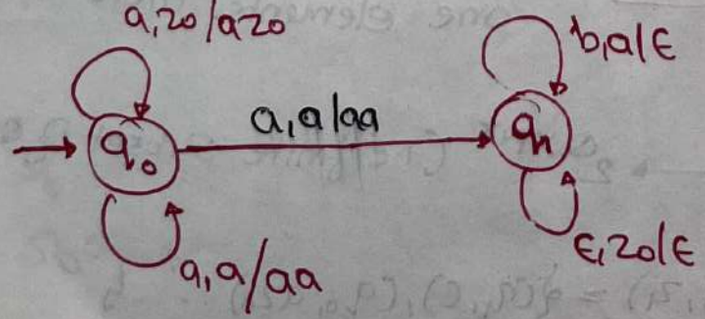
ex: $\delta:(q_0,a,z_1)=(q_0,z_1)$

// remain at same position

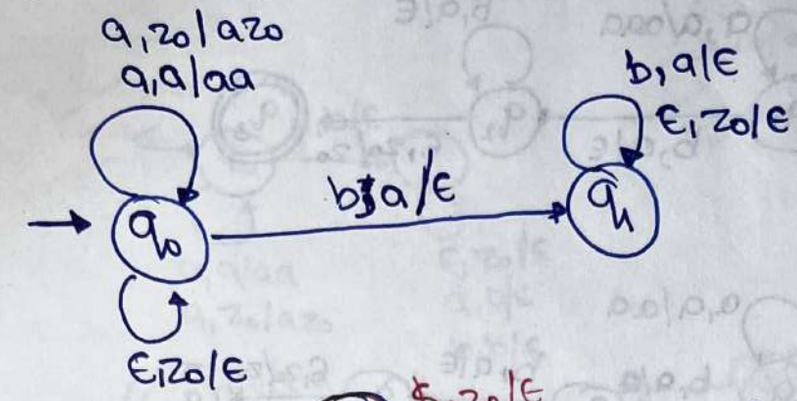


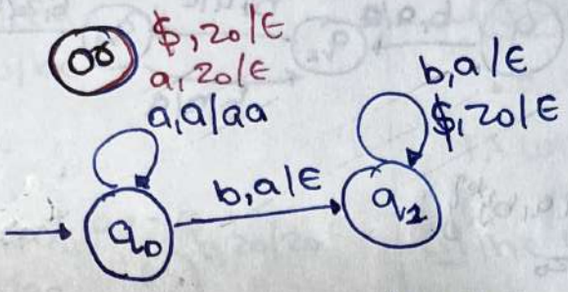
$M=(\{q_1,q_0\},\{a,b\},\{a,z_0\},\delta,q_0,z_0,\phi)$ is a DPDA

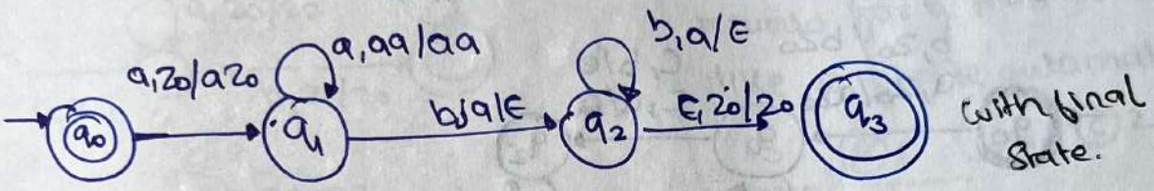
So language is DCFL



ex. Design a PDA for ⇒ $L = \{a^n b^n / n \geq 0\}$.



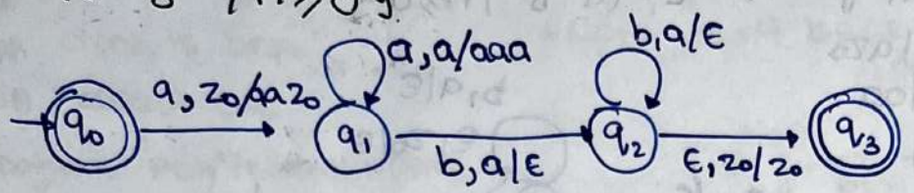


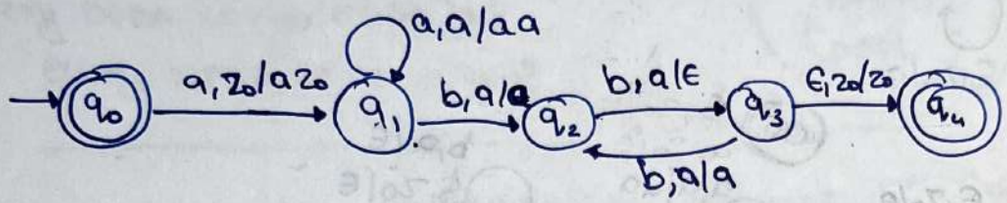


note

- DPDA by empty stack is not possible if it contain 'ε'
- DPDA by final state is powerfull then DPDA by empty stack.
- NPDA is more powerfull than DPDA by final state
- DPDA by empty stack is equal powerfull with DPDA by final state if the concept of end marker is used
- whenever a language have a prefix properties [having 'ε' as a string] for that language DPDA by empty stack is not possible but DPDA by final state may be possible
- NPDA by final state is equal powerful with NPDA by empty stack.

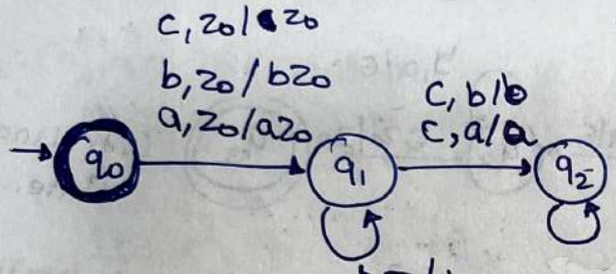
a) $L = \{a^n b^{2n} / n \geq 0\}$





b) $L = \{\omega c \omega^T / \omega \in \{a,b\}^*\}$

abcba

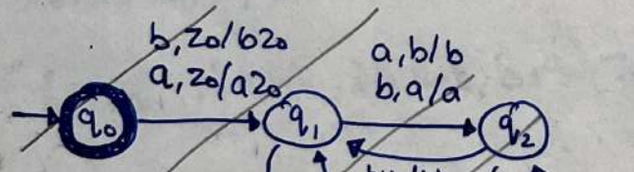


b,a/ba
b,b/bb
a,b/ab
a,a/aa

b,b/ε

a,a/ε
ε,z0/ε

c) $L = \{\omega \in \{a,b\}^* / N_a(\omega) = N_b(\omega)\}$

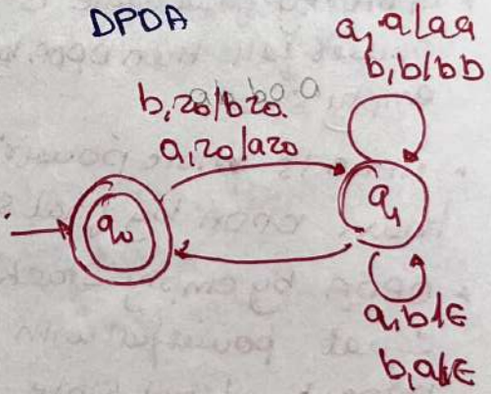


b,b/bb
a,a/aa

a,b/ε
b,a/ε

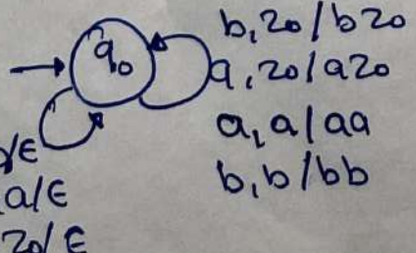
b,b/bb
a,a/aa

DPDA

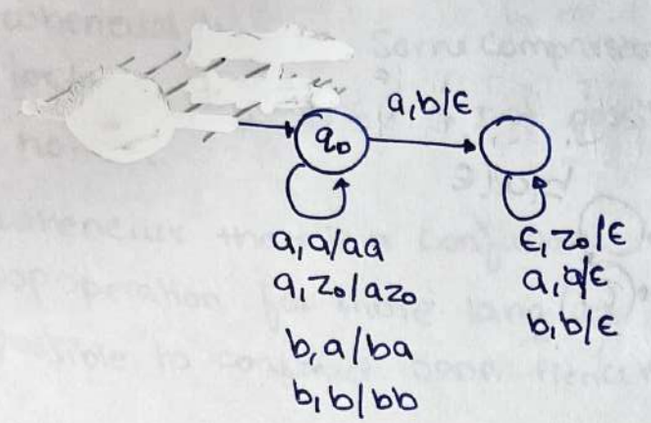


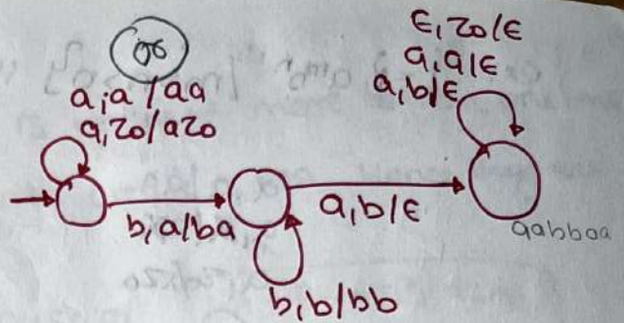
IS DCFL

NPDA :-



ex: $L = \{a^m b^n a^{m+n} / m,n>0\}$ is DCFL.



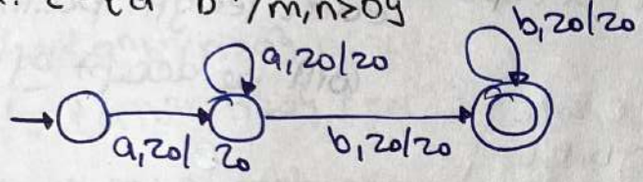


Note

- A PDA can behave like FA without using the stack

Q) the no. of Auxillary memory required for a PDA to behave like a finite automata is (a) 0 (b) 1 (c) 2 (d) 3

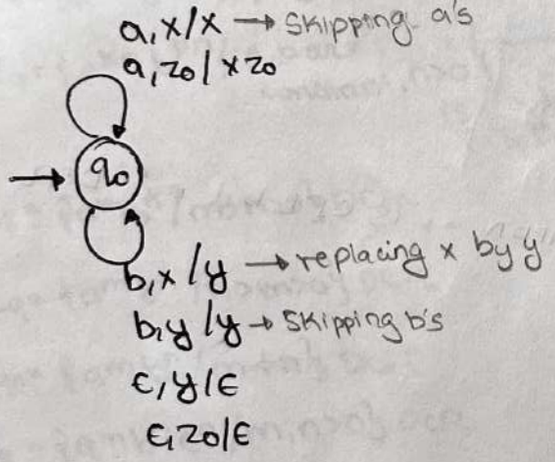
ex: $L = \{a^m b^n / m,n>0\}$

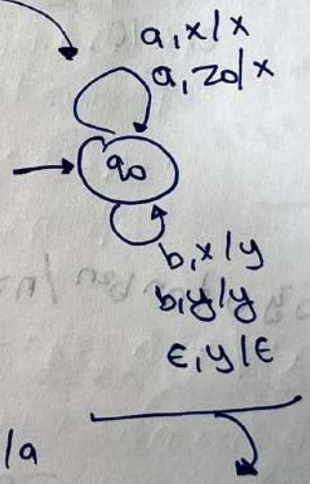


every Regular languag

NPDA By empy Stack

Q)

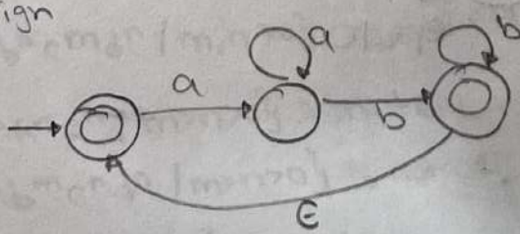




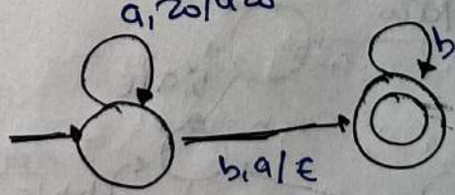
a^+b^+

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

Different Design

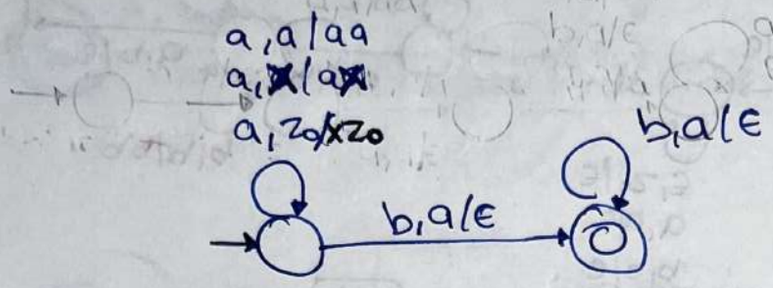


ex:



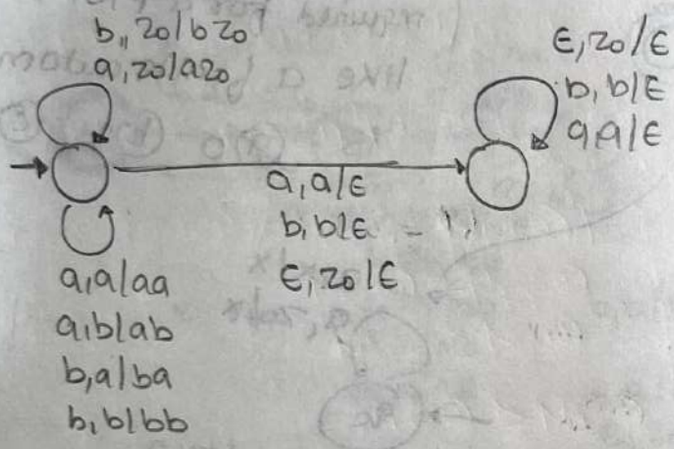
DPDA by final State

ex: $L = \{ a^m b^n / m > n > 0 \}$ IS DCFL



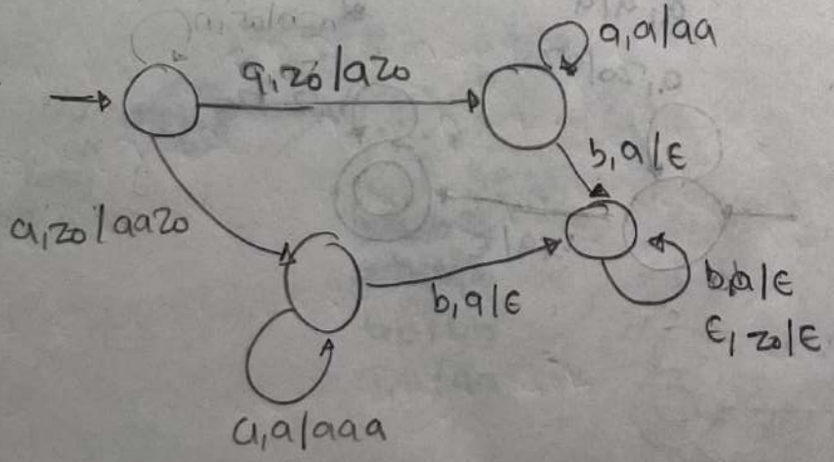
ex: $L = \{ ww^T / w \in \{a,b\}^* \}$ CFL not DCFL

even length palindrome will be accept by NPDA



② $L = \{ a^n b^n / n > 0 \} \cup \{ a^n b^{2n} / n > 0 \}$ NPDA → IS CFL but not DCFL

ab, aabb, aaabbb ... abb, aabbbb



Q) $L = \{a^n b^n c^n / n>0\}$

- whenever there is same comparsion is required more than one time for those language's it is not possible to create a PDA. Hence they are not CFL.
- whenever there is a confusion in push or pop operation for thore language it is not possible to construct DPDA. Hence they are not DCFL

FA = FA + 0 Stack
PDA = FA + 1 Stack
TM = FA + 2 Stack.

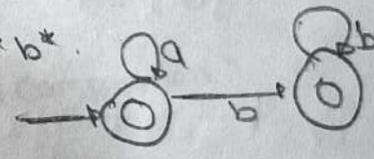
Q) check whether the following language are DCFL or CFL or not?

$L_1 = \{a^n b^n / n \geq 0\}$ DCFL
$L_2 = \{a^n b^n / n>0\}$ DCFL
$L_3 = \{a^n b^{2n} / n \geq 0\}$ DCFL
$L_4 = \{a^{3n} b^n / n>0\}$ DCFL
$L_5 = \{a^n b^n / n \geq 5\}$ DCFL
$L_6 = \{a^{2n} b^{3n} / n \geq 0\}$ DCFL
$L_7 = \{a^{kn} b^{pn} / k,p$ are constant, $n>0\}$ is regular (it is DCFL)

$L_7 = \{a^{kn} b^{pn} / k,p,n \geq 0\}$ is regular

$n=1 \Rightarrow \{a^k b^p / k,p \geq 0\}$

$a^* b^*$



$L_8 = \{a^m b^n / m>n>0\}$ DCFL
$L_9 = \{a^m b^n / n>m>0\}$ DCFL
$L_{10} = \{a^m b^n / m \neq n\}$ DCFL
$L_{11} = \{a^m b^n c^n / m,n>0\}$ DCFL
$L_{12} = \{a^n b^m c^n / m,n>0\}$ DCFL
$L_{13} = \{a^m b^m c^n d^n / m,n>0\}$ DCFL
$L_{14} = \{a^m b^n c^n d^m / m,n>0\}$ DCFL
$L_{15} = \{a^m b^n c^m d^n / m,n>0\}$ is not CFL but CSL
$L_{16} = \{a^m b^n c^n / m>n>0\}$ is not CFL but CSL
$L_{17} = \{a^m b^m c^n d^n / m>n>0\}$ is not CFL but CSL
$L_{18} = \{a^{m+n} b^m c^n / m,n \geq 0\}$ DCFL

19. $\{a^m b^{m+n} c^n / m,n \geq 0\}$ DCFL

20. $\{a^m b^n c^{m+n} / m,n \geq 0\}$ DCFL

21. $\{a^m b^m c^n d^n e^m f^m / m,n > 0\}$ is not CFL but CSL

22. $\{a^l b^l c^m d^m a^n b^n / l,m,n > 0\}$ DCFL
→ using END marker.

23. $\{a^i b^j c^k /$ $i=j$ or $i=k\}$
$i=j$ or $j=k$
$i=k$ or $j=k$
} are CFL but not DCFL

24. $\{a^n b^n / n>0\} \cup \{a^n b^{2n} / n>0\}$ CFL but not DCFL

25. $\{a^m b^n / n \leq m \leq 2n\}$ is CFL but not DCFL

26. $\{w \in \{a,b\}^* / N_a(w) = N_b(w)\}$
$N_a(w) \neq N_b(w)$
$N_a(w) > N_b(w)$
$N_a(w) \geq N_b(w)$
$N_a(w) \leq N_b(w)$
} all are DCFL

27. $\{a^{m-n} b^m c^n / m,n > 0\}$ DCFL → $a^m b^{m+n} c^n$ (1 m, 3 n, 2)

28. $\{a^m b^{m-n} c^n / m,n > 0\}$ DCFL → $a^{m+n} b^m c^n$

29. $\{a^m b^n c^{m-n} / m,n > 0\}$ DCFL

30. $\{a^m b^n c^{m+n} / m,n > 0\}$ not CFL but CSL → $a^{m+n} b^m c^n$

31. $\{a^m b^n c^{m+n} / m,n > 0\}$ not CFL but CSL

32. $\{wcw^T / w \in \{a,b\}^*\}$ DCFL

33. $\{ww^T / w \in \{a,b\}^*\}$ CFL but not DCFL

34. $\{wcw / w \in \{a,b\}^*\}$ not CFL but CSL

35. $\{ww / w \in \{a,b\}^*\}$ not CFL but CSL

36. $\{a^n b^n c^n / n \geq 0\}$ not CFL

37. $\{a^m b^n c^k / m > n > k > 0\}$ not CFL

38. $\{a^m b^n c^k / m > n > k > 0, k \leq 8\}$ CFL

$L_{39} = \{a^m b^n c^k / m > n > k > 0; n \leq 10\}$ DCFL
(it is regular)

$L_{40} = \{a^{i^2} / i \geq 0\}$ not CFL but CSL

$L_{41} = \{a^p / p$ is a prime no.$\}$ not CFL but CSL

$L_{42} = \{a^{n!} / n \geq 0\}$ not CFL but CSL

$L_{43} = \{a^n /$ is a perfect no.$\}$ not CFL but CSL

$L_{44} = \{w c^m w^T / w \in \{a,b\}^*, m > 0\}$ DCFL

$L_{45} = \{w c^m w^T / w \in \{a,b\}^*, m \geq 0\}$ CFL [b/c of $C=0$]

$L_{46} = \{w a^m w^T / w \in \{a,b\}^*, m > 0\}$ is CFL but not DCFL

$L_{47} = \{w c^m w^T / w \in \{a,b,c\}^*, m > 0\}$ is CFL but not DCFL

$L_{48} = \{w x x^T w^T / w, x \in \{a,b\}^*\}$ CFL

$L_{49} = \{w w^T x x^T / w, x \in \{a,b\}^*\}$ CFL

$L_{50} = \{w x w^T x^T / w, x \in \{a,b\}^*\}$ not CFL (Alternative comparison)

$L_{51} = \{w . x x w^T / w, x \in \{a,b\}^*\}$ not CFL

$L_{52} = \{w \in \{a,b\}^* / N_a(w) = 2N_b(w)\}$ DCFL

$L_{53} = \{w x w / w, x \in \{a,b\}^*\}$ is regular & also DCFL

$L_{54} = \{x w w^R . / x, w \in \{a,b\}^*\}$ CFL

$L_{55} = \{w x w / w \in \{a,b\}^*$ & $x \in \{a,b\}^*\}$ not CFL, it's CSL

$L_{56} = \{a^n b^{2n} c^n / n > 0\}$ CFL

$L_{57} = \{a^n b^{2n} c^n / n > 0\}$ CFL

$L_{58} = \{a^i b^j c^k d^l . / i = k$ or $j = l$ & $i,j,k,l > 0\}$ is not CFL and DCFL

$L_{59} = \{a^{2^y} / y \geq 2\}$

$L_{60} = \{a^{x^3} / x \geq 1\}$

$L_{61} = \{a^p$ & $a^3\}$

} not CFL but CSL

- Properties of DCFLs & CFLs

- DCFL's are closed under:
  - Complement
  - Inverse Homomorphism
  - Prefix & suffix
  - Quotient
  - min
  - max
  - union with regular
  - Intersection with regular
  - set difference with regular

  } All language are closed under these operation

- DCFL's are not closed under:
  - Union
  - Intersection
  - Concatenation
  - Kleen's closure
  - Reversal
  - Homomorphism
  - Set difference
  - Subset
  - Superset
  - Infinite Union
  - Infinite Intersectn
  - Infinite set difference

  } No language are closed under these operatn

ex: $L = \{a^n b^n \mid n>0\}$ is DCFL

$L_2 = \{a^n b^{2n} \mid n>0\}$ is DCFL.

$L \cup L_2 = \{a^n b^n\} \cup \{a^n b^{2n}\}$ is CFL

ex: $L = \{a^m b^n c^n \mid m,n>0\}$
$L_2 = \{a^n b^n c^m \mid m,n>0\}$
} are DCFLs

$L \cap L_2 = \{a^i b^i c^i \mid i>0\}$ is not CFL

ex: $L = \{a^n b^n \mid n>0\}$ is DCFL

$L_2 = (a+b)^*$ is regular also DCFL

$L_2 \cdot L = (a+b)^* \cdot a^n b^n$ is CFL but not CFL

ex $aabba^3b^3$

?

for these q there is confusion

ex: Kleen's closure

→ $L=\{a^n b^n a^m / n,m>0\}$ is DCFL

wrong example $L^2=\{a^n b^n a^m a^i b^j a^j / i,j,n,m>0\}$

So L^* is not DCFL

In (union & intersection) if any one is not closed then Set difference is not closed

ex. $a^m b^n (a+b)^* / n>0$ is DCFL

$(R=(a+b)^* b^n a^n$ is CFL

but not DCFL

ex: Let L_1 is DCFL & L_2 is regular

$L_1 - L_2 =$ DCFL ∩ Regular

$L_2 - L_1 =$ DCFL

ex: $L_1 - L_2 = L_1 \cap \overline{L_2}$ (set difference)

⇒ DCFL ∩ $\overline{\text{DCFL}}$

⇒ DCFL ∩ DCFL

⇒ not DCFL

- CFLs are closed under:
  - union
  - concatenation
  - Reversal
  - Kleen's closure
  - Quotient
  - Prefix & suffix
  - Homomorphism
  - Inverse Homomorphism
  - cycle
  - Union with regular
  - Intersection " "
  - set difference

- CFL are not closed under:-
  - Intersection
  - Complement →
  - set difference
  - Subset
  - super set
  - Infinite union
  - Infinite intersection
  - Infinite set difference
  - Min
  - Max
  - Half
  - ALT.

i.e complement of CFL may or may not be CFL

ex:- let L_1 is DCFL L_2 is regular {union}

→ $L_1 \cup L_2$ is DCFL

every regular is DCFL but not every DCFL is not regular

• let $L_2 = \{a^n b^n \mid n>0\}$ is DCFL &

$L_2 = (a+b)^*$ is regular

$L_1 \cup L_2 = \{a^n b^n\} \cup (a+b)^*$

$= (a+b)^*$ is regular.

ex: L_1 is CFL L_2 is regular

then,

$L_1 - L_2 = L_1 \cap \overline{L_2}$

$= CFL \cap \overline{Regular}$

$= CFL \cap Regular$

$= CFL$

$L_2 - L_1 = L_2 \cap \overline{L_1}$

$= Regular \cap \overline{CFL}$

$= Regular \cap \overline{CFL}$ ↑

$= Regular \cap \overline{CSL}$

$= Regular \cap CSL$

$= CSL$

Turing Machine

- 7-type

finit set of I/P

T.F $\delta: Q \times T \rightarrow Q \times T \times \{L, R\}$

$M = (Q, \Sigma, \Gamma, \delta, q_0, \not b, F)$

Special symbol, blank, $\not b \in T$, $\not b \notin \Sigma$ $\not b \in \Gamma - \Sigma$

finite non-empty Set of States

finite non-empty set of tape symbol

Initial State, $q_0 \in Q$

Set of final states $F \subseteq Q$

$\Sigma \subset \Gamma$

$\Sigma \subset \Gamma$

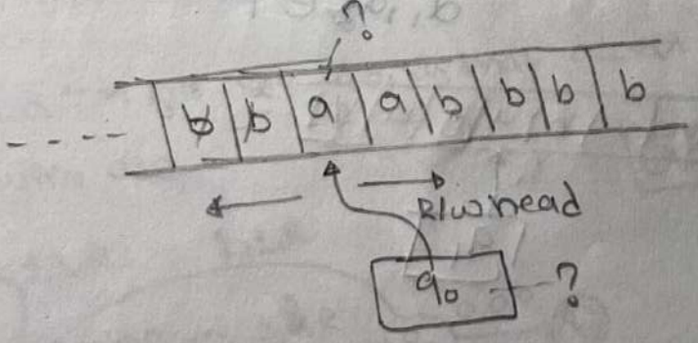
whatever is there in "Σ" it will also in "Γ" & "Γ" also contain more symbol

Turing Machine

- Concept & design
- Variation of TM
- Universal TM
- Church's turing thesis
- Recursive & Rel's
- Properties of Rec & Rel's

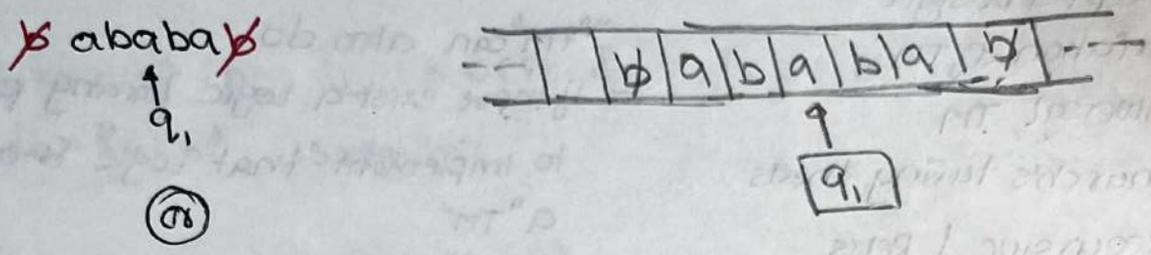
- Alan turning in 1936
  - what ever a computer can do a "TM" can also do
  - if there exist a logic for any problem to implement that logic there exist a "TM"
  - "TM" is used to accept Recursively enumerable language(REL)

- "TM" has a tape of infinite length which is divide into no. of cells & each cell can store only one symbol at a time. the tape is connected to finite control through a Read/write head which can move either one cell left or one cell right only
- in every transition the "TM" take the present state & present symbol in the cell from the tap under Read/write head to give the following
  1. the next state of the machine
  2. to which symbol the present symbol has to replaced
  3. the direction of the Read/write head
  4. wheather to stop or not



$\delta(q_0, a) = (q', x, L/R)$.

- Instantaneous Description(ID)



⊔ abq1aba ⊔

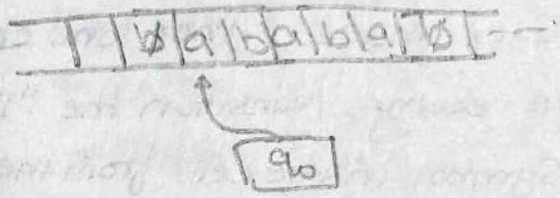
- Initial ID :-

If $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ is a TM

then the initial ID is $q_0 w$ where $w \in \Sigma^*$

⊔ q_0 ababa ⊔

⊔ $q_0 w$ ⊔

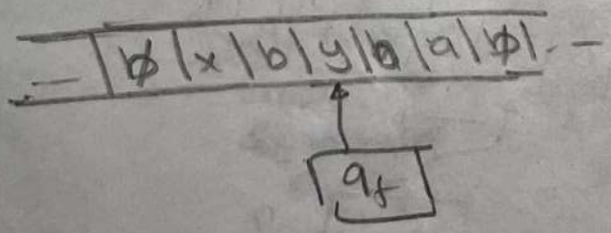


- Acceptance by TM :-

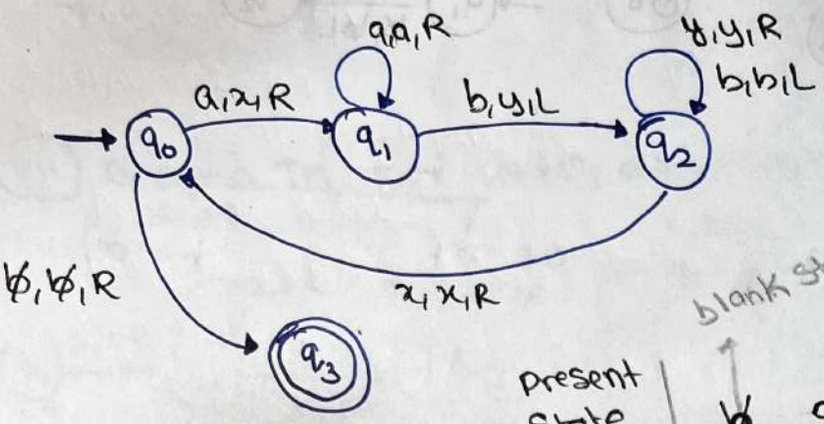
- A string ("w") $\in \Sigma^*$ is said to be accept by "TM" if it start with initial ID & terminate at some final state

i.e ⊔ $q_0 w$ ⊔ $\vdash^*$ $\alpha_1 q_f \alpha_2$ where $q_f \in F$;

$\alpha_1, \alpha_2 \in \Gamma^*$



- Representation of TM By transition diagram.
  - By transition table.

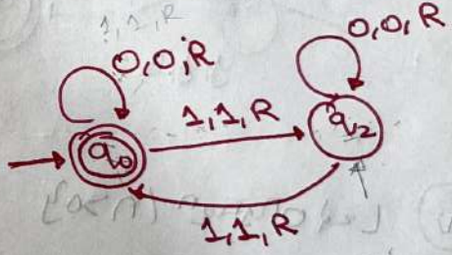


blank state

Present State	Tape symbol ϕ	a	b	x	y
→ q_0	ϕ R q_3	x R q_1	–	–	–
q_1	–	a R q_1	y L q_2	–	–
q_2	–	–	b L q_2	x R q_0	y R q_2
(q_3)	–	–	–	–	–

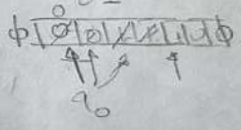
- Design a TM for set {0,1} that accept all string have all even no' of 1's.

i.e ∈, 00, 0000, ……



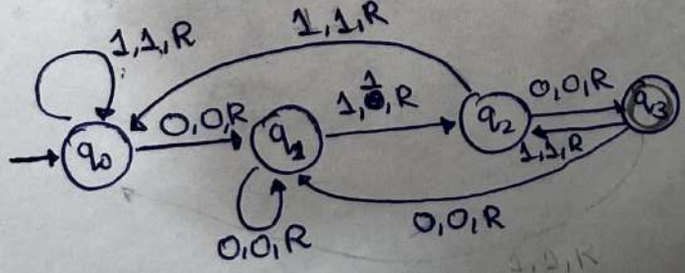
ϕ0010010ϕ

q_0

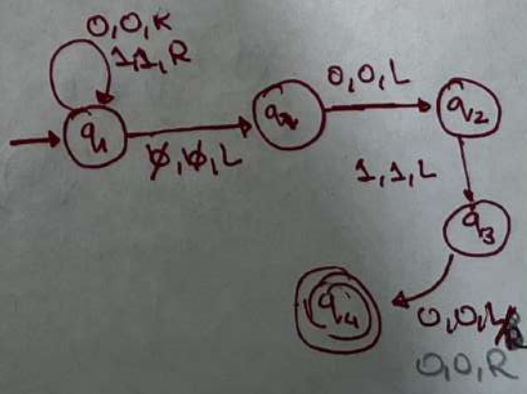


- Design a TM for set {0,1} that accept all string

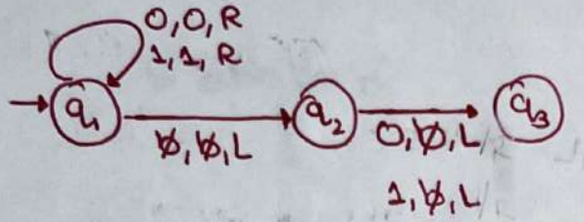
(1) end with 010



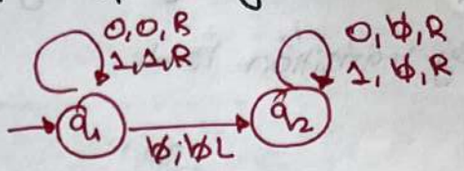
Or

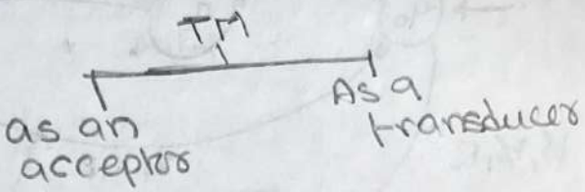


(ii) which can delete the last symbol of the I/p string.



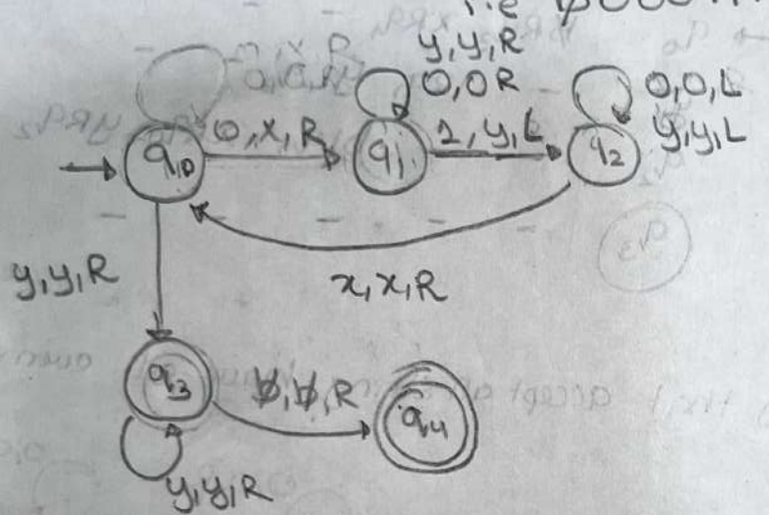
(or)





(iii) $L = \{0^n 1^n / n>0\}$

i.e bb0001111bb



not possible cases

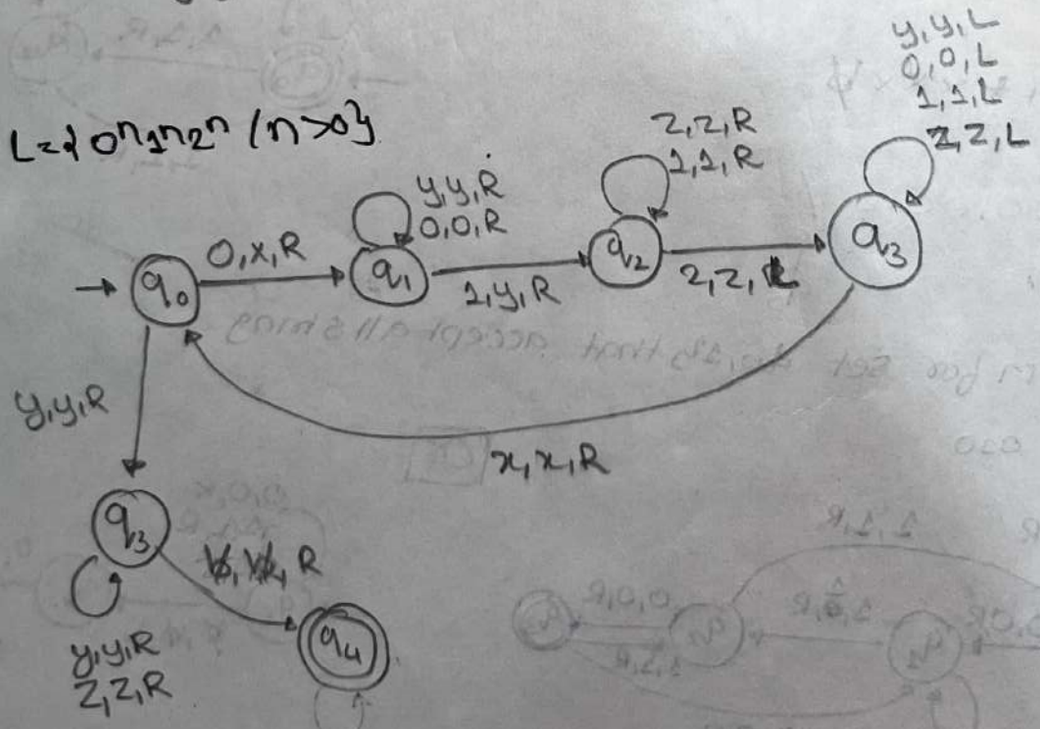
bb0011b1bb

bb0011bb

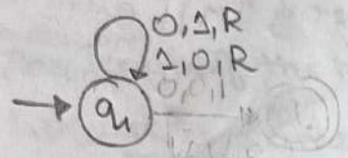
bb0011bb

bb1100b

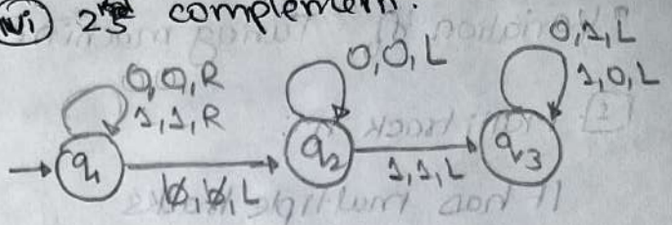
(iv) $L = \{0^n 1^n 2^n / n>0\}$



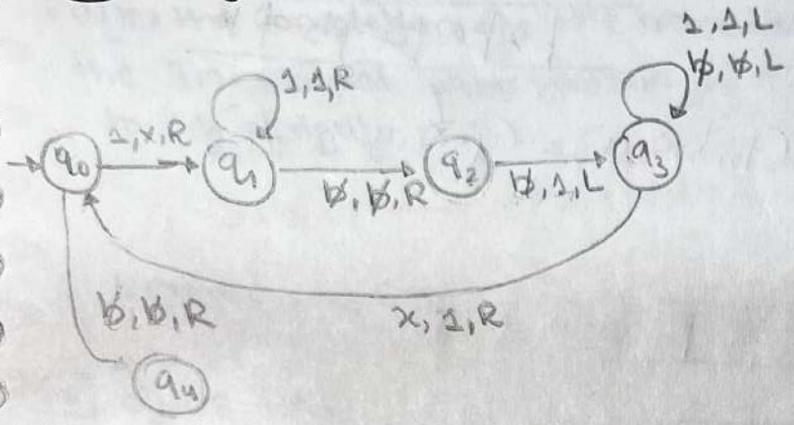
(v) 1st complement



(vi) 2nd complement.



(vii) Design a TM over set {1} which can copy the I/p string

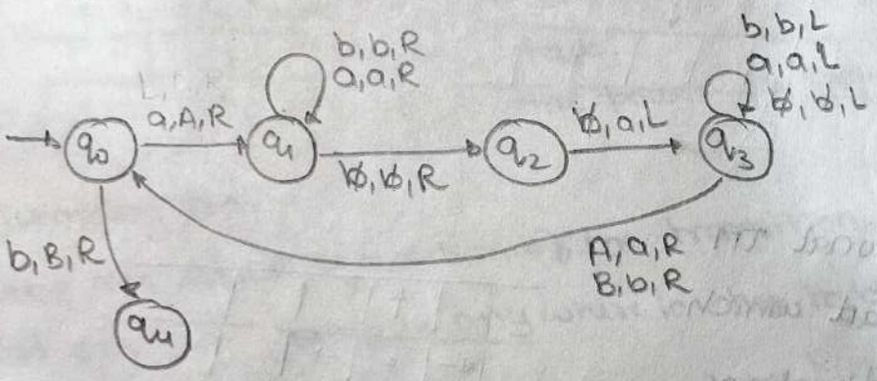


ex 111

qp: 111Ø111

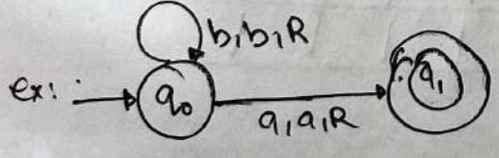
ØØØ 111 Ø 111 Ø ØØ

(viii) Design a TM over set {a,b} which can copy the Ip string



ex: abbab Ø abbab

ex:

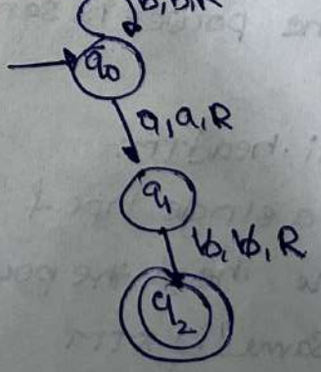


⇒ $b^*a(a+b)^*$

i.e bbbaa

only accepts these

ex. b^*a



in TM there is no trap state.

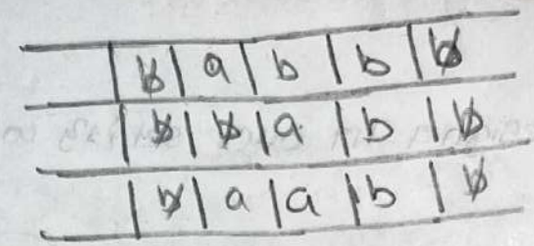
• Variation of Turing machine

[standard TM is more powerful than any TM] [every TM have same power]

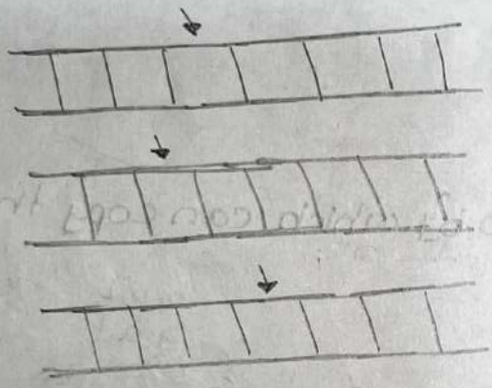
1. Multi track TM:

It has multiple tracks & single R/w head

The power is same as STM



$\delta(q_0, b, a, a) = (q', x, y, a, L/R)$

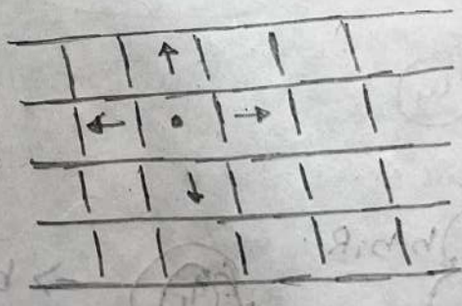


2. Multi-Dimensional TM

- It has R/w head which can move all directions.

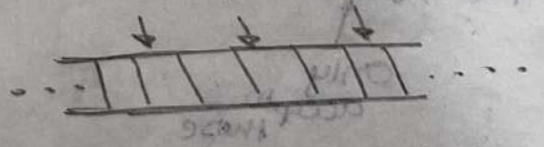
$\delta : Q \times T \rightarrow Q \times T \times \{L, R, U, D\}$

The power is same as STM



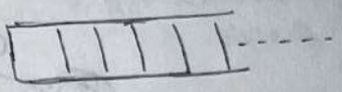
3. Multi-head TM:-

It is a single tape & multiple head the power is same as STM



4 TM with Semi-Infinite Tape / oneway Infinite tape TM

- The length of the tape will restricted in one direction
  - ↳ power is same as STM



5 TM with Stap option:

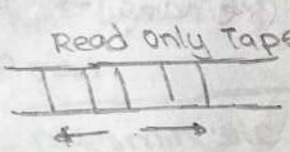
- after the completion of the transition the TM stay at same position or move to left or right

$$\delta : Q \times T \rightarrow Q \times T \times \{L/R/S\}$$

↳ Same as the STM

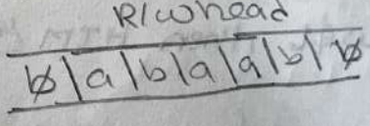
6 Offline TM:

- It has a tape of read only. whenever it process the i/p string it take the help of another Read/write tape.
- the power is same as STM



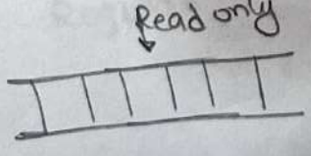
7 Jumping TM:

- these TM after completion of the transition, the Read/write head can jump to any where in the table.
- the power is same as STM



8 Restricted TM's.

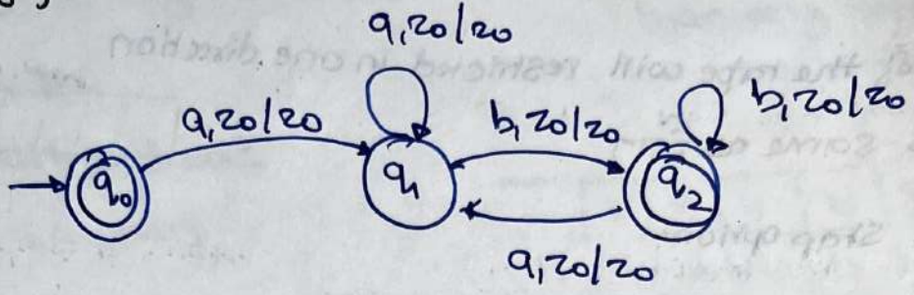
i) ▸ Read only TM:
- → It has a tape of Read only
- → the power is same as FA (2-DFA)



ii) ▸ one way TM:
- ▸ It has a tape of Read/write head but can move in only one direction.

[one way TM & one way Infinite Tape TM] are different.

Ex: $L = (a+b)^*$



Halting TM (HTM)

- $q_0 w \vdash^*$ If the TM halts in a final state → w is accept
- $q_0 w \vdash^*$ If the TM halt in a non final state → w is not accepted
- $q_0 w \vdash^*$ If the TM enters in to infinite loop (i.e hang) → w is not accepted

---

- a halting TM always halt for members as well as non-members
  - ↳ If $w \in L$, then HTM accepts by w by halting in a final state
  - ↳ If $w \notin L$,then HTM reject by w by halting in a non-final state
  - ↳ HTM accepts Recursive language

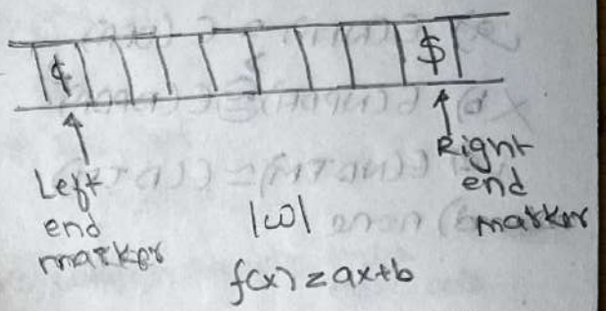
- Linear Bound Automata (LBA):

↳ LBA is used to accept context sensitive language (CSL)

↳ the length of the tape depend on string using end mark (restricting the size of tape)

↳ LBA is non deterministic

↳ In the LBA the length of the tape will be restricted A/C to the length of i/p string by using some linear functn



$M = (Q, \Sigma, T, \delta, q_0, \not b, F, ¢, \$)$

Right end maker

left end marker

$QXT \rightarrow [QXT \cdot X \cdot L, R]$

LBA

NDLBA

X DLBA (no info given) available

↳ TM that use only as much tape as the i/p takes

Non-Deterministic TM (NDTM)

$\delta : Q \times T \rightarrow [Q \times T \cdot X \cdot L, R]^*$

Ex: $\cdot \delta(q_0, a) = \{(q_1, x, R), (q_0, q, L) \cdots\}$ ← these is finite set

- every DTM is also NDTM but every NDTM is not DTM
- For every NDTM there exist an equivalent DTM
- power of NDTM is same as DTM

(DPA ≃ NFA) < DPDA < PDA < LBA < HTM < TM

Regular DCFL CFL CSL Recursive REL.

- Expressive powers of Automata
- It is the no. of languages accepted by the give automata

Q) which of the fall is/are false?

a) $\in$(NFA) $\simeq$ $\in$(DFA)

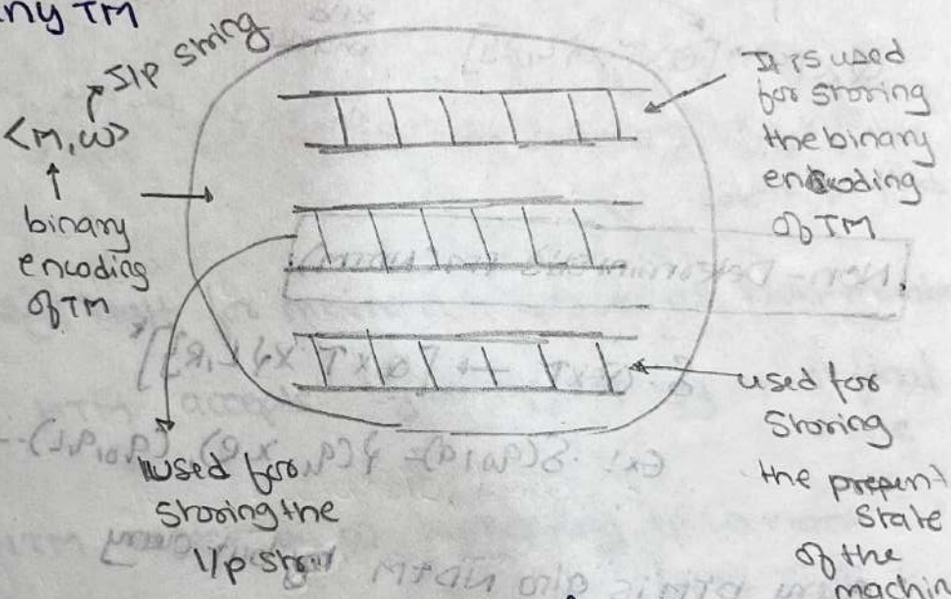
b) $\in$(NPDA) $\supset$ $\in$(DPDA)

c) $\in$(NDTM) $\simeq$ $\in$(DTM)

d) none

∴ the no. of language accept by NPDA is more than no. of language accept by DPDA

- Universal Turning Machine (UTM):
- UTM can simulate any TM
- the I/p to the UTM is binary encoding of the TM & the I/p string ω.



- If TM accepts ω then UTM is also accept ω
- If the TM doesn't accepts ω & halts then UTM is also reject ω & halt.
- If the TM enters into infinite loop on ω then UTM is also hangs on ω.

- The language accepted by UTM is called as universal language (L_u)

$L_u = \{\langle M,\omega\rangle / \omega \in L(M)\}$ is RE but not Rec

$\bar{L}_u = \{\langle M,\omega\rangle / \omega \notin L(M)\}$ is non-RE

↓

language that are not accepted by universal TM.

- Diagonalization language: the set of all TM's which reject their own code.

$L_d = \{e(T) / e(T) \notin L(T)\}$

e(T) is binary encoding of the TM, L(T) the language accepted by TM.

$\bar{L}_d = \{e(T) / e(T) \in L(T)\}$ is RE but non Rec

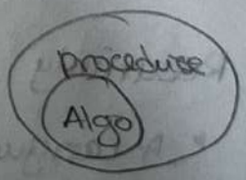
$\bar{L}_d$ is the set of all TM's which accepts their own code.

ex: M_1: $\delta(q_0, a) = (q_1, z, R)$
$\delta(q_0, b) = (q_2, b, L)$
$\delta(q_1, a) = (q_2, y, R)$

111010100100001001001001...

q_0-0 L-0 b-00
q_1-00 R-00 x-000
q_3-000 a-0 y-0000

- Church's Turing thesis:
- Every logically computable functn is Turing computable (i.e Runs on TM)
- For any problem if we can write a procedure, to implement that there exists a TM.
- A procedure may or may not halts in a finite amount of time. An algorithm always halt in some finite amount of time. Every algorithm is a procedure but every procedure in not a algorithm.



- If there exist a procedure to check coneather a string wεL or not? then L is REL
- If there exist an algorithm to check wheather a string wεL or not ? then L is Recursive

TM

as a acceptor ?
Every logically recognizable language inaccepted by TM

as a transducer ?
Every logically computable functn computed by TM

- Recursive & Recursively Enumerable language.

Recursive language :-
A language L is said to be Recursive iff there exist a HTM(Halting TM) to accept "L".

HTM always halts on all i/p
- If wεL → HTM halts in a final state
- If w∉L → HTM halts in a non-final state.

Recursive language are also called as Turing decidable language or Decidable language

L Rec → ∃ HTM
∃HTM → L Rec
(∃ HTM: there exist; L Rec: Language)

L ~Rec → ~∃HTM
~∃ HTM → L ~Rec.
(~: not.)

- Recursively Enumerable language (REL)
  - A Language "L" is Said to be REL iff there exists a TM to accepts "L"
    - if wεL → TM halts in a final state
    - If w∉L → TM may or may not halts.

L is RE → ∃ TM

∃ TM → L is RE

L ~RE → ~∃ TM

∃ ~TM → L ~RE

Recursively enumerable langⁿ are also called as Turing Recognizable language (or) Turing enumerable language (or) semi-decidable language (or) undecidable language

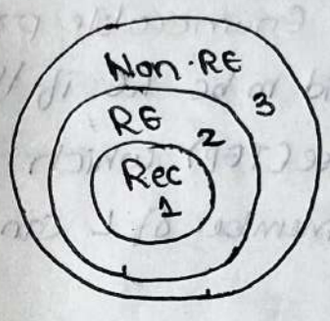
note

- every Rec is also RE but every RE is not Rec

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

→ $\Sigma^* = \{a, b\}^*$

2^{Σ^*} → The power set of Σ^*. i.e the set of all subset of Σ^* [i.e the set of all language]



It is not venn diagram

1 - Recursive (decidable)

1+2 - RE (semi decidable)

2 - RE but not (undecidable) Rec

2+3 - not Rec (undecidable)

3 - non RE (undecidable).

(1+2+3) - 2^{Σ^*}

set of all language

Q] L has TM → L is RE (semidecidable)

L has HTM → L is Rec

L has TM but not HTM → L is RE but not Rec

L has no TM → L is → non RE

L has TM but also HTM Rec.

Properties of RE & Rec language:

- Turning Enumerable procedure (TEP)
- Membership Algorithm (MA)
- Lexicographic ordering
- Closure properties
- L & $\bar{L}$ theorem

Turning Enumerable procedure (TEP)

- L is said to be RE if there exist an Turing Enumeration Procedure (TEP) which can runs on TM, using which all the member of L can be printed one by one on its tape.

Enumeration procedure (EP): It is a way such that we can generate all elements of the set in which any given element will be produced in a finite no. of steps.

$N = \{1, 2, 3, 4, \ldots\}$

1	2	3	4	5	6	...

EP = 1, 1000//

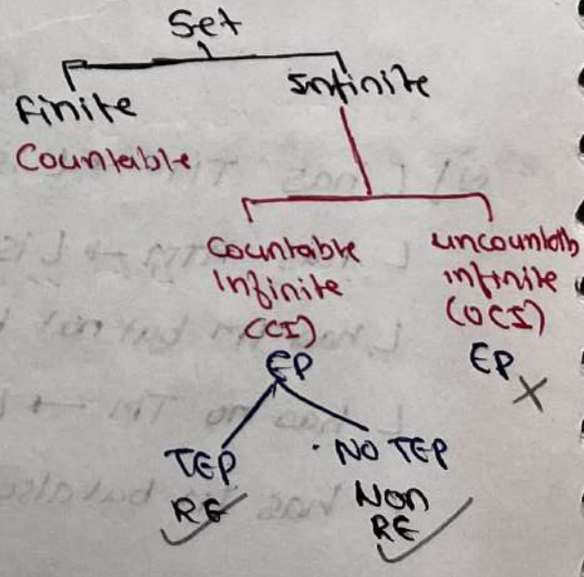
Real no. (R)

(1·1) ≅ (1·11) EP ✗

1·100000-9

Every "EP" can't be runs on TM

The "EP" which can runs on TM is called TEP.



- A language L is said to be Recursive iff both L & $\bar{L}$ has TEP.

i.e If L is Rec then the TM can print member as well as non-member.

1) L has TEP → L is RE
2) Both L & $\bar{L}$ has TEP → L is Rec
3) L has TEP but not $\bar{L}$ → L is RE but not Rec
4) L has no TEP → L is non RE
5) L has EP → L is may be RE or non RE

- Membership algorithm:
- It is process of checking whether a given string $w \in L$ or not?

Decidable language will have membership Algorithm

i.e L is Recursive iff ∃ MA for L

undecidable language won't have MA.

- L has MA → L is Rec
- $\bar{L}$ has MA → L is Rec
- L don't have MA → L is RE but not Rec or non-RE

- If L has MA → $\bar{L}$ also has MA

∴ L is recursive iff both L & $\bar{L}$ has MA

Lexicographic ordering

- L is said to be Recursive if it is enumerated in lexicographic ordering or Alphabatic ordering or Dictionary ordering

Let $\Sigma = \{a, b\}^*$

$\Sigma^* = \{\varepsilon, a, b, aa, ab, \ldots\}$

∴ Σ^* is Recursive & also regular

every Regular is also Recursive but every Recursive need not be regular

Closure properties :

- Recursive language are closed under
  - → union, Intersection, concatenation
  - Set difference, Quotient, Inverse Homomorphism, Reversal, Kleen's closure
  - Complement.

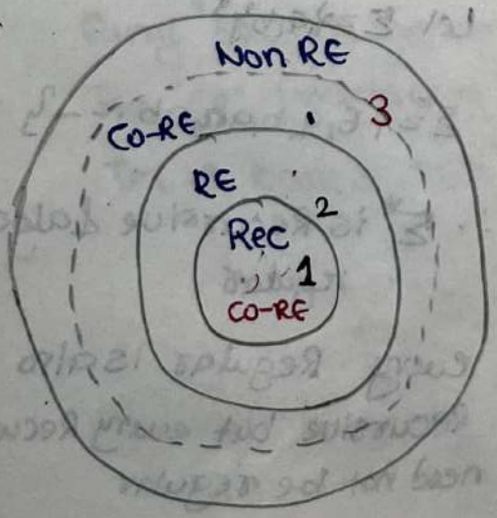
- Recursive language are not closed under.
  - → Homomorphism
  - → subset, superset
  - → Infinite union,
  - → Infinite intersection
  - → Infinite set difference

- Recursive Enumerable language are closed under
  - union, Intersection, concatenation, Quotient, Homomorphism, Inverse Homomorphism, Reverral, Kleen's closure.

- REL are not closed under
  - → Complement
  - → Set difference
  - → subset, superset
  - → Infinite (union, Intersectn Set difference)

L & $\bar{L}$ theorem

- every recursive (rec) is also REL (Recursive enumberable) but not every REL may not be Rec
- If L is Recursive → L is RE (Recursive enumberable)
- If L is RE → L is may or may not be RE
- If L is Recursive → $\bar{L}$ is also Recursive
- If L is REL → $\bar{L}$ is may or may not be RE
- If L & $\bar{L}$ both are RE → L is Rec & $\bar{L}$ is also Rec



- If L is Rec → $\bar{L}$ (co-Rec) is also Rec
- If L is RE → $\bar{L}$ (co-RE) is many or may not be RE.

// given L then find out $\bar{L}$

If two complement language L & $\bar{L}$ are given then one of the following cases will be true.

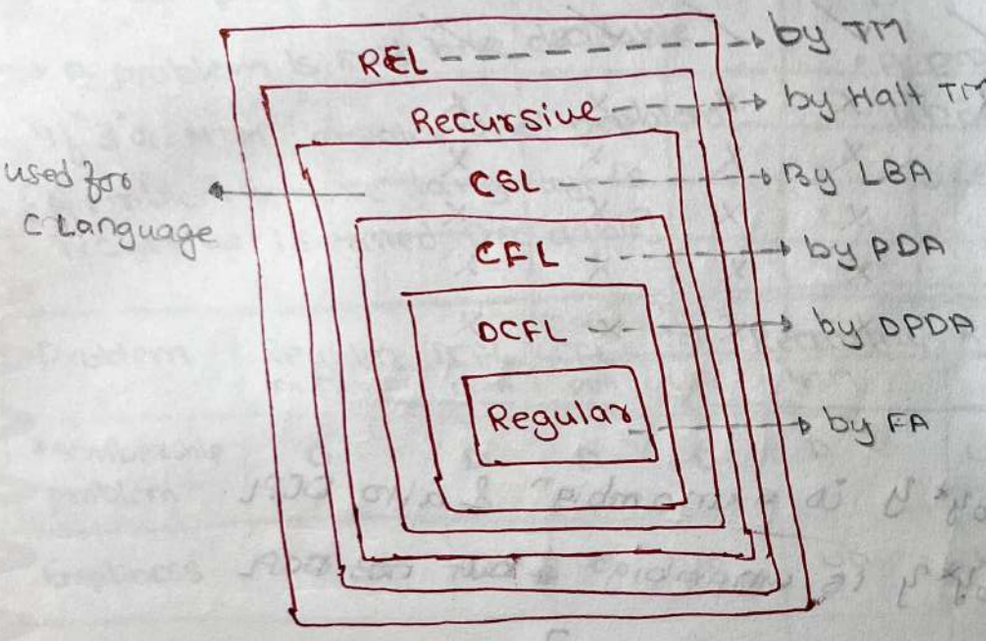
i) Both L & $\bar{L}$ can be Rec

ii) One of them is RE but not Rec & the other one is non RE

iii) Both L & $\bar{L}$ can be non-RE

Note: Both L & $\bar{L}$ are RE but not Rec is never possible.

- chomsky Heirarchy:



- Let L_1 is regular; L_2 is DCFL, L_3 is CFL then.

$L_1^R \cup L_2^* \cap L_3$ is?

CSL//

Order of precedence
[Reversal > Kleens closure > Intersection > union.]

$\Rightarrow$ (Regular)$^R \cup$ (DCFL)$^* \cap$ CFL

$\Rightarrow$ Regular $\cup$ (DCFL$\uparrow$)$^* \cap$ CFL

$\Rightarrow$ Regular $\cup$ (CFL)$^* \cap$ CFL

$\Rightarrow$ Regular $\cup$ (CFL$\uparrow \cap$ CFL$\uparrow$)

$\Rightarrow$ Regular $\cup$ (CSL $\cap$ CSL)

$\Rightarrow$ Regular $\cup$ CSL

$\Rightarrow$ CSL//

	Regular	DCFL	CFL	CSL	Rec	REL
$L_1 \cup L_2$	✓	×	✓	✓	✓	✓
$L_1 \cap L_2$	✓	×	×	✓	✓	✓
$L_1 \cdot L_2$	✓	×	✓	✓	✓	✓
$L_1 - L_2$	✓	×	×	✓	✓	×
L^*	✓	×	✓	✓	✓	✓
L^R	✓	×	✓	✓	✓	✓
$\bar{L}$	✓	✓	×	✓	✓	×
$H(L)$	✓	×	✓	×	×	✓
$H^{-1}(L)$	✓	✓	✓	✓	✓	✓
$\subseteq$	×	×	×	×	×	×
$\supseteq$	×	×	×	×	×	×
$\infty \cup$	×	×	×	×	×	×
$\infty \cap$	×	×	×	×	×	×
$\infty -$	×	×	×	×	×	×

$L = \{ w c w^T \mid w \in \{a,b\}^* \}$ is a unambigⁿ & also DCFL

$L = \{ w a w^T \mid w \in \{a,b\}^* \}$ is unambigⁿ & but not DCFL

$L = \{ w c w \mid w \in \{a,b\}^* \}$

$L = \{ a^n b^n c^n \mid n \geq 0 \}$

$L = \{ N_a(w) = N_b(w) = N_c(w) \mid w \in \{a,b\}^* \}$

all are not CFL but CSL But complement of all these language are CFL

Undecidability

- Definition of problem
- Undecidability problem on TM
- Reducebility
- RICE's Theorem
- Advance RICE's theorem
- Countability

- Decibility
- A problem is said to be decidable if "∃" an algorithm to solve that problem
- A problem is said to be decidable if "∃" a HTM to solve that problem

[A problem whose language is recursive is called Decidable]

- Undecidability
- A problem is said to be undecidable if "∃" a procedure to solve that problem
- A problem is said to be undecidable if "∃" a TM to solve that problem

Problem	Regular (FA & Regular grammer)	DCFL (DPDA)	CFL (CFG) (PDA)	CSL (Type-1) (LBA)	Recursive (HTM)	REL (Type-0) (TM)
Membership problem	D	D	D	D	D	UD
Emptiness	D	D	D	UD	UD	UD
Finiteness	D	D	D	UD	UD	UD
Regularity	D	D	UD	UD	UD	UD
Ambiguity	D	D	UD	UD	UD	UD
Completeness	D	D	UD	UD	UD	UD
Equivalence	D	D	UD	UD	UD	UD
Disjointness	D	UD	UD	UD	UD	UD

Decidability is applicable to all [language grammer, machine, & there complement]

Membership problem

A given string w, a language L,
Is $w \in L$?

- Let L is RE, is $w \in L$? UD
- Let G be a CFG,
  Is $w \in L(G)$? D.
- Let M is a TM, is a $w \in L(M)$? UD.

Ex: L is CSG, Is $w \notin L(G)$? D

L is M is a TM, Is $w \notin L(M)$? UD

$w \in L$ & $w \notin L$
both problem belong to membership problem

note

- If a problem is Decidable
  → complement is also Decidable
- If a proble is undecidable
  → complement is also undecidable

i.e If L is Decidable → L is Decidable

- If L is undecidable → L is undecidable

Emptiness Problem

- A given language L,
  Is $L \neq \phi$ or $L = \phi$

- Let L is a CFL, is $L = \phi$? D
- Let G be a CSG, is $L \neq \phi$? UD
- Let M is a HTM, is $L \neq \phi$? UD
- Let L is regular, is $L \neq \phi$? D

Finiteness Problem

- A given language L,
  Is L a finite or infinite?

- Let L is a Rec, is L a finite? UD
- Let G be a CSG, is L(G) infinite? UD
- Let M is a TM, is L(M) a finite? UD

Regularity problem

- A give language L,
  Is L a regular? or non-regular

- Let L is a Rec, is L a regular? UD
- Let G be DCFG, is L(G) a non-regular? D
- Let M is a TM, is L(M) a regular? UD

Ambiguity problem

- a given language L, is L an ambiguous or unambiguous

ex: Let L is ACSL, Is L an ambiguous ? UD

Let G be a Type 0 grammer, is L(G) unambigous? UD

Let M be a LBA , is L(M an ambiguous? UD

Completeness Problem

- A given language L,

Is $L = \Sigma^*$ or $L \neq \Sigma^*$?

ex: Let L is a RE , Is $L = \Sigma^*$? UD

Let G be a CFG, is $L(G) \neq \Sigma^*$? UD

Let M is a TM, is $L(M) = \Sigma^*$? UD

Equivalence Problem

- A given language L_1, L_2

Is $L_1 = L_2$ or $L_1 \neq L_2$?

ex: Let L_1 & L_2 are 2 DCFL, Is $L_1 = L_2$? D

Let G_1 & G_2 be 2 CFG's, is $L(G_1) \neq L(G_2)$? UD

Let M_1 & M_2 be 2 TM's, is

$L(M_1) = L(M_2)$? UD

Disjointness Problem

- Two given language L_1 & L_2

is $L_1 \cap L_2 = \phi$? or $L_1 \cap L_2 \neq \phi$?

ex: Let L_1 & L_2 are 2 regular language, is $L_1 \cap L_2 = \phi$? D

ex: Let G_1 & G_2 be 2 CSG's,

Is $L(G_1) \cap L(G_2) = \phi$? UD

ex: Let M_1 & M_2 be 2 HTM's

Is $L(M_1) \cap L(M_2) \neq \phi$? UD

- Undecidable problem on TM.
  - Halting problem (HP)
  - Blank Tape Halting problem (BTHP)
  - State entry problem (SEP)
  - POST Correspondance problem (PCP)
  - Modified PCP (MPCP)

All these problem are RE but not REC

- Halting problem:
- Let M is a TM, while processing the string w, will the M halts is undecidable.

- State Entry problem (SEP)
- while processing a string 'w', will the TM halts in a particular State is undecidable
- wheather a TM produce the o/p is undecidable.

- Blank Tape Halting problem
- own on a blank tape will TM halts is undecidable.

- post correspondance problem (PCP)
- Let $x = \{x_1, x_2, \ldots x_n\}$ & $y = \{y_1, y_2, \ldots y_n\}$ are the 2 set having same no. of element & defined over same I/p alphabet.
- Then the PCP is to determine wheather or not there exist a sequence such that for that sequence the string formed from the two set are equal

[you can take x_i any no. of time, at any position]

ex: $x = \{1, 10111, 10\}$ & $y = \{111, 10, 0\}$

	2	1	1	3
x	10	111	111	0
y	10111	1	1	10

ex: $x = \{a, ab, bab\}$, $y = \{aa, abb, baa\}$

for no combination these both are equal

- Modified PCP (MPCP)

In MPCP the sequence will start with 1st element of the two sets.

i.e $x_1 \ldots x_i \cdot y_1 \ldots y_i$

- if MPCP has solution → PCP also has solution
- if PCP has solution → Then MPCP may or may not have the solution
- Both MPCP & PCP are decidable on unary alphable

- **Reducebility.**

- If $P_1 \leq P_2$ → using the solution of P_2 we can solve the Problem P_1

P_1 is reducebld to P_2

- if $P_1 \leq P_2$ If P_1 is undecidable → P_2 is also undecidable (undecidable uses forward)

- If $P_1 \leq P_2$ If P_2 is Decidable → P_1 is also Decidable (Decidable uses backward)

L_1 L_2

Dec ← Dec

Semi-Dec ← Semi Dec

Undecidable → undecidable

Rec ← Rec

RE ← RE → Backward direction

non-RE → non-RE

Forward direction

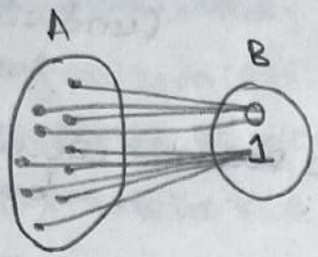
- If $L_1 \leq L_2$ & $L_2 \leq L_3 \rightarrow L_1 \leq L_3$
  - If L_1 is undecidable → L_2 & L_3 are undecidable
  - If L_2 is undecidable → L_3 is also undecidable but L_1 is not know
  - If L_2 is Decidable → L_1 is also Decidable but L_3 is not know
  - If L_3 is Decidable → L_1 & L_2 are also Decidable
  - If L_3 is undecidable → L_1 & L_2 are not known.

RICE's Theorem

- only applicable on RE language

- Every non-trivial property of RE language is undecidable.

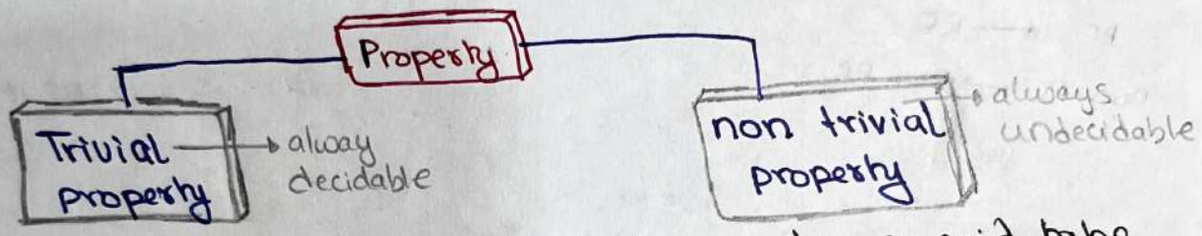
Property: It is a mapping / functn from set of all language to set {0,1}



A = set of all RE language only

P(L)=1 The language satisfies the property "P".

P(L)=0 = The language doesn't satisfies the property "P".

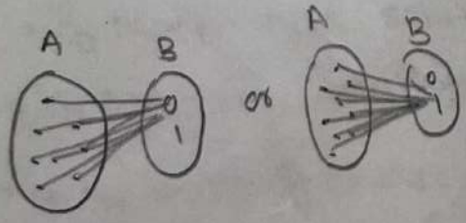


- A property is said to be trivials if either all language are satisfying that property or all language are not satisfying that property.

$\forall L\,(P(L)=1) \vee \forall L(P(L)=0)$

($\forall L(P(L)=1)$: satisfies; $\vee$: or; $\forall L(P(L)=0)$: doesn't satisfies)

every language

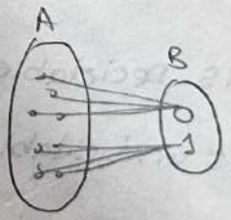


- L is a RE
- L is non-RE
- L is accepted by TM
- L is not accepted by TM

- A property is said to be non-trivial if some language are satisfying that property and atleast one of them is not satisfying that property

$\exists L(P(L)=1) \wedge \exists L(P(L)=0)$

ex:



L is Regular

L is accepted by PDA

L is Recursive

is non-trivial b/c

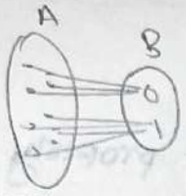
every RE is also Recursive but every Recursive need not be RE.

Q] Does the given String 1001 belongs to RE language ?

↓

membership problem of RE [UD]

Rice's theorem :-



1001

↓

It need to be appear in every String so some satisfy and some not

$L = \{\langle M \rangle / \text{ is } 1001 \in L(M) ?\}$ is UD (is non trivial property)

↓

binary encoding TM

<M> : In general represent encoding of some machine.

these encoding Can be anything binary/unary/decimal

Q] Does the given RE language is empty? is non trivial property UD

$L = \{\langle M \rangle / \text{ is } L(M) = \phi ?\}$

Q] Does the given RE language is Σ^*? is non trivial property UD

$L = \{\langle M \rangle / \text{ is } L(M) = \Sigma^* ?\}$

Q] Does the given RE language is regular?

$L = \{\langle M \rangle / \text{ is } L(M) \text{ a regular } ?\}$ is UD

——— x ——— can't be solved using table.

Q] Does the given RE language is CFL? is non-trivial property UD

$L = \{\langle M \rangle / \text{ is } L(M) \text{ a CFL } ?\}$

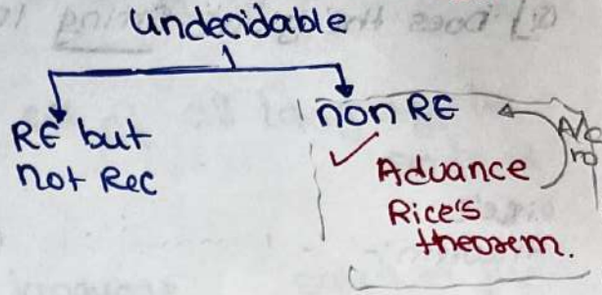
Q] Does the given RE language is recursive?

$L = \{\langle M \rangle / \text{ is } L(M) \text{ a recursive } ?\}$ UD

Q] Does the given RE language is RE?

$L = \{\langle M \rangle / \text{ is } L(M) \text{ a RE } ?\}$ is Decidable.

Advanced Rice's theorem



- A non-monotone property of RE language is non-RE
- monotone property
  - If a language L is satisfying a property (i.e $P(L)=1$) then if all superset of L are also satisfying that property then it is said to be monotone

$$P(L)=1 \rightarrow \forall\, L'\ P(L')=1 \quad ; \text{where } L' \supset L$$

- non-monotone property
  - If a language L is satisfying a property (i.e $P(L)=1$) then if atleast one superset of L is not satisfying that property then it is said to be non-monotone.

$$P(L)=1 \longrightarrow \exists\, L''\ P(L')=0 \quad ; \text{where } L' \supset L$$

Ex: Is $L=\phi$? is Non-monotone

$L'=\{w \mid \_\_\}$

Is $L\neq\phi$? is monotone

$L=\{1,2,3,4\} \neq \phi$

$L' \subset L \qquad L_1 \supset L$

$L'=\phi$

Is L a regular ? is non monotone

$L'=\{a^n b^n \mid n\geq 0\}$ is not regular

$P(L)=0$

$L=\{a^n b^n \mid n \leq 10000\}$ is regular

$L \supset L', \quad P(L')=1$

Is La non regular? is non-monotone

$L = \{a^n b^n / n \geq 0\}$ is not regular.

$P(L) = 1$

$L' = (a+b)^*$ is regular

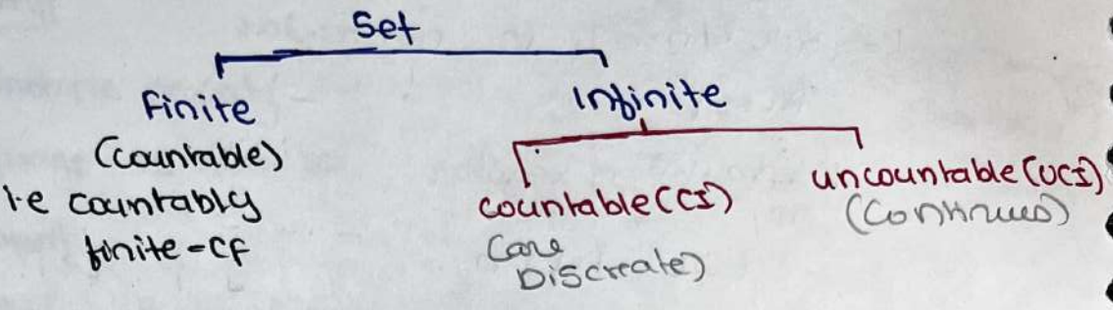
$L' \supset L$, $P(L') = 0$

- Does the TM accepts a string of length 2024? UD.
- Does a given TM accepts ϵ? UD
- Does a given TM halt in 2024 steps? decidable
- Does a given TM prints an op on its tape? UD
- Does a given TM produces an o/p ? UD
- Does a given TM has 100 states? D → can be seen in diagram or shown in diagram
- Does a given TM accept a CFL? D.
  - TM → accept every language

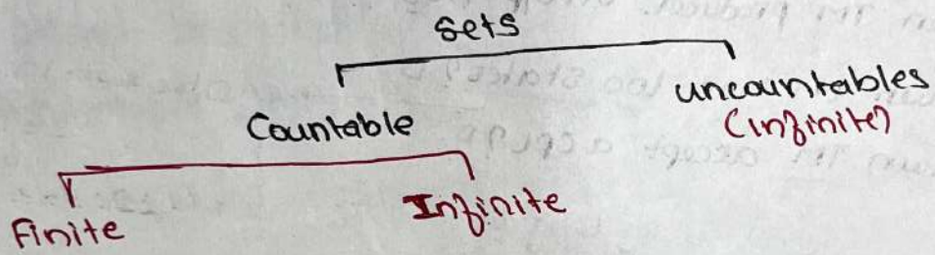
but RE language doesn't accept a CFL every time.

- Does the TM halts on a string of length 2024? UD.
- Does a given TM halt on accept ϵ? UD.
- $L = \{\langle M, w, q \rangle / M$ on i/p w reaches state q in exactly 2023 steps$\}$ is Decidable
- $L = \{\langle M \rangle / L(M)$ is not recursively$\}$ ---- } these involve the membership problem which is undecidable
- $L = \{\langle M \rangle / L(M)$ contain atleast 2023 numbers$\}$ }

Countability



- every finite set is countable
  - finite → countable (one way)
  - countable → finite (false)
  - Infinite → uncountable (false)
  - uncountable → Infinite (True).



- Every Countable set is discreate.
  - Countable → Discreate (2 way)
  - Discreate → Countable ✓
  - uncountable → not discrete ✓ (continues)
  - not Discreate → not countable ✓ (continues)

∴ every uncountable set is continous.

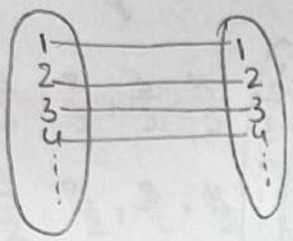
- Countable:
  - A set is said to be countable if "∃" a bijection b/w the given set & any subset of natural no..

or

- A set is said to be countable if ∃ an enumeration procedure (EP) using which all the element of the set can be generated such that for any particular element it takes only a finite no. of step and these step are called as Index of that element.

- Set of natural no. N is.

$N = \{1, 2, 3, 4, \ldots\}$



$N = \{1, 2, 3, \ldots\}$

1 | 2 | 3 | 4 | ----

$EP = +1$.

∴ N is countably Infinite (CI)

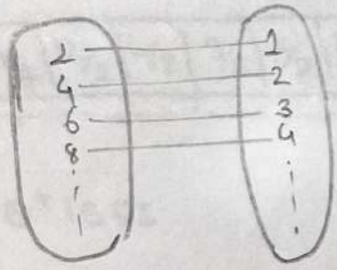
- Set of Real no. (R) is.

1-1 | ?

There is no EP for set of Real no.

∴ R is uncountably Infinite (UCI)

- Set of Even natural no. is



2 | 4 | 6 | ....

$EP = +2$

∴ even natural not is CI.

(Countable Infinite)

Cardinality of set of even natural nos is equal to cardinality of set of natural no.

- set of odd natural no. is CI //

Note

- every subset of countable set is also countable

it can be finite or infinite

- Set of all natural no. of multiples of 3 is CI
- Set of all all prime no. or non-prime no. is CI
- Set of all prefect no. is CI
- Set of all prime no. less than 1 lack is CF → Countable finite

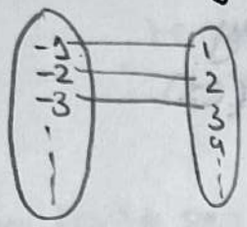
- Set of the integer (Z^+) is CI

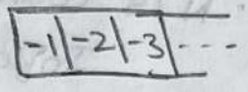
$Z^+ = \{1,2,3,4,5 \ldots\}$

- Set of integers (Z) is CI

$Z = \{\ldots -3,-2,-1,0,1,2,3 \ldots\}$

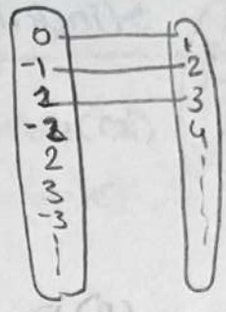
- Set of -ve integers (Z^-) is CI

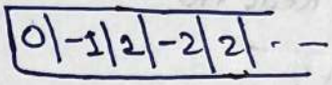




$Ep = -1$

Bijection is possible





$Ep = -1, +1$

- $CF \cup CF = CF$
- $CF \cup CF \cup CF \cup \ldots$ infinite times $= CI$
- $CF \cup CI = CI$
- $CI \cup CI = CI$
- $CI \cup CI \cup CI \cup \ldots$ countable time $= CI$
- $CI \cup UCI = UCI$
- $UCI \cup UCI = UCI$
- $CI \cup CI \cup CI \cup \ldots$ countable time UCI.

- $CF \times CF = CF$
- $CI \times CF = CI$
- $CF \times CI = CI$
- $CI \times CI = CI$
- $CI \times UCI = UCI$
- $UCI \times UCI = UCI$

- the set of all subset of countably finite set is 2^{CF} is CF

$$2^{CF} = CF$$

- The set of all subset of CI (i.e 2^{CI}) is UCI

$$2^{CI} = UCI$$

- Set of +ve Rational no. (Q^+) is **CI**

$Q^+ = \frac{p}{q}$, where $p, q \in Z^+$; $q \neq 0$

$\therefore Q^+ = CI \cup CI \cup CI \cdots$

Countable time = CI

$\frac{1}{1}, \frac{2}{1}, \frac{3}{1}, \frac{4}{1}$ ..... $\infty = CI$

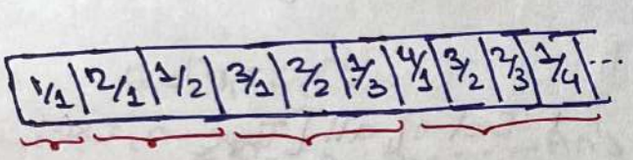
$\frac{1}{2}, \frac{2}{2}, \frac{3}{2}, \frac{4}{2}$ ..... $\infty = CI$

$\frac{1}{3}, \frac{2}{3}, \frac{3}{3}, \frac{4}{3}$ ..... $\infty = CI$

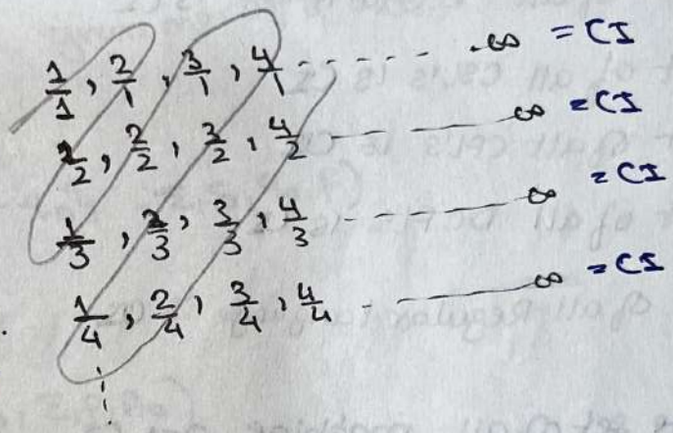
$\therefore Q^+$ is CI

Countable Infinite

Cantor's Digonalization Theorem



$\therefore Q^+$ is CI



countable Infinite

- Set of complex no. is $(x + iy)$ is **UCI**
  UCI UCI

- Set of all string over Σ^* is **CI**

Let $\Sigma = \{a, b\}$ $\quad \epsilon \mid a \mid b \mid aa \mid ba \mid ab \mid bb \mid \ldots$

- Set of all subset of Σ^* is (i.e $2^{\Sigma^*} = 2^{CI}$) is **UCI**

i.e Set of all language over Σ^* is UCI

- Set of all TM's is **CI**

⇒ Rec + RE + non-RE = 2^{Σ^*}

RE + non-RE = 2^{Σ^*} → Set of all Formal language

⇒ RE + non RE = 2^{Σ^*}

CI + non RE = UCI

non RE = UCI

i.e Set of all REL's is CI

Set of all non REL is UCI

Every non-REL is subset of Σ^* → Every non-RE is CI

- Set of all REL's is CI
- Set of all Rec language is CI
- Set of all CSL's is CI
- Set of all CFL's is CI
- Set of all DCFL's is CI
- Set of all Regular language is CI

- here set of all machine are CI, ∴ their cardinality is same

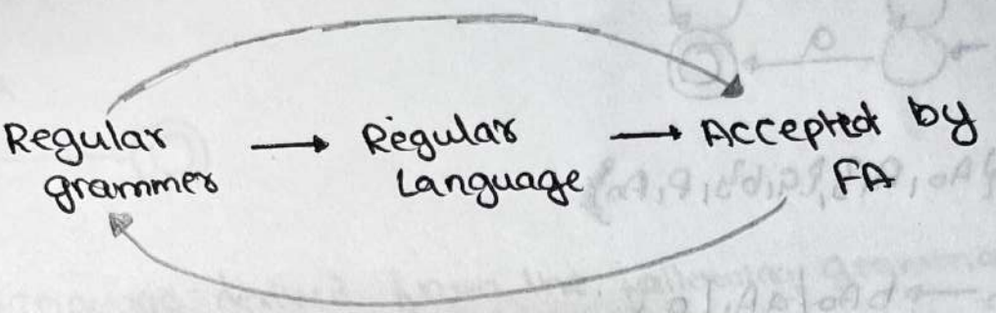
- Set of all { TM, HTMS, LBA, PDA's, DPDA's, NFA's, DFA's } is CI

- Set of all [Type-0, CSG, CFG, linear, Regular] is CI

- Set of all [non REL's; non-Rec language, non CSL's, non CFL's, non-DCFL's, non-regular language] is UCI

- Set of all subset of regular language (i.e 2^{CI}) is UCI
- Set of all subset of a finite language (i.e 2^{CF}) is CF
- Set of all finite languages is CI
- Set of all Syntaxtually correct 'C' program is CI
- Set of all Syntaxtually incorrect 'C' program is CI

- Set of all 'C' program is CI
- Set of all decidable language is CI
- Set of all undecidable language is UCI



- Construction of a Regular grammar for a given Finite Automata :-

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

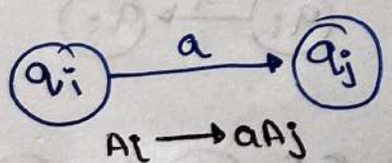
$= (\{q_0, q_1, q_2 \dots\dots q_n\}, \Sigma, \delta, q_0, F)$

$G = (V, T, P, S)$

$= (\{A_0, A_1, A_2 \dots\dots A_n\}, \Sigma, P, A_0)$

The production in P are defined as follows :-

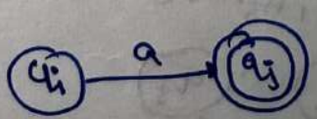
i) $A_i \rightarrow aA_j$ will be in P if $\delta(q_i, a) = q_j$ where $q_j \notin F$



ii) $A_i \rightarrow aA_j$ & $A_i \rightarrow a$ will be in P if $\delta(q_i, a) = q_j$ where

$q_j \in F$

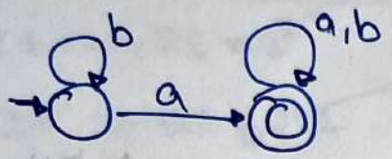
q_j → Can be any State



$A_i \rightarrow aA_j$ & $A_i \rightarrow a$

ex: Construct a regular grammer corresponding to finite automata represented by regular expression?

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



$G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$

$P = \{A_0 \rightarrow bA_0 / aA_1 / a,$

$A_1 \rightarrow aA_1 / a / bA_1 / b\}$

- Construction of a Finite automata for a given Regular grammer

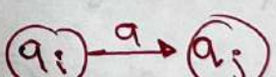
$G = (V, T, P, S)$

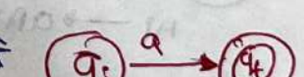
$= (\{A_0, A_1, A_2, \ldots A_n\}, T, P, A_0)$

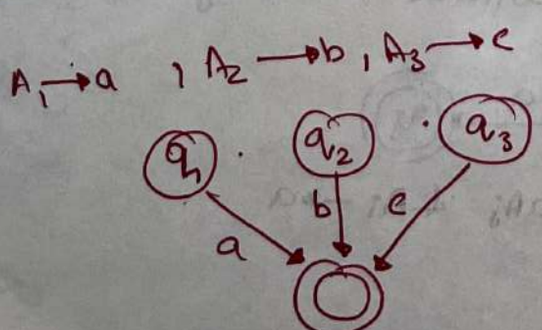
$M = (Q, \Sigma, \delta, q_0, F)$

$= (\{q_0, q_1, q_2, \ldots q_n, q_f\}, T, \delta, q_0, \{q_f\})$

δ is defined as follows.

(i) $A_i \rightarrow a\ A_j$ gives $\delta(q_i, a) = q_j$ 

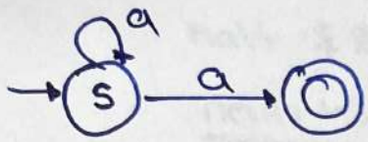
(ii) $A_i \rightarrow a$ give $\delta(q_i, a) = q_f$ where $q_f \in F$ 



(iii) $A_i \rightarrow \epsilon$. give q_i as a final state.

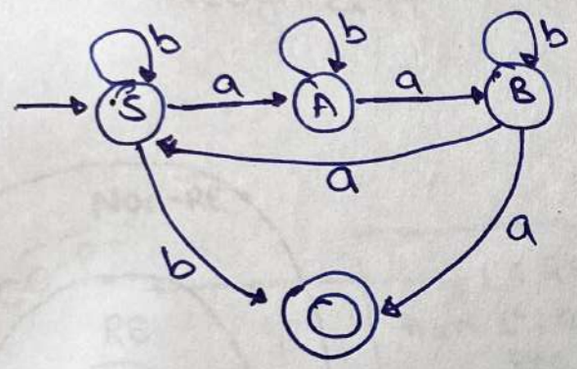
Q] Construct a finite automata equivalent to the following grammer.

$G = (\{S\}, \{a\}, S \to aS/a, S) \to L(G) = \{a^n / n>0\}$



Q] the language derived from the following grammar is.

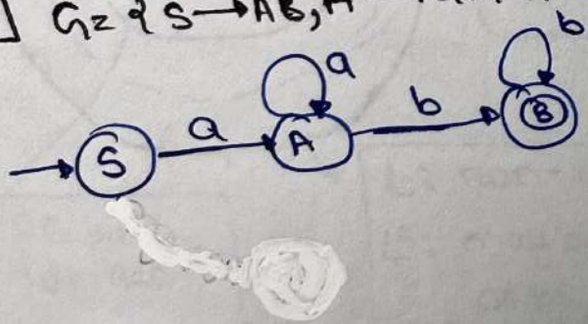
$G = \{S \to bS/aA/b,\ A \to bA/aB,\ B \to bB/as/a\}$.



$\{w \in \{a,b\}^* / Na(w) = 3K,\ K = 0,1,2 \cdots\}$

$a^m / m \geq 1$ $b^n / n \geq 1$

Q] $G = \{S \to AB, A \to aA/a, B \to bB/b\}$



is non regular grammer

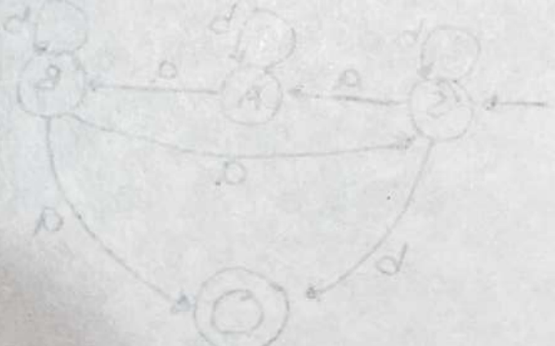
note

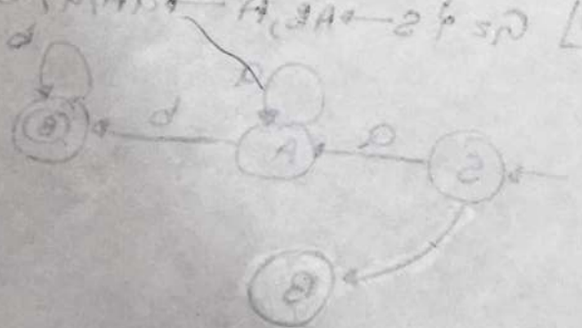
a non-regular grammer Can generate a regulare Language.

$\Rightarrow \{a^m b^n / m,n>0\}$ is regular

- In a case where when Language is DCFL & the question ask if it is equilvalent to CFL or not. is it decibable or not

↳ So it is a problem which is decible b/c we can identify or classify among the both.





Recursive Enumerable Language

(acceptable)
[Turing recognizable]

- L is RE if there is TM
- 3State : Halt & Accept
  Halt & Reject
  never halt.

- A language is called Turing Recognizable if some Turing machine recognize it

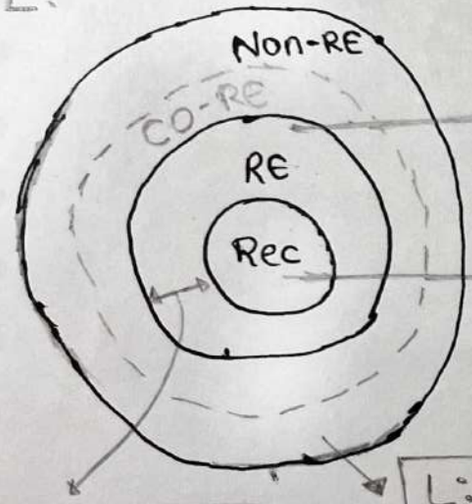
Recursive Language

[Turning decidable]

- L is Recursive if there is Halting / Total TM
- 2State : Halt & Accept
  Halt & Reject

- A Language is called Turing Decidable if some Turing Machine decide it.

L:



If L: RE (then L may be RE or may not be RE)
then L^c: can be Rec or non-RE

If L: Rec → then L is Recursive enumberable
then L^c: Rec

If L: RE but not Rec
L^c: is non-RE

L: non-RE
L^c: may be non-RE or RE (But never Rec)

- if both L & $\bar{L}$ are recursive, then L must be RE
- if L^c is non-regular then L must be non-regular.

- Turning machine should have atleast two state One non-final state & a final state to reject or accept a given string respectively
- For Turning machine the i/p alphabet Σ need not be same as the tape alphabet Γ. (i/p alphabet doesn't contain a blank symbol).
- Turning machine without ink is like 2-DFA
- Computing power of FA + 2 Stacks is equal to
  - ↳ FA + 2 counter
  - ↳ FA + 3 Stack
  - ↳ Turing machine.
- for a given i/p if the transition is not present then that string will get rejected.