TOC

Theory of computation

unit -1

String & alphabet

Alphabet: An alphabet is a finite non-empty set of symbols. These are the input symbols from which string are constructed by following certain operation. we use Σ to denote Alphabet.

$\Sigma = \{0, 1\}$ → alphabet set of binary no.

$\Sigma = \{0, 1, --- 9\}$ → " " " decimal no.

$\Sigma = \{a, b, c --- z\}$ → " " " alphabet

*** String:** String is a finite sequence of symbol selected from some alphabets it is generally denoted by s and ω.

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

$S = 0101$

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

$S = ab = abcd - X$

$= aba$

$= abba$

*** length of string :** $|\omega|$ $|s|$

length of string is the no of symbols present in the string. s. Denoted by $|\omega|$ & $|s|$

if $|\omega| = 0$ or $|s| = 0$ → empty string

denoted by → $\lambda, \varepsilon,$ → Null

↳ epsilon

Kleem positive ($\varepsilon+$)

Kleem star ($\varepsilon*$)

* Kleen star (Σ^*) : It is an infinite set of all possible strings of all possible length over Σ including Null (λ), ε, lembda (λ)

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \dots \Sigma^n$$

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

$\Sigma^0 = \lambda$ (null)

$\Sigma^1 = \{a, b\}$

$\Sigma^2 = \Sigma^1 \times \Sigma^1$

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

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

$\Sigma^3 = \Sigma^2 \times \Sigma^1 = \{aa, ab, ba, bb\} \times \{a, b\}$

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

$\Sigma^* = \{\lambda, a, b, aa, ab \dots\}$

* Kleen positive (Σ^+) : It is an infinite set of all possible strings of all possible length over Σ excluding lembda or Null.

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots \Sigma^n$$

$= \{a, b, aa, ab, ba, bb, aaa \dots\}$

$\Sigma^1 = \{a, b\}$

$\Sigma^2 = \{a \times b\}$

$\Sigma^2 = \Sigma^2 \times \Sigma$

$\Sigma = \{d\}$

$\varepsilon^* = \varepsilon^0 \cup \varepsilon^1 \cup \varepsilon^2 \dots \varepsilon^n$

$\varepsilon^0 = \varepsilon$

$\varepsilon^1 = \{d\}$

$\varepsilon^2 = \{d\} \times \{d\}$

$\{dd\}$

language : A set of strings of symbols that follow well-defined rules

$\Sigma^3 = \{dd\} \times \{d\}$

$\Sigma^3 = \{ddd\}$

$\Sigma^* = \{\Lambda, d, dd, ddd, \dots\}$

$\Sigma^+ = \{d, dd, ddd, \dots\}$

Q. • Elaborate theory of computation.
• Diff b/w Alphabet & String with example of both
• Diff b/w ~~kleen~~ kleen* & kleen+.

* Noam Chomsky classification of language.

$L \subseteq \Sigma^+$ over Σ

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

$\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \dots\}$

~~$\{a, b, aa, ab, ba, bb\}$ finite lan~~

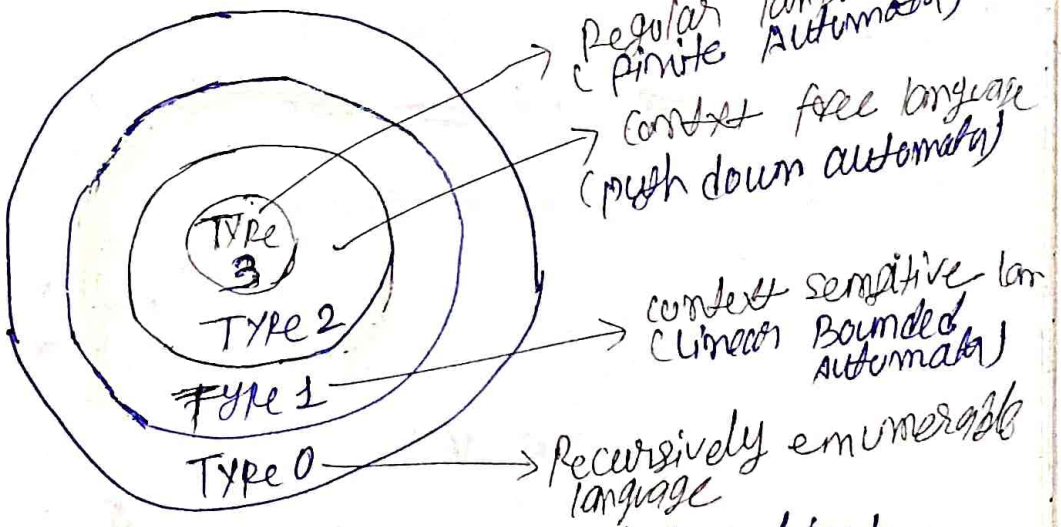
language may be finite or infinite

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

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

$L = \{ab, aabb, aaabbb. \dots\}$ infinite language

* According to Noam chomsky there are 4 types of language or Grammer.



Type 0 > Type 1 > Type 2 > Type 3

*According to Noam chomsky there are 4 types of grammer

(i) type 0 (ii) type 1 (iii) type 2 (iv) Type 3

Type 2 one

(1) Type 0: Type 0 grammer generate recursively enumerable language, it has no restriction and production are in the form of $\alpha \rightarrow \beta$, they generate the language that are recognized by Turing machine.

$\alpha \rightarrow \beta$ where α is strings of terminal and non terminal with atleast one non-terminal and α cannot be null

β is string of terminals and non-terminals

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

V → variable

A → variable, a → terminal T → terminal

SAab → AbB

(can't be null) (can be null)

Ex→ aBab → abAD aB → ∧

(2) Type 1: Type 1 generate context sensitive language (CSL). These grammar have the rules of the form

$\alpha AB \Longrightarrow \alpha \gamma \beta$ $\alpha A \beta = \alpha \gamma \beta$

where A = variable ~~non terminal~~

α, β, γ are strings of terminals and non terminals - strings α and β may be empty but γ must be non-empty.

$A \in V$

$\alpha, \beta, \gamma \in (V \cup T)$

$\alpha, \beta \in \wedge$

$\gamma \notin \wedge$

AB → CDB

↓ γ

αAB → αγβ

Type 2: Type 2 generates context free grammer. It one of the production where rules are the form of

$A \to \alpha$, $A \in V$ (variable)

$A \in V^+ \to$ A is single non terminal (not null)

$\alpha \in (V \cup T)^*$ can be null

A is single non terminal and α is combination of terminals and non-terminals.

$S \Rightarrow qAb$ {$A \to qAb$ | $S \to aaqbbb$
$\Rightarrow aaAbb$ | $S = a^2 b^3$
$\to aaqAbbb$ {$A \to A$} | $S \to a^m b^n$

Type 3: Type 3 is known as Regular grammer accepted by finite automata. It have 2 rule in the form of.

Right Linear Grammar	Left Linear Grammar
$A \to xB$	$A \to Bx$
$A \to x$	$A \to x$
where $A, B \in V$ and $x \in T$	where $A, B \in V$ and $x \in T$
Eg $\to$ $S \to ab S \mid b \to$	Eg $\to$ $S \to Sbb \mid b$

* Operation on language: Language is a subset of Kleen* over Σ. $L \subseteq \Sigma^*$ over Σ language can be finite or infinite.

(i) The complement of language: is defined with respect to Σ^*. L' or $\boxed{L^c = \Sigma^* - L}$

Ex:- $\Sigma = \{a, b\}$

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

$\Sigma^* = \{\wedge, aa, ab, ba, bb, aaa, aab, \ldots\}$

$L' = \{\wedge, a, b, aaa, aba \ldots\}$

(ii) Reverse : Reverse of a language is set of all string reversal.

$L = \{ab, baa, ba\}$

$L^R = \{ba, aab, ab\}$

(iii) concatenation : if L_1 & L_2 is a language of alphabet concatenation of L_1 & L_2 is $L_1 \cdot L_2$.

$L_1 = \{a\}$. $L_2 = \{a, ba\}$

$L_1 \times L_2 = \{a\} \times \{a, ba\}$

$= \{aa, aba\}$

(iv) union : $L_1 + L_2$

$L_1 \cup L_2$

(v) Intersection : $L_1 \cap L_2$

(vi) Kleen closure : $L = \{a\}$

$L^* = a^* = \{\wedge, a, aa, aaa \text{---} \}$

TYPE	Generate	Restriction	Recognized by	Production rules
TYPE 0	Recursively enumerable language	NO restriction	Turing Machine	$\alpha \rightarrow \beta$ $\alpha \in (VUT)^+$] α is strings of terminal & non-terminal with atleast one non-terminal, it cannot be null $\beta \in (VUT)^*$] β is strings of terminal and non-terminal can be null Ex=> $AaBb \rightarrow aAbB$
TYPE 1	context sensitive language	More than type 0	Linear Bounded automata	$\alpha AB = \alpha \gamma B$ $A \in V$ A is variable α, B, γ are strings of terminal and non terminals $\alpha \beta \gamma \in (VUT)$ α, B can be null. $\alpha B \in \wedge$ γ cannot be null. $\gamma \notin \wedge$ Ex=> $A = aAbB$
TYPE 2	context free language	More than TYPE 1	Push down automata	$A \rightarrow \alpha$ $A \in V$ A is non terminal $\alpha \in (VUT)^*$ α is strings of terminal and non terminal it can be null. Ex=> $A \rightarrow aABa$
TYPE 3	Regular language	more restriction than all.	Finite automata	Right linear Grammar: $A \rightarrow xB$, $A \rightarrow x$; $A, B \in V$ & $x \in T$; Ex=> $A \rightarrow aBab$ Left Linear grammar: $A \rightarrow Bx$, $A \rightarrow x$; $A, B \in V$ and $x \in T$; $A \rightarrow BabAa$

Unit - 2 (4M) Definition + construction FA:

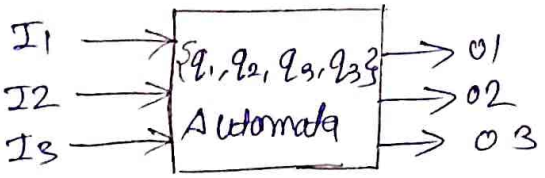
Finite Automata is also known as finite state machine

* finite automata: The term finite is used because we have finite no of state. It has fix no of state and no of input is fixed. They are good Models for computer with limited amount of memory.

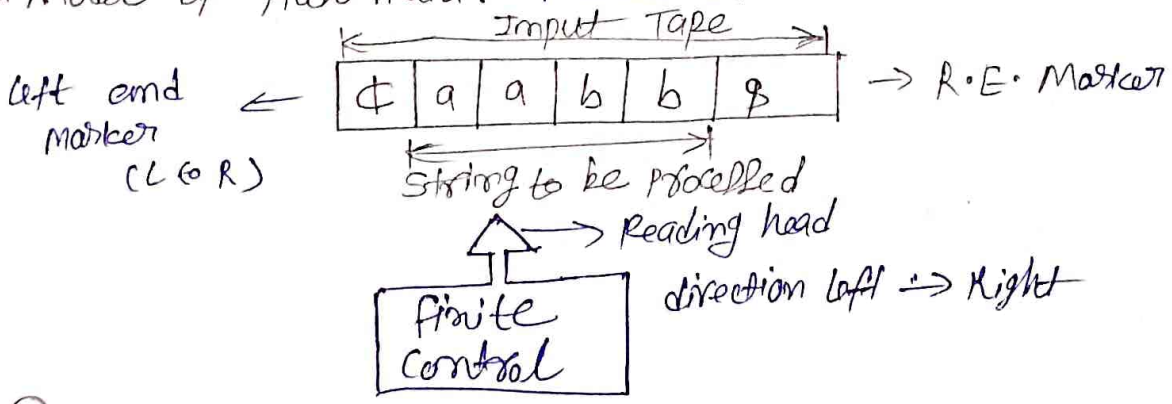
Ex: Controller for an automatic door.

An Automation is defined as system where energy material and information are transformed, transmitted and are used for performing some functionalities without the participation of human.

Automatic machine
Automatic package machine
Automatic photo printing



*Model of Automata: 4Marks



(i) Input Tape: It takes input and produce the output. Input Automata is given in the input tape. Input Tape is divided into sequences, each sequence containing a single symbol from input alphabet Σ. Input containg two end markers. Absence of end markers indicate that tape is of infinite length. Symbol b/w end markers indicate string to be processed by the automata.

(ii) Reading head: The reading head scans the input tape one symbol at a time, starting from the leftmost symbol to rightmost symbol. It moves to the next symbol on the tape after each transition.

(iii) Finite control: The finite control is the 'brain' of the automata. It determines the next state based on the current state and the input symbol read by the reading head. It consists of the transition function and a record of the current state.

* finite Automata is of 2 type
(i) DFA (ii) NDFA

* DFA (Deterministic Finite Automata): A DFA is a finite automata where for each state and input symbol there is exactly one transition to a next state. In DFA through given the current state we know what the next state will be, it has only one unique next state, it has no choice of random walk, it is simple and easy to design.

DFA = (Q, Σ, q0, F, δ)

Q → Set of all states

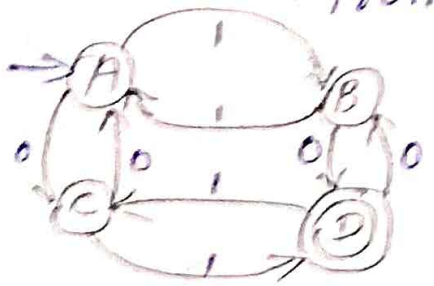
Σ → input symbols

q0 → Start state/initial state

F → Set of final states

δ → transition function from Q × Σ → Q

Eg →



Q → {A, B, C, D}

Σ → {0, 1}

q0 → A

F → {D}

δ →

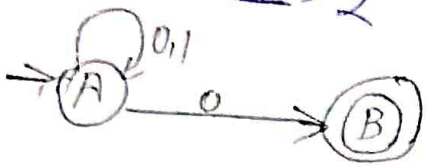
	0	1
A	C	B
B	D	A
C	A	D
D	B	C

Eg →

* Nondeterministic Finite Automata): An NDFA is a finite automata where for each state and input symbol there can be multiple possible transitions, including transitions without consuming any input (ε transitions). In NDFA, given the current state there could be multiple next states, the next state may be chosen at random. All the next states may be chosen in parallel.

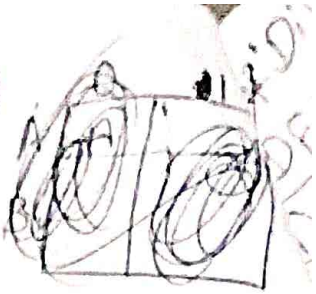
NDFA = (Q, Σ, q0, F, δ).

$Q \rightarrow$ Set of all states
$\Sigma \rightarrow$ inputs
$q_0 \rightarrow$ start state
$F \rightarrow$ Set of final states
$\delta \rightarrow Q \times \Sigma \rightarrow 2^Q$

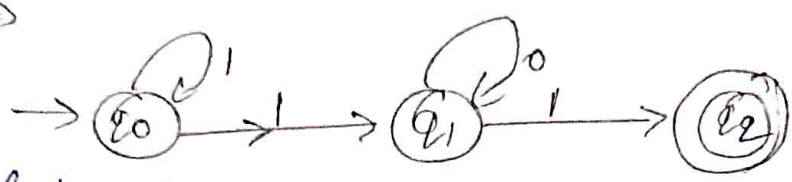


$A \times 0 \rightarrow A$
$A \times 0 \rightarrow B$
$A \times 1 \rightarrow A$
$B \times 0 \rightarrow \phi$
$B \times 1 \rightarrow \phi$

$A \xrightarrow{1} A, B, AB, \phi.$



Ex ⇒



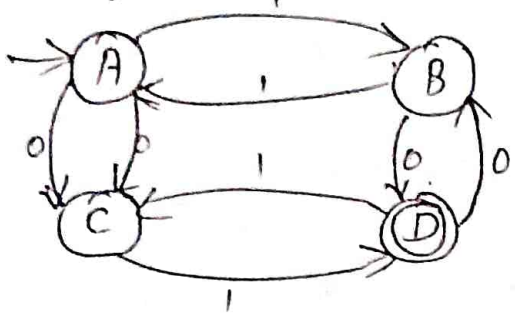
Explain diagram: here in above example set of all states are $\{q_0, q_1, q_2\}$, input and start state is q_0, final state is q_2 and δ will be, if by choosing 1 transition will be either from q_0 to q_0 or from q_0 to q_1 since multiple transition could be possible here that's why it is an NDFA after that at q_1 if we choose 0 then the transition again will be from q_1 to q_1 itself but if we choose 1 then transition will be from q_1 to q_2 at q_2 since it will not have any transition possible so even if we select 0 or 1 the next transition will be ϕ.

Diff b/w DFA and NDFA

DFA

(i) From one state, there is only single transition to next state : $Q \times E = Q$

(ii) diagram :



$Q = \{A, B, C, D\}$

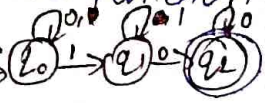
$\Sigma = \{0, 1\}$

$q_0 = A$

$F = \{D\}$

δ =

	0	1
A	C	B
B	D	A
C	A	D
D	B	C

(iii) In DFA transition function cannot be empty : 

(iv) In DFA we cannot move from one state to another state without conforming an input symbol.

(v) DFA is a deterministic means no choice in state transitions

(vi) It can be converted to an equivalent NDFA.

(vii) Generally harder to construct, as it required deterministic transitions.

(viii) Generally requires less memory

NDFA

(i) From one state, there is multiple transition to the next state : $Q \times E = 2^Q$

(ii) diagram :

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

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

$q_0 = q_0$

$F = q_2$

$\delta \rightarrow$ $q_0 \times 0 = q_0$

$q_0 \times 1 = q_1$

$q_1 \times 0 = q_1$

$q_1 \times 1 = q_2$

$q_2 \times 0 = \phi$

$q_2 \times 1 = \phi$

(iii) In NDFA transition function can be : 

(iv) In NDFA we can move from one state to another state without conforming input symbol.

NDFA is Nondeterministic means multiple choices in state transition allowed.

It can be converted to an equivalent DFA.

Easier to construct as it allows non determinism and ε-moves.

May require more memory due to multiple

*Equivalence of DfA and NDfA

DFA can simulate the behaviour of NDfA by increasing the no of states. ~~that is we can~~, Any NDfA is a more general machine without being more powerful that is both DfA and NDfA are same in power.

Q. 2m → ~~Are~~ NDfA and DfA same in power Juffify.

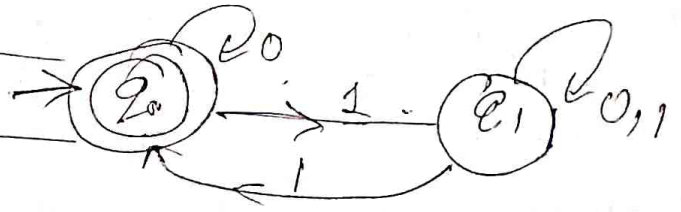
Juffify → Yes. NDfA and DfA both are same in power because NDfA can be converted into a equivalent DfA. This means that any language reangnized by an NDfA can also be rerognized by an DfA and vice versa. Since they can rerognized the same set of languages, NDfA and DfA are Confidered to have the same computational power.

*Conversion of NDfA to DfA.

Construct a non-deterministic automata equivalent to ~~machine~~ Deterministic automata.

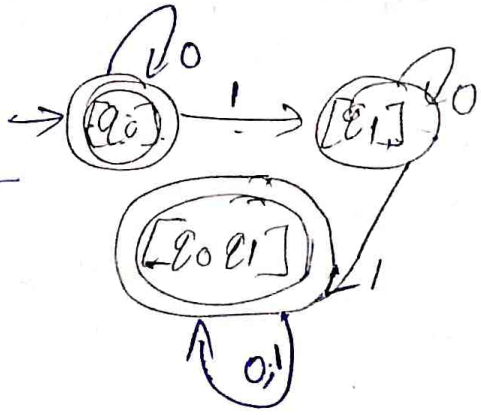
$M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_0\})$

State/Σ	0	1
→(q0)	q0	q1
q1	q1	q0,q1



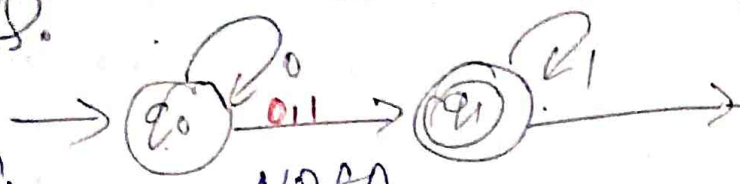
Solution:

	0	1
→[q0]	[q0]	[q1]
[q1]	[q1]	[q0q1]
[q0q1]	[q0q1]	[q0q1]



Convert NDFA to DFA

Q.



Sol^n:

NDFA

δ	0	1
→ q_0	q_0, q_1	q_1
q_1	q_2	q_1, q_2
q_2	ϕ	q_2

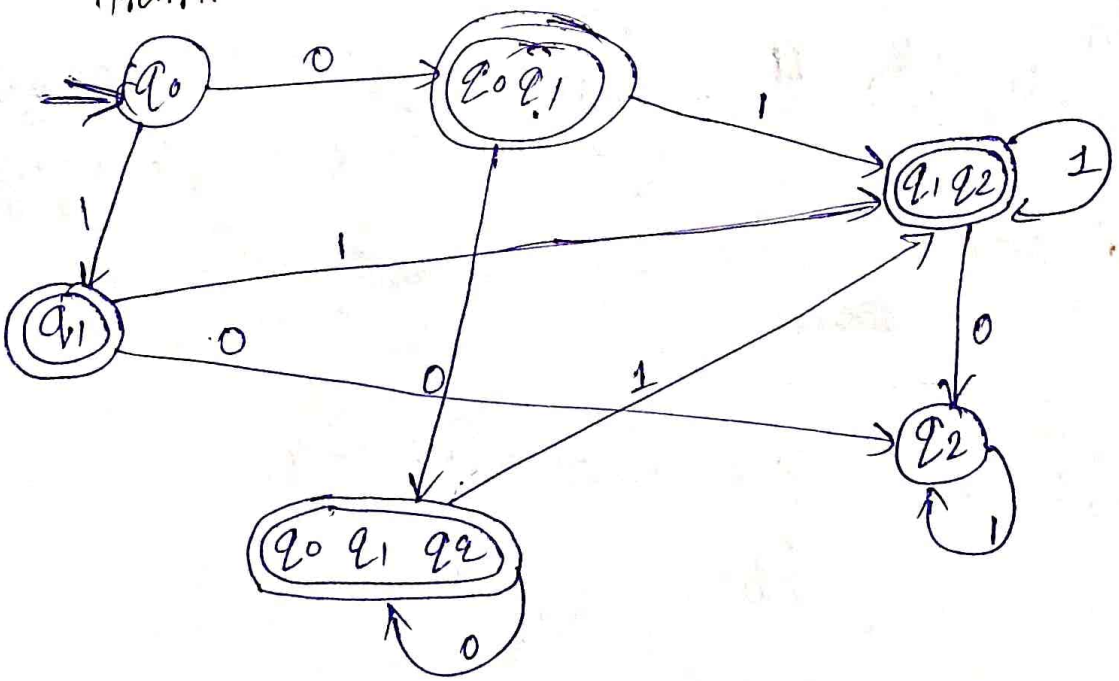
Transition table

DFA →

The transition table for converted DFA

δ	0	1
$[q_0]$	$[q_0\ q_1]$	$[q_1]$
$[q_0\ q_1]$	$[q_0, q_1, q_2]$	$[q_1, q_2]$
$[q_1]$	$[q_2]$	$[q_1\ q_2]$
$[q_0\ q_1\ q_2]$	$[q_0\ q_1\ q_2]$	$[q_1, q_2]$
$[q_1\ q_2]$	$[q_2]$	$[q_1\ q_2]$
$[q_2]$	ϕ	$[q_2]$

Transition diagram



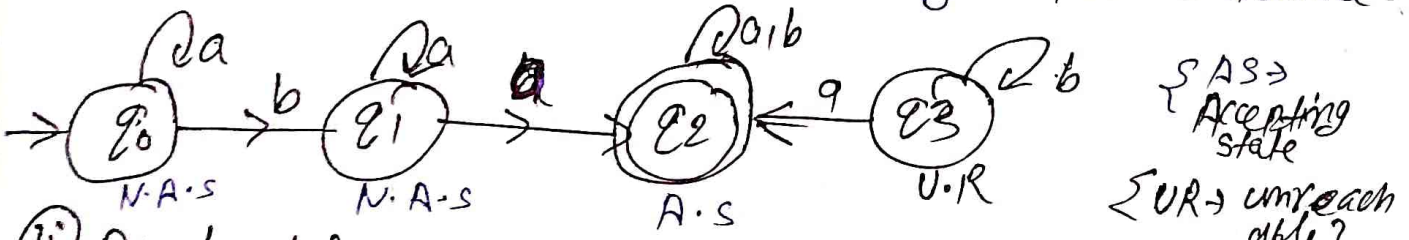
* Minimization of automata (DFA)

Minimization refers to construction of finite machine with minimum no of states which is equivalent to the given finite machine. The no of states of an automata has direct effect to the size of automata. When we minimize the automata, we detect those state whose presence or absence does not affect language accepted by the finite automata.

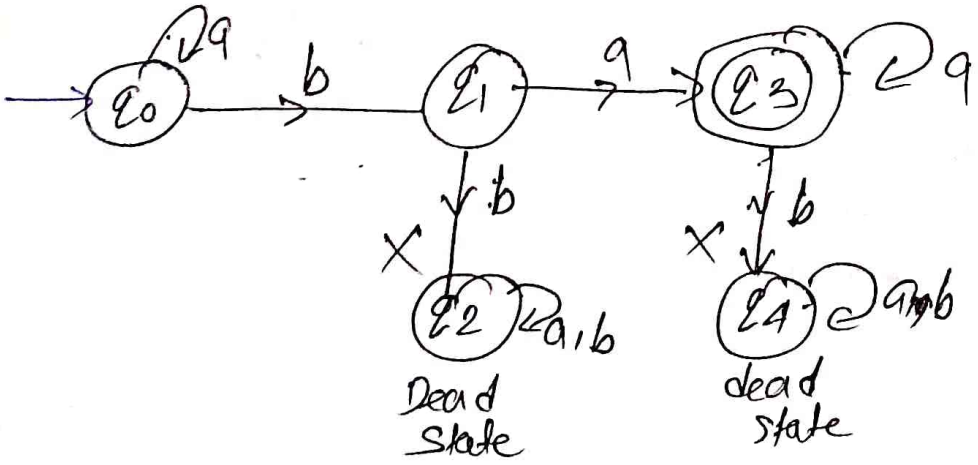
(i) unreachable state (iii) equivalent state

(ii) Dead state

(i) Unreachable state: Unreachable means a state where the machine never reaches. It is a state of automata which are not reachable from the initial state of DFA on any input sequence.

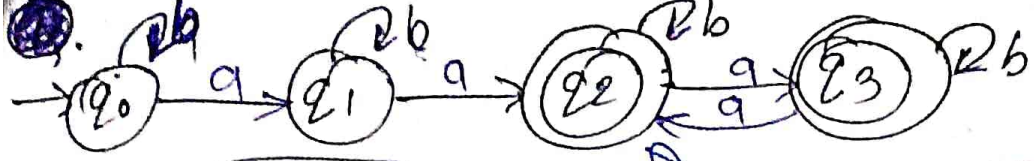


(ii) Dead state: A dead state is non-accepting state which does to itself on every input symbol.



(iii) Equivalent state: q and q' are the equivalent if both are final or both are non-final.

Q.



$f \subseteq Q$

Minimize the DFA

Q.1

	a	b
q0	q1	q0
(q1)	q2	q1
(q2)	q1	q2

π_0 (0-equivalent)

→ {q0}, {q1, q2}

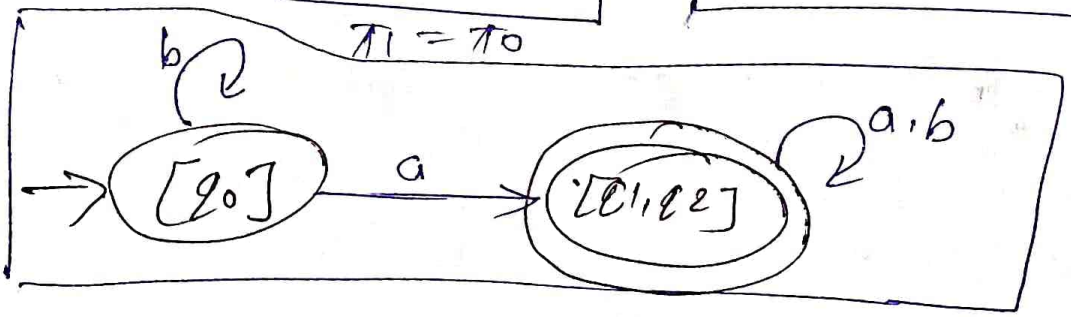
non final, final

π_1 → (π_1 → equivalent) {q0}, {q1, q2}

$\pi_{k+1} = \pi_k$

$(q, a)(q', a) \in E$

$\pi_1 = \pi_0$



Q2.

	a	b
a	b	d
b	c	e
c	b	e
d	c	e
(e)	e	e

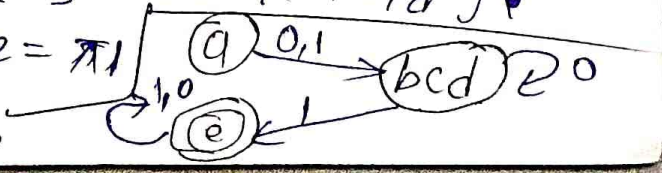
ans →

$\pi_0 = \{a, b, c, d\}, \{e\}$

$\pi_1 = \{a\}, \{b, c, d\}, \{e\}$

$\pi_2 = \{a\}, \{e\}, \{b, c, d\}$

$\pi_2 = \pi_1$



tra dia →

Q3.

	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

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

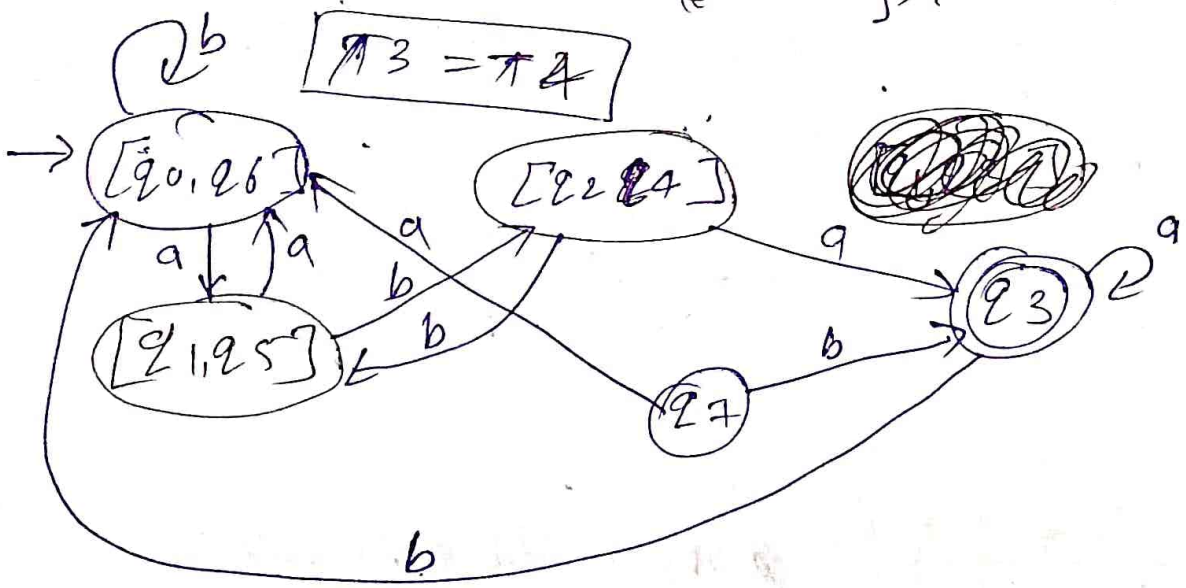
$\pi_1 = \{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4, q_7\}$

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

$\pi_3 = \{q_3\}, \{q_7\}, \{q_2, q_4\}, \{q_0, q_6\}, \{q_1, q_5\}$

$\pi_4 = \{q_3\}, \{q_7\}, \{q_2, q_4\}, \{q_0, q_6\}, \{q_1, q_5\}$

$\pi_3 = \pi_4$



Q. What is the need of Minimization

Minimization of DFA is essential for several reasons

(i) Reduced complexity: Minimization simplifies the DFA by eliminating redundant states and transitions and making it easier to understand.

(ii) Resource saving: By reducing the number of states the minimized DFA consumes less memory and power, saving resources.

(iii) less memory use: By reducing the no of states the minimized DFA takes up less memory, it is useful where memory is limited.

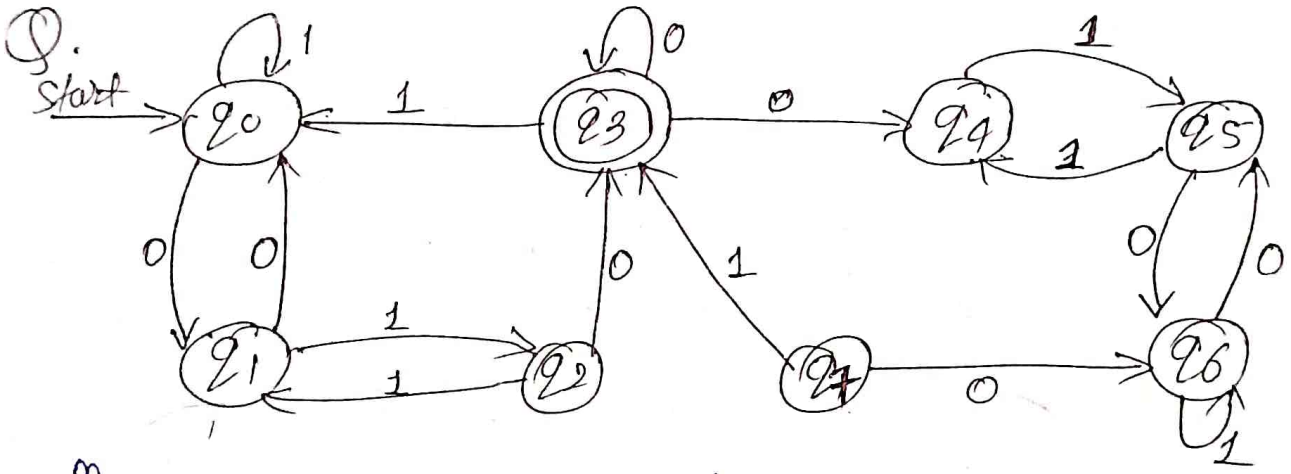
(iv) Improved Efficiency: By reducing the no of states the minimized DFA leads to faster execution and low power consumption.

(v) Optimized performance: The minimized DFA is the most efficient representation of the language reduces computational cost.

Q. What is the Need of minimization and Minimize the given DFA.

Need of DFA Minimization: Reducing / minimizing the no of states in a DFA while preserving its behavior. Improved efficiency and understanding of the DFA's functionality.
other words.
Minimization of DFA reduces computational cost and enhance the performance for tasks such as pattern matching and text processing by minimizing the no of state transitions also it makes system less complex in order to perform tasks in fast and easy manner.

Q.



Sol^n: from the given Transition diagram, let's us draw the transition table:

state	0	1
→ q0	q1	q0
q1	q0	q2
q2	q3	q1
(q3)	q3	q0
q4	q3	q5
q5	q6	q4
q6	q5	q6
q7	q6	q3

~~firstly we will eliminate unreachable state.~~

Now, let us divide the transition table into final & non-final states:

(i) final state set : {q3}

(ii) Non-final state set : {q0, q1, q2, q4, q5, q6, q7}

Now, distinguishing the states:

For that let us check the transition i/p '0' and '1' for each pair of states within the same group or diff.

If group is different then states are distinguishable & can't merged.

If group are same then state can be merged.

Let's check:

π_0(zero-equivalence) = {q0, q1, q2, q4, q5, q6, q7}, {q3}
(non-final states; final state)

For q0, q1: We observe from the table that the group are same (belongs to the same set)

For q0, q2: We observe that group are not same of q0 after consuming '0' reaches to the q1 state & q2 after consuming '0' reaches to the q3 state, since q1 & q3 belongs to different sets.

For q0, q4: We observe that the group are not same of q0 after consuming '0' reaches to the q1 state whereas, q4 after consuming '0' reaches to q3 state, since q1 & q3 belongs to different sets.

For q0, q5: We observe from the table that group are same (belongs to same set).

for q_0, q_6: we observe that group are same (belong to same set).

for q_0, q_7: we observe that group are same for i/p 0 and different for i/p '1' of q_0 after consuming '1' reaches to the q_0 state whereas q_7 after consuming '1' reaches to q_3 state. and both belongs to different sets.

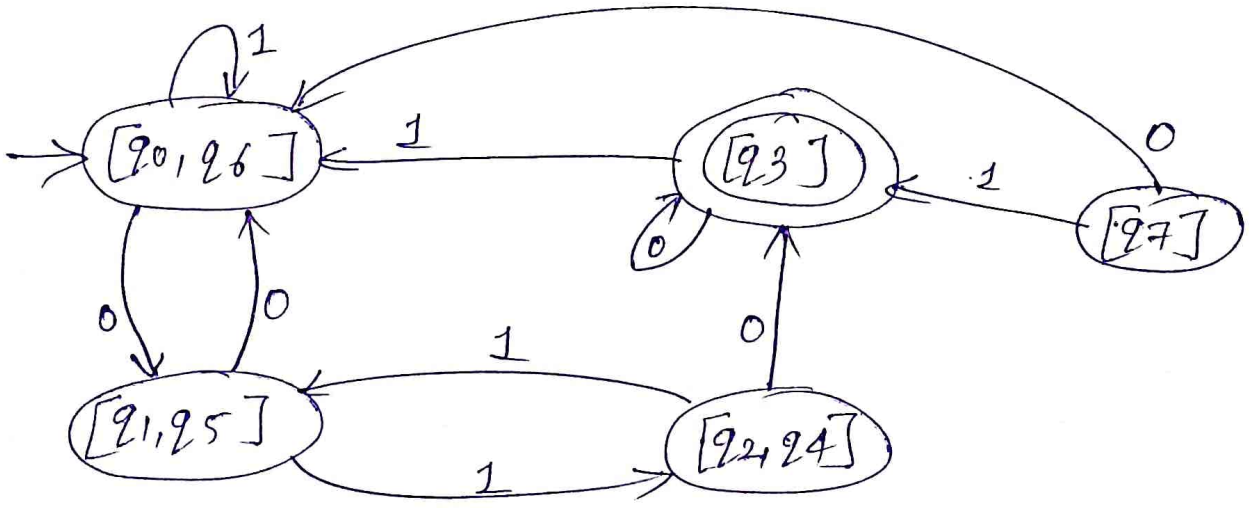
$\pi_1 = \{q_3\}, \{q_0, q_1, q_5, q_6\} \{q_2, q_4, q_7\}$

Similarly by following the above steps.

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

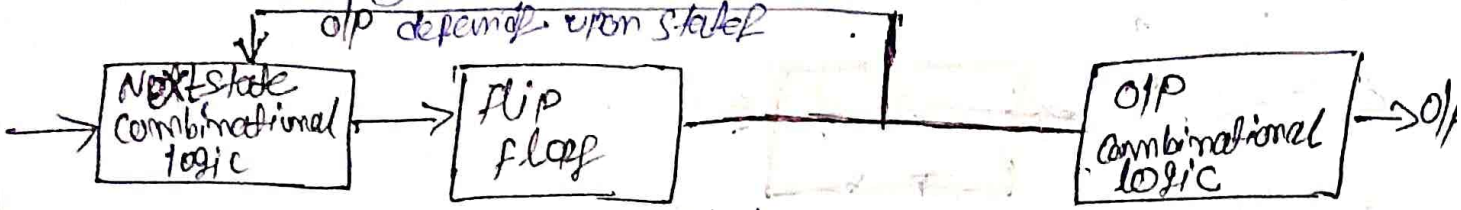
$\pi_3 = \{q_3\}, \{q_7\}, \{q_2, q_4\}, \{q_0, q_6\}, \{q_1, q_5\}$

Hence minimized, lets us draw transition diagram



* Moore and Meally Machine (F)

Moore machine is a machine where output depends only state of the machine



(circuit dia)

* Moore machine is a 6 tuple machine.

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

$Q \rightarrow$ finite set of states

$\Sigma \rightarrow$ input alphabet

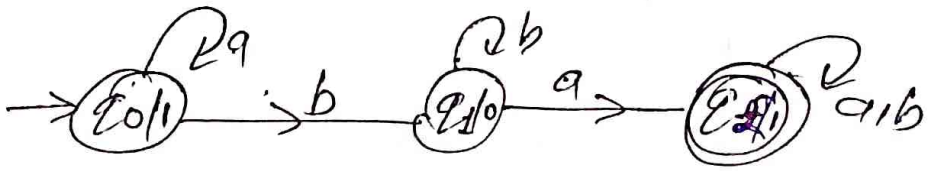
$\Delta \rightarrow$ o/p alphabet

$\delta \rightarrow Q \times \Sigma \rightarrow Q$

$\lambda \rightarrow$ o/p function $\lambda : Q \rightarrow \Delta$

$q_0 \rightarrow$

(Transition diagram)



$Q = q_0, q_1, q_2$

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

$\Sigma = a, b$

$\delta =$

$\lambda =$

$q_0 = q_0$

$\lambda = Q \times \Delta$

$q_0 \rightarrow 1$

$q_1 \rightarrow 0$

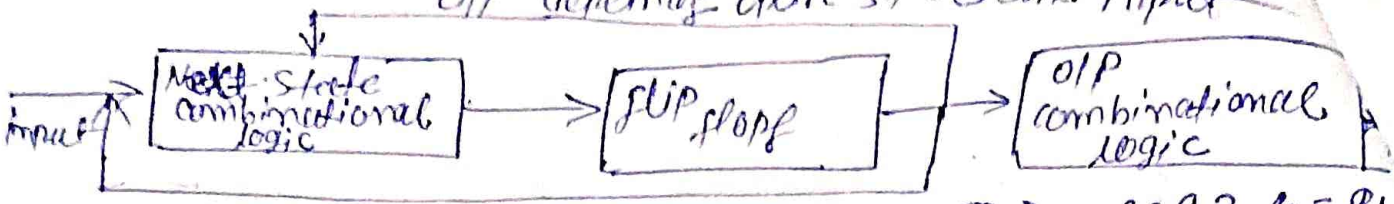
$q_2 \rightarrow 1$

$\delta =$

	a	b	Δq_2
q_0	q_0	q_1	1
q_1	q_2	q_1	0
q_2	q_2	q_2	1

Mealy machine: Mealy machine is a machine in which o/p depends upon states and input at any instance of time.

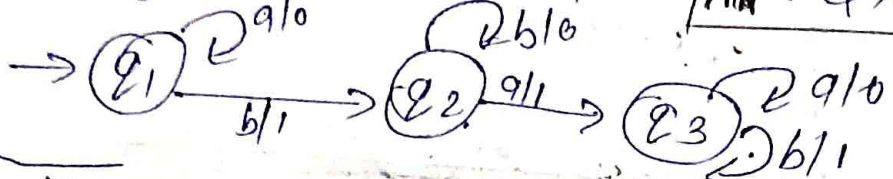
o/p depends upon state and input



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

$Q \to q_1, q_2, q_3$ $q_0 = q_1$
$\Sigma = \{a, b\}$
$\Delta = \{0, 1\}$

Mealy o/p → state + input $\lambda : Q \times \Sigma \to \Delta$



Moc $\delta : Q \times \Sigma \to Q$

λ = o/p fun $Q \to \Delta$

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

curr state	inp	output	inp	o/p
	0	1	1	1
q1	a	0	0	1
q2	a	1	b	0
q3	a	0	b	1

any →

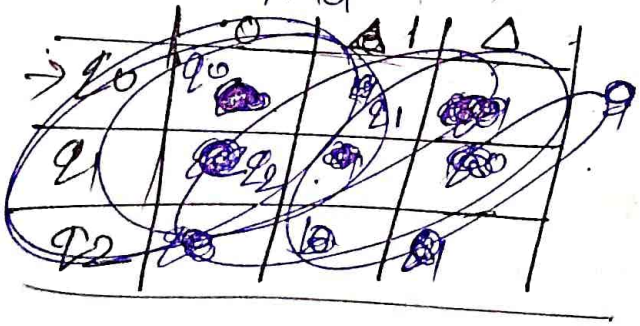
→ q_0

↓

a

inversion :

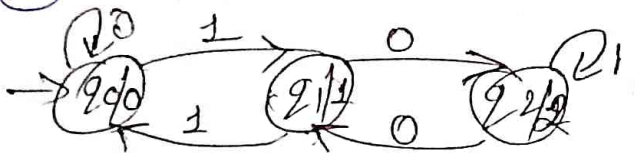
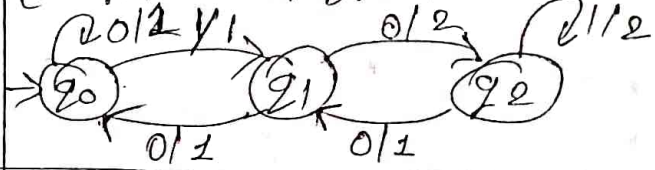
o/p → aaabaa



curr state	0	Δ	1	Δ
q_0	q_0	a	q_1	b
q_1	0	b	q_0	a
q_2	q_0	a	q_1	b

Difference between Moore and Mealy

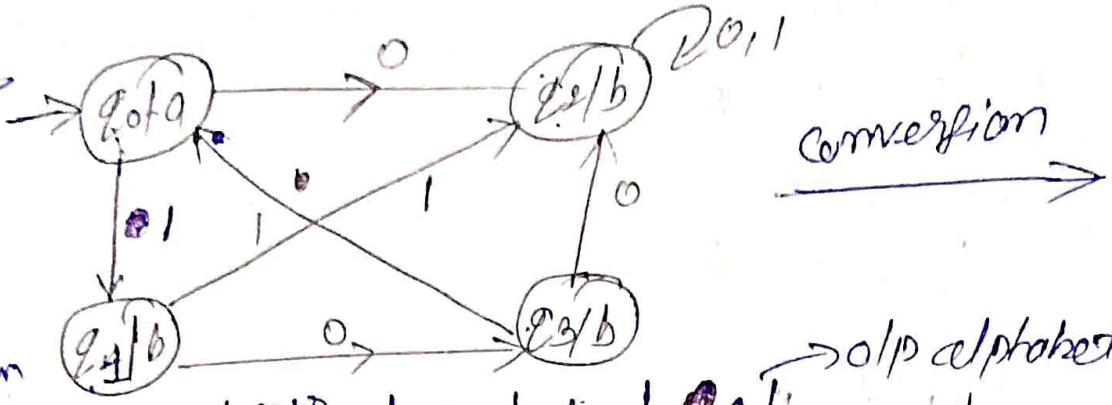
(12 marks with a example / remember each one example question.)

Moore	Mealy
(i) In moore output depends only on present state of the machine.	In mealy machine output depends upon states and present input.
(ii) Input string length (n) → output string length (n+1)	Input string length (n) → output length is also (n)
(iii) $\lambda : Q \rightarrow \Delta$	$\delta : Q \times \Sigma \rightarrow \Delta$
(iv) transition diagram 	transition diagram 
(v) output is synchronous with clock.	output is asynchronous.
(vi) if input changes, output does not change	If input changes, output also changes.
(vii) output is placed on state	output is placed on transitions.
(viii) Easy to design	Difficult to design.
(ix)	

* Conversion from Moore to Mealy Machine

MST

am



Conversion

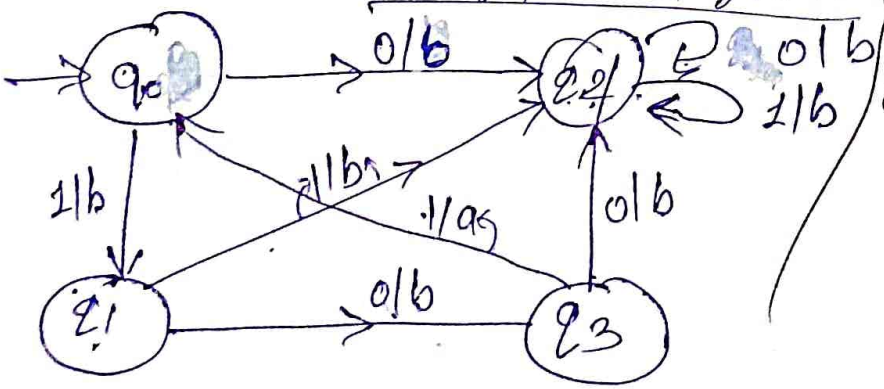
transition Table

curr state	0	1	Δ → o/p alphabet
→ q0	q2	q1	a
q1	q3	q2	b
q2	q2	q2	b
q3	q2	q0	b

Mealy ↓↓

curr state	0	Δ	1	Δ
→ q0	q2	b	q1	b
q1	q3	b	q2	b
q2	q2	b	q2	b
q3	q2	b	q0	a

Transition diagram



theory:

The output of the states of q0, q1, q2, q3 are a, b, b, b. In mealy machine the next ~~state~~ State is similar to a Moore machine for all state and input. →So, assigning output of a mealy machine for all inputs corresponding to their next state. for example the next state of q0 is q2 on input symbol 0 so the output of state q0 on input 0 is 'b'.

~~the output of state q0~~

Q.

	0	1	Δ 0/1
→q0	q1	q3	a
q1	q3	q1	b
q2	q1	q3	a
q3	q2	q1	b

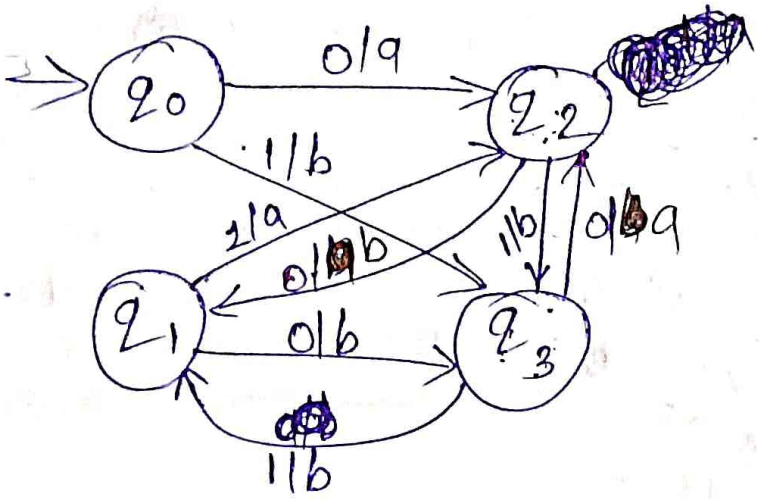
conversion from moore to mealy.

draw transition diagram

mealy

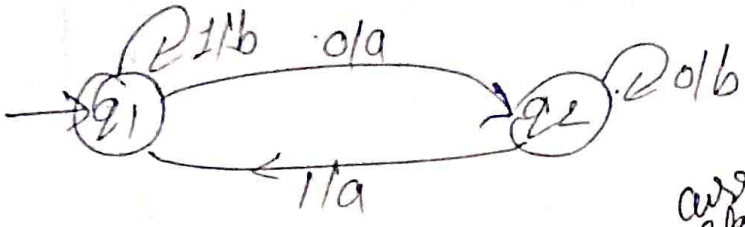
	0	Δ	1	Δ
→q0	q2	a	q3	b
q1	q3	b	q2	a
q2	q1	b	q3	b
q3	q2	a	q1	b

Transition diagram



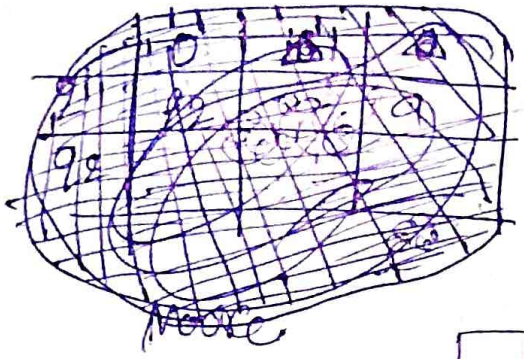
* Mealy to Moore Machine

convert the following Mealy to Moore



transition table

curr state	0	Δ	1	Δ
→ q_1	q_2	a	q_1	b
q_2	q_2	b	q_1	a

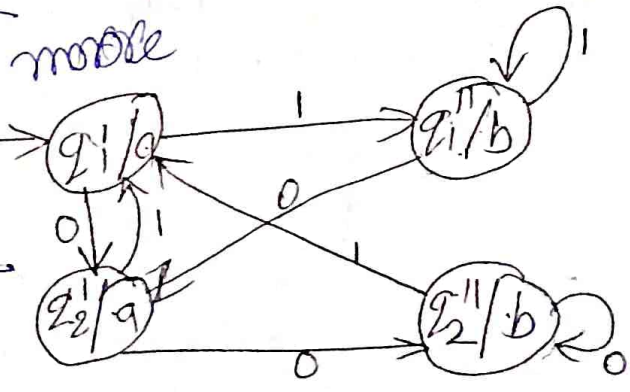


$q_1' \to a$ $q_1'' \to b$

$q_2' \to a$ $q_2'' \to b$

curr state	0	Δ	1	Δ
→ q_1'	q_2'	a	q_1''	b
→ q_1''	q_2''	a	q_1''	b
q_2'	q_2''	b	q_1'	a
q_2''	q_2''	b	q_1'	a

curr state	0	1	Δ
→ q_1'	q_2'	q_1''	a
→ q_1''	q_2'	q_1''	b
q_2'	q_2''	q_1'	a
q_2''	q_2''	q_1'	b



Q.

	State 0	Δ	State 1	Δ
q_0	q_1	a	q_3	b
q_1	q_2	b	q_0	a
q_2	q_3	a	q_1	b
q_3	q_0	b	q_2	a

$q_0' \to a$, $q_0'' \to b$

$q_1' \to a$, $q_1'' \to b$

$q_2' \to a$, $q_2'' \to b$

$q_3' \to a$, $q_3'' \to b$

(0)

Curr State	State	Δ	1	Δ
→ q_0'	q_1'	a	q_3''	b
→ q_0''	q_1'	a	q_3''	b
q_1'	q_2''	b	q_0'	a
q_1''	q_2''	b	q_0'	a
q_2'	q_3'	a	q_1''	b
q_2''	q_3'	a	q_1''	b
q_3'	q_0''	b	q_2'	a
q_3''	q_0''	b	q_2'	a

Curr state	0	Δ	1	Δ
→ q_0'	q_1'		q_3''	a
→ q_0''	q_1'		q_3''	b
q_1'	q_2''		q_0'	a
q_1''	q_2''		q_0'	b
q_2'	q_3'		q_1''	a
q_2''	q_3'		q_1''	b
q_3'	q_0''		q_2'	a
q_3''	q_0''	b	q_2'	b

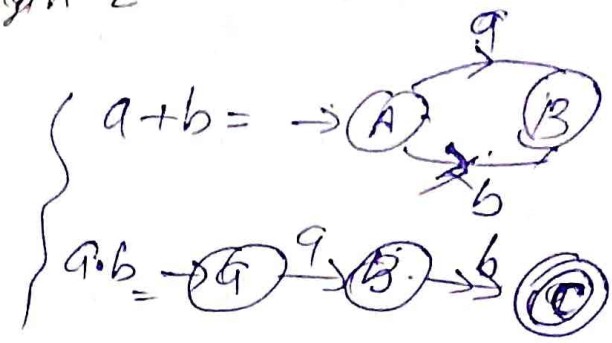
transition diagram बनाओ

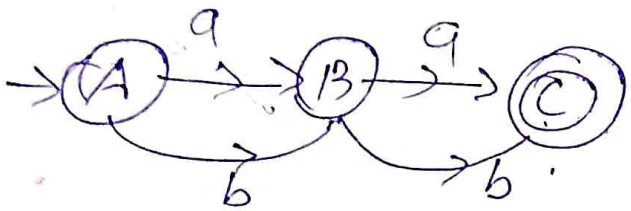
* Designing of finite Automata

1Q. Design a automata or a machine in which over length of string is length 2.

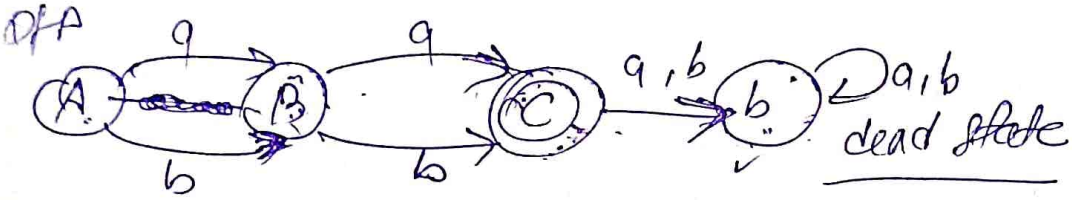
$\Sigma = \{a, b\}$, $|w| = 2$

language = {aa, ab, ba, bb}

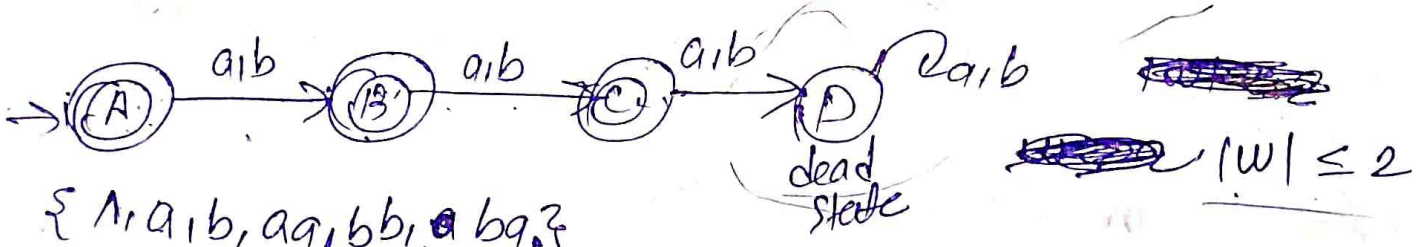




NDFA



Q. Design a automata over length of $|w| \geq 2$ and $|w| \leq 2$ $L = \{aa, ab, ba, bb, aaa, \ldots bbb \ldots\}$



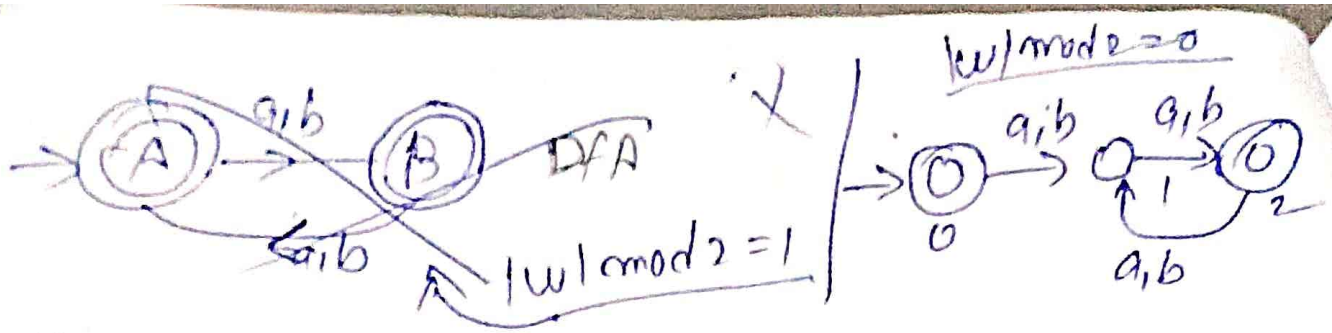
{Λ, a, b, aa, bb, ba, ab}

Q Design a automata machine $|w| \bmod 2 = 0$ leng of string is zero.

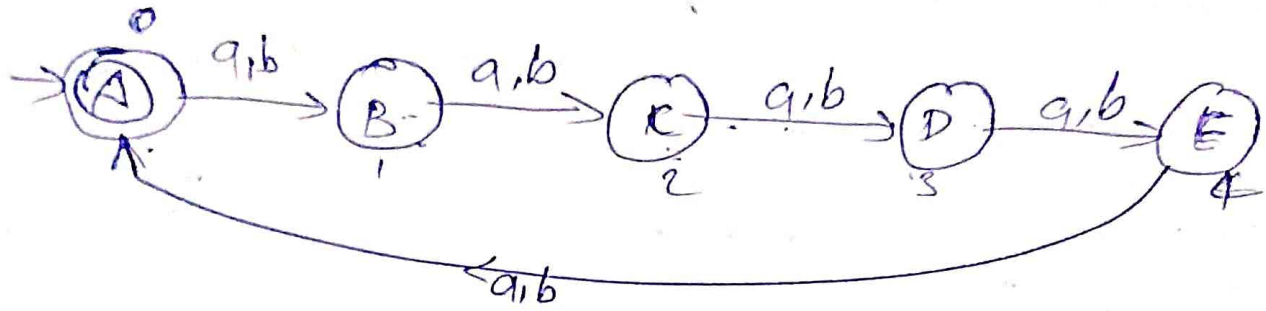
$w \cdot 2 = 0$

{2, 4, 6, 8, 10}

$L = \{\Lambda, aa, bb, baba, abbbab\}$ ....



Q. Design a Automata length $|w| \% 5 = 0$



Q. Design a Automata length $|w| \bmod 4 = 2$

$|w| \% 4 = 0$ ~~{0, 4, 8, ...}~~

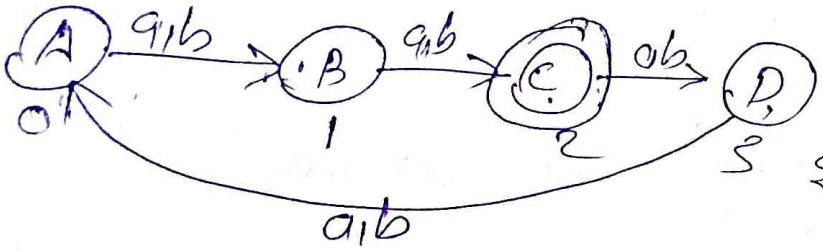
$w = 4n + 2$ | $n = 0, 1, 2, 3, \ldots$

$w = 2$

$w = 6$

$w = 10$

$w = 14$

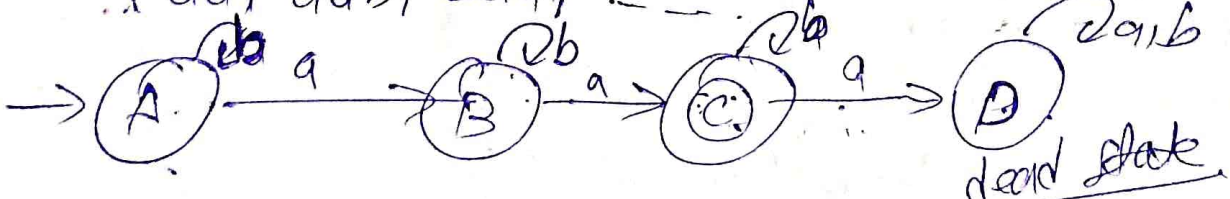


{aa, abaab, abcaaabb, abcaaaabb ...}

Q Design a automata in which no of a's are 2.

ans → ~~n(w) = 2~~ $n_a(w) = 2$

= {aa, aab, baa, ...}

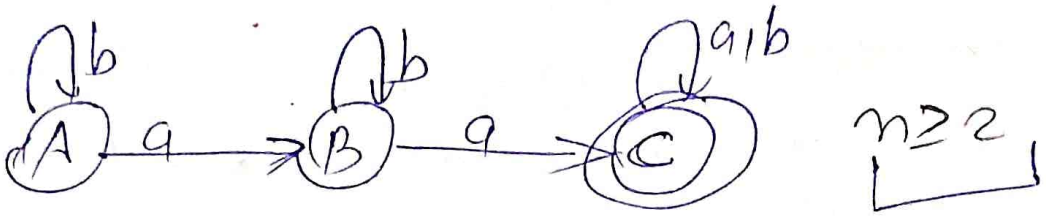


Q1 no of a's @ $n_a(w) \geq 2$ $n \leq 2$

Q2

Q3 - no of a's mod 2 = 0

{aa, aaa, bbbabbabb, abaabaab ... }

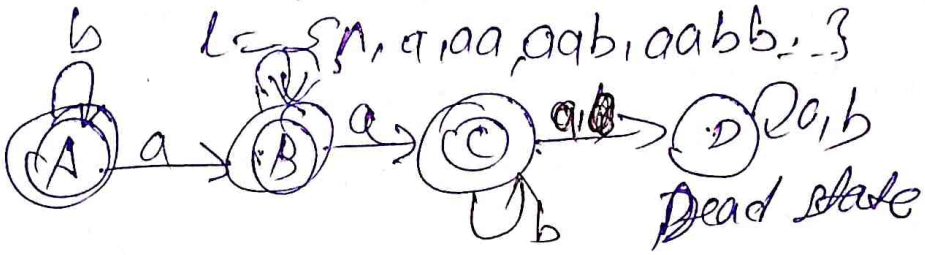


$n \geq 2$

(b) $n \leq 2$ a's

~~{ ∧, a, ab, aa, ba, bb }~~

L = {∧, a, aa, aab, aabb ... }



(c) no of a's mod 2 = 0

$a \% 2 = 0$

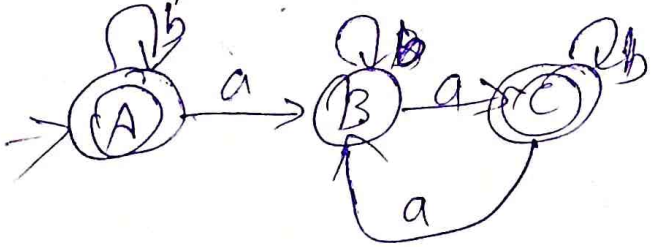
$a = 2n + b$

$b = 0$

$n = 2$

$n = 0$

lan = { ∧, aa, aaaa, aaaaaa ... }



bbabab

Unit -3
Regular Expression

Regular expression: Regular expressions are the algebric description, They are used by many text editor to search a block of text for a certain pattern. They are the representation of language accepted by finite automa.

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

$\phi = \{\phi\}$

* write a expression for the language in which length of string is exactly 2.

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

$R = ab + ba + aa + bb$

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

$= (a+b)(a+b) \Rightarrow$

Q2. $L = \{ab, ba, bb, aa, \ldots\}$

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

$|w| \geq 2$

Q3 $|w| \leq 2$

$\{0, 1, 2\}$

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

$R = \{\lambda + a + b + aa + ab + bb + ba\}$

$R = (\lambda + a + b)(\lambda + a + b)$

Q4. $|w| \bmod 2 = 0$

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

$(a+b)^{2n+1}$ → odd =

Q5. Leng of the string is divisible by 3

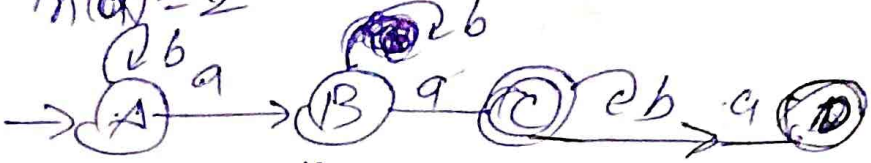
$|w| \% 3 = 0$ $\quad (a+b)^{3n}$

$((a+b)(a+b)(a+b))^*$ or $(a+b)^{3n}$

Q6 $|w| \bmod 3 = 2$

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

Q7 $n(a) = 2$

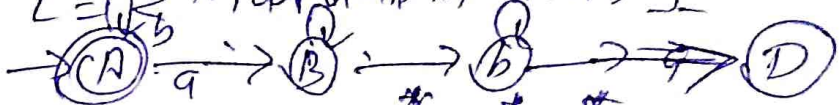


aab^*

$b^*ab^*ab^*$

Q7. $n(a) \geq 2$ $\quad b^*ab^*a(a+b)^*$

Q8. $n(a) \leq 2$ $\quad L = \{\wedge, a, aa, ab, ba, bb, aba, bab\}$



$b^*ab^*ab^*$

Q9. Set of all ~~the~~ string start with a.

Q10 Set of all the string end with a.

Q11. Set of all string containing a.

Q12. " " " starting and ending with diff symbol.

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

Wrong

Q12 " " " starting and ending with same symbol.

Q13. Identities of

$\phi + R = R$

$\phi \cdot R = R$

$\Lambda \cdot R = R$

$\phi^* = \Lambda$

all string having a single b. (a^*ba^*)

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

4 times b as substring.

all string end with ab.

" start with bq

all string in which single a followed by any no of b and single b followed by any no of a's.

Set of all strings over the alphabet start and end with 0.

Generate the expression.

* Arden's theorem

: Let P, Q, R be the three regular expression given by finite automata then the equation.

$R = Q + RP$ has unique solution given by

$R = QP^*$

(2M) proof

$R = Q + RP$ → $R = QP^*$ — (1)

put $R = QP^*$

$R = Q + (QP^*)P$

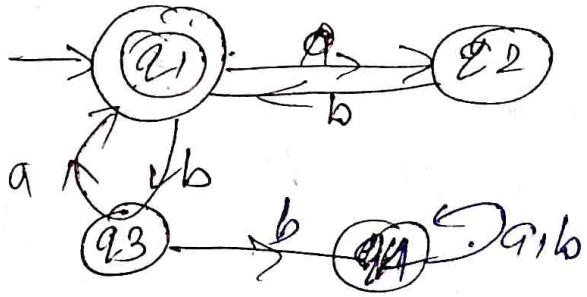
$R = Q(\Lambda + P^*P)$ [$\because PP^* = P^+$]

→ since $(\Lambda + RR^*) = R^*$

$R = Q(\Lambda + P^*P)$

$R = QP^*$

Q convert into regular expression:



We can apply the direct method here, since the graph does not contain any null move & there is only one initial state.

$q_1 = q_2b + q_3a + \Lambda$ — (1)

$q_2 = q_1a$ — (2)

$q_3 = q_1b$ — (3)

$q_4 = q_4a + q_4b + q_3b$ — (4)

put equation (2) and eq^n (3) in eq (1)

$q_1 = q_1ab + q_1ba + \wedge$ ——— (5)

since: $R = RP + Q$

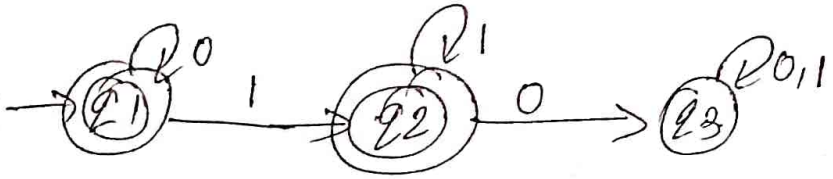
$R = QP^*$

$q_1 = q_1(ab + ba)^* + \wedge$

$q_1 = \wedge(ab+ba)^*$

$q_1 = (ab+ba)^*$ any

Q_2.



$q_1 = q_10 + \wedge 1$ ——— (1)

$q_2 = q_11 + q_21$ ——— (2)

$q_3 = q_20 + q_30 + q_31$ ——— (3)

from equation (1)

$q_1 = q_10 + \wedge 1$ ——— since $R = RP + Q$, $R = QP^*$

$q_1 = \wedge 0^*$

$q_1 = 0^*$ ——— (4)

$q_2 = q_11 + q_21$ from $R = RP + Q$, $R = QP^*$

$q_2 = q_21 + 0^*1$

$q_2 = 0^*1 + q_21$

$R = SP^* \Rightarrow QP^* \Rightarrow (0^*1)(1)^* \Rightarrow 0^*1^*1 = 0^*1^*$

$q_1 + q_2$

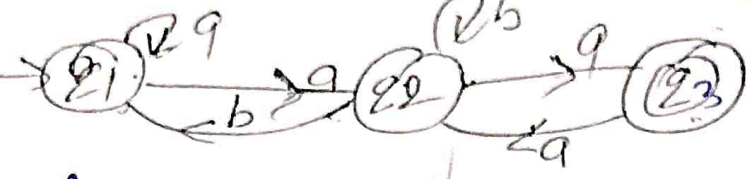
$0^* + 0^*1^* \Rightarrow 0^*(1 + 1^*)$

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

$\Rightarrow 0^* 1^*$

answer

Q3.



$a_{ij} = (a + a(b+aa)^*b)^* a(b+aa)^*a$

Soln:

$q_1 = q_1a + q_2b + \wedge$ — (1)

$q_2 = q_2b + q_1a + q_3a$ — (2)

$q_3 = q_2a$ — (3)

put eqn (3) in eqn (2)

~~$q_2 = q_2b + q_1a + q_3aa$ — (4)~~

$q_2 = q_2b + q_1a + q_2aa$ — (4)

$q_2 = q_2(b+aa) + q_1a$ —

$R \quad R \quad P \quad Q$

$\therefore R = R + RQ$

$R = QP^*$

$\boxed{q_2 = q_1a(b+aa)^*}$

Substituting q_2 in q_1

$q_1 = q_1a + q_1a(b+aa)^*b + \wedge$

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

$q_1 = q_1(a + a(b+aa)^*b) + \Lambda$ $\quad \because R = R + Qp \Rightarrow Qp^*$

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

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

put value of q_1

we have to find

$q_3 = q_2 a$

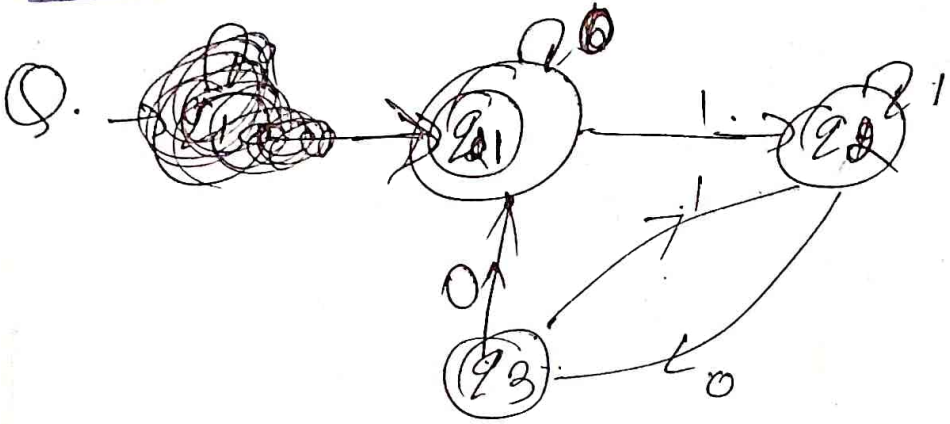
$q_3 = [q_1 a(b+aa)^*]a$

↑ we need q_1

$q_3 = [q_1 a(b+aa)^*]a$

$q_3 = [(a + a(b+aa)^*b)^* a(b+aa)^*]a$

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



$q_1 = q_{10} + q_{30} + \Lambda$ ——— (1)

$q_2 = q_{21} + q_{11} + q_{31}$ ——— (2)

$q_3 = q_{20}$ ——— (3)

~~put eq 3 in q1~~ from above question q_1 is final state so vision is clear we have to find q_1 but in order to find q_1 we need q_3 then we find q_3 first but in order to find q_3 we need q_2 so we will find q_2 first.

$q_2 = q_{21} + q_{11} + q_{31}$ from eq (2) put eq (3) in eq (2)

$q_2 = q_{21} + q_{11} + (q_{20})1$

$q_2 = q_{11} + q_2(01 + 01)$

~~$q_2 = q_2(1+01) + q$~~

$q_2 = q_{11} + q_2(1+01)$

$Q + R(P)$

Since, $R = Q + RQ$
then $R = QP^*$

$\boxed{q_2 = q_{11}(1+01)^*}$ — (4)

Now, we can find q_3

$q_3 = q_{20}$. put eq (4) in q_3

$\boxed{q_3 = q_{11}(1+01)^*0}$ — (5)

Now we can find q_1

$q_1 = q_{10} + q_{30} + \lambda$ put eq (5) in q_1

$q_1 = q_{10} + q_{11}(1+01)^*00 + \lambda$

$q_1 = q_1(0 + 1(1+01)^*00) + \lambda$

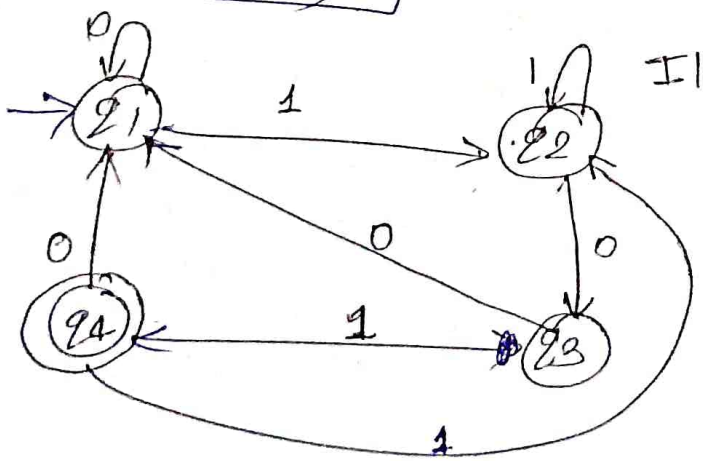
$q_1 = \lambda + q_1(0+1(1+01)^*00)$ Since, $R = Q + RP$
then $R = QP^*$

$Q + R(P)$

$q_1 = \lambda(0+1(1+01)^*00)^*$

$\boxed{q_1 = (0+1(1+01)^*00)^*}$

PYQ



Illustrate Ardemi & theorem.

Convert into Regular Expression using Arden Thm.

Sol^n: There is only one initial state, and there are no $\wedge$-moves, so, we form the equations corresponding to q_1, q_2, q_3, q_4

$$q_1 = q_10 + q_30 + q_40 + \wedge \quad \text{——} (1)$$

$$q_2 = q_11 + q_21 + q_41 \quad \text{——} (2)$$

$$q_3 = q_20 \quad \text{——} (3)$$

$$q_4 = q_31 \quad \text{——} (4)$$

Now, $q_4 = q_31 \Rightarrow (q_20)1 \Rightarrow q_201$ ——

$$\boxed{q_4 = q_201} \quad \text{——} (5)$$

Put eq (5) in eq (2)

$$q_2 = q_11 + q_21 + q_2011$$

$$q_2 = q_11 + q_2(1 + 011) \qquad R = Q + RP \rightarrow QP^*$$

$$\boxed{q_2 = q_11(1 + 011)^*} \quad \text{——} (6)$$

$$q_2 = q_1(1(1 + 011)^*) \quad \text{——} (6)$$

from eq (1)

$$q_1 = q_10 + q_200 + q_2010 + \wedge$$

$$q_1 = q_10 + q_2(00 + 010) + \wedge \qquad \text{put } q_2 \text{ means eq (6) here}$$

$$q_1 = q_10 + q_1(1(1 + 011)^*)(00 + 010) + \wedge$$

$$q_1 = q_1(0 + 1(1 + 011)^*)(00 + 010) + \wedge$$

$$q_1 = \wedge + q_1(0 + 1(1 + 011)^*)(00 + 010) \qquad \because R = Q + RP = QP^*$$

$$\boxed{q_1 = \wedge(0 + 1(1 + 011)^*(00 + 010))^*} \quad \text{——} (7)$$

from eqn (5)

$q_4 = q_2 01$ put eqn 6

$q_4 = q_1(1(1+011)^*01$ again put eq (7)

$$q_4 = (\varnothing + 1(1+011)^*(00+010))^*(1(1+011)^*01$$

ans

Ardemis theorem : Ardemis theorem state that if P and Q are two regular expressions over Σ ; and if P does not contain $\in/\Lambda$, then the following equation in R given by $R = Q + RP$ has a unique solution i.e. $R = Qp^*$. That means whenever we get any equation in the form of $R = Q + RP$, then we can directly replace it with $R = Qp^*$.

$R = \varnothing + RP$

$R = Qp^*$

* pumping lemma : The term pumping lemma is made up of two word pumping & second is lemma, the word pumping means to generate many input strings by putting a symbol in an input string again and again. the word lemma refers to an intermediate theorem in a proof : Pumping lemma is used to proof that the given language is not regular.

A language is regular if language is accepted by finite automata
A regular grammer can be accepted

Q. How can we say a language is regular?
A language is regular if language is accepted by finite automata. A regular ~~grammar~~ language can be constructed ~~or regular expression exists.~~ using regular grammar and can be described by a regular expression.

(answer)

Pumping lemma theorem:

~~Step~~ 1: To prove that certain sets are not regular
Assume that L is regular, let's small n be the no of states in a corresponding finite automata.

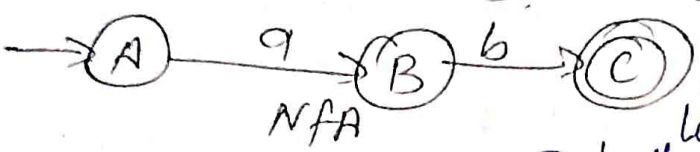
Step 2: choose a string w such that $|w| \geq n$
choose pumping lemma to write $w = xyz$ such that
$|xy| \leq n$, $|y| > 0$

Step 3: find a suitable integer i such that $xy^i z$ does not belong to L. ($xy^i z \notin L$)

Question: show that $L = \{a^p b^p \mid p \geq 1\}$ is not a regular language.

To prove that certain sets are not regular Assume that
Step 1: ~~Suppose~~ L is regular, let n be the no of states in a finite automata accepting L.

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



NfA

$n = 3$

Step 2: choose a string w such that $|w| \geq n$, choose pumping lemma to write $w = xyz$ such that $|xy| \leq n$ & $|y| > 0$

$w = aabb$

$|w| \geq n \quad |w| \geq 3$

$w = xyz$

$|xy| \leq n$ & $|y| > 0$

$x = a$

$y = a$

$z = bb$

Step 3: find a suitable integer i, such that $xy^i z$ does not belong to L.

$i = 0 \quad xy^i z$

$\Rightarrow xy^0 z$

$\rightarrow xz$

$= abb \notin L$

since $abb \notin L$

am Mf

Q2. $L = \{ a^p \mid p \text{ is prime} \}$

Sol: $L = \{ a, a^3, a^5, a^7, a^{11} \ldots \}$

$L = \{ aa, aaa, aaaaa \ldots \}$ $\quad |w| \geq n$

$\Rightarrow aa \quad n = 3$

$w = aaaaa$

$w = xyz$

$|xy| \leq n$

$|y| > 0$

$x = aa$

$y = a$

$z = aa$

$i = 0$ $\quad xy^i z \Rightarrow aa\overset{0}{a}aa \rightarrow aaaa$ length 4 which are not prime no.

$aaaa \notin L$

Q3. Show that $L = \{ a^{p^2} / p \geq 1 \}$ H.W

Q4. Show that $\Rightarrow$

$L = \{ ww / w \in \{a,b\}^* \}$ H.W

Q3. $L = \{ a, aaaa, aaaaaaaaa \text{ ------} \}$

$n = 2$

See the first term a, its length is 1 so you have to add (+1) to it, it will be your n

$n = 1+1$

$n = 2$

$w = aaaa$

$w = xyz$

$x = aa$, $y = a$, $z = aa$

$i = 0 \quad xy^i z \Rightarrow aa\overset{0}{a}aa \Rightarrow aaa = a^3 \notin L$

Q4. Show that $L = \{ ww / w \in \{a,b\}^* \}$

Step 1: Let $w = aba$ $\quad L = \{ abaaba \}$

$n = 4$

$ww = abaaba$

Choose $w = xyz$

$x = ab$ $y = a$, $z = aba$

$|xy| \leq n$

$w = xy^i z$

Step 3: (a) $i = 0$

$w = ab a^0 aba \Rightarrow$

$w = ababa$

Since $ababa \notin L$

So, this is not regular lang.

Q4. Show that $L = \{\gamma\gamma / \gamma \in (a,b)^*\}$

any $\Rightarrow$

where $\gamma \in \{a, b, ab, aa, bb, ba, \text{----}\}$

$\gamma\gamma = \{aa, bb, abab, aaaa, bbbb, baba, \text{----}\}$

$\downarrow$

$n = n+1$

$\boxed{n = 3}$

$w = \gamma\gamma = abab$

$w = xyz \quad |w| \geq n$

$x = a, y = b, z = ab$

$|xy| \leq n \quad |y| > 0$

$w = xy^i z \rightarrow$

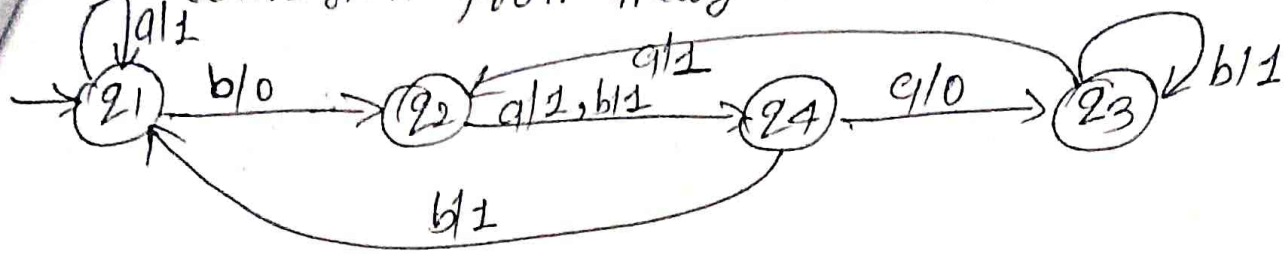
$i = 0$

$xy^0 z \Rightarrow xz \rightarrow aab$

$w = aab \notin L$

lang is not regular

(1). Conversion from Mealy machine to Moore machine



Current State	input a	output	input b	output
→ q1	q1	1	q2	0
q2	q4	1	q4	1
q3	q2	1	q3	1
q4	q3	0	q1	1

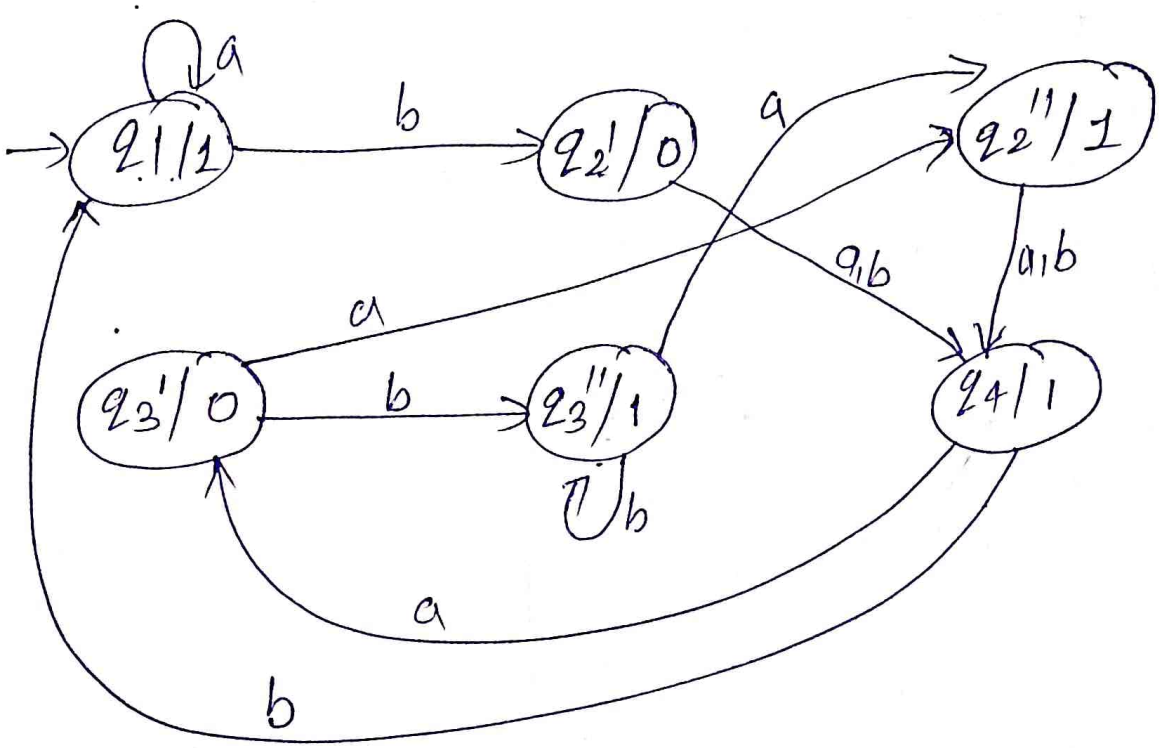
$q_2' \Rightarrow 0$, $q_2'' \rightarrow 1$

$q_3' \rightarrow 0$, $q_3'' \rightarrow 1$

curr state	a	o/p	b	o/p
→ q1	q1	1	q_2'	0
q_2'	q4	1	q4	1
q_2''	q4	1	q4	1
q_3'	q_2''	1	q_3''	1
q_3''	q_2''	1	q_3''	1
q4	q_3'	0	q1	1

moore table

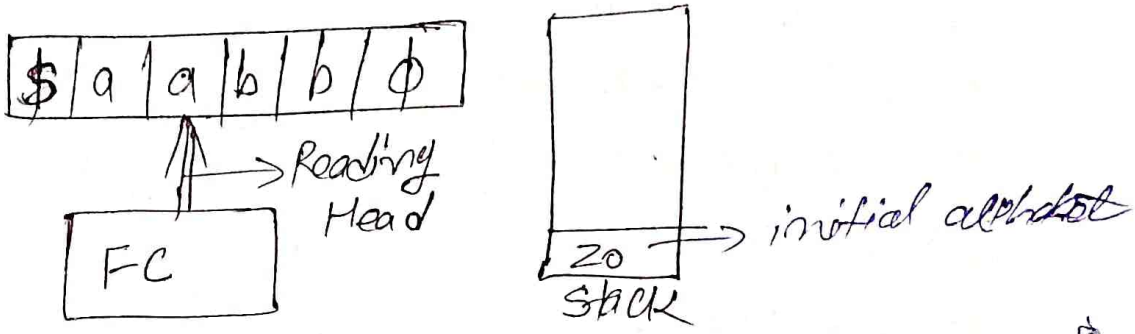
curr state	a	b	o/p
→ q1	q1	q_2'	1
q_2'	q4	q4	0
q_2''	q4	q4	1
q_3'	q_2''	q_3''	0
q_3''	q_2''	q_3''	1
q4	q_3'	q1	1


a
b
a
a,b
a,b
a
b
a
b
$q_1/1$
$q_2'/0$
$q_2''/1$
$q_3'/0$
$q_3''/1$
$q_4/1$

Pushdown Automata

* Pushdown automat is 7 tuple machine that accept context free grammar, it is more powerfull than finite automat because it can accept language that which not accepted by finite Automata. It is a 7 tuple machine.

* Model of Pushdown Automata:



H/W → PDA and finite Automata → 6 diff Model of FA, Design PDA
PDA

7 tuple: {Q, Σ, δ, q0, Γ, F, Z0}

Γ → stack initial alphabet set
Z0 → Stack initial alphabet

Define PDA: Pushdown automata is 7 tuple machine that accept context free grammar, it is more powerfull than finite automata because, it can accept language that are not accepted by finite automata. PDA is a type of finite automata with consisting of stack as additional memory to recognize a broader type of languages than finite automata.
It is a 7 tuple machine. There are two main ways a PDA can accepting string (i) By final state (ii) By Empty stack.

PDA accepted by final state, Accepted by empty stack

$T(N) = \{w \in \Sigma^* (q_0, w, z_0) \vdash (\cdot$

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

$Q \rightarrow$ Set of finite states

$\Sigma \rightarrow$ input alphabet

$\Gamma \rightarrow$ stack alphabet

$\delta \rightarrow$ transition function

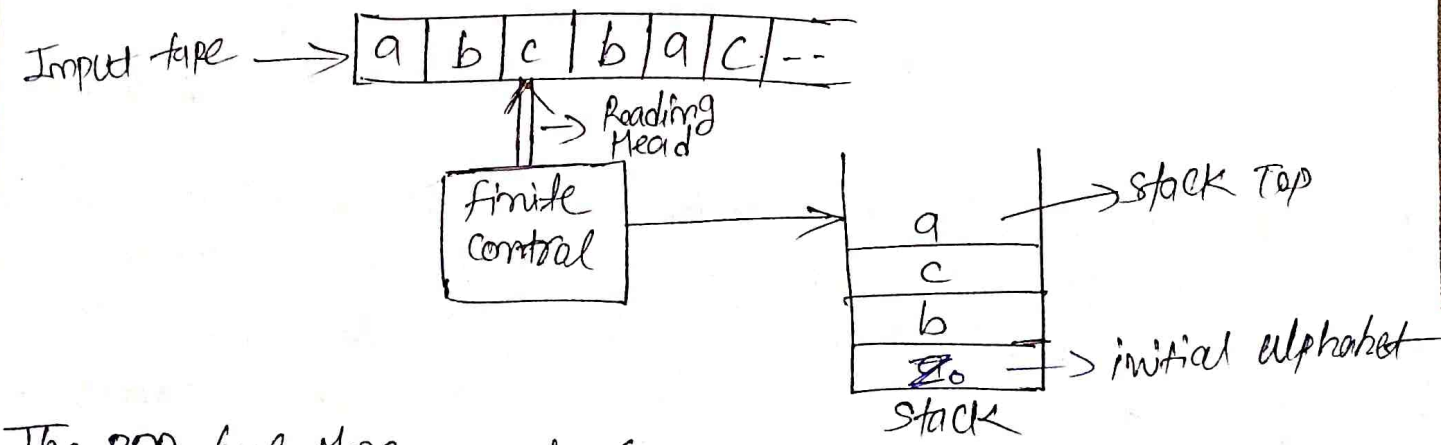
$$\delta = Q \times (\Sigma \cup \{\wedge\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

$q_0 \rightarrow$ initial state

$z_0 \rightarrow$ initial stack alphabet/symbol

$F \rightarrow$ final state/accepting state

Model of push Down Automata



The PDA has three main components:

(i) Input tape

(ii) finite control

(iii) stack

① Input tape: The input tape is divided in many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.

② Finite control: Finite control has some pointer which points the current symbol which is to be read.

③ Stack: The stack is a data structure in which we can push and pop (remove) the items from one end only. It has an infinite size. In PDA, the stack is used to store the items temporarily.

Q Difference between finite Automata and push Down Automata.

	Push Down Automata (②)	finite Push Down Automata (①)
(i) Memory	Utilize a stack for memory	No memory component, only tracks the current state
(ii) Language recognize	Recognizes context-free languages	Recognize regular language
(iii) memory operation	Can push, pop or read symbols from the stack	No memory operations
(iv) components	7 tuple (includes stack)	5 tuple (no stack)
(v) Transition function	Depends on the current state, input symbol, and stack	Depends only on the current state and input symbol
(vi) computational power	more powerful due to stack memory	Less powerful
(vii) Complexity	More complex to design.	Less complex to design.
(viii) Example	Recognizes languages like palindrome	Recognizes languages like basic strings patterns.

PDA (FA + Stack)

- Deterministic PDA
- Non-Deterministic PDA

$\delta = Q \times \{\Sigma \cup \{\in\}\} \times \Gamma \to Q \times \Gamma^*$ | $\delta = Q : \{\Sigma \cup \{\in\}\} \times \Gamma \to 2^{(Q \times \Gamma^*)}$

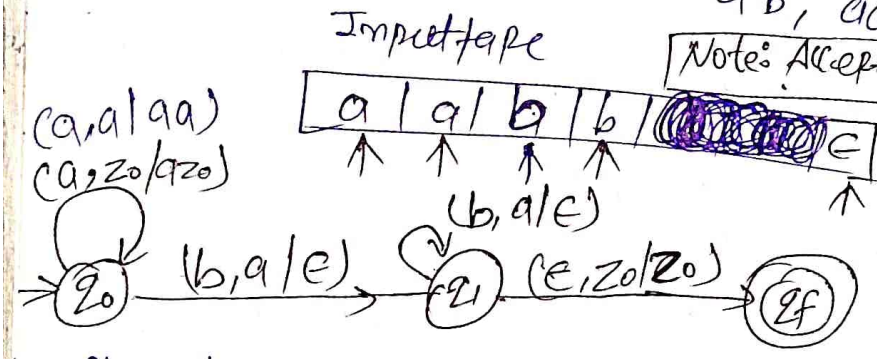
PDA

- Accepting by final state
- Accepting by empty stack

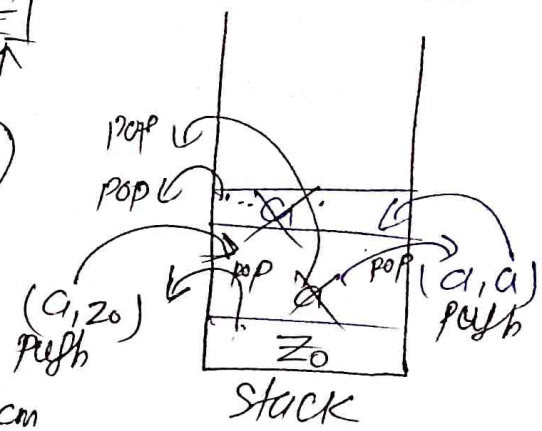
Designing of Push Down Automata

Q1. Design a PDA for accepting a language $\{a^n b^n \mid n \geq 1\}$

$a^n b^n \mid n \geq 1 \to ab, aabb, aaabbb$



using transition diagram



using transition function

$\delta(q_0, a, z_0) \vdash (q_0, az_0)$

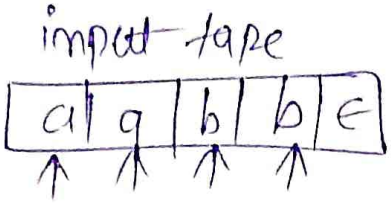
$\delta(q_0, a, a) \vdash (q_0, aa)$

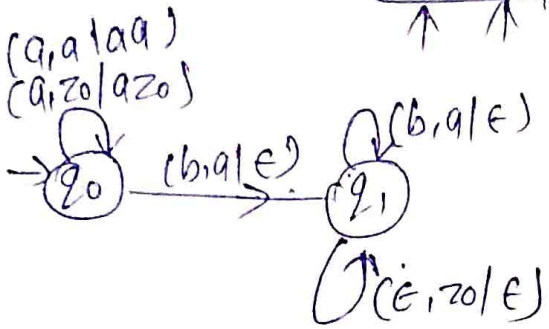
$\delta(q_0, b, a) \vdash (q_1, \in)$

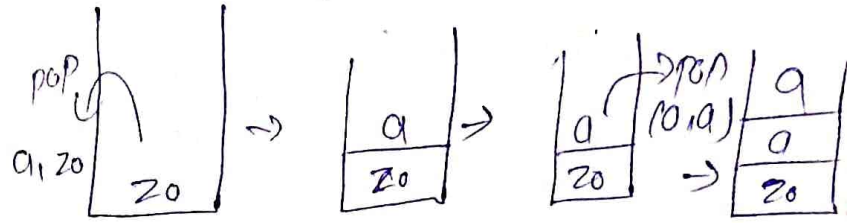
$\delta(q_1, b, a) \vdash (q_1, \in)$

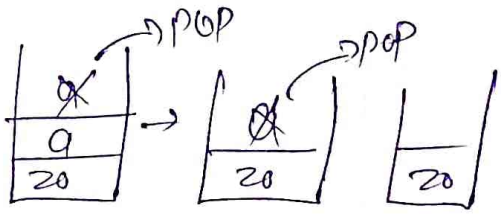
$\delta(q_1, \in, z_0) \vdash (q_f, z_0)$

Acceptance by empty stack









using transition function

$\delta(q_0, a, z_0) \vdash (q_0, aaz_0)$

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

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

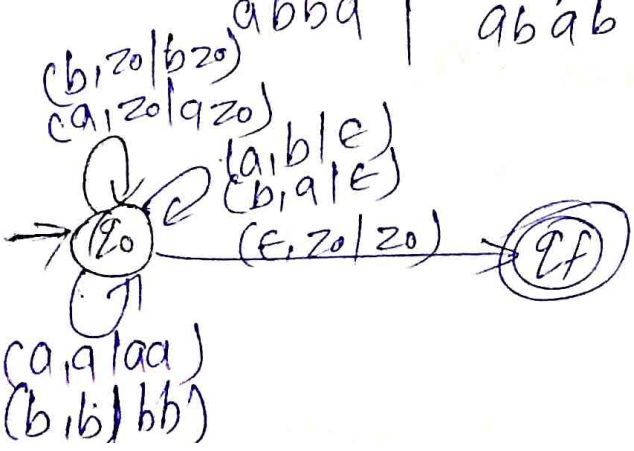
$\delta(q_1, b, a) \vdash (q_1, \epsilon)$

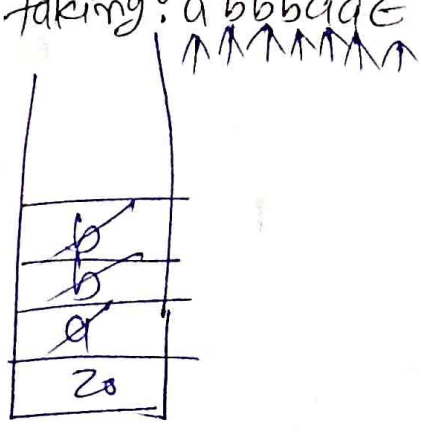
$\delta(q_1, \epsilon, z_0) \vdash (q_f, \epsilon)$ $a^n b^n$ provided

Q. Design a PDA for accepting language $n(a) = n(b)$

Soln: ~~bbaa~~ | ~~bba a~~ bcab
aabb | baba
abba | abab

taking: a bbbaa ∈





Using transition function

$\delta(q_0, a, z_0) \vdash (q_0, az_0)$

$\delta(q_0, b, z_0) \vdash (q_0, bz_0)$

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

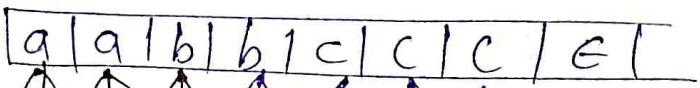
$\delta(q_0, b, b) \vdash (q_0, bb)$

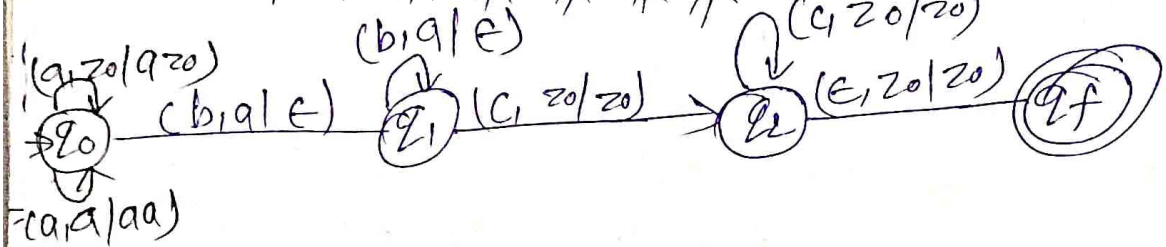
$\delta(q_0, a, b) \vdash (q_0, \epsilon)$

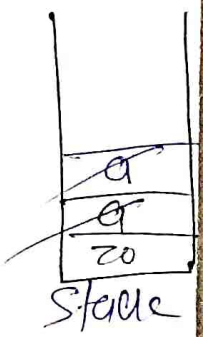
$\delta(q_0, b, a) \vdash (q_0, \epsilon)$

$\delta(q_0, \epsilon, z_0) \vdash (q_f, z_0)$

$a^n b^m c^m \quad (n, m \geq 0)$







$\delta(q_0, a, z_0) \vdash (q_0, az_0)$

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

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

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

$\delta(q_1, c, z_0) \vdash (q_2, z_0)$

$\delta(q_2, c, z_0) \vdash (q_2, z_0)$

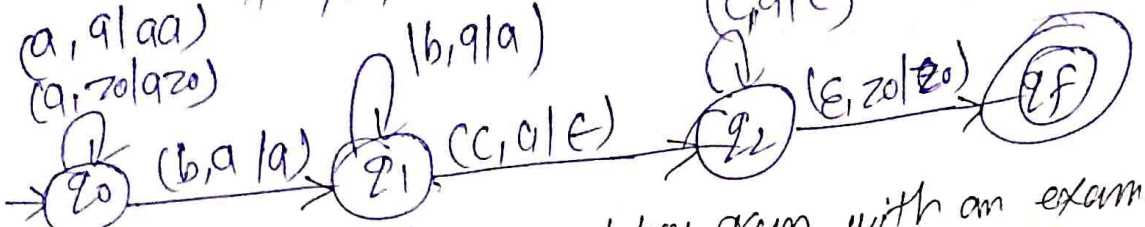
$\delta(q_2, \epsilon, z_0) \vdash (q_f, z_0)$

$P_{DA} = \{Q, \Sigma, \delta, q_0, \Gamma, f, z_0\}$

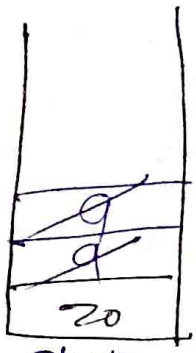
$PDA = \{\{q_0, q_1, q_2, q_f\}, \{a, b, c\}, \{\delta\}, \{q_0\}, \{a, z_0\}, \{q_f\}, \{z_0\}\}$

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

a	a	b	b	b	c	c	∈



Explain 2 is times and dry run with an example



$\delta(q_0, a, z_0) \vdash (q_0, qz_0)$

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

$\delta(q_0, b, a) \vdash (q_1, a)$ // do nothing with b

$\delta(q_1, b, a) \vdash (q_1, a)$ // //

$\delta(q_1, c, a) \vdash (q_2, \in)$

$\delta(q_2, c, a) \vdash (q_2, \in)$

$\delta(q_2, \in, z_0) \vdash (q_f, z_0)$

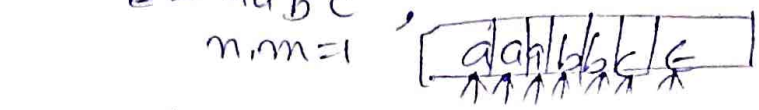
PDA $\Rightarrow \{ Q, \Sigma, \delta, q_0, \Gamma, f, z_0 \}$

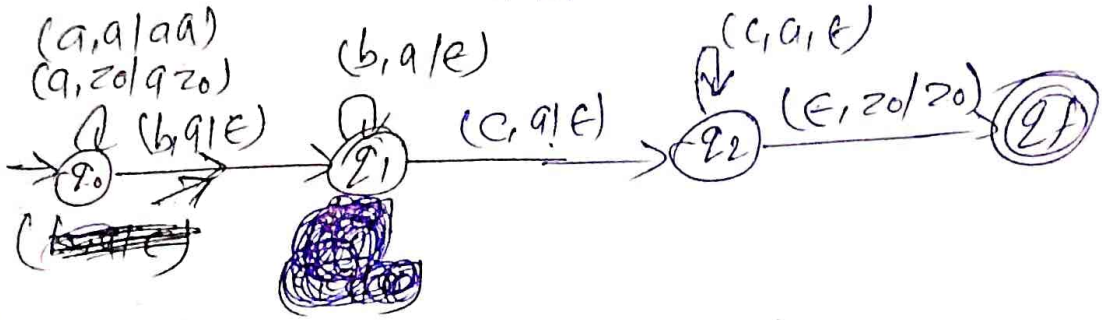
PDA $\Rightarrow \{ \{q_0, q_1, q_2, q_f\}, \{a, b, c\}, \{\delta\}, \{q_0\}, \{q, z_0\}, \{q_f\}, z_0 \}$

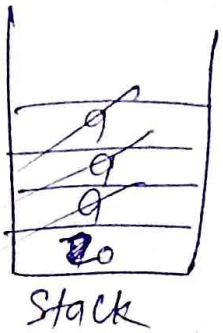
$L = a^{m+n} b^{m} c^{n} \mid m, n \geq 1$

$L = aab^1c^1$, $m=2, n=1$

$n, m = 1$







$(q_0, a, z_0) \vdash (q_0, a z_0)$

$(q_0, a, a) \vdash (q_0, aa)$

$(q_0, b, a) \vdash (q_1, \in)$

$(q_1, b, a) \vdash (q_1, \in)$

$(q_1, c, a) \vdash (q_2, \in)$

$(q_2, c, a) \vdash (q_2, \in)$

~~$(q_2, \in, z_0) \vdash (q_f, z_0)$~~

$(q_2, \in, z_0) \vdash (q_f, z_0)$

PDA $= \{q_0, q_1, q_2, q_f\}, \{a, b, c\}, \{a, z_0\}$

$\{q_0\} \{a, z_0\}, \{q_f\} \{z_0\}$.

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

Soln: $a^n b^{m+n} c^m \Rightarrow a^n b^m \cdot b^n \cdot c^m$

$\Rightarrow a^n b^m b^n c^m$

Each encountering of a push into stack, each encountering of b pop & each eco of b push, each encounter of c pop it

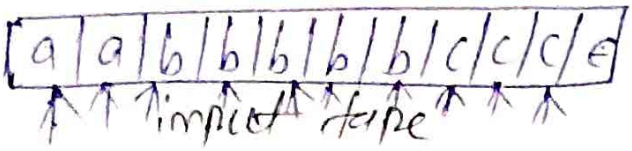
~~push into stack~~ ~~by each encountering of c & c pop from stack~~

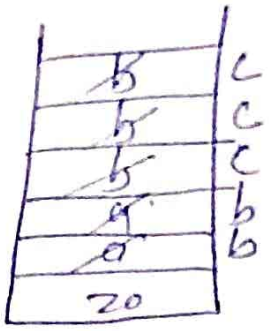
$$L = a^n b^{m+m} c^m \mid n \geq 1$$

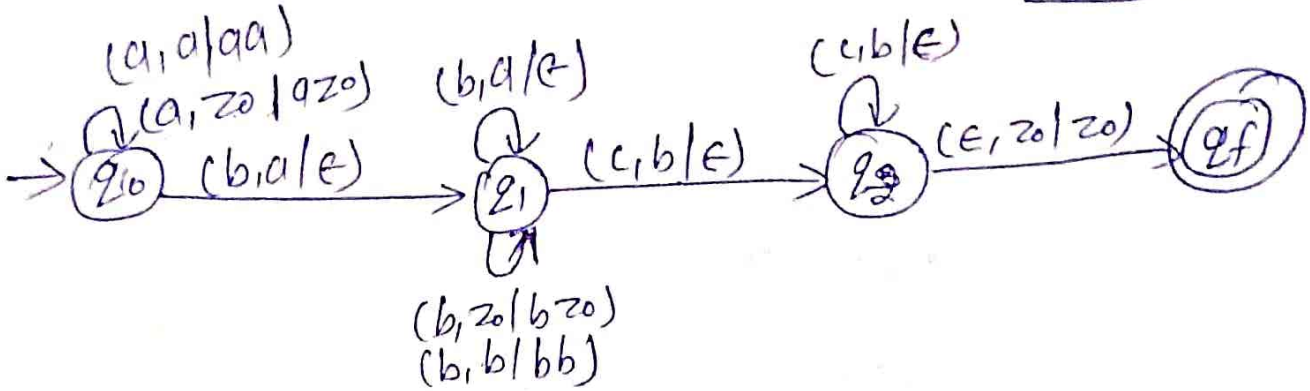
taking $n = 2$, $m = 3$

$$L = a^2 b^{3+2} c^3$$

$$L = aabbbbbccc$$







dry run

$(q_0, a, z_0) \vdash (q_0, az_0)$

$(q_0, a, a) \vdash (q_0, aa)$

$(q_0, b, a) \vdash (q_1, \epsilon)$

$(q_1, b, a) \vdash (q_1, \epsilon)$

$(q_1, b, z_0) \vdash (q_1, bz_0)$

$(q_1, b, b) \vdash (q_1, bb)$

$(q_1, c, b) \vdash (q_2, \epsilon)$

$(q_2, c, b) \vdash (q_2, \epsilon)$

$(q_2, \epsilon, z_0) \vdash (q_f, z_0)$

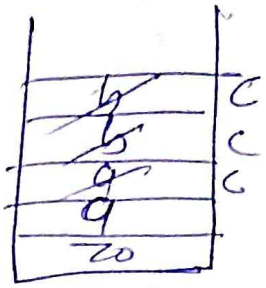
Q. $a^n b^m c^{m+m} \mid m, n \geq 1$

Sol^n: $a^n b^m c^{m+m} \Rightarrow a^n b^m c^m \cdot c^n \Rightarrow$

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

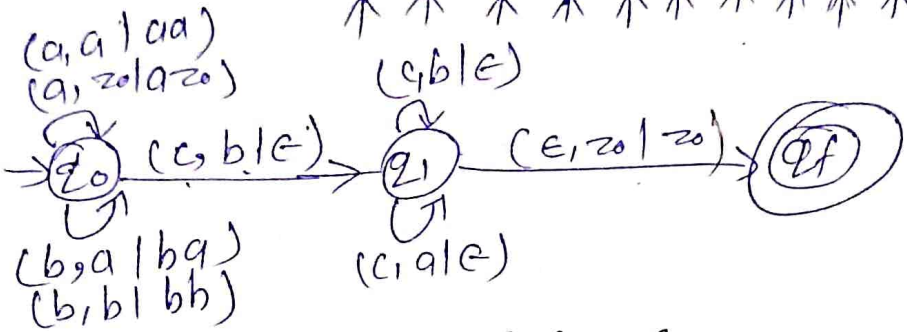
taking $m=2,\ n=3$

$a^3 b^2 c^{3+2} \Rightarrow aaabbccccc$



input tape

a	a	a	b	b	c	c	c	c	c	∈



Dry run

$\delta(q_0, a, z_0) \vdash (q_0, a z_0)$

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

$\delta(q_0, b, a) \vdash (q_0, ba)$

$\delta(q_0, b, b) \vdash (q_0, bb)$

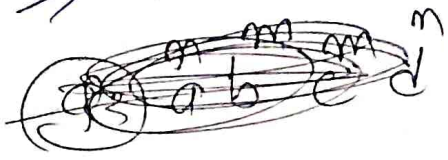
$\delta(q_0, c, b) \vdash (q_1, \epsilon)$

$\delta(q_1, c, b) \vdash (q_1, \epsilon)$

$(q_1, c, a) \vdash (q_1, \epsilon)$

$(q_1, \epsilon, z_0) \vdash (q_f, z_0)$

$q_1.\ a^n b^n c^m d^m$ (q2) $a^n b^m c^n d^m$ → Not possible using PDA



$q_3.\ a^m b^n$ $q_4.\ a^n b^{2n}$ (q5) $a^n b^m c^n$

↓

Not possible using PDA

Design PDA for $L = \{a^{2m} b^m \mid m \geq 1\}$

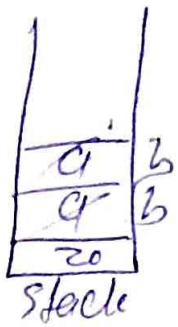
$m = 1$

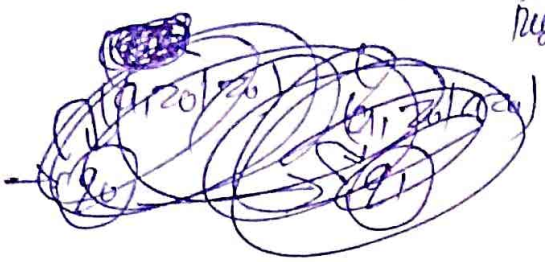
$L = \{aab, aaaabb, \ldots\}$

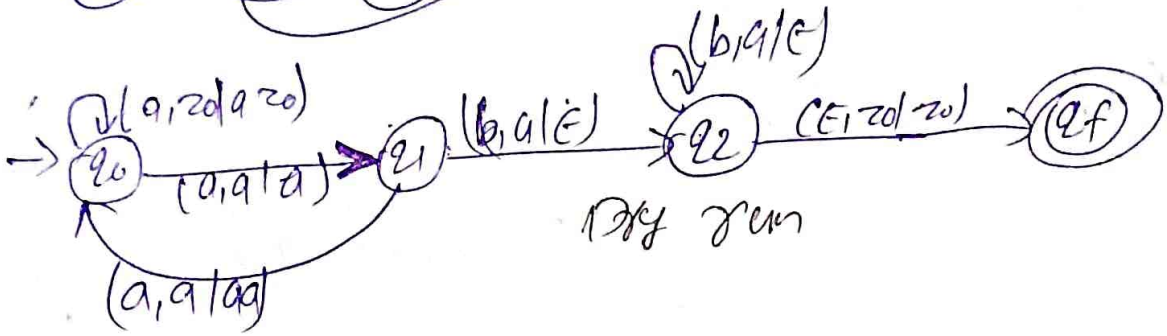
taking

1	2	3	4	5	6	
a	a	a	a	b	b	∈

Push Skip Pop Pop







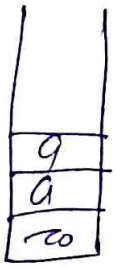
Design PDA $\{L = a^n b^{2n} \mid n \geq 1\}$

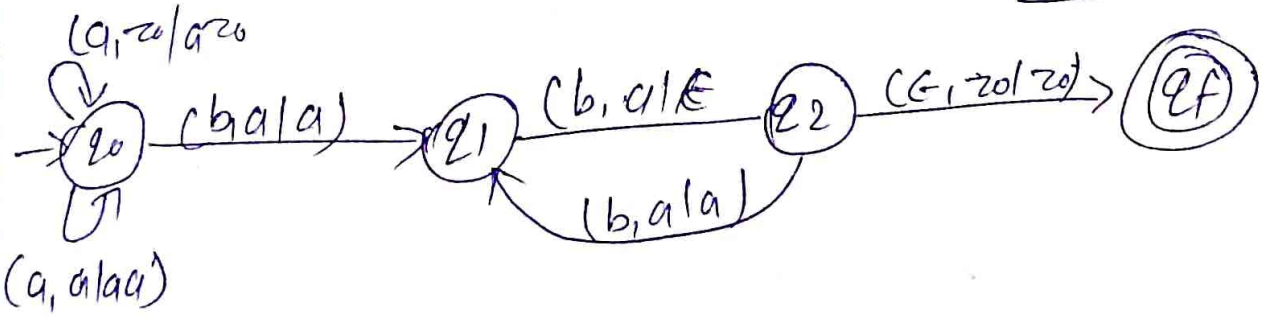
$L = \{abb, aabbbb, aaabbbbbb \ldots\}$

taking

a	a	a	b	b	b	b	b	b	∈

Push Push Push Pop Pop Pop



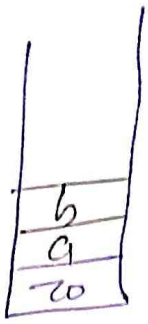


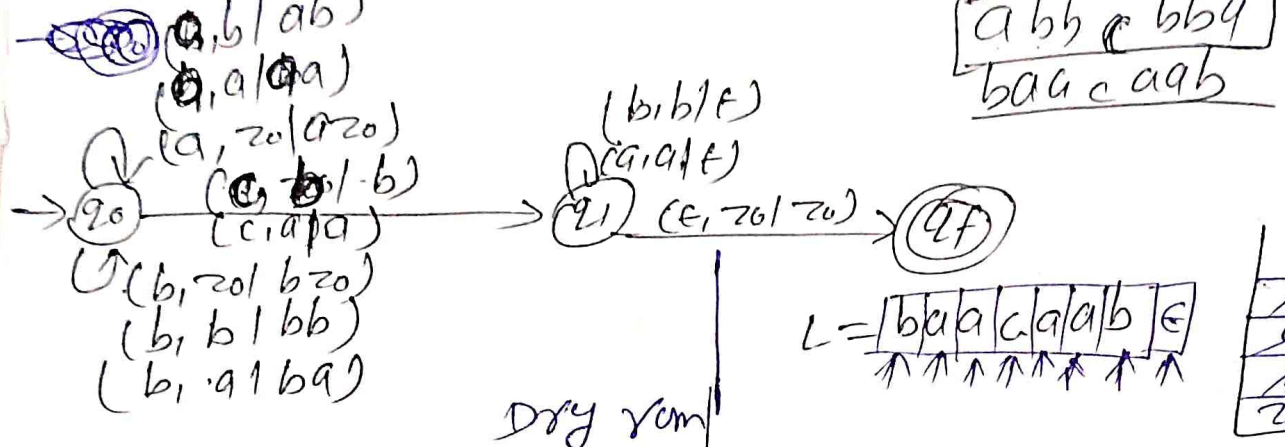
designing a PDA of odd no of palindrom

Q $wcw^R \mid w \in (a,b)^+$

taking $w = abb$

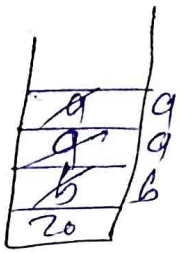
$w = abbcbba$





abbcbba
bacaab

$L = \boxed{b}\boxed{a}\boxed{a}\boxed{c}\boxed{a}\boxed{a}\boxed{b}\boxed{\epsilon}$



Dry run

$\delta(q_0, a, z_0) \vdash (q_0, az_0)$
$\delta(q_0, a, a) \vdash (q_0, aa)$
$\delta(q_0, a, b) \vdash (q_0, ab)$
$\delta(q_0, b, z_0) \vdash (q_0, bz_0)$
$\delta(q_0, b, b) \vdash (q_0, bb)$
$\delta(q_0, b, a) \vdash (q_0, ba)$
$\delta(q_0, c, b) \vdash (q_1, b)$
$\delta(q_0, c, a) \vdash (q_1, a)$
$\delta(q_1, b, b) \vdash (q_1, \epsilon)$
$\delta(q_1, a, a) \vdash (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) \vdash (q_f, z_0)$

$\delta(q_0, b, z_0) \vdash (q_0, bz_0)$
$\delta(q_0, a, b) \vdash (q_0, ab)$
$\delta(q_0, a, a) \vdash (q_0, aa)$
$\delta(q_0, c, a) \vdash (q_1, a)$
$\delta(q_1, a, a) \vdash (q_1, \epsilon)$
~~$\delta(q_1, a, a) \vdash$~~
$\delta(q_1, b, b) \vdash (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) \vdash (q_f, z_0)$

* design a PDA of even no of palindrom.

Note : for even no palindrom it will be NDPDA → non deterministic push down automata.

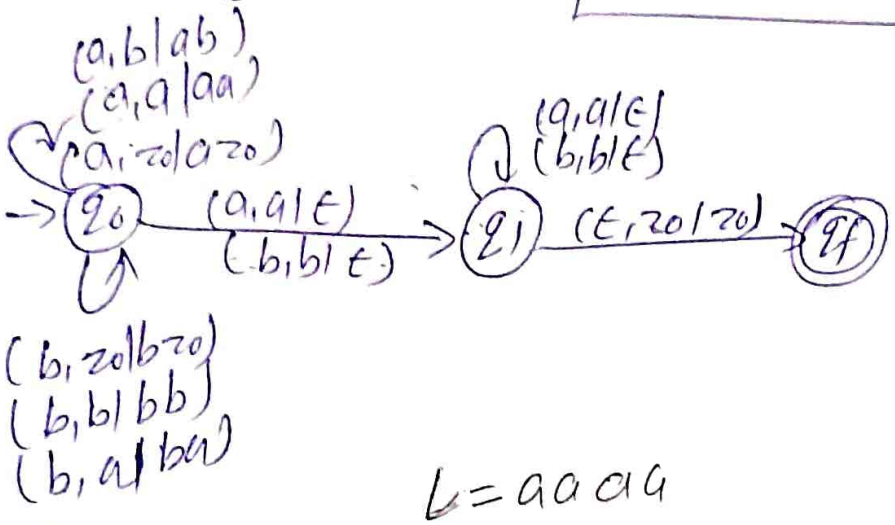
Q ... $L = \{ ww^R \mid w \in (a,b)^* \}$

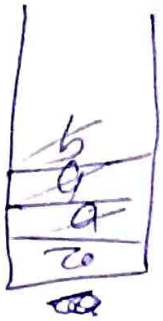
$w = aab$

$w^R = baa$

$L = aabbaa$

push	pop
aa	aa
bb	bb
ab	ba
aba	aba
bab	bab





$L = aaaa$

Instantaneous Description :- for string aaaa

$(q_0, aaaa, z_0) \vdash (q_0, aaa, az_0)$

N.C | C

(q_0, aa, aaz_0) | (q_1, aa, z_0) ✗ not transition defined

N.C | C

$(q_0, a, aaaz_0)$ | (q_1, a, az_0)

NC | C

$(q_0, \epsilon, aaaaz_0)$ ✗ dead configuration | (q_1, ϵ, aaz_0) Dead configuration | (q_1, ϵ, z_0)

final state Accepted

C → combo

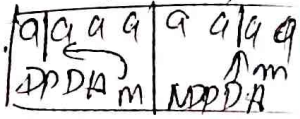
NC → not combo

~~MSTAM~~

Q Why NDPDA is more powerful than DPDA? Or Is NDPDA more powerful than DPDA? Justify with suitable example

Yes, NDPDA is more powerful than DPDA. This means that there are languages that can be recognized by NDPDA but not by a DPDA.

Example: $L = \{ w w^R \mid w \in (a,b)^+ \}$ this language consists of all even palindroms over the alphabet $\{a,b\}$.



A DPDA cannot recognize this language because it require making a decision about the middle of the palindrom without knowing the entire input string.

An NDPDA can recognize this language by guessing the middle of the palindrome and then comparing the first half with the second half in reverse order. It can do this by pushing symbols onto the stack as it reads the first half of the string. Then it can compare the symbols on the stack with the remaining symbols, poping them off the stack as it matches them. if the stack is empty at the end of the input, the string is accepted as a palindrome.

3 b/w push Down Automata and Turing machine

Features	pushDown Automata	Turing machine
Language recognize	It recognizes context-free languages.	It recognizes recursively enumerable languages.
Tape	It has only one tape	It has infinite tape.
memory	It has single stack for Memory	It has an infinite memory.
Acceptance	Acceptance based on empty stack or final state.	Acceptance on halting in an accepting state.
Computational power	Less powerful	More powerful
Type of Automata	PDA can be deterministic and non deterministic.	Turing machine can also be deterministic and non deterministic.
Transition function	$\delta: Q\Sigma\varepsilon \cup \Lambda\Sigma \times \Gamma \rightarrow Q \times \Gamma^*$	$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times (L, R)$
Use case	used in parsing and syntax analysis in compilers	Serves as a theoretical model for general-purpose computers.
Head movement	Head can only move to ~~one~~ one direction left to right.	Head can move in both direction left and right.
Read/write	It can only read input tape	It can read or write on input tape

Q) Difference b/w Finite Automata and Turing Mach

Features	Finite Automata	Turing machine
Language recognition	It recognize regular language.	It regonize regular, context free, context-semitive and recursively enumerable language.
Input Tape	Input tape is finite in length. from both sides.	Input tape is finite on the left side but infinite on the right side.
Components	Consists of a finite number of states, a finite set of input symbols, an initial state, and transition rules.	Consists of finite states, input symbols, a blank symbol, an initial state and transition rule
Head Movement	Head moves only to the right side.	The head can move in both directions (left and right).
Read/write capacity	The head only reads symbols from the tape, it cannot write.	The head can read as well as write symbols.
computational power	Less powerful than a Turing machine	More powerful than finite Automata.
Design Complexity	Easier to design	difficult to design as well as complex.
Transition function	$\delta : Q \times \Sigma^* \rightarrow Q$	$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times (L,R)$
Use Cases	Useful for simpler applications such as pattern matching.	Useful for complex computational task such as algorithmic processing.

Pumping Lemma (CFL)

Pumping Lemma is used to prove that a language is not a context free.

Step 1: Assume L is a context free language & pumping length n such that $|w \in L, |w| \geq n$

Step: 2 choose w → string

Step 3: break w into 5 parts, $w = uvxyz$ such that

$|vxy| \leq n$

$|vy| \geq 1$

where v and y only contains one type of symbol.

Step 4: $w = uv^i xy^i z$ at any value of i uvxyz does not belong to L ($uv^i xy^i z \notin L$)

Q. Show that $L = \{a^n b^n c^n | n \geq 0\}$ is not context free.

Step 1: Assume L is a context free language & pumping length = n

Step 2: $L = \{abc, aabbcc, aaabbbccc, \ldots\}$

$w = aaabbbccc$ taking $n = 3$

$w \geq 3$, break w into 5 parts $w = uvxyz$

Step 3:

$|vxy| \leq n$

$|vy| > 0$

taking $n = 3$

$w = a\,a\,a\,b\,b\,b\,c\,c\,c$

$u\ v\ \ x\ y\ \ z$

$u = a$
$v = a$
$x = b$
$y = b$
$z = ccc$

$|vxy| \leq n$

$|abb| \leq 3$ ✓

$|vy| \geq 1$ ✓

Step 4: choose value for i such that

$uv^i xy^i z \notin L$

$i = 0$ $\quad aa^i bb^i ccc \Rightarrow aa^0 b^0 ccc$

$\Rightarrow abccc$

$\Rightarrow a^1 b^1 c^3$

$a^1 b^1 c^3 \notin L$ [no of a, b, c should be equal]

under the assumption that L is a context free language leads to a contradiction, L is not a context free language.

Eg: Explain ambiguity of context free languages. Test the following language is ambiguous or not.

$S \rightarrow S1S/0$

A grammar is said to be ambiguous if there exists more than one left most derivation or more than one right most derivation or more than one parse tree for the given string. No method can detect and remove ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

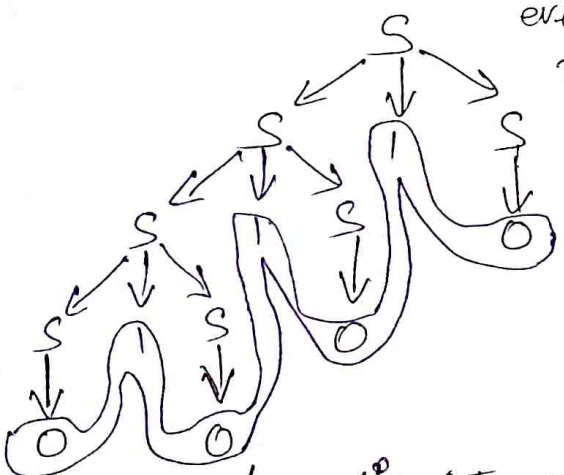
Given $\rightarrow S \rightarrow S1S/0$

taking string = "0101010"

$S \rightarrow S1S$

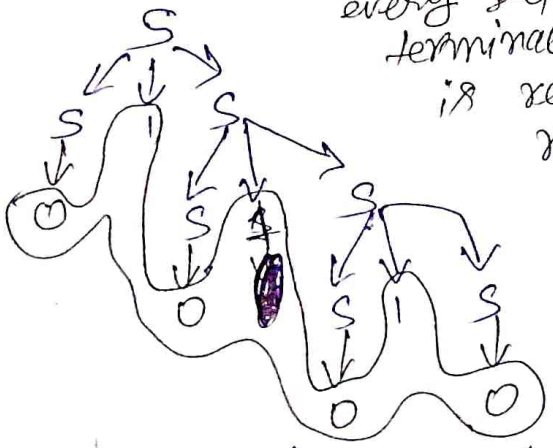
$S \rightarrow 0$

(i) Left Most derivative: In a left most derivation, at every step, the leftmost non-terminal in the current string is replaced using a production rule.



0101010

(ii) Right Most derivative: In a right most derivation, at every step, the rightmost non-terminal in the current string is replaced using a production rule.



0101010

You can clearly see that in both way left most derivative and right most derivative both parse tree are different hence it is ambiguous.

* Turing Machine: Turing machine is a 7 tuple machine that can accept regular, context free, context sensitive, recursively enumerable language. In turing machine head movement can be possible in both direction left and right. (1)

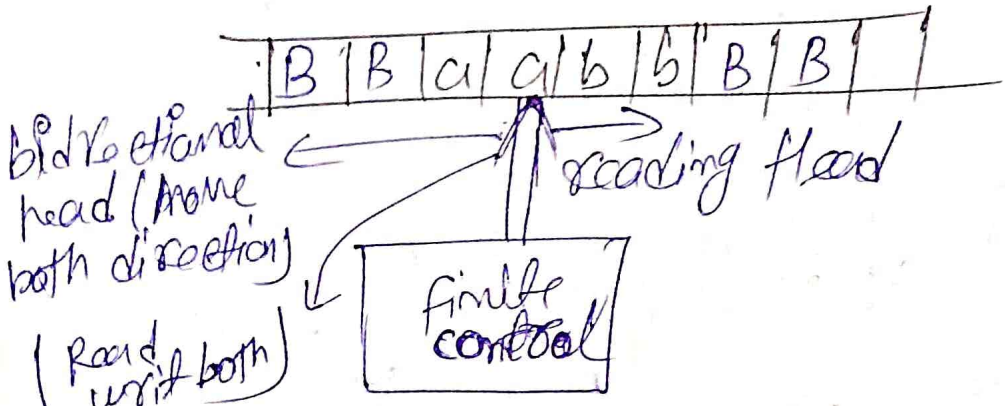
$(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ $\Gamma \rightarrow$ Tape symbol

$B \rightarrow$ Blank symbol

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

Turing Machine

Model of TM



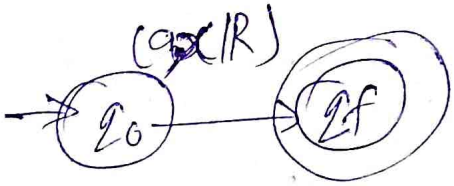
It is a 7 tuple machine

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

B → Blank symbol

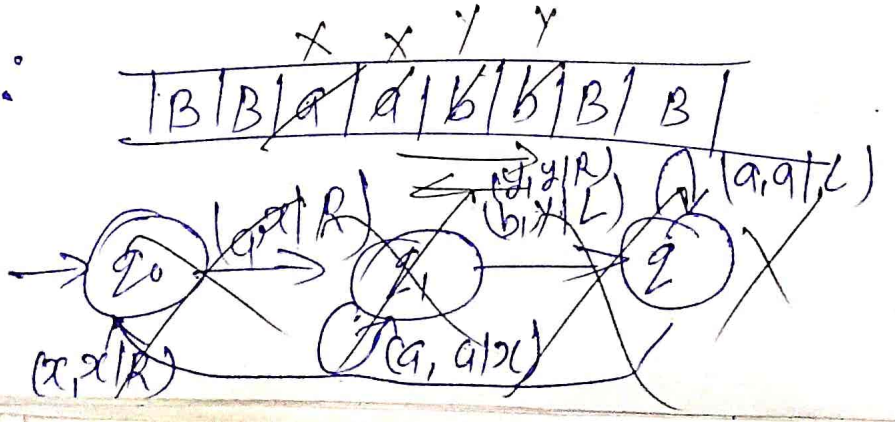
Γ ← tape symbol

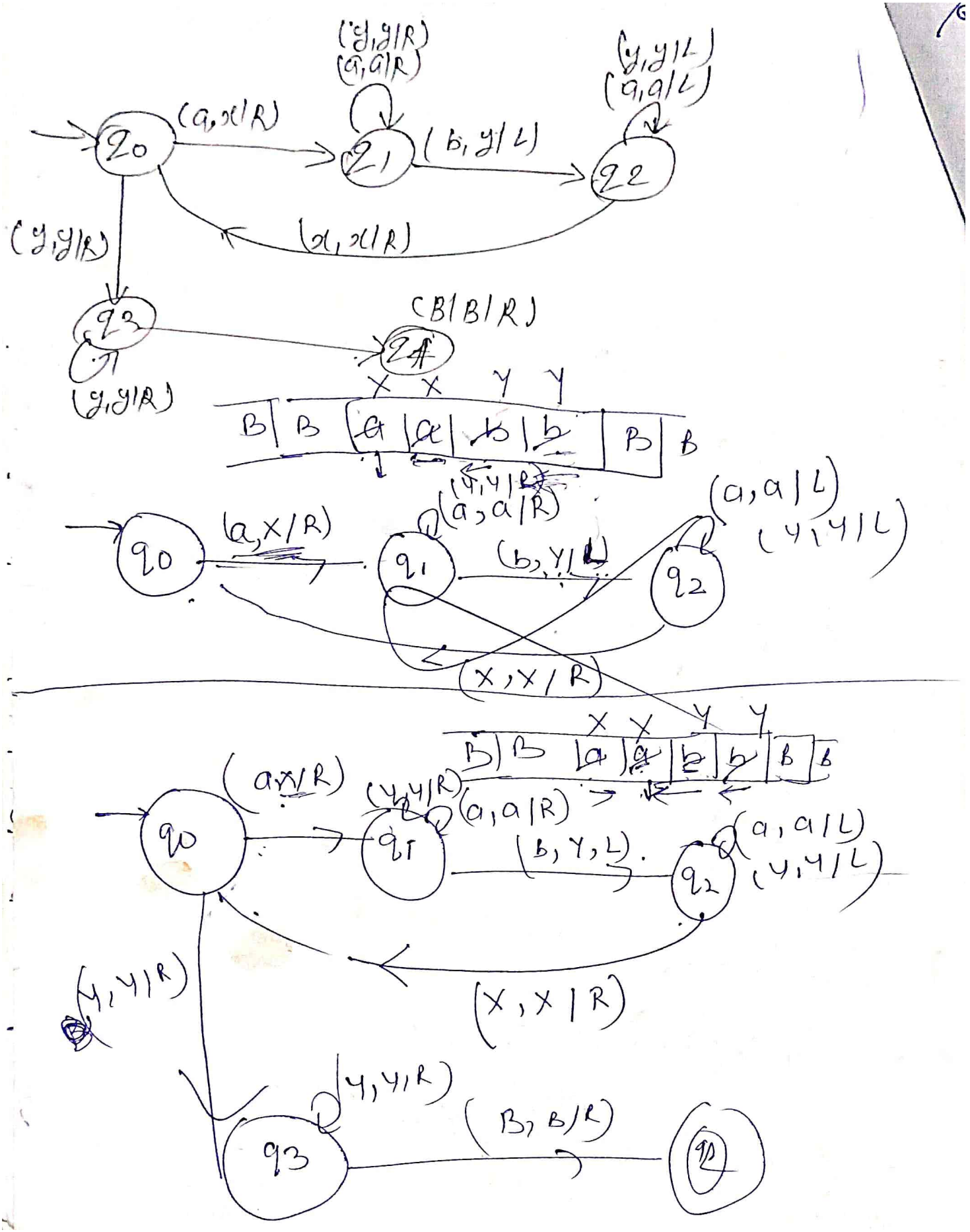
$\delta: Q \times \Gamma \to Q \times \Gamma \times (L, R)$



Q $a^n b^n \mid n \geq 1$ 8 marks Mpt.

Sol^n:




(Y,Y/R)
(a,a/R)
(Y,Y/L)
(a,a/L)
(a,x/R)
(b,y/L)
q0
q1
q2
(Y,Y/R)
(x,x/R)
q3
(B/B/R)
q4
(Y,Y/R)
X X Y Y
B B a a b b B B
(Y,Y/R)
(a,a/R)
(a,a/L)
(Y,Y/L)
(a,x/R)
(b,Y/L)
q0
q1
q2
(X,X/R)
X X Y Y
B B a a b b B B
(a,a/R)
(a,x/R)
(Y,Y/R)
(a,a/L)
(Y,Y/L)
(b,Y,L)
q0
q1
q2
(Y,Y/R)
(X,X/R)
(Y,Y/R)
(B,B/R)
q3
q4

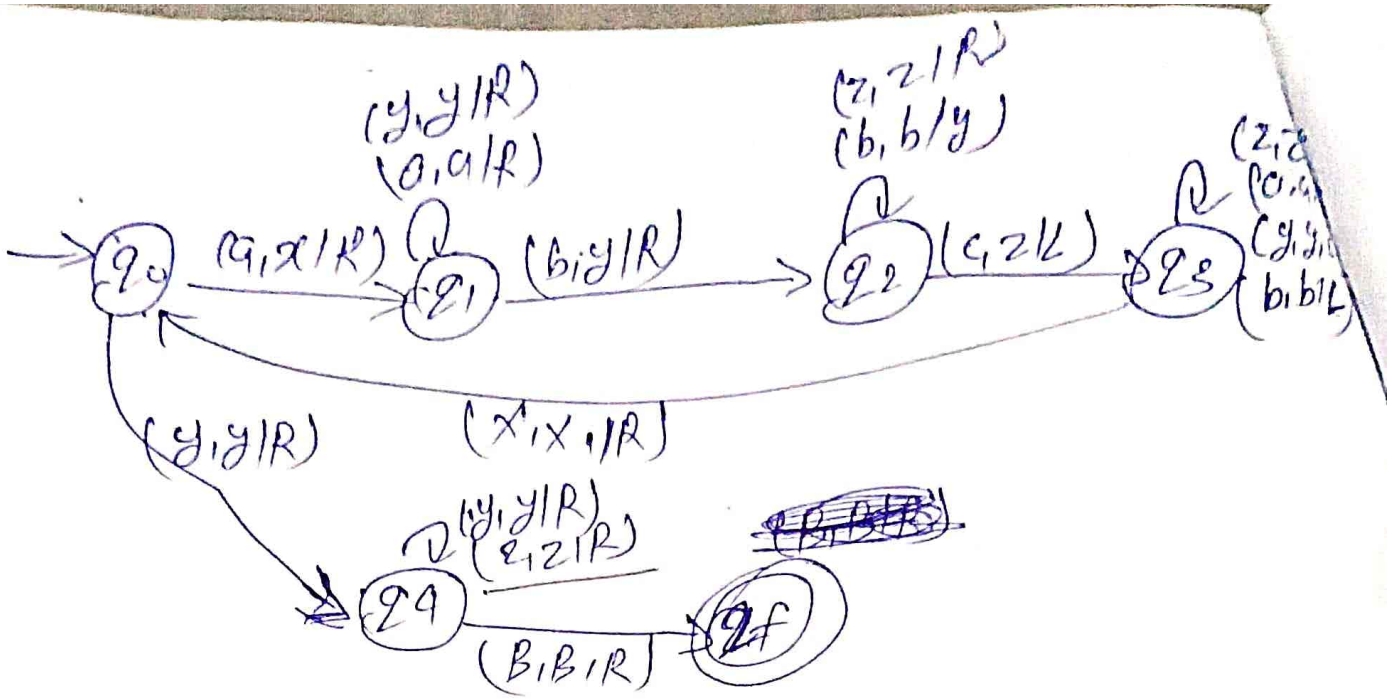
- $a^n b^{nn} c^{nn} \mid n \geq 1$
- Univer TM
- Multitop TM
- Single TM

Design a automata for equal no of a's b's and c's.

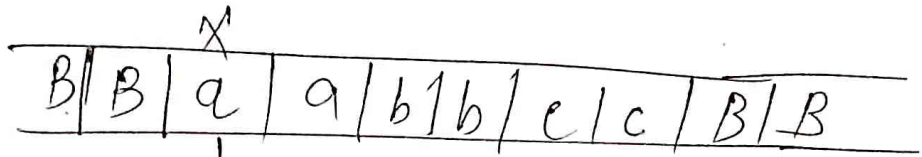
$$L = \{ a^n b^n c^n \mid n \geq 1 \}$$

x x y y z z

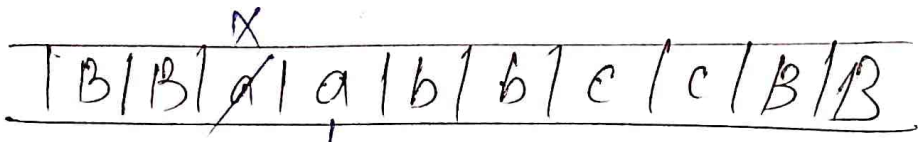
B	B	X	a	b	Y	c	Z	B	B



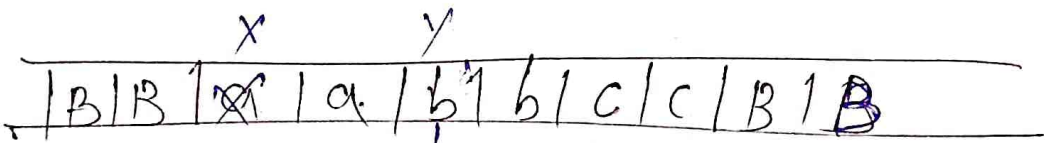
Q



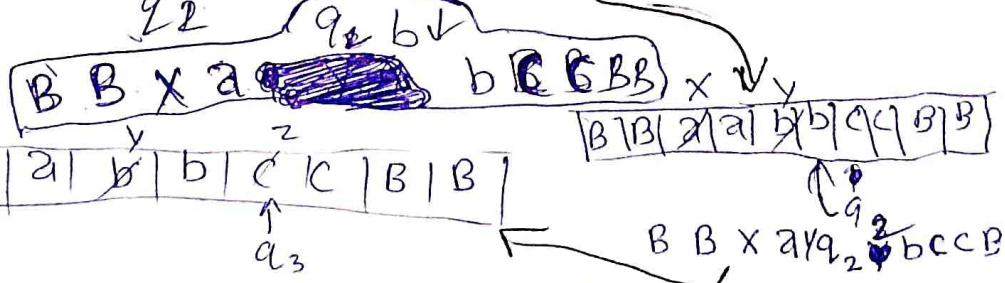
q_0 $Bq_0aabbccB$



$Bxq_1abbccB$

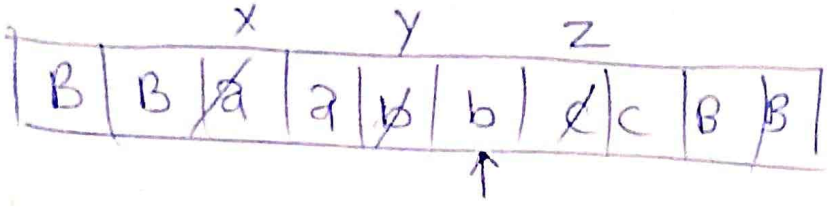


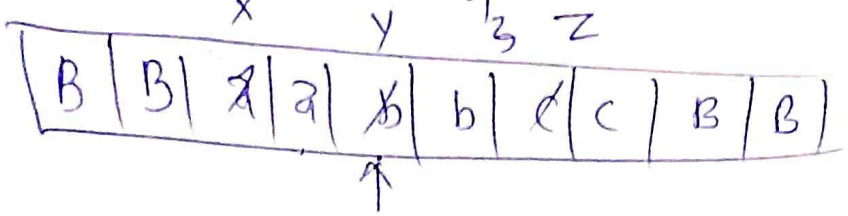
q_2

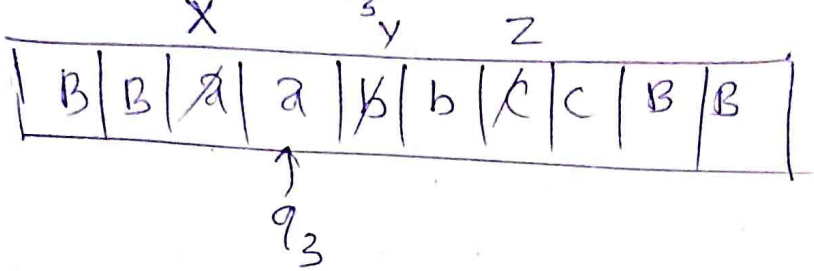


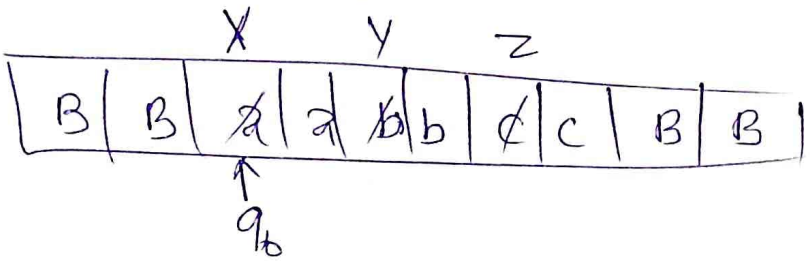
$Bxaq_2bccBB$

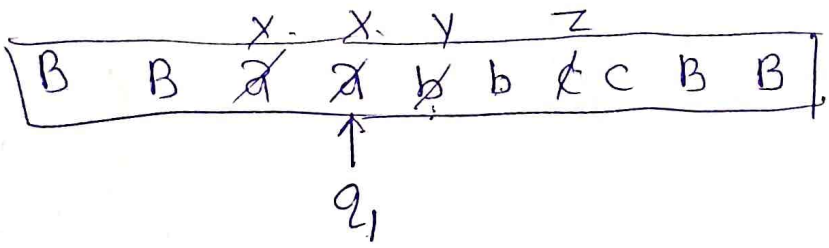
$BBxayq_3zccBB$



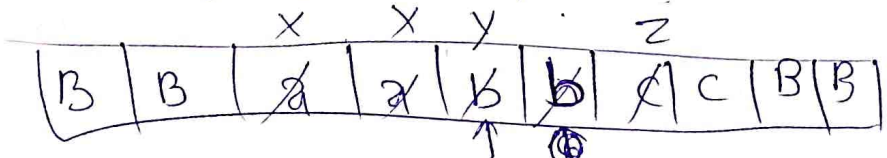


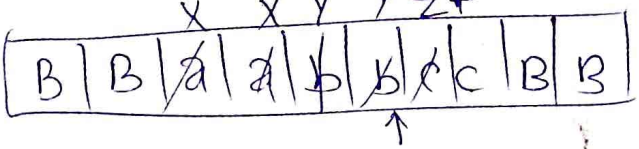


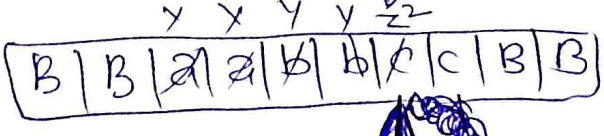




B B X X q_1 Y b Z C B B







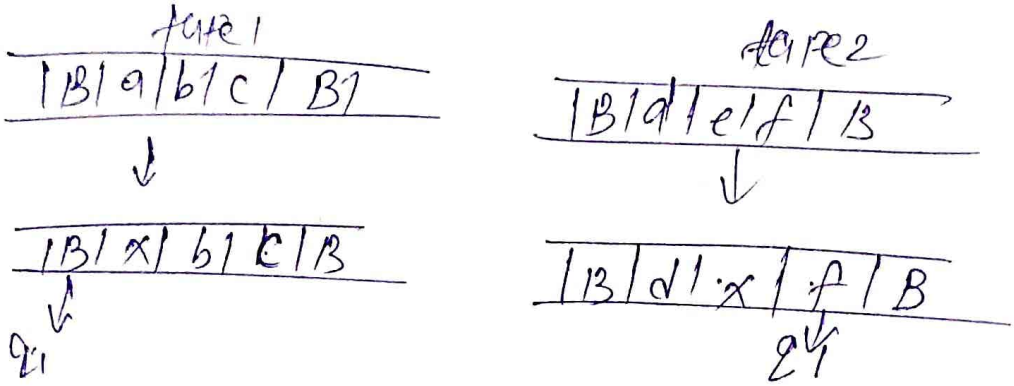
Varients of Turing machine

① Multitape Turing machine

A Turing machine with several tapes every tape have their own read write head.

$$\delta : Q \times \Gamma^N \to Q \times \Gamma^N \times \{L, R\}^N$$

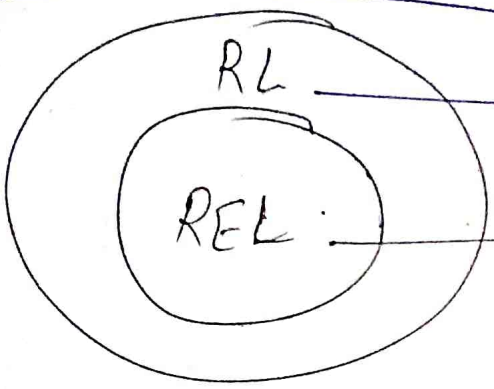
$$(q_0, a, e) \longrightarrow (q_1, x, y, L, R)$$



② Multihead Turing machine

NST
~~→ PDA~~
~~→ $a^n b^n$~~ can (code can)
~~PDA vs TM~~
~~PDA vs FA~~
~~TM vs FA~~
~~Turing machine~~
~~RL vs RGL~~

REL & RL



(i) Halt & Accept
(ii) Halt & reject

→ (1) Halt and Accept
(2) Halt and reject
(3) Never halt (Loop)

Recursive lang: A language L is recursive if there is halting and total turing machine.

Recursively Enumerable lang: A language is REL if

Q. Diff b/w Recursive language and recursively enumerable

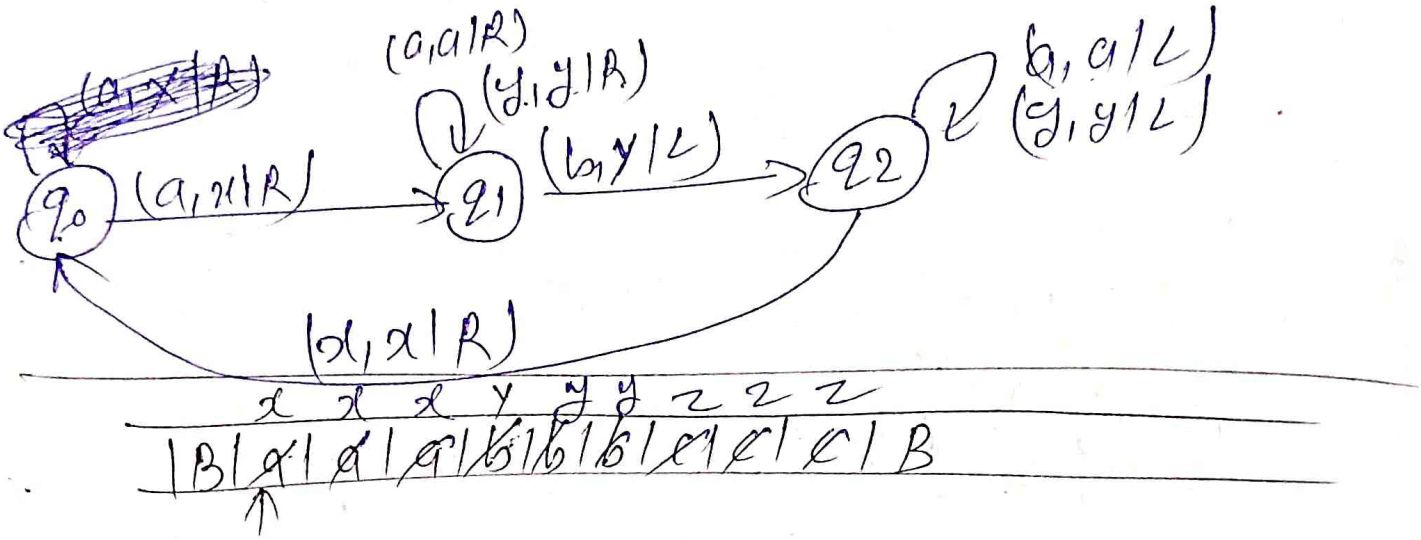
Features	Recursive Language	Recursively enumerable language
Also known as	Turing decidable languages	Turing recognizable languages
Definition	In Recursive language, the Turing machine accepts all valid strings that are part of the language and rejects all the strings that are not part of a given language and halt.	In Recursively enumerable language the Turing machine accepts all valid strings that are part of the language and rejects all the strings that are not part of the given language but do not halt and starts an infinite loop.
States	(i) Halt and accept (ii) Halt and Reject	(i) Halt and accept (ii) Halt and reject (iii) Never halt (infinite loop)
Loop	Finite Loop	infinite loop
Halting	Halting Turing machine	Non Halting Turing machine
Acceptance	A string is accepted if the machine halts in an accepting state.	A string is accepted if the machine halts in an accepting state.
Examples	The set of prime numbers	The Halting problem
Computational complexity	less complex than REL.	more complex than RL.

$L = a^n b^n c^n \quad | n \geq 1.$

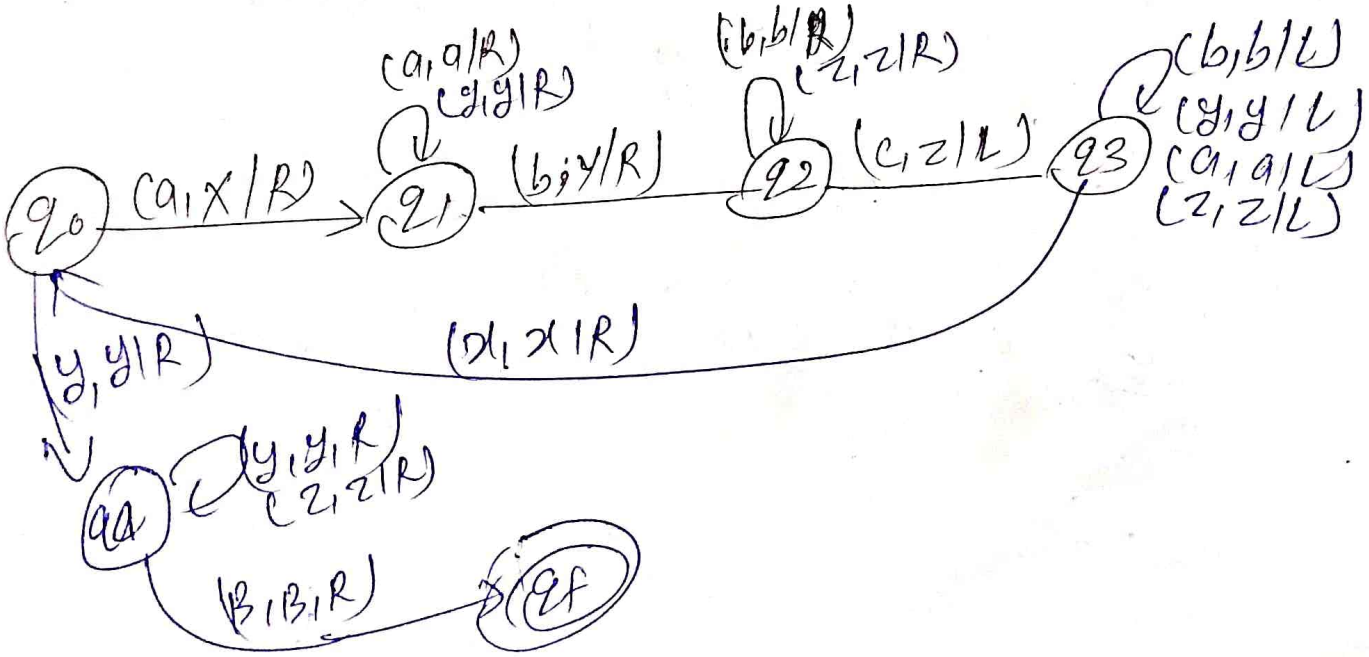
?

$L = a\,a\,a\,b\,b\,b\,c\,c\,c$

	X	X	X	Y	Y	Y				
B	a	a	a	b	b	b	c	c	c	B



	x	x	x	y	y	y	z	z	z	
B	a	a	a	b	b	b	c	c	c	B



Machines	DFA	NFA	Moore	Mealy
tuples	DFA = 5tuple $\{\Sigma, Q, q_0, \delta, f\}$ Σ → input symbols Q → Set of states q_0 → initial state δ → transition fun f → final state	NFA = 5tuple $\{\Sigma, Q, q_0, \delta, f\}$	$M = 6$ $\{\Sigma, Q, q_0, \delta, \Delta, \lambda\}$ Δ → o/p alphabet λ → o/p function $\delta: Q \times \Sigma \to Q$ $\lambda: Q \to \Delta$	$M = 6$ $\{\Sigma, Q, q_0, \delta, \Delta, \lambda\}$
transition function	$\delta: Q \times \Sigma \to Q$	$\delta: Q \times \Sigma \to 2^Q$	$\delta: Q \times \Sigma \to Q$ $\lambda: Q \to \Delta$	$\delta: Q \times \Sigma \to Q$ $\lambda: Q \times \Sigma \to \Delta$

PDA	TM	MTTM	LBA
PDA = 7 tuple $\{\Sigma, Q, q_0, \delta, F, Z_0, \Gamma\}$ Γ → Stack alphabet Z_0 → initial stack symbol	TM = 7 tuple $\{\Sigma, Q, q_0, \delta, F, \Gamma, B\}$ Γ → Tape symbol B → Blank symbol	MTTM = 7 tuple $\{\Sigma, Q, q_0, \delta, F, \Gamma, B\}$	LBA = 8 $\{\Sigma, Q, q_0, \delta, F, \Gamma, [,]\}$ [→ Left end marker] → Right end marker
transition function $\delta: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$	$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$	$\delta: Q \times \Gamma^N \to Q \times \Gamma^N \times (L, R)^N$	$\delta: Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ where (-1, 0, +1) indicates movement of the tape.

TOC

PYQ property of LR(k) Grammar

(i) Every LR(k) Grammar G is unambiguous.

(ii) There exists a DPDA that accepts the language generated by an LR(k) grammar.

(iii) LR(k) grammars are widely used in compiler construction due to their robustness and efficiency.

(iv) LR(k) parsers are known for their efficiency in parsing input strings. They can quickly determine the correct parse tree.

(v) LR(k) parsers often have built-in error recovery mechanisms. when encountering syntax errors they can attempt to recover and continue parsing.

(vi) LR(k) grammars can express a wide range of programming language.

(vii) The 'k' in LR(k) refers to the no of lookahead symbols used by the parser to make decisions.

PYQ Define Halting problem. Comment on ATM = {(M,w) | M is a TM and M halts at input w}.

PYQ Explain Halting problem of Turing Machine.

ans ⇒ The Halting problem is not a problem it is a question that is asked to turing machine that a Given algorithm will ever halt or not? Halting means that the program on certain input will accept it and halt or reject it and it would go into an infinite loop. Basically halting means terminating. So the answer of this question is no we Cannot design a generalized algorithm which can say that given a program will halt or not? The only way is to run the program and check whether it halts or not

Proof by Contradiction.

Let's assume that we can design that kind of machine called HM(P, I) where HM is the machine/program, P and I is the input on the basis of it machine will tell that the P either halts or not.

Now let is construct a new program paradox. ~~that takes~~ ~~that passed to HM, it will return either true or false~~ HM is called in paradox program.

```
HM(P, I)
{
  Halt
  or
  may not Halt
}
```

```
paradox (P)
{
  if (HM(x, x) == Halt)
  { loop forever
  } // it means it will never halt
  else
    return
  } // it will never halt
  because of the above non halting condition.
```

So the both condition is non halting. So this is the contradiction and we can say that our assumption was wrong that is why this problem is undecidable.

$A_{TM} = \{(M, w) \mid M \text{ is a TM and } M \text{ halts at input } w\}$

A_{TM} represents the set of all pairs (M, w) where M is a Turing machine and M halts when given input w. This set is directly related to the halting problem.

Since the halting problem is undecidable, it follows that A_{TM} is an undecidable language. This means that there is no algorithm that can definitively determine, for any given pair (M, w) whether M will halt on input w.

* Simplification of CFG

(i) Elimination of Null (ϵ) Production.

$S \rightarrow aS \mid A$

$A \rightarrow \epsilon$

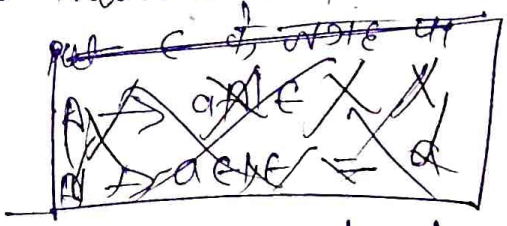
Soln: remove A

- if starting String S containing null (ϵ) we can not remove it

$L(G) = \epsilon$

- $S \rightarrow ABC$
  $A \rightarrow aA \mid \epsilon$
  $B \rightarrow bB \mid \epsilon$
  $C \rightarrow c$

Soln: Nullable variable = A, B



$S \rightarrow ABC \mid BC \mid AC \mid C$

$A \rightarrow aA \mid a$ remove B because we can not reach B and also C.

(PYQ) given grammar reduce it has no unit production.

(ii) Elimination of unit production:

unit production:

$\alpha \rightarrow B$

$\alpha, B \in V$ (Variable)

$S \rightarrow AB \mid E \rightarrow a$

$A \rightarrow a$

$B \rightarrow a \mid b$

$C \rightarrow D$

$D \rightarrow E$

Let's

Co

r

Sol^n:

$B \to C$
$C \to D$
$D \to E$
] Unit P

Put

If

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

Put value of E in D
D in C C in B

New:

$E \to a$
$D \to a$
$C \to a$
$B \to a|b$
$A \to a$
$S \to AB$ { reachable only AB
not C and D and E}
then remove
C D E

after removing C, D, E

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

any

(PYQ) Explain chomsky normal form with an example.

Ans ⇒ Chomsky Normal form (CNF) is a way to represent context free grammar. A grammar is in CNF if every production rule is of one of the following forms:

(i) $A \to BC$ where A, B, C are non-terminal
(ii) $A \to a$ where A is non terminal and a is terminal
(iii) $S \to \epsilon$ only allowed if ϵ is in language and S is the start symbol.

Example:

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

Remove null production

Remove $A \to \epsilon$ to $A \to a$

Now, $S \to AB|B|b$ $A \to a$, $B \to b$

Remove all unit production

$S \to B$ to

$S \to b$

Now, $S \to AB|b$

$A \to a$

$B \to b$

chomsky normal form (CNF)

* Conversion of CFG to Chomsky normal form

Convert the following CFG to CNF:

P: $S \to ASA|aB, A \to B|S, B \to b|\epsilon$

Steps to convert a given CFG to chomsky normal form.

Step1: If the start symbol occurs on the right side, then create a new start symbol S' and a new production $S' \to S$

Step2: Remove Null production.

Step3: Remove unit production.

Step4: Replace each production $A \to ABB$ $\{n>2\}$

with let $P = AB$

$A \to PB$

] repeate this step for all production have more than two symbols on right side

Step5: If the right side of any production is in the form $A \to aB$ where 'a' is a terminal and A and B are non-terminals, then the production is replaced by $A \to XB$ where $X \to a$.

Q P: $S \to ASA|aB, A \to B|S, B \to b|\epsilon$

(1) since, S appears in RHS, we add a new state S'

and $S' \to S$

P: $S' \to S,\ S \to ASA|aB,\ A \to B|S,\ B \to b|\epsilon$

(2) Remove the null productions: $B \to \epsilon$ and $A \to \epsilon$

After removing $B \to \epsilon$: $S' \to S,\ S \to ASA|aB|a,\ A \to B|S|\epsilon,\ B \to b$

After removing $A \to \epsilon$: $S' \to S,\ S \to ASA|aB|a|SA|AS|S,\ A \to B|S,\ B \to b$

(3) Remove unit productions: $S \to S,\ S' \to S,\ A \to B,\ A \to S$

After removing $S \to S$:

$S' \to S$

$S \to ASA|aB|a|SA|AS$

$A \to B|S$

$B \to b$

After removing $S' \to S$

$S' \to ASA|aB|a|SA|AS$

$S \to ASA|aB|a|SA|AS$

$A \to B|S$

$B \to b$

After removing $A \to B$

$S' \to ASA|aB|a|SA|AS$

$S \to ASA|aB|a|SA|AS$

$A \to b|S$

$B \to b$

After removing $A \to S$

$S' \to ASA|aB|a|AS|SA$

$S \to ASA|aB|a|AS|SA$

$A \to b|ASA|aB|a|AS|SA$

$B \to b$

(4) Now find out the product that has more than two variables

$S' \to ASA,\ S \to ASA,\ A \to ASA \qquad X \to SA$

After removing that $\to$

$S' \to AX|aB|a|AS|SA$

$S \to AX|aB|a|AS|SA$

$A \to b|AX|aB|a|AS|SA$

$B \to b$

$X \to SA$

(5) Now change the production $S' \to aB,\ S \to aB,\ A \to aB$

P: $S' \to AX|YB|a|AS|SA$

$S \to AX|YB|a|AS|SA$

$A \to b|AX|YB|a|AS|SA$

$X \to SA$

$Y \to a$

Chomsky Normal form

(PYQ) Design a Turing machine for the Addition of 2 numbers

$a = 4$, $b = 2$ ans $= 6$

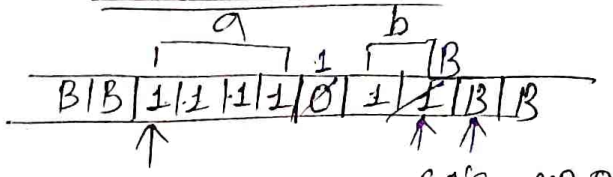
unary representation

$a = 1111$

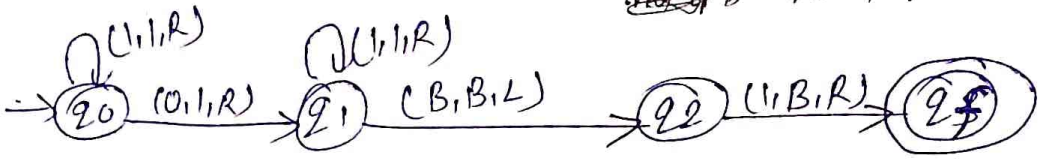
$b = 11$

0 → Seperator

Input tape



no of 1's = 6



no of a's = 6 ans = 6

Greibach normal form

PYQ 12M

* Greibach normal form: A CFG is an Greibach Normal form if the productions are in the following forms:

$A \to b$

$A \to bC_1C_2 \ldots C_n$

where A, C_1, C_2 are Non-Terminals and b is a Terminal.

Steps to convert CFG to GNF

Step1: check if the given CFG has any unit productions or NULL productions and Remove if there are any.

Step2: check whether the CFG is already in chomsky normal form and convert it to CNF if it is not.

Step3: Change the names of the Non-Terminal Symbols into some A_i in ascending order of i.

Step4: If the production is of the form

$A_i \longrightarrow A_j x$ then

$i < j$ and never be $i \geq j$

Step5: Remove left recursion by introducing a new variable to remove left recursion.

Eg ⇒ · S → CA | BB (pyq 12m)

B → b | SB

C → b

A → a

Step 1 & 2: No null/unit production and it is in form CNF.

Step 3: Change the names of non terminal symbols into A_i

S with A1

C with A2

A with A3

B with A4

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step 4: If the production P8 of the form $A_i \rightarrow A_jx$, then $i<j$ and never be $i \geq j$

$A_4 \rightarrow b \mid A_1A_4$

$A_4 \rightarrow b \mid A_2A_3A_4 \mid A_4A_4A_4$ { $A_1 \rightarrow$ put value } → put value of $A_2 \rightarrow b$

A_4 produces itself left recursion

Step 5: Remove left recursion by introduce a new variable to remove the left Recursion.

$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$

$Z \rightarrow A_4A_4Z \mid A_4A_4$

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

Now the grammar is:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$ → look A_2A_4 starting will variable at first terminal

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$A_2 \rightarrow b$, $A_3 \rightarrow a$ $Z \rightarrow A_4A_4 \mid A_4A_4Z$

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$ { $A_2 \rightarrow b$ putting, $A_4 \rightarrow$ putting }

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \rightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid$ { putting value of A_4 again here

$bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

$A_2 \rightarrow b$, $A_3 \rightarrow a$ { Greibach normal form }

(PYQ) Formulate a grammar in Greibach normal for equivalent to grammar

$S \rightarrow AA \mid a$

$A \rightarrow SS \mid b$

Step 1: check if given grammar has any null or unit productions if yes then remove.

No null or unit production there.

Step 2: it is also in normal (CNF)

Step 3: change the names of the non-terminal symbols into A_i in ascending order of i.

$A_1 \rightarrow A_2A_2 \mid a$

$A_2 \rightarrow A_1A_1 \mid b$

$S \rightarrow A_1$

$A \rightarrow A_2$

Step 4: if the production is of the form $A_i \rightarrow A_jx$ then $i < j$

$A_2 \rightarrow A_1A_1 \mid b$ } put value of A_1

$A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$ } $A \rightarrow A\alpha \mid \beta$

Left recursion

Step 5: remove left recursion introducing a new variable z

$Z \rightarrow A_2A_1Z \mid A_2A_1$

$A_2 \rightarrow aA_1 \mid b \mid aA_1Z \mid bZ$

Now the grammar is

$A_1 \rightarrow A_2A_2 \mid a$ ——— (1)

$A_2 \rightarrow aA_1 \mid b \mid aA_1Z \mid bZ$ ——— (2)

$Z \rightarrow A_2A_1Z \mid A_2A_1$ ——— (3) put value of A_2 in eq^n (1)

$A_1 \rightarrow qA_1A_2 / bA_2 / qA_1Z_1A_2 / bZ_2A_2 / q$

again put value of A_2 in eq (3)

$Z \rightarrow qA_1A_1Z / bA_1Z / qA_1ZA_1Z / bZA_1Z /$

$qA_1A_1 / bA_1 / qA_1ZA_1 / bZA_1$

Now the grammar is:

$A_1 \rightarrow qA_1A_2 / bA_2 / qA_1ZA_2 / bZA_2 / q$ —— (1)

$A_2 \rightarrow qA_1 / b / qA_1Z / bz$ —— (2)

$Z \rightarrow qA_1A_1Z / bA_1Z / qA_1ZA_1Z / bZA_1Z /$

$qA_1A_1 / bA_1 / qA_1ZA_1 / bZA_1$ in Greibach normal form.

(Py①) CFG to GNF

$S \rightarrow AB$

$A \rightarrow BS / b$

$B \rightarrow SA / a$

Step1: There is no any null and unit production.

Step2: productions are also in CNF.

Step3: $A_1 \rightarrow A_2A_3$ $\quad S \rightarrow A_1$

$A_2 \rightarrow A_3A_1 / b$ $\quad A \rightarrow A_2$

$A_3 \rightarrow A_1A_2 / a$ $\quad B \rightarrow A_3$

Step4: $A_3 \rightarrow A_1A_2 / a$ {put value of A_1} $A_i \rightarrow A_jx$, $i < j$

$A_3 \rightarrow A_2A_3A_2 / a$ put value of A_2

$A_3 \rightarrow A_3A_1A_2A_2 / bA_3A_2 / a$

left recursion

Step5: introduce a variable Z and remove left recursion

$Z \rightarrow A_1A_3A_2Z \mid A_1A_3A_2$

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$

Now the grammar:

$A_1 \rightarrow A_2A_3$ —①

$A_2 \rightarrow A_3A_1 \mid b$ —②

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$ —③

$Z \rightarrow A_1A_3A_2Z \mid A_1A_3A_2$ —④

put eq ③ in eq ②

Now

$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2ZA_1 \mid aZA_1 \mid b$ —⑤

put eq ⑤ in ①

$A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2ZA_1A_3 \mid aZA_1A_3 \mid bA_3$ —⑥

—⑥

Put eq ⑥ in ④

$Z \rightarrow bA_3A_2A_1A_3A_3A_2Z \mid aA_1A_3A_3A_2Z \mid bA_3A_2ZA_1A_3A_3A_2Z \mid$

$aZA_1A_3A_3A_2Z \mid bA_3A_3 \ldots A_2Z \mid \ldots$

$\ldots bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2ZA_1A_3A_3A_2 \mid$

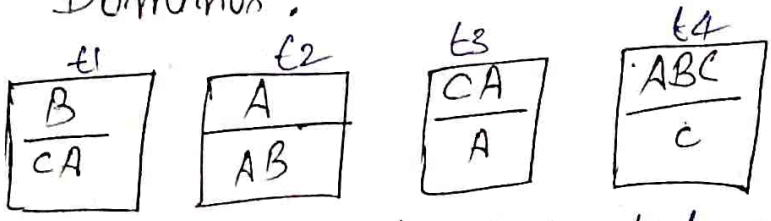
$aZA_1A_3A_3A_2 \mid bA_3 \ldots A_3A_2 \ldots$

Now the grammar is in Greibach normal form.

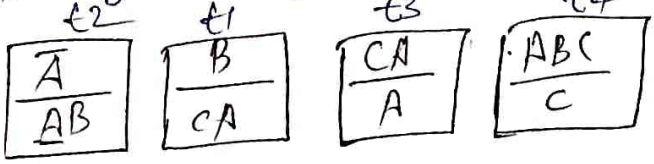
Now, check double side all are in GNF

PYQ Post correspondence problem: Post correspondence problem is a popular undecidable problem that was introduced by Emil Leon in 1946. It is similar simpler than Halting problem. In this problem we have N number of Dominos (tiles). The aim is to arrange tiles in such order that string made by numerators is same as string made by Denominators.

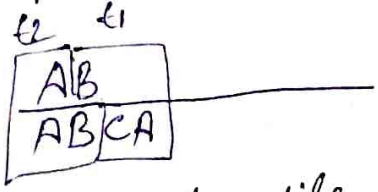
Ex→ Dominos:



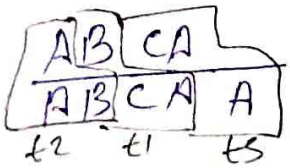
Step1: We will start with tile in which numerator and denominator are starting with same number/alphabet (t2)



Step2: We need a tile in which numerator containg B (t1)

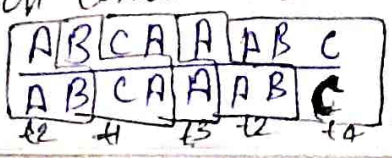


Step3: We need a tile which containg CA in numerator (t3)



Step3: We need a tile in which containg A in numerator (t2)

Step4: Again we need a tile which contain AB in numerator (t4)



Look numerator string is same as Denominator string.

Undecidability: The post correspondence problem is known to be undecidable, meaning there is no general algorithm that can solve this problem. This was proven by showing that if we could solve the PCP, we could also solve the halting problem, which is known to be undecidable.

PYQ Compare LBA and context sensitive language

PYQ Discuss in detail Linear Bounded Automata.

ans ⇒ A Linear Bounded is a type of Turing machine but with a bounded finite length of the tape. It is more powerful than pushdown automat but less powerful than Turing machine. It is a type machine. It accept context sensitive language.

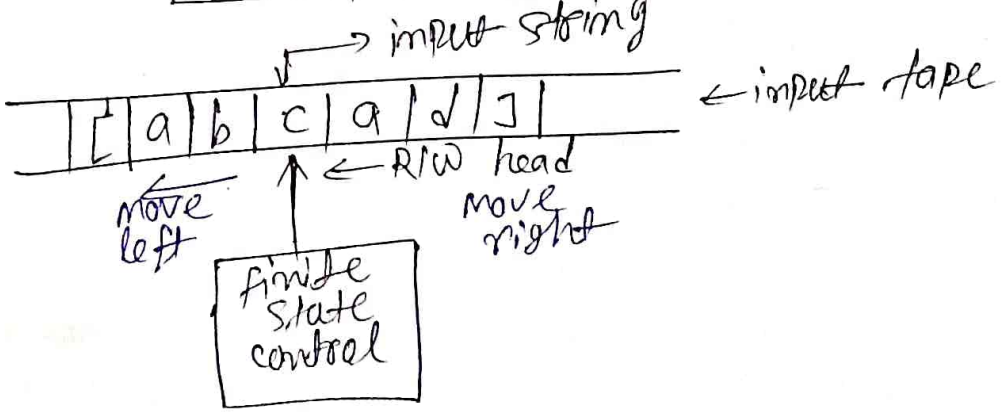
LBA = {Q, Σ, Γ, F, [,], δ, $_0$}

Q → finite set of states	F → set of final states
Σ → input alphabet	[→ left end marker
Γ → tape alphabet	] → Right end marker
q_0 → initial state	δ → transition function.

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 0, +1\}$ where -1, 0, +1 indicate movement of the tape.

Ex ⇒ $L = \{a^n b^n c^n \mid n \geq 1\}$ Language accepted by LBA.

Model of LBA



(19M) Compare linear Bound automata and context-sensitive language.

① Definition: Linear Bounded Automata (LBA): LBA is a type of Turing machine with a bounded finite length of tape. It accept context sensitive language. It is more powerful than automata but less than Turing Machine.

Context-Sensitive language (CSL): A context sensitive language is a type of formal language that can be generated by a context sensitive grammar. The production rule is

$\alpha A \beta \rightarrow \alpha \gamma \beta$ where A is non-terminal, γ is a string of terminals and/or non-terminals

② Relationship: LBA: Every Context-Sensitive language is recognized by a Linear Bounded Automata.

CSL: Context-Sensitive grammars define context sensitive languages. Every CSL can be recognized by LBA.

③ Grammar and Representation:

LBA: Operates based on states, tape and transitions within linear tape bounds.

CSL: Uses context sensitive grammars, which are more powerful than context-free grammar.

④ Application: LBA: practical for languages requiring space-efficient computation.

CSL: Defines computational problems requiring more power than context free languages.

⑤ Expressive power: Both are equivalent in expressive power recognizing exactly the same class of languages.

⑥ Tape usage:

LBA: Tape size is linearly proportional to input length.

CFG: No direct concept of tape.

⑦ Example: $L = \{ a^m b^m c^m \mid m \geq 1 \}$

this example is recognized by both languages