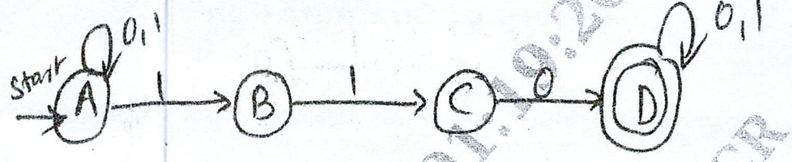
		OR			
Q.4	**a.**	State and prove pumping theorem for regular languages. Show that the language $L = \{a^n b^n \mid n \geq o\}$ is not regular.	**06**	**L3**	**CO2**
	b.	Convert the following FA to RE using state elimination method. 	**06**	**L3**	**CO2**
	c.	Prove that the Regular Languages are closed under : i) Union ii) Complementation iii) Intersection iv) Difference	**08**	**L3**	**CO2**
		Module – 3			
Q.5	**a.**	Design CFG for the following Languages. i) $L = \{ ww^R$ where $W \in \{a, b\}^* \}$ ii) $L = \{ 0^m 1^m 2^n \mid m \geq 1$ and $n \geq 0 \}$ iii) $L = \{ a^n b^{n+3} \mid n \geq 0 \}$	**10**	**L3**	**CO3**
	b.	Consider the grammar G with productions $S \rightarrow aSbS \mid bSaS \mid \in$ Obtain LMD, RMD and parse tree for the string aababb. Is the grammar ambiguous.	**10**	**L3**	**CO3**
		OR			
Q.6	**a.**	Obtain PDA to accept the language $L = \{ wCw^R \mid W \in (a + b)^* \}$ and show the moves made by the PDA for the string aabCbaa.	**10**	**L3**	**CO3**
	b.	Convert the following CFG to PDA $S \rightarrow aABC$ $A \rightarrow aB \mid a$ $B \rightarrow bA \mid b$ $C \rightarrow a$	**10**	**L3**	**CO3**
		Module – 4			
Q.7	**a.**	Define CNF. Convert the following CFG to CNF $E \rightarrow E + T/T$ $T \rightarrow T * F/F$ $F \rightarrow (E) / I$ $I \rightarrow Ia / Ib \mid a \mid b$	**10**	**L3**	**CO4**
	b.	State and prove pumping lemma for context Free Grammars. Show that $L = \{ 0^n 1^n 2^n \mid n \geq 1 \}$ is not content free.	**10**	**L3**	**CO4**
		OR			
Q.8	**a.**	Define CNF convert the following CFG to CNF $S \rightarrow AB$ $A \rightarrow aA \mid bB \mid b$ $B \rightarrow b$	**10**	**L3**	**CO4**
	b.	Prove that the content free languages are closed under i) Union ii) Concatenation iii) Homomorphism	**10**	**L3**	**CO4**
		Module – 5			
Q.9	**a.**	Define a Turing Machine. Explain the working of a basic Turing machine with neat diagram.	**08**	**L2**	**CO5**
	b.	Design a Turing Machine to accept the language $L = \{ a^n b^n c^n \mid n \geq 1 \}$. Draw the transition diagram and show the moves made by TM for the string : aabbcc.	**12**	**L3**	**CO5**
		OR			
Q.10	**a.**	What are the programming Techniques for Turing Machine. Explain.	**10**	**L2**	**CO5**
	b.	Write short notes on : i) Multi Tape Turing Machine ii) Non Deterministic Turing Machine	**10**	**L2**	**CO5**

CBCS SCHEME

USN | | | | | | | | | | |

BCS503

Fifth Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Theory of Computation

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	**a.**	Define the following with example. i) Alphabet ii) String iii) Power of an Alphabet iv) Language v) Problem	**05**	**L2**	**CO1**
	b.	Design DFA for the following languages. i) DFA to accept string of 0's, 1's and 2's beginning with a '0' followed by odd number of 1's and ending with a '2'. ii) $L = \{ W \in \{0,1\}^* : w$ does not have 001 as a substring$\}$	**08**	**L3**	**CO1**
	c.	Convert the following NFA to DFA. 	**07**	**L3**	**CO1**
		OR			
Q.2	**a.**	Design an NFA to recognize the following set of strings 0101, 101 and 011.	**05**	**L2**	**CO1**
	b.	Design an DFA to accept binary numbers divisible by 5.	**08**	**L3**	**CO1**
	c.	Obtain DFA for the following $\in$ – NFA δ | $\in$ | a | b | c $\rightarrow p$ | ϕ | {p} | {q} | {r} q | {p} | {q} | {r} | ϕ *r | {q} | {r} | ϕ | {p}	**07**	**L3**	**CO1**
		Module – 2			
Q.3	**a.**	Write the Regular Expression for the following languages. i) $L = \{a^n b^m \mid n >= 4, m <= 3\}$ ii) RE to accept words with two or more letters but beginning and ending with the same letter where $\sum = \{a, b\}$	**06**	**L3**	**CO2**
	b.	Obtain $\in$ – NFA for given Regular Expression $(a + b)^* aa (a + b)^*$	**06**	**L3**	**CO2**
	c.	Find the minimized DFA of the following δ | a | b $\rightarrow A$ | B | F B | G | C *C | A | C D | C | G E | H | F F | C | G G | G | E H | G | C	**08**	**L3**	**CO2**

Fifth Semester B.E Degree Examination Dec 2025-2026

Theory of Computation

SOLUTION

1a. Define the following i)Alphabet ii) String iii) power of an alphabet,iv)language v) Problem

An alphabet is a finite non empty set of symbols, which used to represent the input of a machine. Alphabets are typically thought of as represented by letters, characters, digits, signs, punctuation, etc. Conventionally we use the symbol ∑ for an alphabet. Common alphabets include:

- ∑ = {0, 1}: The binary alphabets.

A string is a finite ordered sequence of symbols chosen form some set of alphabet or ∑. For example, 'aababbbbaa' is a valid string from the alphabet ∑ = {a, b}, similarly '001111000101' is a valid string form the alphabet ∑ = {0, 1}.

If ∑ is an alphabet, we can express the set of all strings of a certain length form that alphabet by using an exponential notation. We denoted it $\sum^n$ to be the set of strings of length n, each of whose symbols is in given input alphabet ∑.

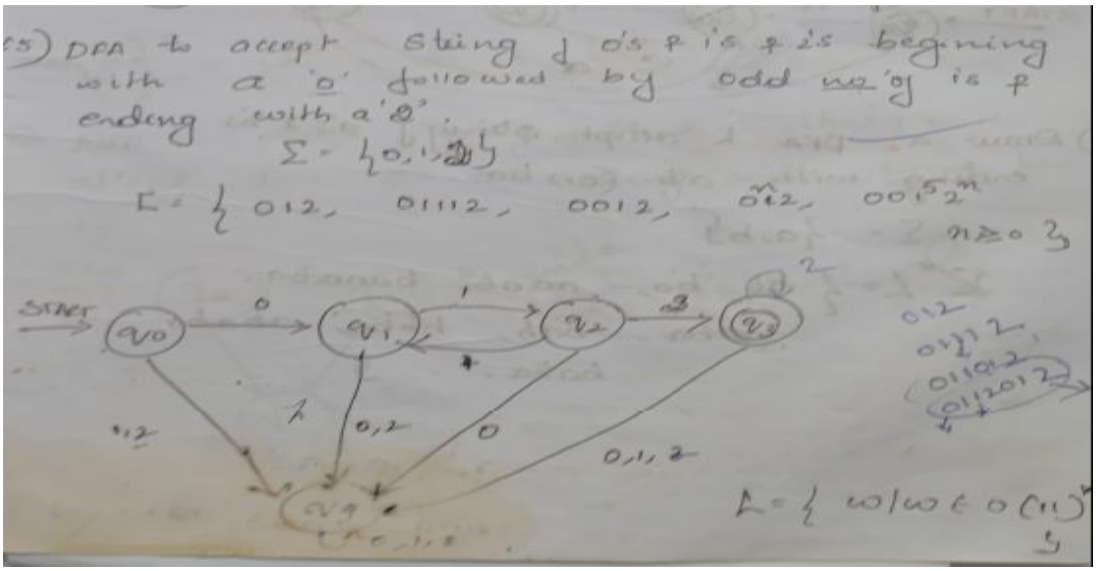
For example: $\sum^0 = \{\varepsilon\}$ regardless of what alphabet ∑ is, i.e. ε is the only string whose length is zero.

Language (L): A set of strings that adhere to specific rules or patterns. For example, L = all even-length binary numbers where Σ = {0, 1}.

Languages can be defined by their structure, such as L = {w ∈ Σ* | w has equal numbers of 0s and 1s}.

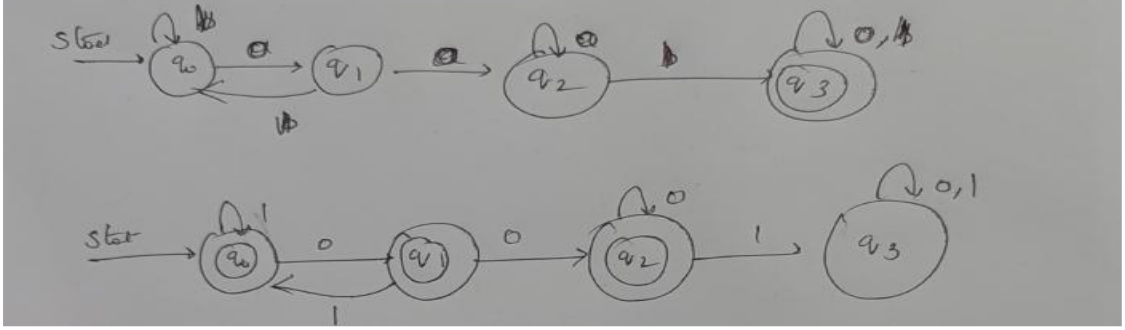
b.	Design DFA for the following languages. i) DFA to accept string of 0's, 1's and 2's beginning with a '0' followed by odd number of 1's and ending with a '2'. ii) L = { W∈ {0,1}* : w does not have 001 as a substring}	08	L3	CO1

Solution

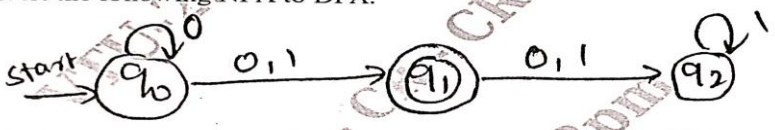


1 (b)

(ii) $L = \{w \in \{0,1\}^* : w$ does not have 001 as substring.



c.	Convert the following NFA to DFA.	07	L3	CO1

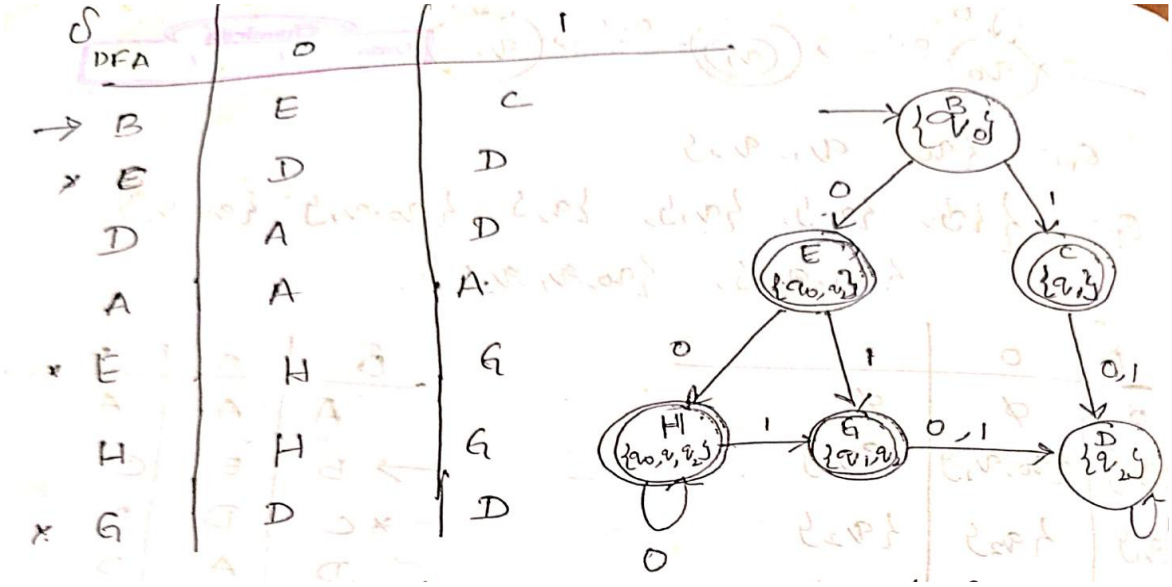


$Q_N = \{q_0, q_1, q_2\}$

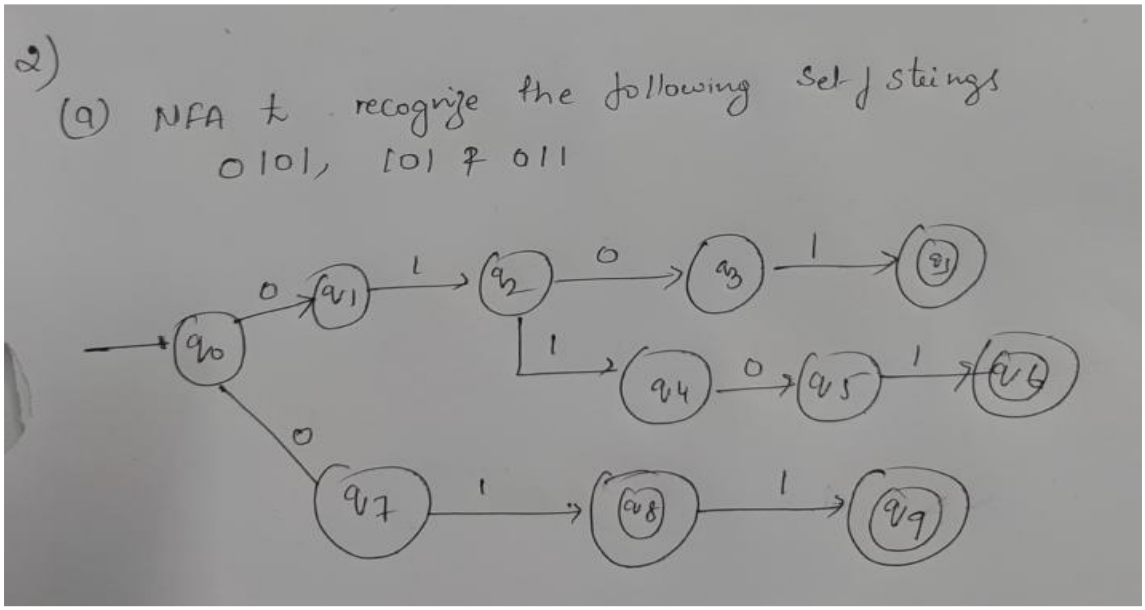
$Q_D = \{\{\}, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$.

δ	0	1
A ϕ	ϕ	ϕ
B $\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_1\}$
* C $\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	ϕ	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$.

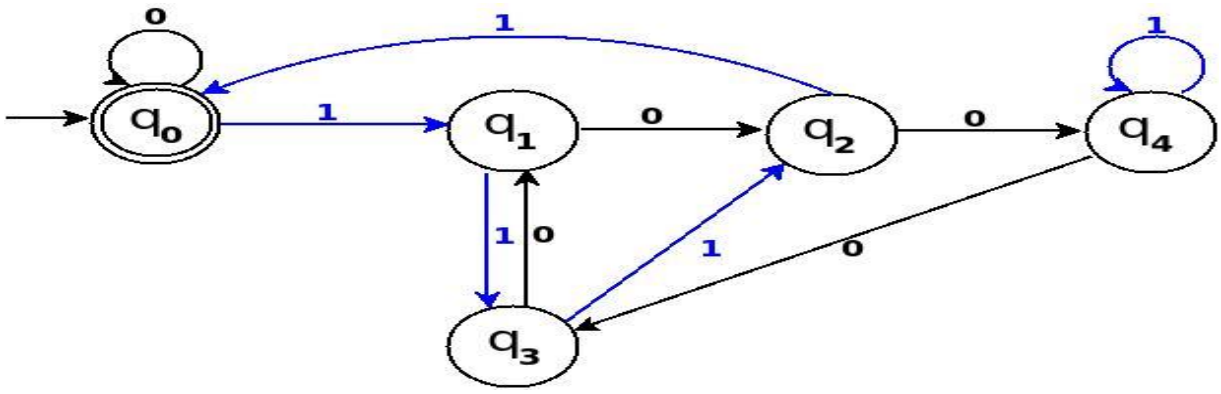
δ	0	1
A	A	A
$\rightarrow$ B	E	C
* C	D	D
D	A	D
* E	H	G
F	E	G
* G	D	D
H	H	G.



Q.2	a.	Design an NFA to recognize the following set of strings 0101, 101 and 011.	05	L2	CO1



2b. **b.** Design an DFA to accept binary numbers divisible by 5.



c. Obtain DFA for the following ∈ – NFA

δ	∈	a	b	c
→p	ϕ	{p}	{q}	{r}
q	{p}	{q}	{r}	ϕ
*r	{q}	{r}	ϕ	{p}

07 | L3 | CO1

ε-closure means: state itself + all states reachable using only ε

- ε-closure(p) = { p }
- ε-closure(q) = { q, p }
- ε-closure(r) = { r, q, p }

From A = {p}

- on a: move({p}, a) = {p} → ε-closure = {p}
- on b: move({p}, b) = {q} → ε-closure = {p, q}
- on c: move({p}, c) = {r} → ε-closure = {p, q, r}

So new DFA states:

- B = {p, q}
- C = {p, q, r}

From B = {p, q}

- on a: {p} ∪ {q} = {p, q} → ε-closure = {p, q}
- on b: {q} ∪ {r} = {q, r} → ε-closure = {p, q, r}
- on c: {r} ∪ φ = {r} → ε-closure = {p, q, r}

From C = {p, q, r}

- on a: {p} ∪ {q} ∪ {r} = {p, q, r}
- on b: {q} ∪ {r} ∪ φ = {q, r} → ε-closure = {p, q, r}
- on c: {r} ∪ φ ∪ {p} = {p, r} → ε-closure = {p, q, r}

Step 4: Identify final states
Any DFA state containing r is a final state.

- Final DFA state: C = {p, q, r}

DFA State	a	b	c
→ A = {p}	A	B	C
B = {p, q}	B	C	C
* C = {p, q, r}	C	C	C

3.a) Write Regular Expression for the following Languages

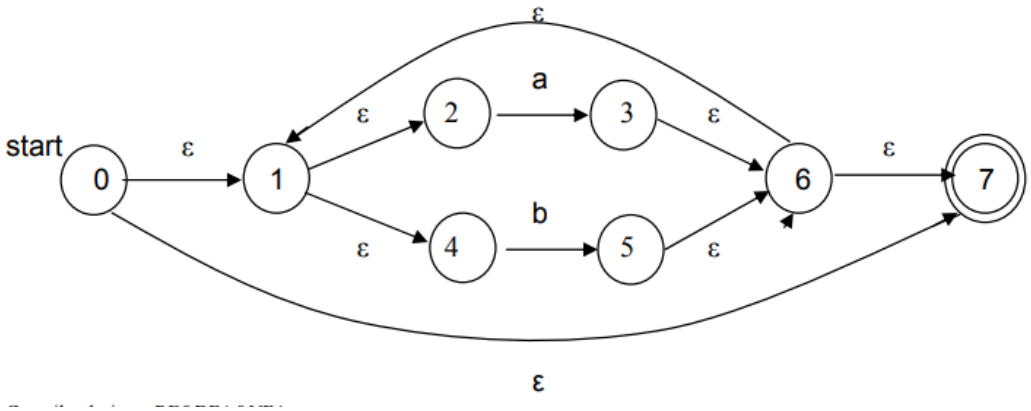
i) $L=\{a^n b^m \mid n>=4, m<=3\}$

ii) RE to accept words with two or more letters but beginning and ending with same letter where ∑={a,b}

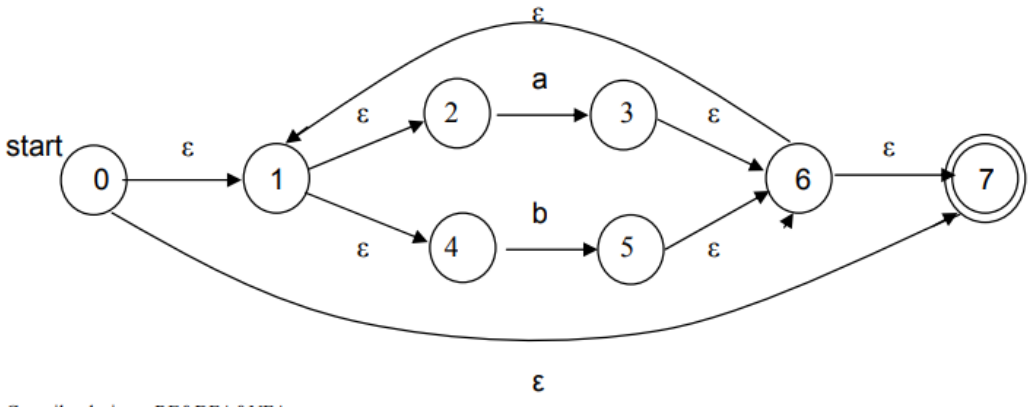
solution :i) aaaaa*(€+b+bb+bbb)

ii) a(a|b)*a | b(a|b)*b

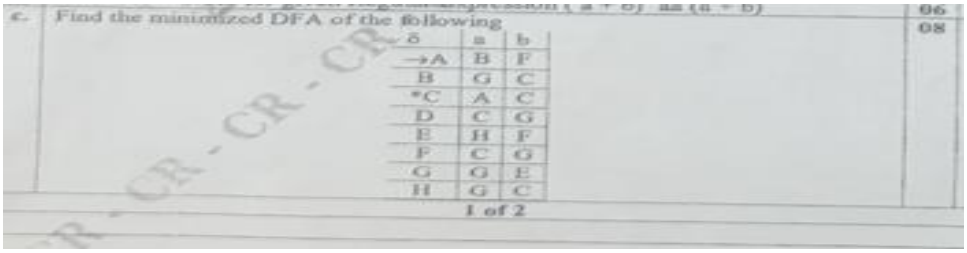
3b) Obtain €-NFA for the given Regular Expression (a+b)* aa(a+b)*

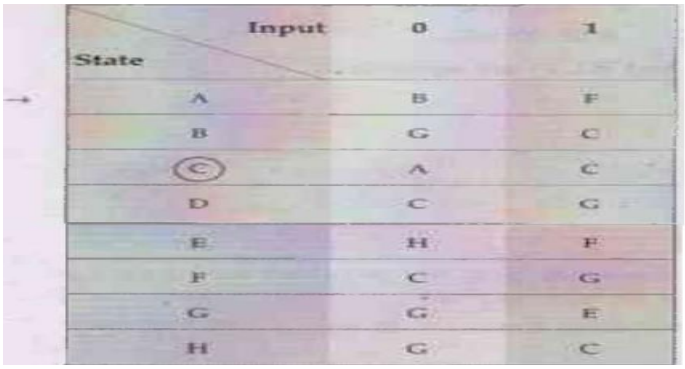


Concatenate a a and join with below one

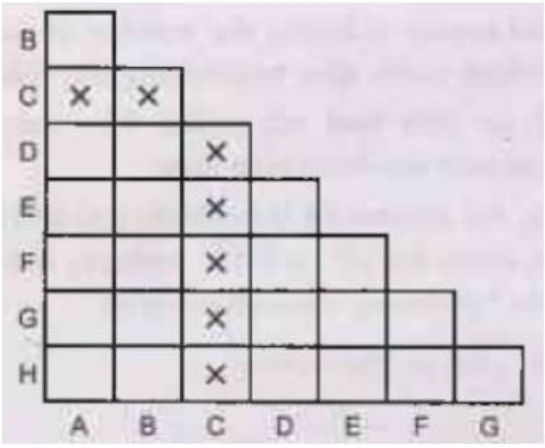


3c. Find the Minimized DFA for the following





Step 1: We will build a table in following format for finding equivalent states and mark final and non-final states with x



Step 2: The states that need to be marked are (A, B), (A, D), (A, E), (A, F), (A, G), (A, H), (B, D), (B, E), (B, F), (B, G), (B, H), (D, E), (D, F), (D, G), (D, H), (E, F), (E, G), (E, H), (F, G), (F, H), (G, H).

If we observe transition table then

δ (B, 0) = G

δ (B, 1) = C

δ (H, 0) = G

δ (H, 1) = C

Similarly

δ (D, 0) = C

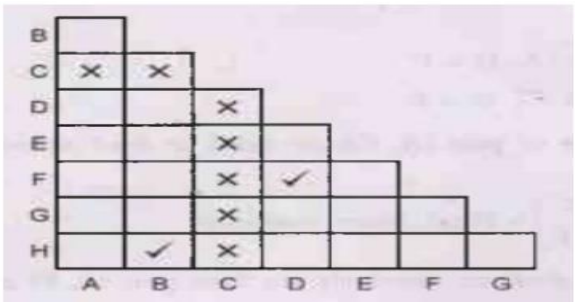
δ (D, 1) = G

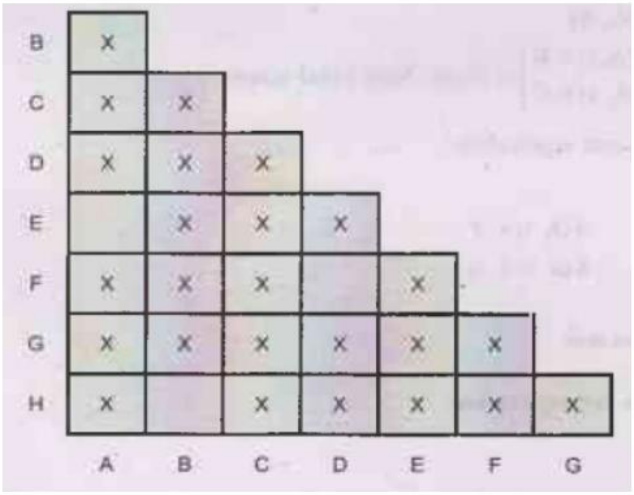
δ (F, 0) = C

(F, 1) = G

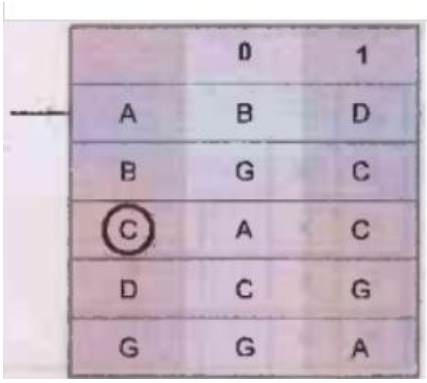
Thus pairs (B, H) and (D, F) are equivalent

Step 3:

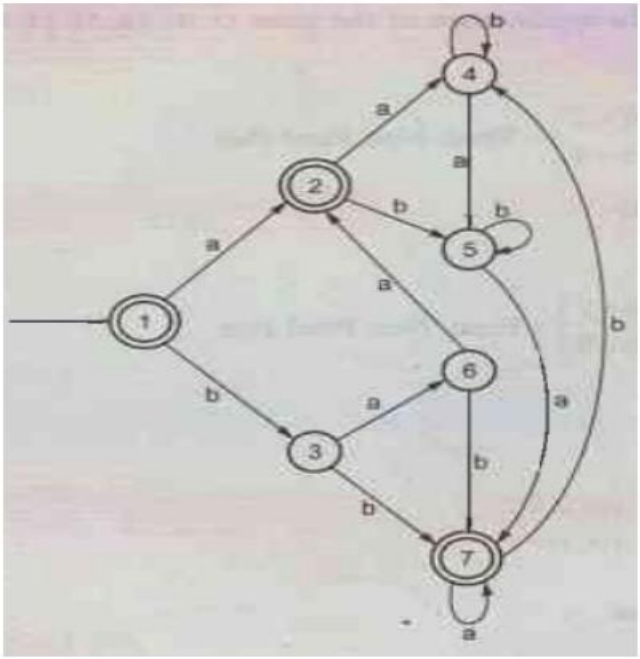




Thus we get equivalent pairs as (A, E), (B, H), (D, F). Hence the minimized DFA will be,



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



4a). State and Prove Pumping Lemma for regular Languages. Show that the Language L= {$a^n b^n$|n>=0 is not regular}

The Pumping Lemma can be formally stated as follows −

If L is a regular language, then there exists an integer n (the pumping length) such that any string w in L with | w | ≥ n can be decomposed into three parts, w = xyz, satisfying the following conditions −

|xy| ≤n |xy|,

|y|>0,

xyiz∈L for all i≥0

These conditions states that for any sufficiently long string in a regular language, there is a section of the string that can be "pumped" (repeated or removed) to produce new strings that also belong to the language.

rove that L = {a^n b^n | n ≥ 1} is not regular.

Solution

Assume the set L is regular. Let n be the number of states of the FA accepting the set L.

Let w = a^n b^n, where |w| = 2^n. By the Pumping Lemma, we can write w = xyz with |xy| ≤ n and |y| > 0.

We want to find a suitable *i* so that $xy^iz \notin$ L.

The string **y** can be one of the following −

- y is a string of only 'a's, so y = a^k for some k ≥ 1.
- y is a string of only 'b's, so y = b^k for some k ≥ 1.
- y is a string of both 'a's and 'b's, so, y = a^k b^l for some k, l ≥ 1.

For case (i), take *i* = 0. As xyz = a^n b^n, xy^0 z = xz will be a^{n-k} b^n.

Ask≥1,(n−k)≠n,soxy0z∉L

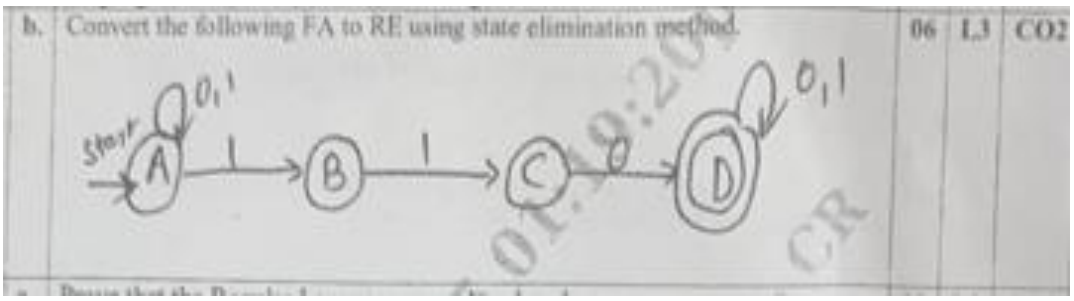
For case (ii), take *i* = 0. As xyz = a^n b^n, xy^0 z = xz will be a^n b^{n-k}.

Ask≥1,(n−k)≠n,soxy0z∉L

Which is not in the form a^n b^n, so **xy^2 z ∉ L**.

For all three cases, we find a contradiction. Therefore, L is not regular.

4b) Convert the following FA to RE using State elimination method



To State A –(0+1)*

A→ B==1. A→C=11, A→ D=110

RE= (0+1)*110(0+1)*

4.c) Prove that Regular Languages are closed under i) Union, ii) complementation, iii) Intersection iv) Difference

Union

Imagine L1 = {a, aa} and L2 = {b, bb}. The union of L1 and L2 is,

L1∪L2={a,aa,b,bb}

Closure Property − Regular languages are closed under union. We can prove this in two ways –

- Regular Expression − If L1 and L2 are regular, they have regular expressions R1 and R2. The union of L1 and L2 can be expressed as R1 + R2, which is also a regular expression.
- Finite Automata (DFA) − If L1 and L2 are regular, they have DFAs D1 and D2. We can construct a DFA for L1 U L2 by combining D1 and D2, introducing a new start state with epsilon transitions to the start states of D1 and D2. This combined DFA represents L1 U L2, proving that it is regular.
- **Complementation**

- Proof: Regular languages are closed under complementation.
- L1 is a regular language ⇒∃ a DFA M such that L1=L(M).
- Construct DFA M′ such that:
- final states in M are nonfinal states in M′.
- nonfinal states in M are final states in M′.
- w∈L(M′)⟺w∈L¯⇒ closed under complementation.

Intersection

- DeMorgan's Law: L1∩L2=L1∪L2

(2) L1 and L2 are regular languages ⇒∃ DFAs M1 and M2 such that L1=L(M1) and L2=L(M2).
L1=L(M1) and L2=L(M2)
M1=(Q,Σ,δ1,q0,F1)
M2=(Q,Σ,δ2,p0,F2)
The idea is to construct a DFA so that it accepts only if both M1 and M2 accept. There is an algorithm for that.

Difference Operation

The difference of L1 and L2, denoted as L1 - L2, includes all strings that are in L1 but not in L2.
Closure Property − Regular languages are closed under difference.
Proof − We can express L1 - L2 as L1 ∩ L2', where L2' is the complement of L2. We've already established that both intersection and complement preserve regularity. Therefore, L1 - L2 is also regular.

5a) Design CFG for the following Language

i) L={WW^R where w€ {a,b}*}
ii) L={$0^m1^m2^n$ |m>=1 and n>=0}
iii) L={a^nb^{n+3}|n>=0}

i) S → aSa | bSb | ε
ii) S → A B , A → 0A1 | 01 ,B → 2B | ε
iii) S → aSb | bbb

5b) Consider the Grammar G with the productions S→aSbS | bSaS |€,Obtain LMD ,RMD and parse tree for the string aababb, Is the grammar is ambiguous

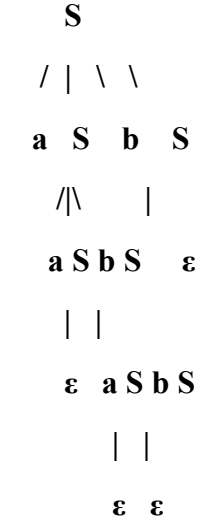
LMD for the String

S
⇒ a S b S
⇒ a a S b S b S
⇒ a a ε b S b S
⇒ a a b S b S
⇒ a a b a S b S
⇒ a a b a ε b S
⇒ a a b a b S
⇒ a a b a b ε

RMD for the String

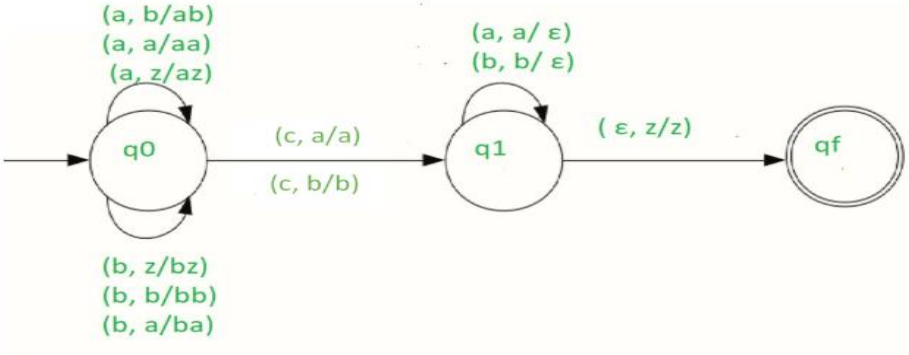
S
⇒ a S b S
⇒ a S b ε
⇒ a S b
⇒ a a S b S b
⇒ a a S b ε b
⇒ a a S b b
⇒ a a ε b b

The Parse Tree is ,The Grammar is ambiguous



6a).Obtain PDA to accept the Language .

L={WCW^R |W€(a+b)* } and show the moves made by the PDA for the String aabCbaa



Stack Transition Functions

$$\delta(q_0, a, z) \vdash (q_0, az)$$
$$\delta(q_0, a, a) \vdash (q_0, aa)$$
$$\delta(q_0, b, z) \vdash (q_0, bz)$$
$$\delta(q_0, b, b) \vdash (q_0, bb)$$
$$\delta(q_0, a, b) \vdash (q_0, ab)$$
$$\delta(q_0, b, a) \vdash (q_0, ba)$$
$$\delta(q_0, c, a) \vdash (q_1, a)$$
$$\delta(q_0, c, b) \vdash (q_1, b)$$
$$\delta(q_1, a, a) \vdash (q_1, \in)$$
$$\delta(q_1, b, b) \vdash (q_1, \in)$$
$$\delta(q_1, \in, z) \vdash (q_f, z)$$

Where, q_0 = Initial state
q_f = Final state
$\in$ = indicates pop operation

For the String aabCbaa

(q0, aabCbaa, Z0)
), abCbaa, aZ0)
), bCbaa, aaZ0)
), Cbaa, baaZ0)
l, baa, baaZ0)
l, aa, aaZ0)
l, a, aZ0)
l, ε, Z0)
, ε, Z0)

6b) Convert the following CFG to PDA

S→aABC, A→aB|a, B→bA|b,C→a

On Non Terminal

$\delta(q, \varepsilon, S) \rightarrow (q, a\ A\ B\ C)$
$\delta(q, \varepsilon, A) \rightarrow (q, a\ B)$
$\delta(q, \varepsilon, A) \rightarrow (q, a)$
$\delta(q, \varepsilon, B) \rightarrow (q, b\ A)$
$\delta(q, \varepsilon, B) \rightarrow (q, b)$
$\delta(q, \varepsilon, C) \rightarrow (q, a)$
On Terminal
$\delta(q, a, a) \rightarrow (q, \varepsilon)$
$\delta(q, b, b) \rightarrow (q, \varepsilon)$

7a.) Define CNF Convert the following CFG to CNF

E→E+T/T, T→T*F/F , F→(E)/ I , I→Ia|Ib|a|b

S → E
E → E X | T M F | L Y | I A | I B | a | b
T → T M F | L Y | I A | I B | a | b
F → L Y | I A | I B | a | b
X → P T
Y → E R
I → I A | I B | a | b
P → +
M → *
L → (
R →)
A → a
B → b

7b)**State and Prove pumping lemma for Context free grammars. Show that $L=0^n1^n2^n|n>=1\}$ is not context free**

If L is a context-free language, there is a pumping length p such that any string w ∈ L of length ≥ p can be written as w = uvxyz, where vy ≠ ε, |vxy| ≤ p, and for all i ≥ 0, $uv^ixy^iz \in L$.

Find out whether the language $\{L=0^n1^n2^n|n>=1\}$ is context free or not.
Solution
Let L is context free. Then, L must satisfy pumping lemma.
At first, choose a number n of the pumping lemma. Then, take z as $0^n1^n2^n$.
Break z into uvwxy, where
|vwx| ≤ n and vx ≠ ε.
Hence vwx cannot involve both 0s and 2s, since the last 0 and the first 2 are at least (n+1) positions apart. There are two cases −
Case 1 − vwx has no 2s. Then vx has only 0s and 1s. Then uwy, which would have to be in L, has n 2s, but fewer than n 0s or 1s.
Case 2 − vwx has no 0s.
Here contradiction occurs.
Hence, L is not a context-free language.

8.a) Define CNF, convert the following CFG to CNF

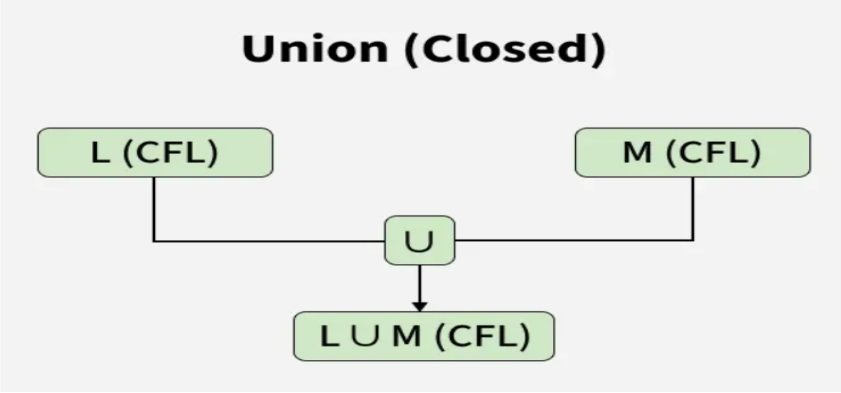
S→AB , A→aA|bB|b , B→b

A Context-Free Grammar (CFG) is said to be in Chomsky Normal Form (CNF) if every production is of one of the following forms:
A → BC
where A, B, C are non-terminals
A → a
where a is a terminal
S → ε (optional)
only if the start symbol S can derive the empty string
Final Grammar is

S → AB
A → CA | DB | b
B → b
C → a
D → b

8.b) Prove that the context free languages are closed under i)union, ii)concatenation iii) Homomorphism

CFLs are closed under the following operations:
1. Union
If L and M are two context-free languages, then their union L ∪ M is also a CFL.
Construction:
1. Consider two context-free grammars, G and H, for L and M respectively.
2. Assume that G and H have no common variables (this can be ensured by renaming variables if needed).
3. Introduce a new start symbol S and add the rule: S → S_1 | S_2 Here, S_1 and S_2 are the start symbols of G and H, respectively.
4. The resulting grammar generates L ∪ M, proving that CFLs are closed under union.



Example: Let L_1 = { $a^n b^n c^m$ | m ≥ 0, n ≥ 0 } and L_2 = { $a^n b^m c^m$ | n ≥ 0, m ≥ 0 }.

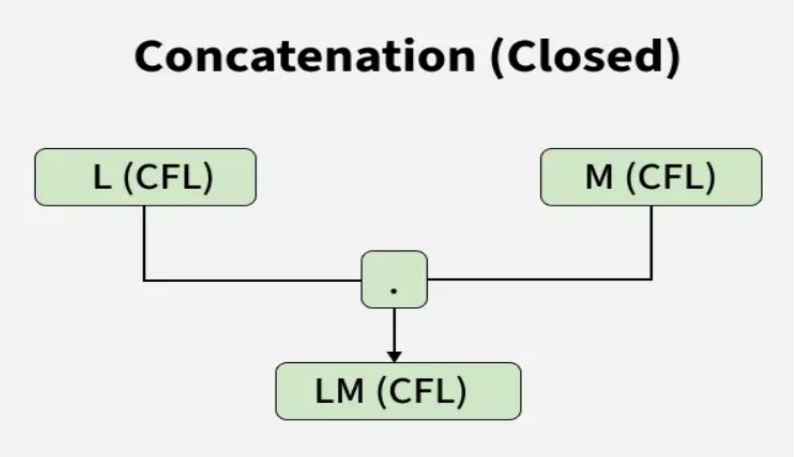
- *L_1 enforces that the number of a's equals the number of b's.*
- *L_2 enforces that the number of b's equals the number of c's.*

- *Their union states that either of these conditions must be satisfied, making the resulting language context-free.*

Note: So CFL are closed under Union.

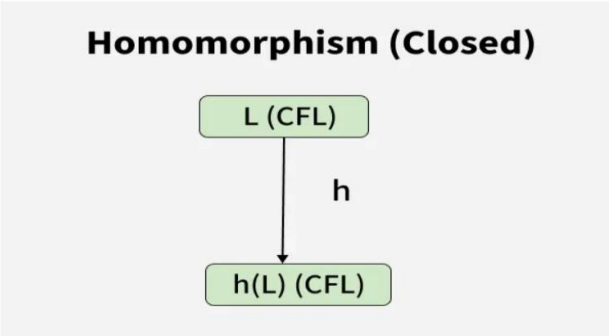
2. Concatenation

If L and M are CFLs, then their concatenation LM is also a CFL.
Construction:
1. Let G and H be the context-free grammars for L and M, respectively.
2. Assume that G and H have no common variables.
3. Introduce a new start symbol S and add the production: $S \rightarrow S_1 S_2$ Here, S_1 and S_2 are the start symbols of G and H, respectively.
4. This ensures that any derivation from S will first generate a string in L and then a string in M, proving that CFLs are closed under concatenation.



homomorphism

A homomorphism is a function that replaces each symbol in a string with another string.
If **L** is a CFL and **h** is a homomorphism, then **h(L)** is also a CFL.



9.a) Define Turing Machine ,Explain the working of basic Turing machine with neat diagram

- A Turing Machine consists of an infinite tape, a read/write head, and a set of rules that determine how it reads, writes, and moves on the tape. Despite its simplicity, a TM can simulate any real-world computation, making it as powerful as modern computers but with infinite memory.

- The Turing machine's behavior is determined by a finite state machine, which consists of a finite set of states, a transition function that defines the actions to be taken based on the current state and the symbol being read, and a set of start and accept states.
- The TM begins in the start state and performs the actions specified by the transition function until it reaches an accept or reject state. If it reaches an accept state, the computation is considered successful; if it reaches a reject state, the computation is considered unsuccessful.
- In the context of automata theory and the theory of computation, Turing machines are used to study the properties of algorithms and to determine what problems can and cannot be solved by computers. They provide a way to model the behavior of algorithms and to analyze their computational complexity, which is the amount of time and memory they require to solve a problem.

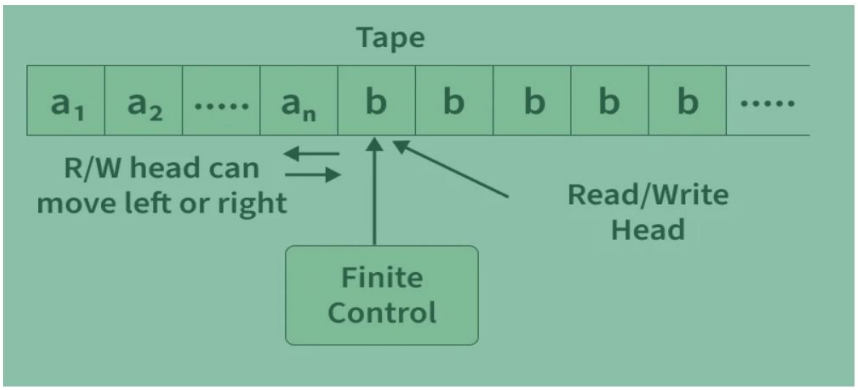
Why Turing Machines?

Turing machines are an important tool for studying the limits of computation and for understanding the foundations of computer science. They provide a simple yet powerful model of computation that has been widely used in research and has had a profound impact on our understanding of algorithms and computation.

While one might consider using programming languages like C to study computation, Turing Machines are preferred because:

- They are simpler to analyze.
- They provide a clear, mathematical model of computation.
- They possess infinite memory, making them even more powerful than real-world computers.

A Turing Machine consists of a tape of infinite length on which read and writes operation can be performed. The tape consists of infinite cells on which each cell either contains input symbol or a special symbol called blank. It also consists of a head pointer which points to cell currently being read and it can move in both directions.



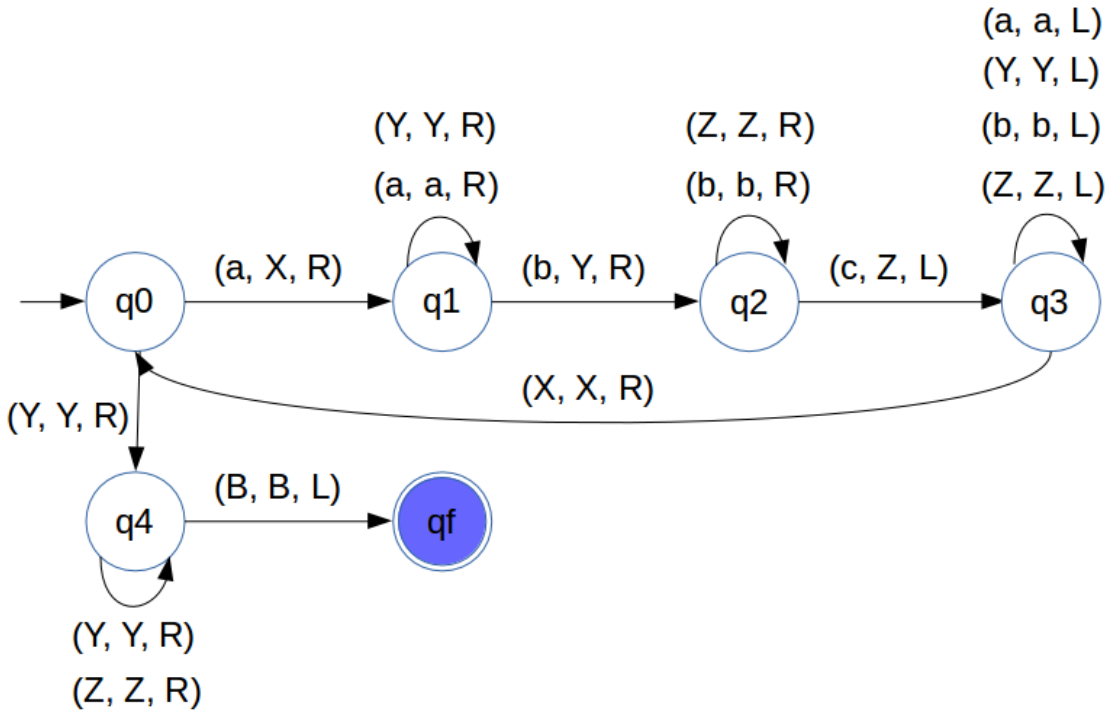
Turing Machine Formalism

A Turing Machine is defined by:

1. A finite set of states (Q).
2. An input alphabet (Σ).
3. A tape alphabet (Γ) that includes Σ.
4. A transition function (δ).
5. A start state (q0).
6. A blank symbol (B).

7. A set of final states (F).

9.b) **Design Turing machine to accept the language L={$a^n b^n c^n$| n>=1}**
Draw the transition diagram and show the moves made by the TM for the string aabbcc



10 a. What are the Programming techniques for Turing machine Explain

Alan Turing invented the Turing machine in 1936. It manipulates symbols on a strip of tape according to a table of rules specified. Although it is a simple model it is capable of implementing any computer algorithm. It is an accepting device which accepts Recursive Enumerable Language generated by type 0 grammar.

Various features of Turing machine:

- It has external memory which remembers long sequences of input.
- Input at the left or right of the tape can be easily read.
- Unlimited memory capability.
- Machine uses a certain input to generate an output, often it may use the same input to generate another output. Hence, we can say that the distinction between input/output has been removed here. A common set of alphabets can be used in the Turing machine.

Techniques of Turning machines

- **Storage in the finite control:**

A state that consists of a fixed number of fixed-size components can be made into a tuple. The behaviour of a TM programme can be made simpler by allowing the tuple's components to store a predetermined amount of data.

Example: Keep track of an additional symbol. The actual states of the TM can be [q0, A], [q1, A], [q0, B], or [q1, B] if the "additional data" can be A or B and the "state" can be q0 or q1.

- **Multiple Tracks:**

A particular kind of multi-tape Turing machine called a multi-track Turing machine has numerous tracks, but only one tape head can read and write on each track. One tape head reads n symbols from n tracks in this instance. Recursively enumerable languages are accepted, just like they are for single-track, single-tape Turing machines.

A Multi-track Turing machine can be formally described as a 6-tuple (Q, X, ∑, δ, q0, F) where

- Q is a finite set of states
- X is the tape alphabet
- ∑ is the input alphabet
- δ is a relation on states and symbols where δ(Qi, [a1, a2, a3,….]) = (Qj, [b1, b2, b3,….], Left_shift or Right_shift)
- q0 is the initial state
- F is the set of final states

Note − For every single-track Turing Machine S, there is an equivalent multi-track Turing Machine M such that L(S) = L(M).

Example

We will create a Turning machine that accepts the language L={ $wcw \mid w \in \mathbf{a} + \mathbf{b} *$ } using many tracks and storage in the finite control.This equations means any string w, followed by a "c", followed by a second copy of w.

This TM will work by "checking off" corresponding symbols in each copy of w.

- **Shifting Symbols Over**

In a Turning Machine , you might need to add extra space in the centre of a group of symbols. Holding a tiny buffer of symbols in the finite control will allow you to achieve this.

Example: Shifting a string of nonblank symbols over by two spaces.

States will be of the form q, $A1$, $A2$, where q is $q1$ or $q2$ and $A1$ and $A2$ are in Γ

- **Subroutines**

TMs can emulate subroutines, including those that send parameters and use recursion.

Example: Multiplication using a "copy" subroutine

- Given 0m10n1 as input, produce 0mn as output.

- This can be done by "copying" n 0s m times

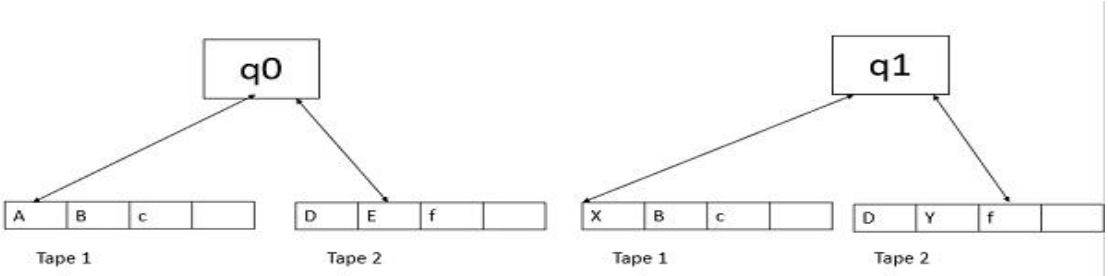
Here's how the overall TM will work:

1. While processing, the tape will contain a string of the form 0i10n0kn for some k.
2. In one "iteration", we will change a 0 in the first group of i 0s to B and append n 0s to the last group of 0s.This will require copying the group of n 0s to the end of the string.
3. Eventually, there will be no more 0s at the start of the string, and the Turning Machine can delete the 10n1, leaving only 0nm on the tape.

10.b.Write Short notes on :

a) Multi tape Turing machine

Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head. Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank. At first, the first tape is occupied by the input and the other tapes are kept blank. Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.



A Multi-tape Turing machine can be formally described as a 6-tuple (Q, X, B, δ, q_0, F) where –

- **Q** is a finite set of states
- **X** is the tape alphabet
- **B** is the blank symbol
- **δ** is a relation on states and symbols where

$\delta: Q \times X^k \rightarrow Q \times (X \times \{Left\_shift, Right\_shift, No\_shift\})^k$

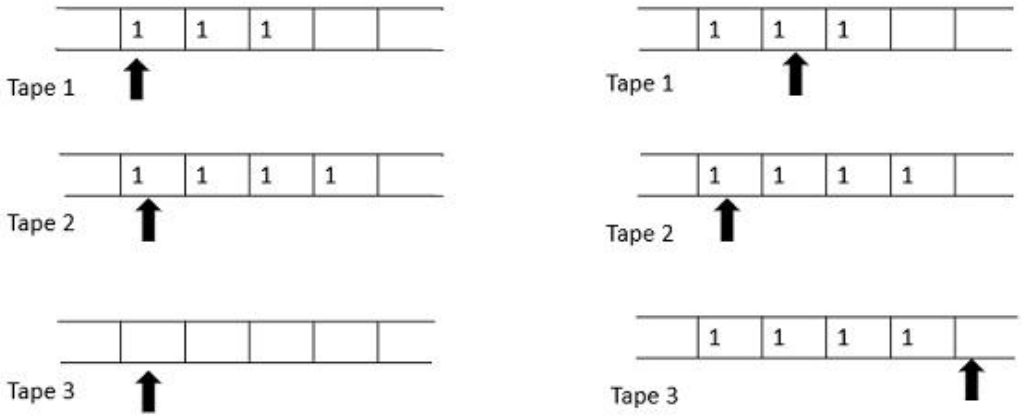
where there is **k** number of tapes

- **q_0** is the initial state
- **F is the set of final states**

Example

Multiplying two numbers (each represented as a unary string of ones) to get a third would be difficult to do with a simple Turing machine, but is fairly straightforward with a three tape machine.

Tapes before starting to compute Tapes before starting the second addition



Start by checking to see whether either number is zero −

- (0,(B,B ,B),(B,B ,B),(S, S, S), Halt) Both are zero
- (0,(B, 1, B),(B, 1,B),(S, S, S), Halt) First is zero
- (0,(1,B ,B),(1,B ,B),(S, S, S), Halt) Second is zero
- (0,(1, 1, B),(1, 1B,),(S, S, S), 1) Both are nonzero

Add the number on the second tape to the third tape −

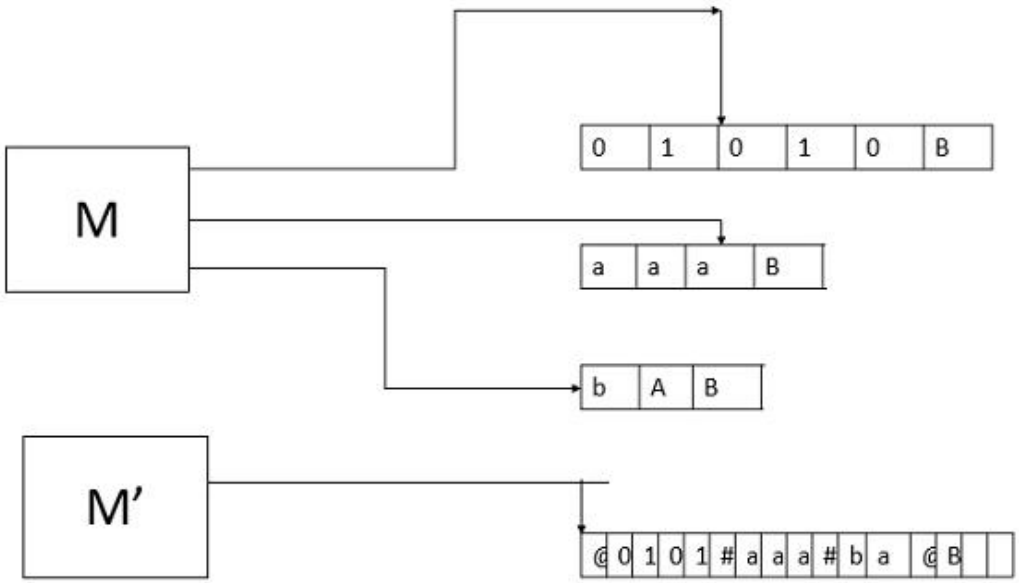
- (1, (1, 1,B),(1, 1, 1),(S, R, R), 1) Copy
- (1,(1, B ,B),(1,B ,B),(S, L, S), 2) Done copying

Move the tape head of the second tape back to the left end of the number; move the tape head of the first number one cell to the right −

- (2,(1, 1, B),(1, 1,B),(S, L, S), 2) Move to the left end
- (2,(1,B ,B),(1,B ,B),(R, R, S), 3) Both types to the right one cell

Check the first tape head to see if all the additions have been performed −

- (3,(B, 1,B),(B, 1,B),(S, S, L), Halt) Done
- (3,(1, 1,B),(1, 1,B),(S, S, S), 1) Do another add



Every multi-tape TM has an equivalent single tape TM

ii) Non-Deterministic Turing machine

In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic. The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted. If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

A non-deterministic Turing machine can be formally defined as a 6-tuple $(Q, X, \sum, \delta, q_0, B, F)$ where −

- **Q** is a finite set of states
- **X** is the tape alphabet
- $\sum$ is the input alphabet
- **δ** is a transition function;

$\delta : Q \times X \rightarrow P(Q \times X \times \{Left\_shift, Right\_shift\})$.

- $\mathbf{q_0}$ is the initial state
- **B** is the blank symbol
- **F** is the set of final states