CS8501 THEORY OF COMPUTATION L T P C 3 0 0 3

OBJECTIVES:

To understand the language hierarchy
To construct automata for any given pattern and find its equivalent regular expressions
To design a context free grammar for any given language
To understand Turing machines and their capability
To understand undecidable problems and NP class problems

UNIT I AUTOMATA FUNDAMENTALS 9

Introduction to formal proof – Additional forms of Proof – Inductive Proofs –Finite Automata – Deterministic Finite Automata – Non-deterministic Finite Automata – Finite Automata with Epsilon Transitions

UNIT II REGULAR EXPRESSIONS AND LANGUAGES 9

Regular Expressions – FA and Regular Expressions – Proving Languages not to be regular – Closure Properties of Regular Languages – Equivalence and Minimization of Automata.

UNIT III CONTEXT FREE GRAMMAR AND LANGUAGES 9

CFG – Parse Trees – Ambiguity in Grammars and Languages – Definition of the Pushdown Automata – Languages of a Pushdown Automata – Equivalence of Pushdown Automata and CFG, Deterministic Pushdown Automata.

UNIT IV PROPERTIES OF CONTEXT FREE LANGUAGES 9

Normal Forms for CFG – Pumping Lemma for CFL – Closure Properties of CFL – Turing Machines – Programming Techniques for TM.

UNIT V UNDECIDABILITY 9

Non Recursive Enumerable (RE) Language – Undecidable Problem with RE – Undecidable Problems about TM – Post's Correspondence Problem, The Class P and NP.

TOTAL :45PERIODS

OUTCOMES:

Upon completion of the course, the students will be able to:

Construct automata, regular expression for any pattern.
Write Context free grammar for any construct.
Design Turing machines for any language.
Propose computation solutions using Turing machines.
Derive whether a problem is decidable or not.

TEXT BOOK:

1. J.E.Hopcroft, R.Motwani and J.D Ullman, "Introduction to Automata Theory, Languages and Computations", Second Edition, Pearson Education, 2003.

REFERENCES:

1. H.R.Lewis and C.H.Papadimitriou, "Elements of the theory of Computation", Second Edition, PHI, 2003.
2. J.Martin, "Introduction to Languages and the Theory of Computation", Third Edition, TMH, 2003.
3. Micheal Sipser, "Introduction of the Theory and Computation", Thomson Brokecole, 1997.

CS8501- Theory of Computation.

UNIT-1

AUTOMATA FUNDAMENTALS

Introduction:

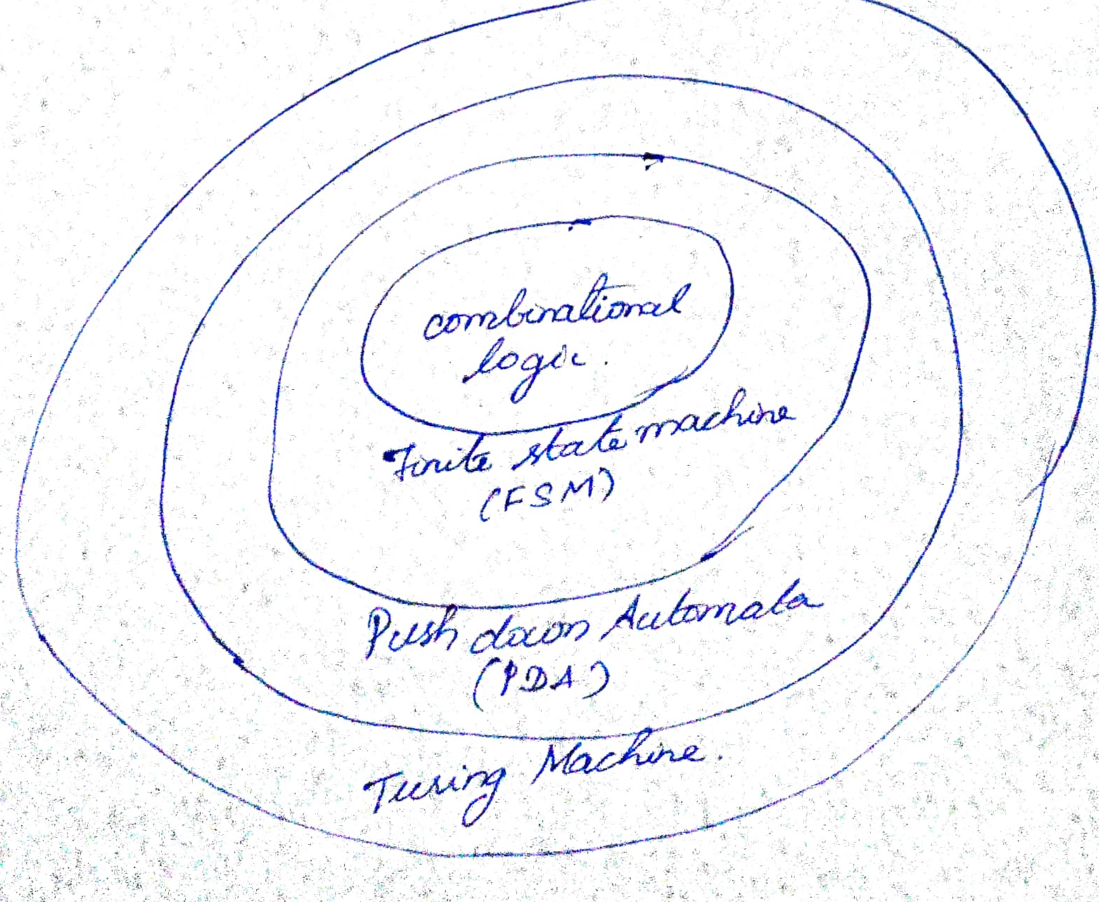
Theory of computation is the branch that deals with whether and how efficiently problems can be solved on a model of computation using an algorithm.

The field is divided into 3 major branches.

1) automata theory (what)
2) computability theory (How)
3) Computational complexity theory (time & space).

Applications:

* Text Processing
* Compilers
* Hardware Design



1. INTRODUCTION TO FORMAL PROOF

A proof of a statement is essentially just a convincing or justifying argument that the statement is true.

There are 2 types of proofs.

1) Deductive proof
2) Inductive proof.

1) Deductive proof:

A detective proof consists of a sequence of statements whose truth leads us from initial statement called hypothesis or given statement to a conclusion statement.

The theorem that is proved when we go from a hypothesis H to a conclusion C is the statement "if H then C", we say that "C is deduced from H."

Theorem 1:

If $x \geq 4$ then $2^x \geq x^2$

Proof:

Here $x \geq 4$ is hypothesis

$2^x \geq x^2$ is conclusion

x is the parameter

For $x = 6$

H is true i.e. $6 \geq 4$ is true

C is also true i.e. $2^6 \geq 6^2$

$64 \geq 36$ is true.

For $x = 3$.

H is false i.e. $3 \geq 4$ is false

C is also false i.e. $2^3 \geq 3^2$

$8 \geq 9$ is false

Therefore whenever H is true C will be also true.

Theorem 2:

If x is the sum of square of four positive integers, then $2^x \geq x^2$

Statement	Justification
1) $x = a^2 + b^2 + c^2 + d^2$	Given
2) $a \geq 1, b \geq 1, c \geq 1, d \geq 1$	Given
3) $a^2 \geq 1,\ b^2 \geq 1,\ c^2 \geq 1,\ d^2 \geq 1$	(2) and properties of arithmetic
4) $x \geq 4$	(1), (3) & properties of arithmetic
5) $2^x \geq x^2$	(4) and theorem if $x \geq 4$ then $2^x \geq x^2$

Reduction to Definition:

Theorem:

Let S be a finite subset of some infinite set U. Let T be the complement of S with respect to U. Then ~~a~~ T is infinite

Proof:

Original stmt	New Stmt
i) S is finite	There is an integer n such that $\|S\| = n$
ii) U is infinite	For no integer P is $\|U\| = P$
(iii) T is complement of S	$S \cup T = U$ & $S \cap T = \phi$

The above theorem can be proved by the principle of "Proof by contradiction"

This says, "if Contradiction of Hypothesis then Contradiction of conclusion"

Solution to the above theorem with "proof by contradiction"

Let us assume T is finite

$||S|| = n$

$||T|| = m$

$S \cup T = U$

So $n+m$ must be element of U.

$||U|| = n+m$, So U is finite

U is finite contradicts the given statement that U is infinite.

Hence proved

Other Theorem forms:

"If_Then" other forms are "if H then C"

- H implies C $\Rightarrow$ $x \geq 4$ implies $2^x \geq x^2$
- H only if C $\Rightarrow$ $x \geq 4$ only if $2^x \geq x^2$
- C if H $\Rightarrow$ $2^x \geq x^2$ if $x \geq 4$
- whenever H holds, C follows.

whenever $x \geq 4$, $2^x \geq x^2$ follows.

ADDITIONAL FORMS OF PROOF

How to construct proofs:

1) Proofs about sets
2) Proofs by contradiction
3) Proofs by countirexample.

1. Proofs about sets:

Contrapositive of the statement "if H then C" is "if not C then not H".

A statement & its contrapositive are either both true or both false.

commutative law $\Rightarrow$ $R \cup S = S \cup R$.

E is $R \cup S$

F is $S \cup R$

Commutative law says $E = F$.

This can be written as, set equality $E = F$ as an if-and-only if statement, an element x is in E if and only if x is in F.

Theorem:

$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$. ⇒ Distributive law of Union over intersection.

Proof

$E = R \cup (S \cap T)$

$F = (R \cup S) \cap (R \cup T)$

" if H then C"

" if x is in E then x is in F "

Statement	Justification
1) x is in $R \cup (S \cap T)$	Given
2) x is in R or x is in $S \cap T$	(1) and definition of Union
3) x is in R or x is in both S and T	(2) & definition of intersection
4) x is in $R \cup S$	(3) & Union
5) x is in $R \cup T$	(3) & Union
6. x is in $(R \cup S) \cap (R \cup T)$	(4), (5) & definition of intersection

↑ Steps in the if part.

Statement	Justification
1. x is in $(R \cup S) \cap (R \cup T)$	Given
2. x is in $R \cup S$	(1) & def. of intersection
3. x is in $R \cup T$	(1) & def. of intersection
4. x is in R or x is in both S & T	(2), (3) & reasoning about unions.
5. x is in R or x is in $S \cap T$	(4) & def. of intersection
6. x is in $R \cup (S \cap T)$	(5) & def. of union.

↑ Steps in the only if part.

2) Proof by contradiction:

A statement of the form "if H then C" can be proved using the statement "H and not C" implies falsehood.

3) Proof by counterexample:

Theorems are generally statements about infinite number of cases perhaps all values of its parameters. It is easier to prove that a statement is not a theorem than to prove it is a theorem.

Q1. All primes are odd.

$\Rightarrow$ if integer x is a prime, then x is odd.

Disproof :- integer 2 is a prime, but 2 is even.

Q2: There is no pair of integers a & b such that

a mod b = b mod a.

Disproof :- Let a = b = 2, then.

a mod b = b mod a = 0.

3. INDUCTIVE PROOFS

Induction on integers

i) Basis step: In basis step we show the statement $S(i)$ for a particular integer i usually $i=0$ or $i=1$

ii) Inductive: In inductive step we assume that $n \geq i$ where i is the basis integer and we show that for $S(n)$ and $S(n+1)$

Prove that for all $n \geq 0$ $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Basis step:

Let $n = 1$

L.H.S $= 1$ (a) $\sum_{i=1}^{n} i^2 = 1$

$$RHS = \frac{1(1+1)(2\times1+1)}{6} = \frac{1(2)(3)}{6} = 1$$

L.H.S = R.H.S

Inductive step:

Assume $n = k+1$

$$L.H.S = \sum_{i=1}^{k+1} i^2 = \sum_{i=1}^{k} i^2 + (k+1)^2$$

$$= \frac{k(k+1)(2k+1)}{6} + (k+1)^2$$

$$= \frac{(k^2+k)(2k+1)}{6} + (k+1)^2$$

$$= \frac{2k^3+3k^2+k}{6} + (k^2+2k+1)$$

$$L.H.S = \frac{2k^3+9k^2+13k+6}{6}$$

$$R.H.S = \frac{n(n+1)(2n+1)}{6}$$

Sub. $n = k+1$ in the above eqn

$$= \frac{(k+1)(k+1+1)(2(k+1)+1)}{6}$$

$$= \frac{(k+1)(k+2)(2k+3)}{6}$$

$$= \frac{(k^2+3k+2)(2k+3)}{6}$$

$$= \frac{2k^3+9k^2+13k+6}{6}$$

L.H.S = R.H.S

Hence Proved.

Theorem: Prove that for every integer $n \geq 0$ the number $4^{2n+1} + 3^{n+2}$ is a multiple of 13.

Basis:

$n = 0$

$$L.H.S = 4^{2(0)+1} + 3^{0+2}$$

$$= 4 + 3^2 = 4+9 = 13.$$

= xple of 13

= R.H.S

Induction hypothesis:

$n = k.$

$$4^{2k+1} + 3^{k+2} = 13k = 13(x) \quad \text{—— ①}$$

Inductive step: $n = k+1$

$$4^{2(k+1)+1} + 3^{(k+1)+2} = 13(k+1) = 13(x)$$

L.H.S. $= 4^{2(K+1)+1} + 3^{(K+1)+2}$

$= 4^{2(K+1)+1} - 3 \cdot 4^{2K+1} + 3 \cdot 4^{2K+1} + 3^{(K+1)+2}$

$= 4^{2K+1}[4^2 - 3] + 3[4^{2K+1} + 3^{K+2}]$

$= 4^{2K+1}(13) + 3(13K)$

$= 13[4^{2K+1} + 3K]$

$= 13(x)$

$=$ R.H.S.

Hence proved.

The Central Concepts of Automata Theory.

1. Alphabets : (Σ)
 - finite, non empty set of symbols.

 $\Sigma = \{0, 1\}$ $\rightarrow$ binary alphabet

 $\Sigma = \{a, b, .. z\}$ $\rightarrow$ set of all lower case letters.

2) Strings (w)

- finite sequence of symbols chosen from some alphabet.

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

string ⇒ 01101

101

0

1001

3. Empty string : (ϵ)

- Zero occurrences of symbols, denoted by ϵ

4. Length of a string: $|w|$

- no. of symbols in the string

01101 → length 5

- length is denoted by $|w|$, w is the string

$|011| = 3$.

$|\epsilon| = 0$

5. Power of an Alphabet : Σ^k

- Set of all strings with certain length,

- denoted by Σ^k

$\Sigma^0 = \{\epsilon\}$

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

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

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$ $\quad * \rightarrow 0$

$+ \rightarrow 1$

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

$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

6. Concatenation of strings:

x & y $\rightarrow$ string

$xy \rightarrow$ denotes the concatenation of x and y.

7. Language:

- set of all strings which are chosen from
- empty language is denoted by ϕ

eg: The language of all strings ~~contains~~ consisting of n 0's followed by n 1s, $n \geq 0$

$\{\epsilon, 01, 0011, 000111, \ldots\}$

The set of strings of 0's and 1's with an equal number of each.

$\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$

FINITE AUTOMATA:(FA)

Finite Automata (FA) is the simplest machine to recognize patterns.

A finite automata consists of the following.

Q : Finite set of states

Σ : Set of Input symbols.

q : Initial state

F : Set of Final states.

δ : Transition Function

Formal specification of machine is

$\{Q, \Sigma, q, F, \delta\}$

Two types of FA

1) Deterministic FA

2) Non Deterministic FA

DETERMINISTIC FINITE AUTOMATA (DFA)

In DFA, for a particular input character, there is only one transition from its current state.

In DFA, null or ϵ move is not allowed

DFA consists of 5 tuples.

$\{Q, \Sigma, q, F, \delta\}$

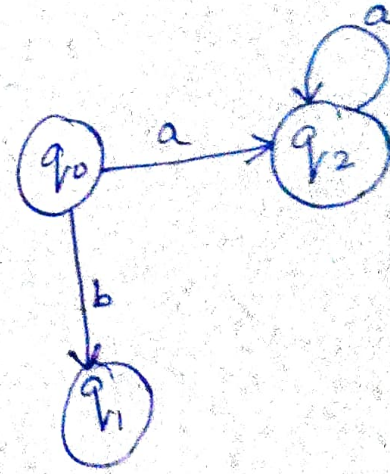
Q : Set of all states

Σ : set of input symbols

q : initial state

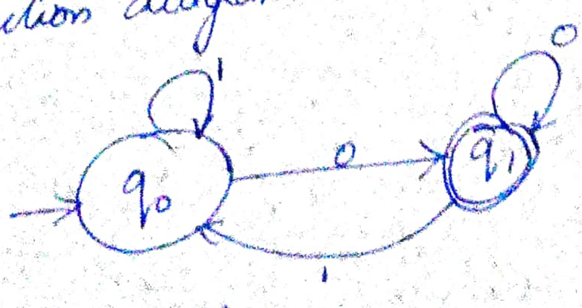
F : Set of final state

δ : Transition function.



Q1. Draw the DFA with $\Sigma = \{0,1\}$ that accepts all strings ending with 0.

Transition diagram.



Transition table:

States	I/P 0	1
$\rightarrow q_0$	q_1	q_0
$* q_1$	q_1	q_0

A Non-Deterministic Finite Automata can be represented by a 5 tuples

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

Q is a finite set of states

Σ is a finite set of input symbols.

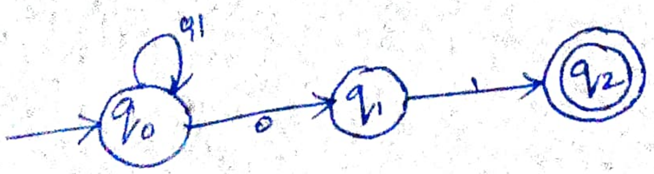
δ is a transition function

q_0 is the initial state

F is the set of final states.

eg) A NDFA accepting all string that end in 01

Transition diagram.



Transition table:

state	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
$* q_2$	ϕ	ϕ

Input 00101

$\hat{\delta}(q_0, \epsilon) = \{q_0\}$

$\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \epsilon), 0) = \delta(q_0, 0) = \{q_0, q_1\}$

$\hat{\delta}(\{q_0, q_1\}, 00) = \delta(\hat{\delta}(q_0, 0), 0) = \delta(\{q_0, q_1\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \phi$

$= \{q_0, q_1\}$

$\hat{\delta}(\{q_0, q_1\}, 001\} = \delta(\hat{\delta}(\{q_0, q_1\}, 00), 1) = \delta(\{q_0, q_1\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\}$

$= \{q_0, q_2\}$

$\hat{\delta}(\{q_0, q_2\}, 0010) = \delta(\hat{\delta}(\{q_0, q_1\}, 001), 0) = \delta(\{q_0, q_2\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_2, 1) = \{q_0, q_1\} \cup \phi$

$= \{q_0, q_1\}$

$\hat{\delta}(\{q_0, q_1\}, 00101) = \delta(\hat{\delta}(\{q_0, q_2\}, 0010), 1) = \delta(\{q_0, q_1\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\}$

$= \{q_0, q_2\}$

q_2 is the final state and hence the string is accepted by NFA.

Input:

1100

$\hat{\delta}(q_0, \epsilon) = q_0$

$\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_0$

$\hat{\delta}(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_0, 1) = q_0$

$\hat{\delta}(q_0, 110) = \delta(\hat{\delta}(q_0, 11), 0) = \delta(q_0, 0) = q_1$

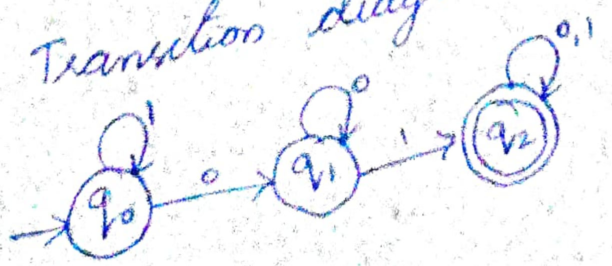
$\hat{\delta}(q_0, 1100) = \delta(\hat{\delta}(q_0, 110), 0) = \delta(q_1, 0) = q_1$

q_1 is the final state and hence the string is accepted by DFA.

x ——— x

Eg2 Draw a DFA for a string that contain a zero followed by 1 and check whether 001 is accepted or not.

Transition diagram:



States	I/O 0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$* q_2$	q_2	q_2

Input : 001

$\hat{\delta}(q_0, \epsilon) = q_0$

$\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \epsilon), 0) = \delta(q_0, 0) = q_1$

$\hat{\delta}(q_1, 00) = \delta(\hat{\delta}(q_0, 0), 0) = \delta(q_1, 0) = q_1$

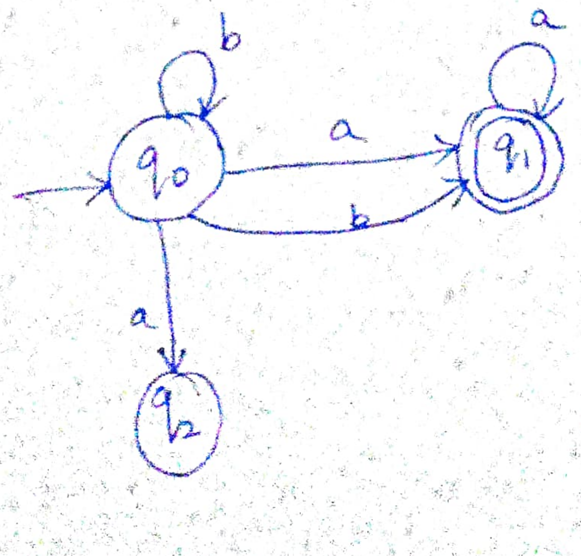
$\hat{\delta}(q_1, 001) = \delta(\hat{\delta}(q_1, 00), 1) = \delta(q_1, 1) = q_2$

q_2 is the final state and hence the string is accepted by DFA.

————

NON-DETERMINISTIC AUTOMATA: NFA:

The finite automata is called Non-deterministic Finite Automata if there is many paths for a specific input from current state to next state.



FINITE AUTOMATA WITH ε TRANSITION

The ε transition in NFA are given in order to move from one state to another state without having any symbol from input set Σ.

NDFA with ε-transition is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

where

Q. finite set of states

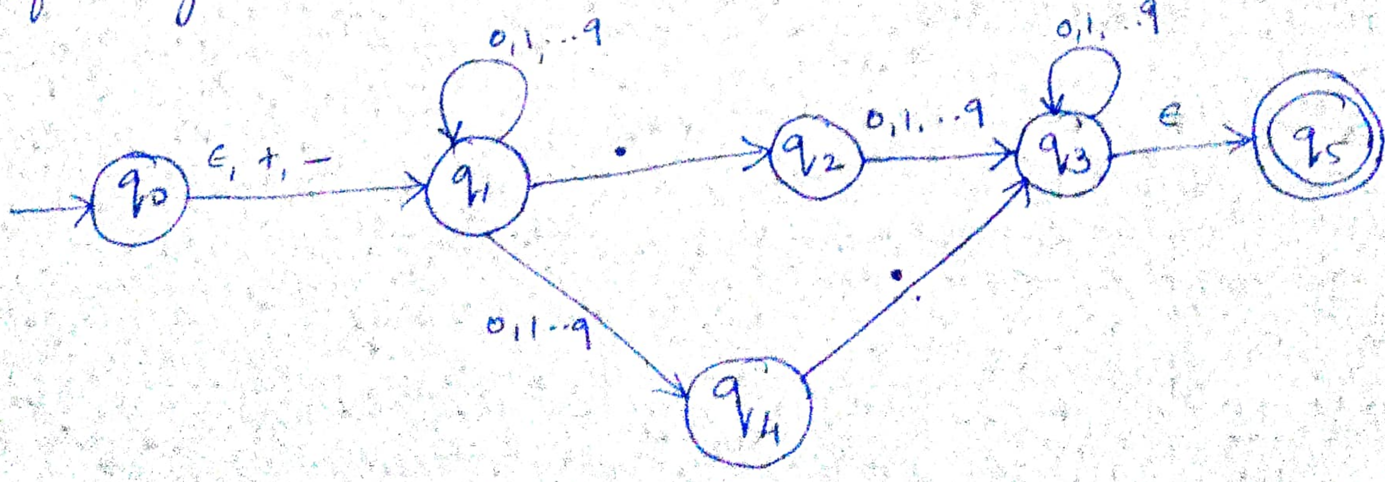
Σ. finite set of symbols

δ - be a transition function

q_0 . initial state

F. set of final state.

eg. Draw the ε-NFA that accepts decimal nos. consisting of (i) An optional + or - sign (ii) String of digits (iii) A decimal point and (iv) another string of digits.



* ———— *

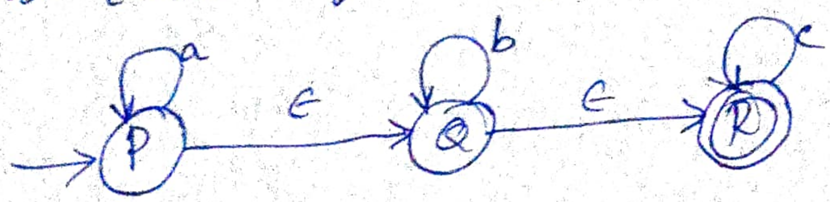
Epsilon closures:

The ϵ closure (P) is a set of all states which are reachable from state P on ϵ-transition such that,

i) ϵ-closure (P) = P where $P \in Q$

ii) if there exist ϵ-closure (P) = {P} and $\delta(q, \epsilon) = r$ then ϵ-closure (P) = {q, r}

eg). Find ϵ-closure for the following NFA with ϵ.



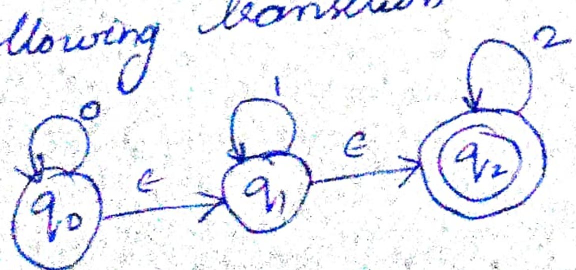
ϵ-closure (P) = {P, Q, R}

ϵ-closure (Q) = {Q, R}

ϵ-closure (R) = {R}

ϵ-NFA to NFA without ϵ

Obtain the NFA without ϵ transition with following transition



States	0	1	2	ϵ
$\rightarrow q_0$	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$

$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$

$\epsilon\text{-closure}(q_2) = \{q_2\}$

$\hat{\delta}(q_0, \epsilon) = \epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$

$\hat{\delta}(q_1, \epsilon) = \epsilon\text{-closure}(q_1) = \{q_1, q_2\}$

$\hat{\delta}(q_2, \epsilon) = \epsilon\text{-closure}(q_2) = \{q_2\}$

$\delta'(q_0, 0) = \hat{\delta}(q_0, 0)$

$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), 0))$

$= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 0\}$

$= \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

$= \epsilon\text{-closure}(q_0 \cup \phi \cup \phi)$

$= \epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$

$\delta'(q_0, 1) = \hat{\delta}(q_0, 1)$

$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), 1))$

$= \epsilon\text{-closure}(\delta\{q_0, q_1, q_2\}, 1)$

$= \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1)\ \delta(q_2, 1))$

$= \epsilon\text{-closure}(\phi \cup \{q_1\} \cup \phi)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1, q_2\}$

$$\delta'(q_0,2) = \hat{\delta}(q_0,2)$$
$$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon),2))$$
$$= \epsilon\text{-closure}(\delta(\{q_0,q_1,q_2\},2))$$
$$= \epsilon\text{-closure}(\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2))$$
$$= \epsilon\text{-closure}(\phi \cup \phi \cup \{q_2\})$$
$$= \epsilon\text{-closure}(q_2)$$
$$= \{q_2\}$$

$$\delta'(q_1,0) = \hat{\delta}(q_1,0)$$
$$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1,\epsilon),0))$$
$$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},0))$$
$$= \epsilon\text{-closure}(\delta(q_1,0) \cup \delta(q_2,0))$$
$$= \epsilon\text{-closure}(\phi \cup \phi)$$
$$= \epsilon\text{-closure}(\phi) = \phi$$

$$\delta'(q_1,1) = \hat{\delta}(q_1,1)$$
$$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1,\epsilon),1))$$
$$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},1))$$
$$= \epsilon\text{-closure}(\delta(q_1,1) \cup \delta(q_2,1))$$
$$= \epsilon\text{-closure}(\{q_1\} \cup \phi)$$
$$= \epsilon\text{-closure}(\{q_1\})$$
$$= \{q_1,q_2\}$$

$$\delta'(q_1,2) = \hat{\delta}(q_1,2)$$
$$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1,\epsilon),2))$$
$$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},2))$$
$$= \epsilon\text{-closure}(\delta(q_1,2) \cup \delta(q_2,2))$$

$= \epsilon\text{-closure}(\phi \cup \{q_2\})$

$= \epsilon\text{-closure}(\{q_2\})$

$= \{q_2\}$

$\delta'(q_2, 0) = \hat{\delta}(q_2, 0)$

$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_2, \epsilon), 0))$

$= \epsilon\text{-closure}(\delta(q_2, 0))$

$= \epsilon\text{-closure}(\phi) = \phi$

$\delta'(q_2, 1) = \hat{\delta}(q_2, 1)$

$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_2, \epsilon), 1))$

$= \epsilon\text{-closure}(\delta(q_2, 1))$

$= \epsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_2, 2) = \hat{\delta}(q_2, 2)$

$= \epsilon\text{-closure}(\delta(\hat{\delta}(q_2, \epsilon), 2))$

$= \epsilon\text{-closure}(\delta(q_2, 2))$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$

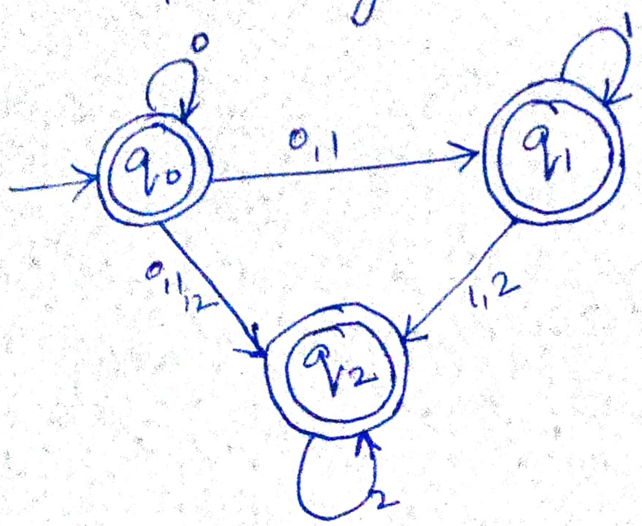
Transition table.

states

Transition table:

States	0	1	2
$\rightarrow * q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$* q_1$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
$* q_2$	ϕ	ϕ	$\{q_2\}$

NFA diagram:



* ———————— *

NFA to DFA

δ	a	b
$\rightarrow p$	$\{p\}$	$\{p,q\}$
q	$\{r\}$	$\{r\}$
$*r$	ϕ	ϕ

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

$M = (\{p, q, r\}, \{a, b\}, \delta, p, \{r\})$.

Convert it to DFA.

solution:-

Applying δ transition on each state

$\delta(p, a) = \{p\}$

$\delta(p, b) = \{p, q\}$ ——— new state ①

$\delta(q, a) = \{r\}$

$\delta(q, b) = \{r\}$

$\delta(r, a) = \phi$

$\delta(r, b) = \phi$

Considering new state ① $\{p, q\}$

$\delta(\{p,q\}, a) = \delta(p, a) \cup \delta(q, a)$

$= \{p\} \cup \{r\} = \{p, r\}$ ——— new state ②

$$\delta(\{P,q\},b) = \delta(P,b) \cup \delta(q,b)$$
$$= \{P,q\} \cup \{r\}$$
$$= \{P,q,r\}$$ —— new state ③

Considering new state ②

$$\delta(\{P,r\},a) = \delta(P,a) \cup \delta(r,a)$$
$$= \{P\} \cup \phi = \{P\}$$

$$\delta(\{P,r\},b) = \delta(P,b) \cup \delta(r,b)$$
$$= \{P,q\} \cup \phi = \{P,q\}$$

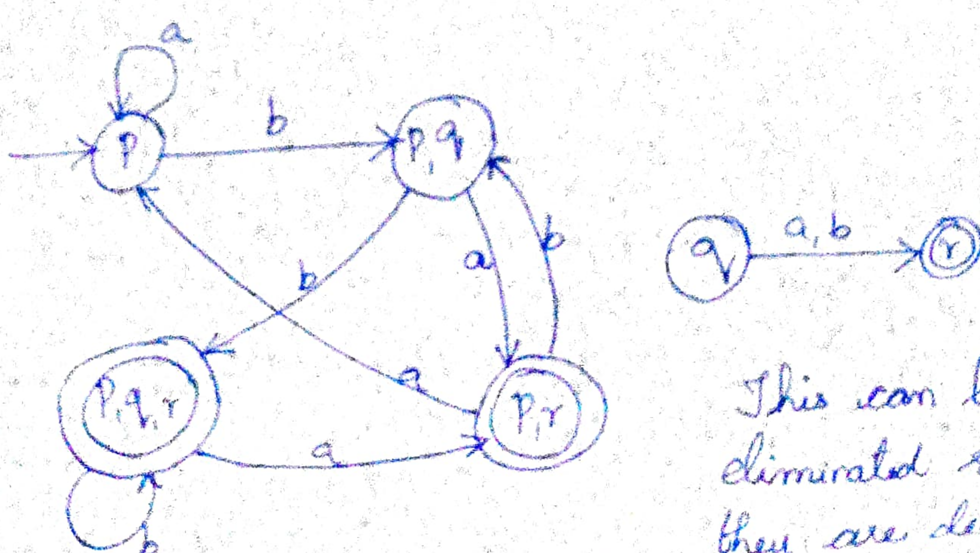
Considering new state ③

$$\delta(\{P,q,r\},a) = \delta(P,a) \cup \delta(q,a) \cup \delta(r,a)$$
$$= \{P\} \cup \{r\} \cup \phi$$
$$= \{P,r\}$$

$$\delta(\{P,q,r\},b) = \delta(P,b) \cup \delta(q,b) \cup \delta(r,b)$$
$$= \{P,q\} \cup \{r\} \cup \phi$$
$$= \{P,q,r\}$$

Transition table.

States	a	b
→ {p}	{p}	{p,q}
{q}	{r}	{r}
* {r}	φ	φ
{p,q}	{p,r}	{p,q,r}
* {p,r}	{p}	{p,q}
* {p,q,r}	{p,r}	{p,q,r}

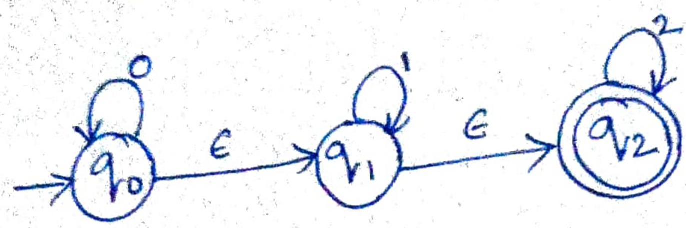


This can be eliminated since they are dead states.

* ——————— *

ϵ-NFA to DFA:

Consider the following ϵ-NFA. Compute the ϵ-closure of each state and find its equivalent DFA.



Transition table.

states	0	1	2	ϵ
$\rightarrow q_0$	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
$* q_2$	ϕ	ϕ	q_2	ϕ

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$ ——— ①

ϵ closure $(q_1) = \{q_1, q_2\}$ ——— ②

ϵ-closure $(q_2) = \{q_2\}$

Step 1: ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$ — ①

$\delta(\{q_0, q_1, q_2\}, 0\} = \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

$= \epsilon\text{-closure}(q_0 \cup \phi \cup \phi)$

$= \epsilon\text{-closure}(q_0)$

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

$\delta(\{q_0, q_1, q_2\}, 1) = \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$

$= \epsilon\text{-closure}(\phi \cup q_1, \cup \phi)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1, q_2\}$

$\delta(\{q_0, q_1, q_2\}, 2) = \epsilon\text{-closure}(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2))$

$= \epsilon\text{-closure}(\phi \cup \phi \cup q_2)$

$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$

From 2

$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$

$\delta(\{q_1, q_2\}, 0) = \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0))$

$= \epsilon\text{-closure}(\phi \cup \phi) = \phi$

$\delta(\{q_1, q_2\}, 1) = \epsilon\text{-closure}(\delta(q_1, 1) \cup \delta(q_2, 1))$

$= \epsilon\text{-closure}(q_1 \cup \phi)$

$= \epsilon\text{-closure}(q_1)$

$= \{q_1, q_2\}$

$\delta(\{q_1, q_2\}, 2) = \epsilon\text{-closure}(\delta(q_1, 2) \cup \delta(q_2, 2))$

$= \epsilon\text{-closure}(\phi \cup q_2)$

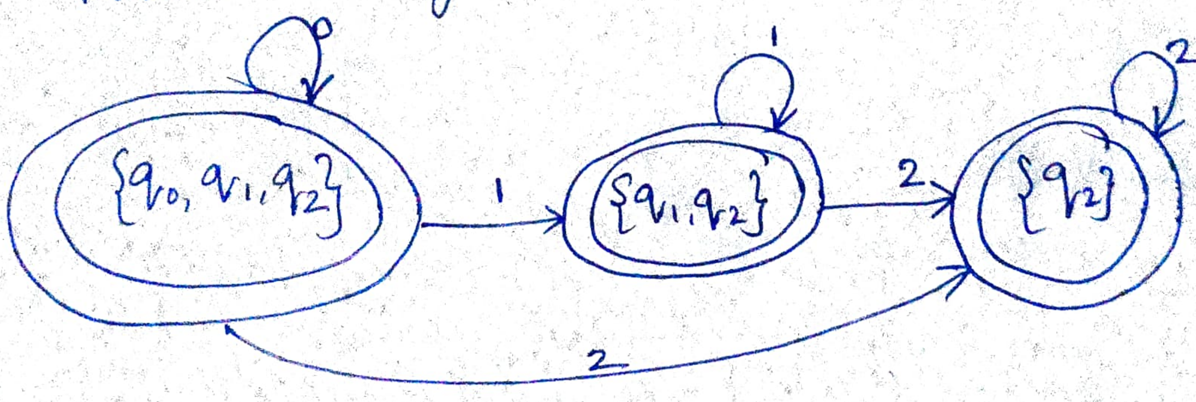
$= \epsilon\text{-closure}(q_2)$

$= \{q_2\}$

Transition table:

states	0	1	2
$\rightarrow * \{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$* \{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
$* \{q_2\}$	ϕ	ϕ	$\{q_2\}$

Transition diagram - (DFA)



* ———— *

UNIT-II

REGULAR EXPRESSION & LANGUAGES.

REGULAR EXPRESSIONS:

The regular expression are the algebraic notation that describes exactly the same languages as finite automata.

Operations of regular expressions:

① Union of 2 languages

L & M denoted by L∪M, is the set of strings that are in either L or M or both. L={001,10,111} & M={ϵ,001}

L∪M={ϵ,10,001,111}

② Concatenation of 2 languages.

L & M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M. LM={001,10,111,001001,10001,111001}

③ Closure (or star or Kleene closure) of a language denoted by L^*

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

④ Positive closure of a language L is denoted by L^+

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Building regular expression.

For each regular expression E, the language is denoted by $L(E)$

Basis:

The basis consists of 3 parts.

① φ is ~~a~~ a regular expression denoting the language $\{\varphi\}$

② ϵ is a regular expression denoting the language $\{\epsilon\}$

③ a is any symbol, then a is a RE denoting the language $\{a\}$.

Induction:

If r, s are RE denoting languages R, S then,

1. $r+s$ is a RE denoting $R \cup S$
2. rs is a RE denoting RS
3. $r*$ is a RE denoting R^*

Precedence of RE operators:

1. The star operator is of highest precedence
2. Next concatenation or dot operator.
3. Finally all unions (+ operators) are grouped with their operands.

Write the Regular Expression for the following.

1. $L = \{ w / w$ has the substring $101 \}$.

solution: $(0+1)^*.101(0+1)^*$

2. $L = \{ w / w$ has an even length$\}$.

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

3. The language of all string not ending with 11

$\epsilon + 0 + 1 + (0+1)^* 00 + (0+1)^* 01 + (0+1)^* 10$

$\epsilon + 0 + 1 + (0+1)^* (00 + 01 + 10)$

4. The set of all strings whose no. of 0 is multiple of 5.

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

5. The language of all strings that contain atmost one 1 or atleast 2 0's.

- atmost one 1

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

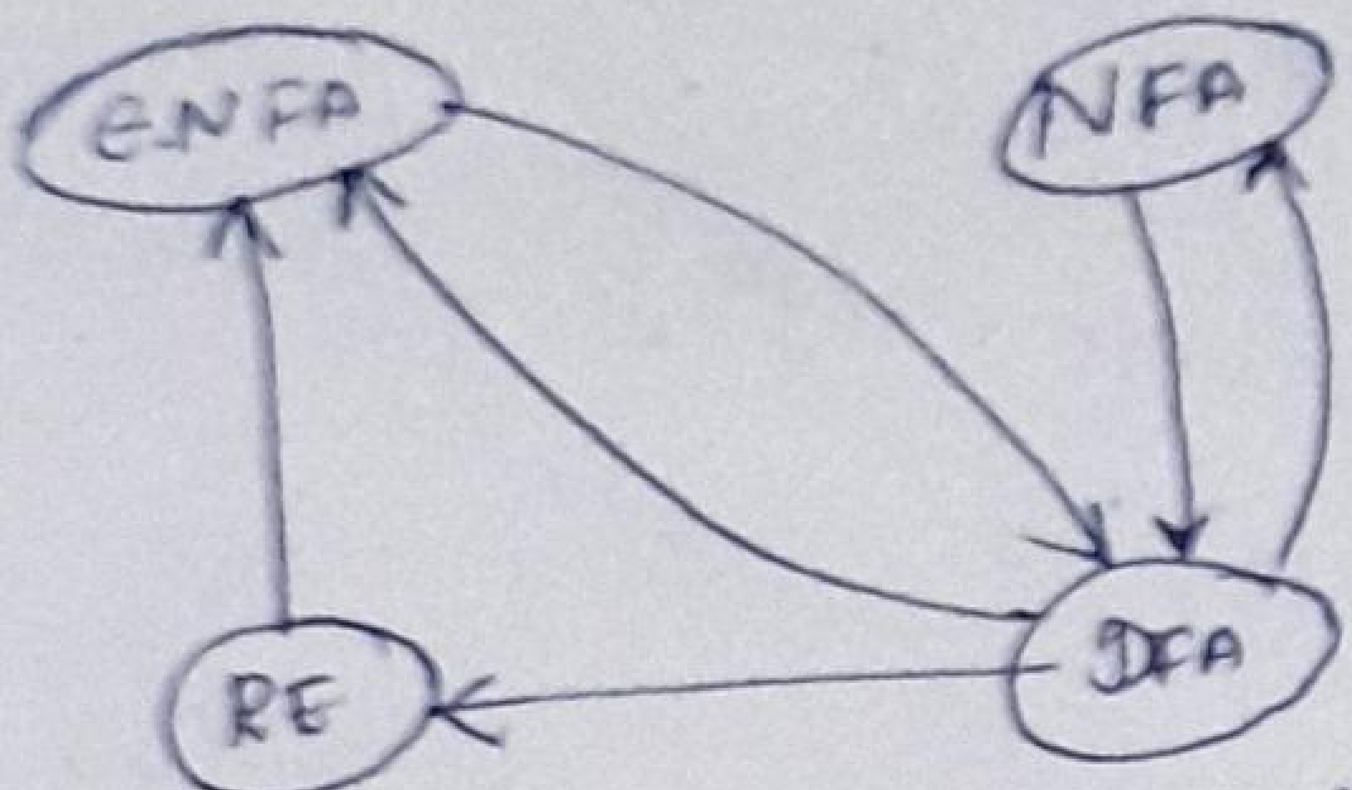
- atleast 2 0's.

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

- atmost one 1 or atleast 2 0's.

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

FA and REGULAR EXPRESSIONS



Plan for showing the equivalence of 4 different notations for Regular languages.

From DFA's to RE.

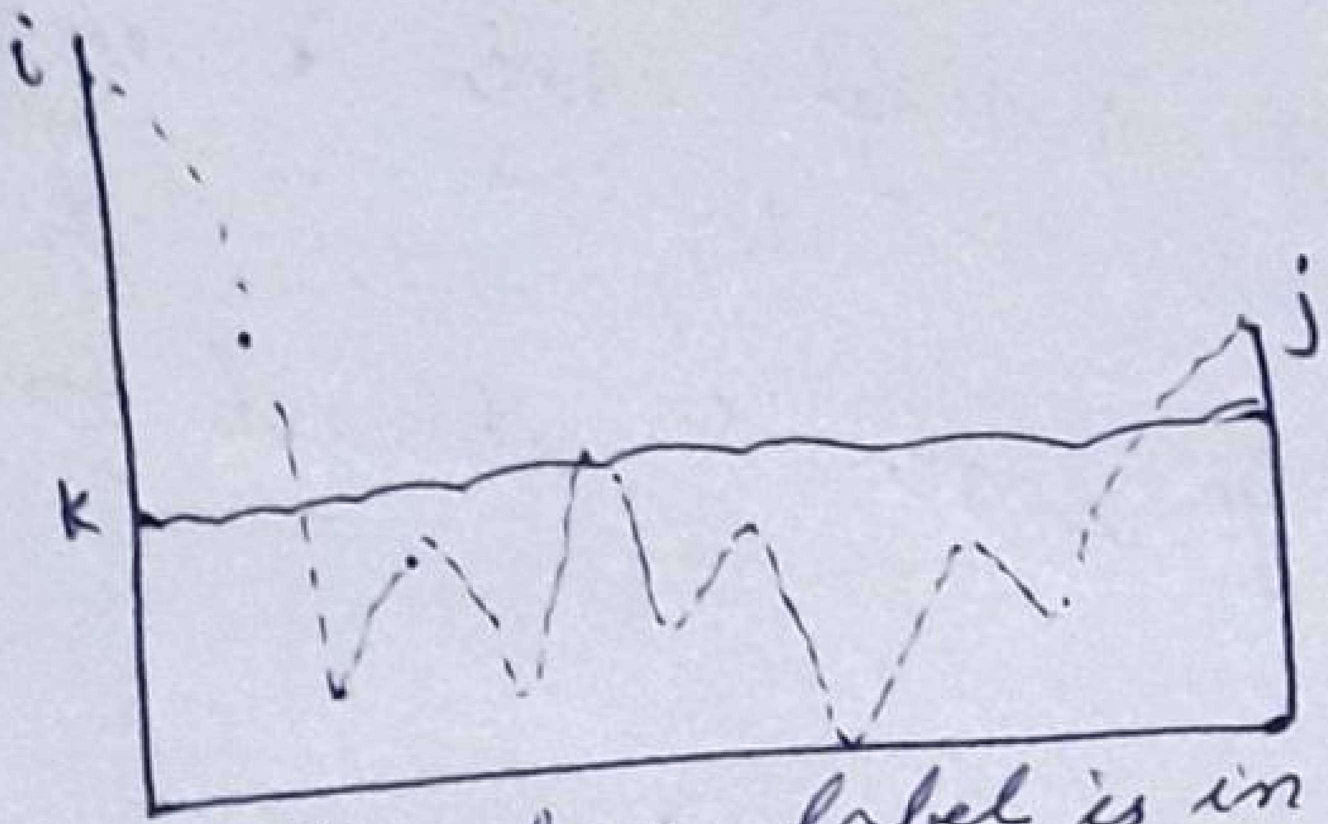
Theorem: If $L = L(A)$ for some DFA A, then there is a RE R such that $L = L(R)$

Proof:

Let us assume that A's states are $\{1, 2, \ldots n\}$ for some integer n.

Let us use $R_{ij}^{(k)}$ as the name of a RE whose language is the set of strings w such that w is the label of a path from state i to state j in A.

The path has no intermediate node whose number is greater than k.



A path whose label is in the language of RE $R_{ij}^{(k)}$

Basis:

The basis is $k=0$

The path must have no intermediate states

There are only 2 kinds of paths.

1. An arc from node i to node j
2. A path of length 0 that consists of only same node i.

If $i \neq j$ then only case (1) is possible

Now examine DFA A and find input symbols a.

a) If there is no such symbol 'a', then $R_{ij}^{(0)} = \phi$

b) If there is exactly one such symbol 'a' then $R_{ij}^{(0)} = a$

c) If there are symbols $a_1, a_2, \ldots a_k$, then

$$R_{ij}^{(0)} = a_1 + a_2 + \ldots + a_k$$

Induction:

Suppose there is a path from state i to j that goes through no state higher than k.

There are 2 possible cases.

① The path does not go through state k at all.
The label of the path is $R_{ij}^{(k-1)}$

② The path goes through state k atleast once.

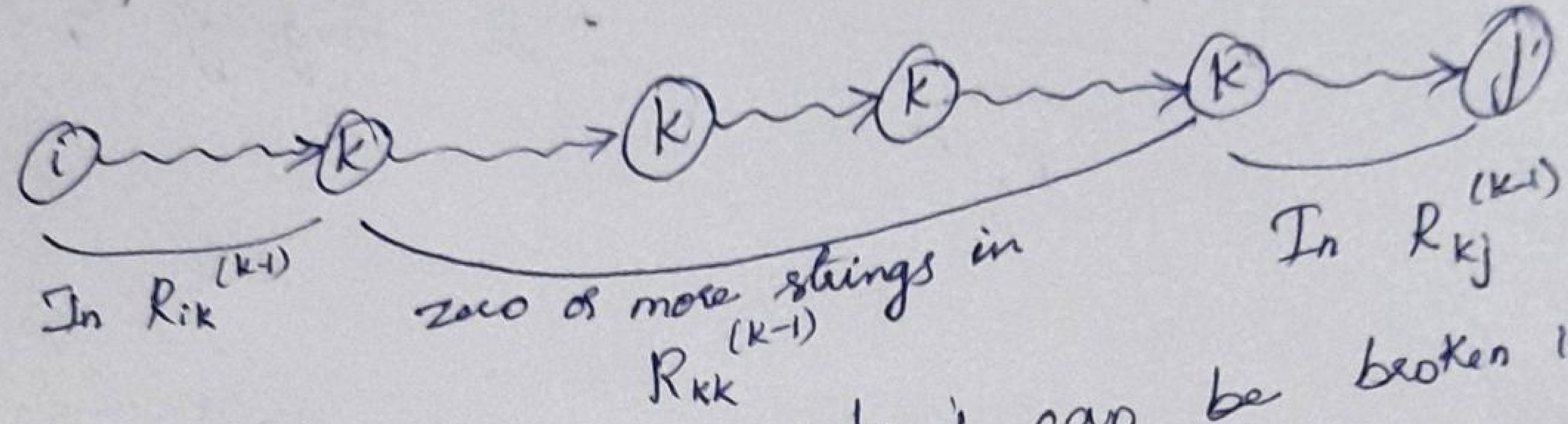


fig:- A path from i to j can be broken into segments.

The set of labels for all path of this type is represented by the regular expression,

$$R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

By combining the expressions for the paths of the 2 types, we get the expression.

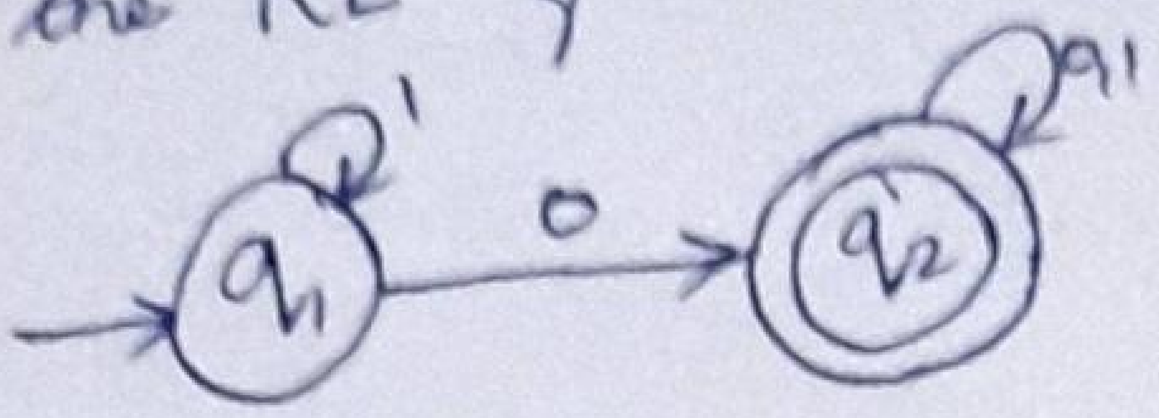
$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

for the labels of all paths from i to state j that go through no state higher than k.

Hence proved.

Problem.

Find the RE for the DFA (Equation method).



$R_{ij}^{(k)}$ k-no of states
i-starting
j-final state

sln

Let $K=0$

$R_{11}^{(0)} = \epsilon + 1$

$R_{12}^{(0)} = 0$

$R_{22}^{(0)} = \epsilon + 0 + 1$

$R_{21}^{(0)} = \phi$

Let $K=1$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} (R_{11}^{(0)})^* R_{1j}^{(0)}$$

$i=1, j=1$

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= (\epsilon+1) + (\epsilon+1)(\epsilon+1)^*(\epsilon+1)$$

$[\epsilon + R = R]$

$$= 1 + 11^*1$$

$$= 1(\epsilon + 11^*)$$

$[\epsilon + RR^* = R^*]$

$$R_{11}^{(1)} = 11^*$$

Scanned by TapScanner

$i=1,\ j=2$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)}\left(R_{11}^{(0)}\right)^{*} R_{12}^{(0)}$$

$$= 0 + (\epsilon+1)(\epsilon+1)^{*}0$$

$$= 0 + 11^{*}0$$

$$= 0(\epsilon + 11^{*})$$

$$R_{12}^{(1)} = 01^{*}$$

$i=2,\ j=1$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)}\left(R_{11}^{(0)}\right)^{*} R_{11}^{(0)}$$

$$= \varphi + \varphi\,(\epsilon+1)^{*}(\epsilon+1)$$

$$= \varphi + \varphi$$

$$R_{21}^{(1)} = \varphi$$

$$\left[\begin{array}{l} \varphi R = \varphi \\ \varphi + R = R \end{array}\right]$$

$i=2,\ j=2$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)}\left(R_{11}^{(0)}\right)^{*}\left(R_{12}^{(0)}\right)$$

$$= (\epsilon+0+1) + \varphi\,(\epsilon+1)^{*}(0)$$

$$= (\epsilon+0+1) + \varphi$$

$$R_{22}^{(1)} = \epsilon + 0 + 1$$

$$= 0+1$$

Let $K = 2$.

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} \left(R_{22}^{(1)}\right)^* R_{22}^{(1)}$$

$$= 01^* + 01^* (0+1)^* (0+1)$$

$$= 01^* \left(\epsilon + (0+1)^* (0+1)\right)$$

$$R_{12}^{(2)} = 01^* (0+1)^*$$

The RE for the given DFA is,

$$01^* (0+1)^*$$

Converting DFA's to RE by Arden's Theorem:

Let P & Q be 2 REs over Σ. If P does not contain ϵ, then the equation $\underline{R = Q + RP}$ has a solution $\underline{R = QP^*}$.

Using this theorem, it is easy to find the RE.

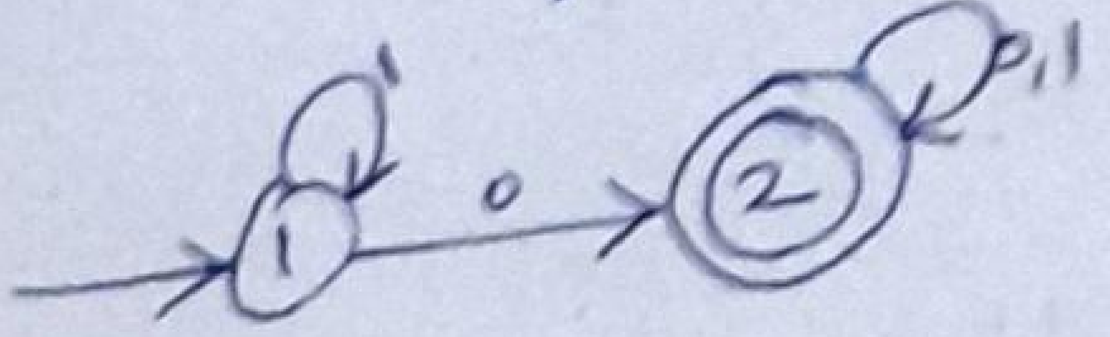
The condition to apply this theorem are,

i) Finite automata does not have ϵ-moves

ii) It has only one start state.

Pbm.

Find the RE for the following DFA



Sln

For starting state add ϵ.

$q_1 = q_1 1 + \epsilon$ ——— ①

$q_2 = q_1 0 + q_2 (0+1)$ ——— ②

Take eqn ①

$\underset{R}{q_1} = \underset{R}{q_1} \underset{P}{1} + \underset{Q}{\epsilon}$

It can be written as $R = QP^*$

$q_1 = \epsilon 1^*$

$\boxed{q_1 = 1^*}$

sub q_1 in ②.

$q_2 = q_1 0 + q_2 (0+1)$

$q_2 = 1^* 0 + q_2 (0+1)$

$\overset{R}{q_2} = \overset{R}{q_2} \overset{P}{(0+1)} + \overset{Q}{1^* 0}$

Applying Arden's theorem,

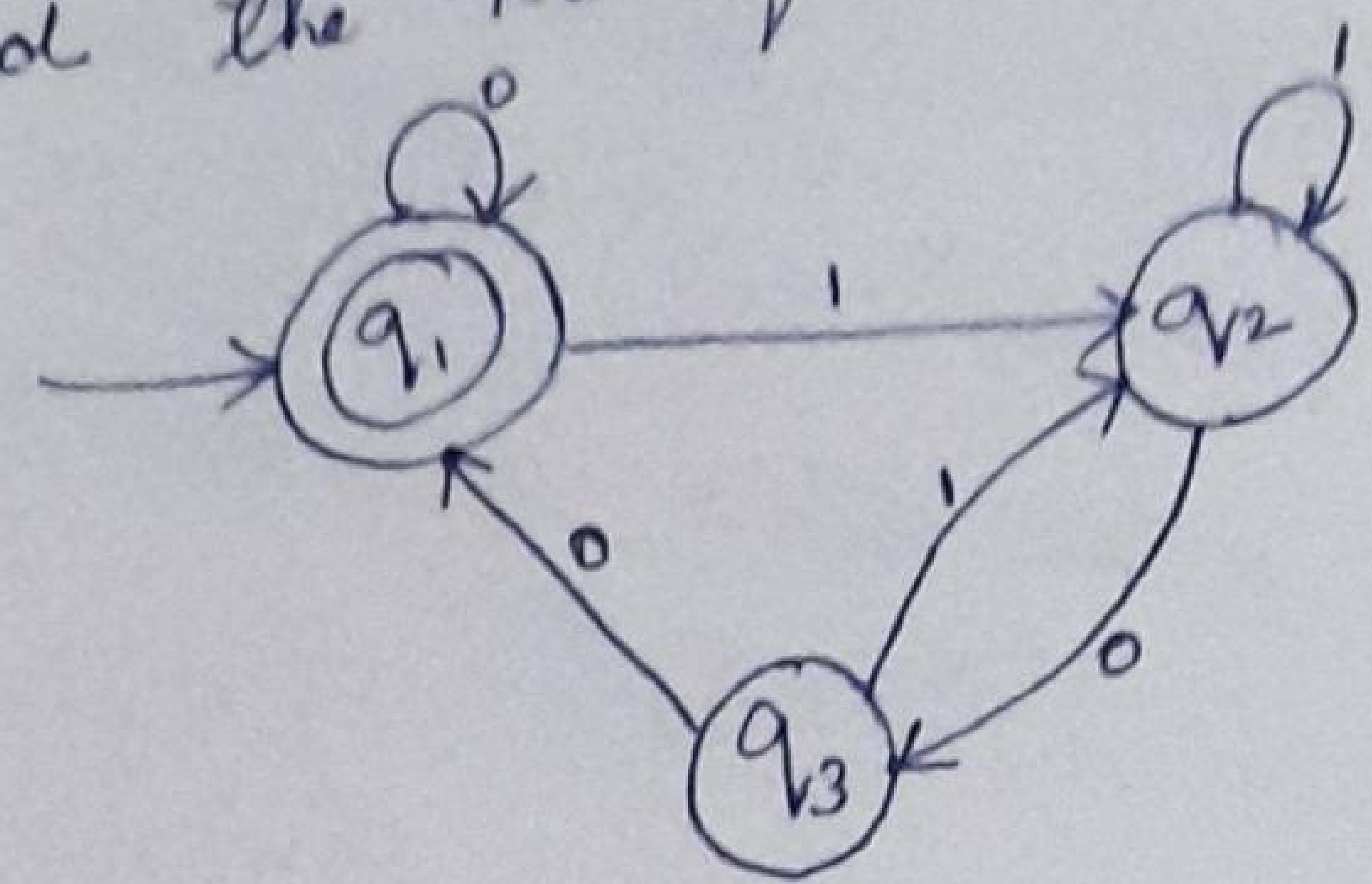
$q_2 = 1^* 0 (0+1)^*$

Since q_2 is the final state, the RE is,

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

Pbm.

Find the RE for DFA.



Sln. For starting state add ϵ

$q_1 = q_1 0 + q_3 0 + \epsilon$ ——— 1

$q_2 = q_1 1 + q_2 1 + q_3 1$ ——— 2

$q_3 = q_2 0$ ——— 3

Sub. 3 in 1.

$q_1 = q_1 0 + q_2 0 0 + \epsilon$ ——— 4.

Sub. 3 in 2

$q_2 = q_1 1 + q_2 1 + q_2 01$

$q_2 = q_1 1 + q_2 (1 + 01)$

$\overset{R}{q_2} = \overset{R}{q_2} \overset{P}{(1+01)} + \overset{Q}{q_1 1}$

Applying Arden's theorem

$q_2 = q_1 1 (1+01)^*$ ——— 5

Sub 5 in 4.

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

$\overset{R}{q_1} = \overset{R}{q_1} \overset{P}{(0 + (1 (1+01)^* 00)} + \overset{Q}{\epsilon}$

Applying Arden's theorem,

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

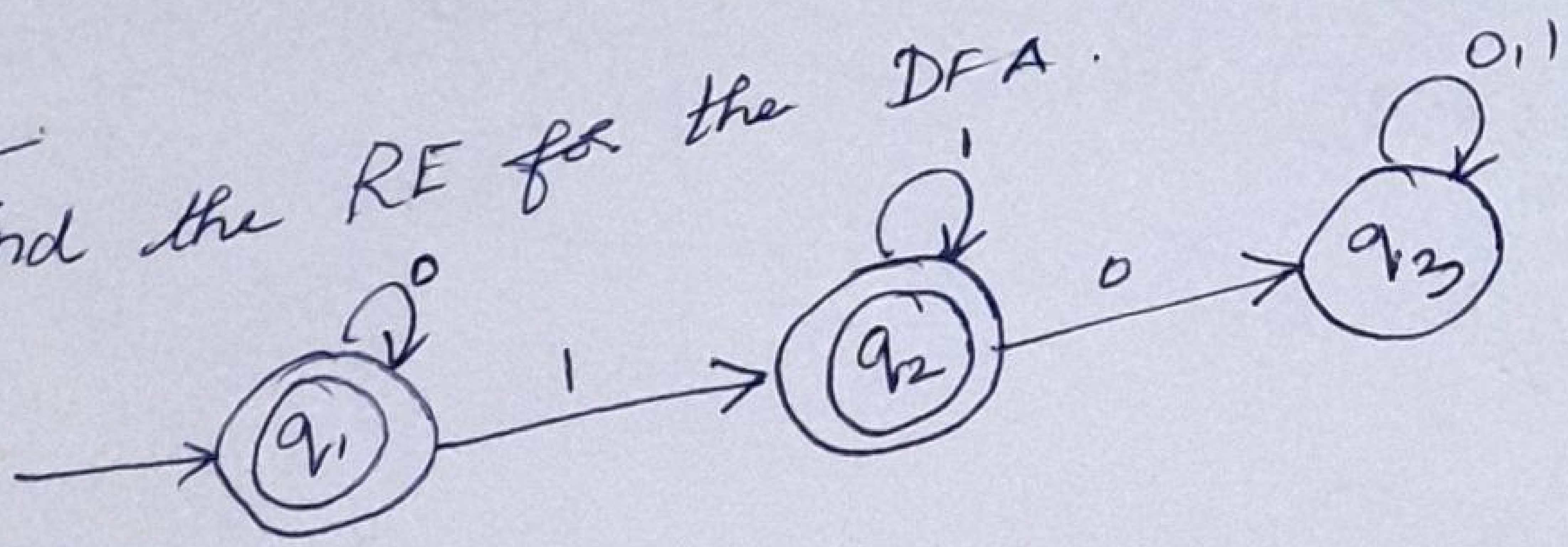
$$q_1 = (0 + (1(1+01)^*00))^*$$

Since q_1 is the final state, the RE is,

$$(0 + (1(1+01)^*00))^*$$

x ———————————————— x

Pom. Find the RE for the DFA.



Sln

For starting state add ϵ.

$$q_1 = q_1 0 + \epsilon \quad \text{——— (1)}$$

$$q_2 = q_1 1 + q_2 1 \quad \text{——— (2)}$$

$$q_3 = q_2 0 + q_3(0+1) \quad \text{——— (3)}$$

Apply Arden's theorem for (1)

$$q_1 = \epsilon 0^*$$

$$\boxed{q_1 = 0^*}$$

sub q_1 in (2).

$$q_2 = 0^* 1 + q_2 1$$

$$q_2 = q_2 1 + 0^* 1$$

Apply Arden's theorem

$$q_2 = 0^* 1 1^*$$

Since q_1 & q_2 are final states, the RE is,

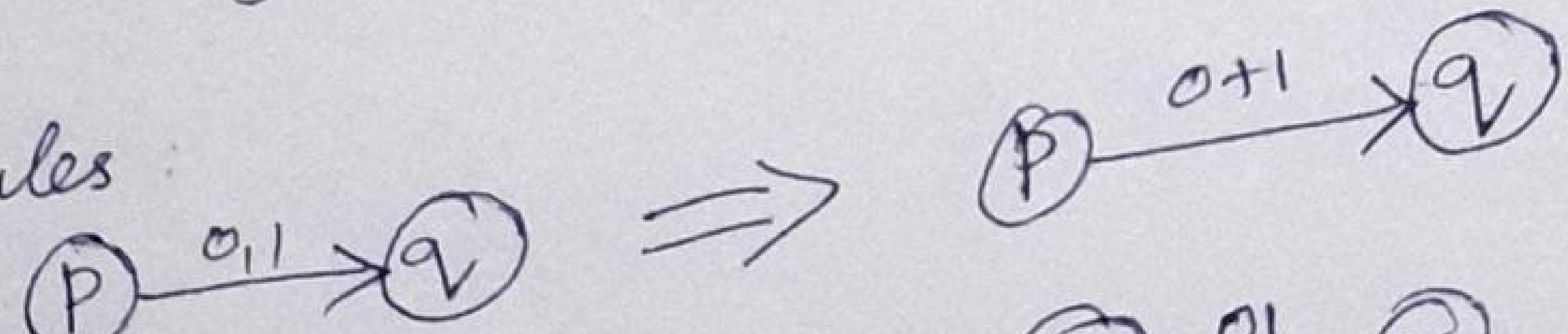
$$0^* + 0^* 1 1^*$$

x ———————— x

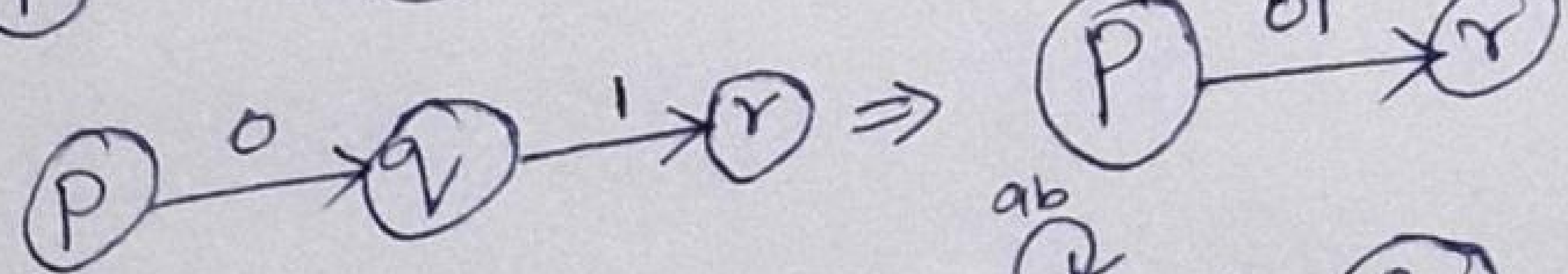
Converting DFA's to RE by eliminating states:

RE

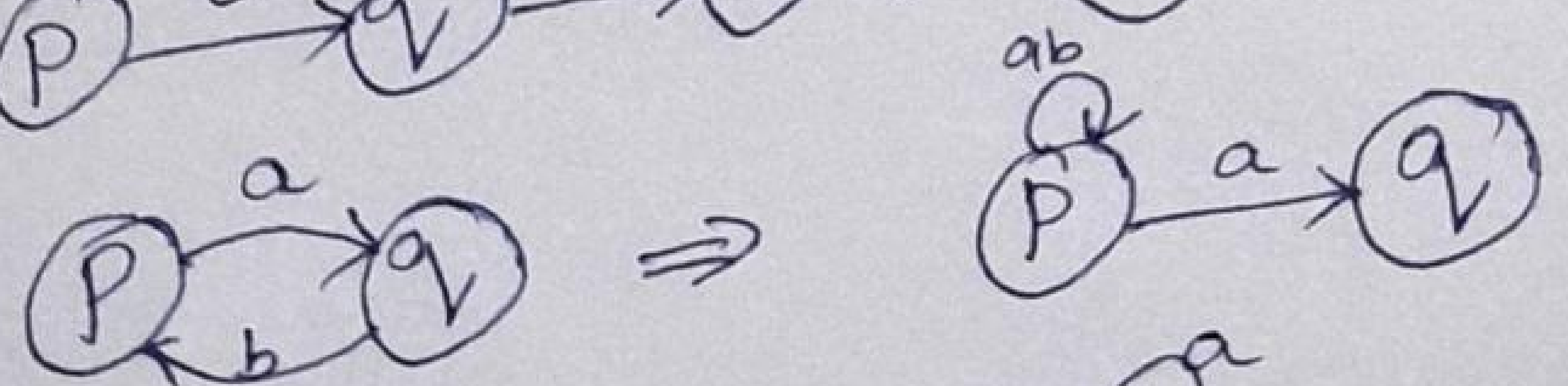
Rules:

1) 

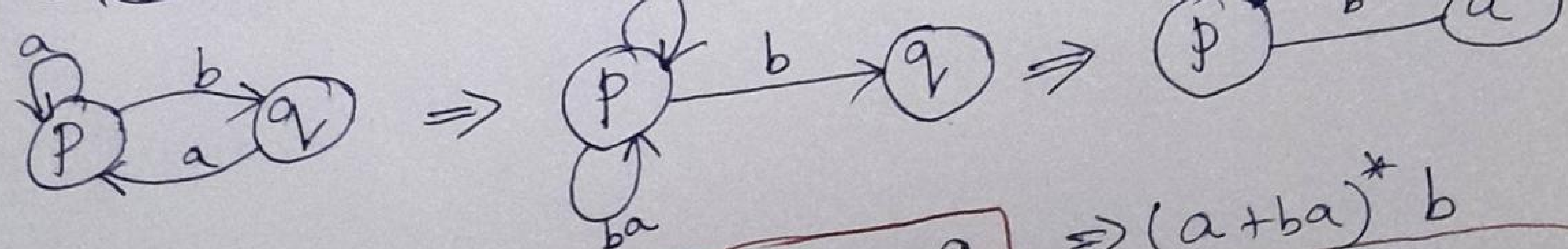
⇒ $0+1$

2) 

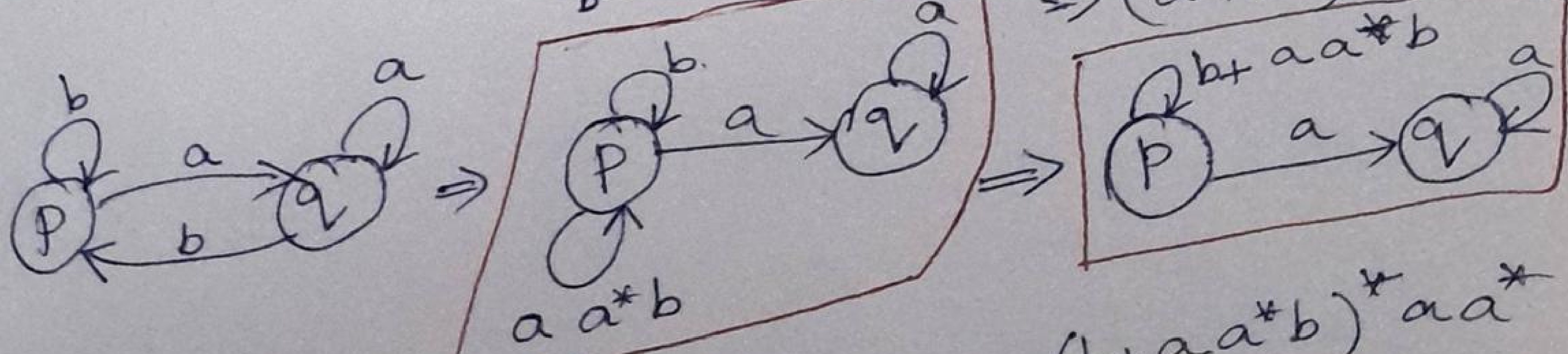
⇒ 01

3) 

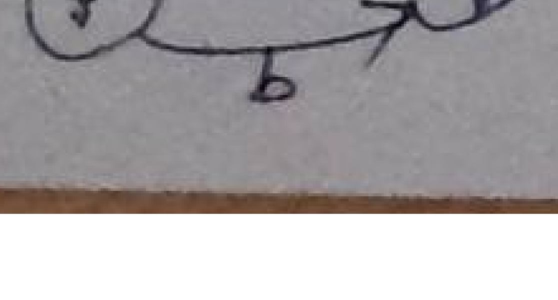
⇒ $(ab)^* a$

4) 

⇒ $(a+ba)^* b$

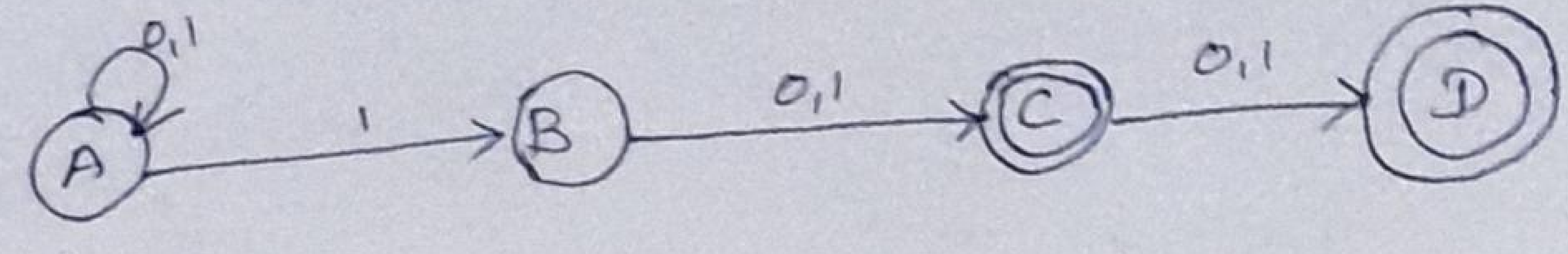
5) 

⇒ $(b+aa^*b)^* aa^*$

6) 

⇒ $a+b$

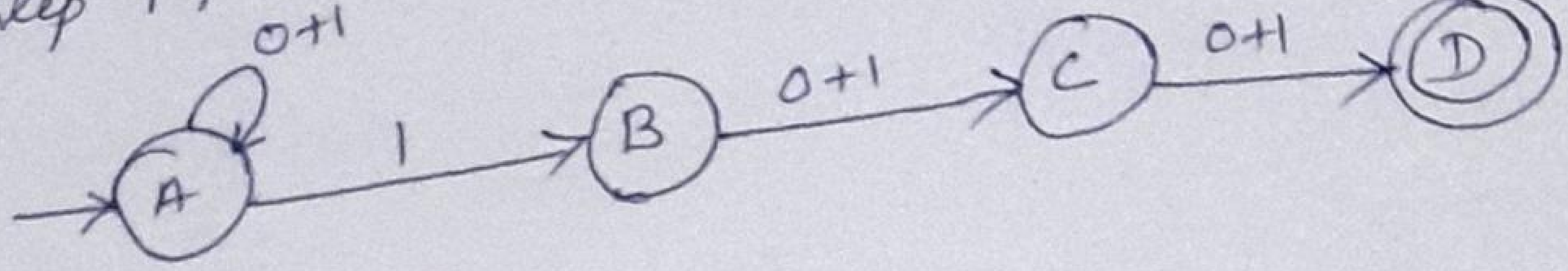
Find the RE for the DFA.



Sol

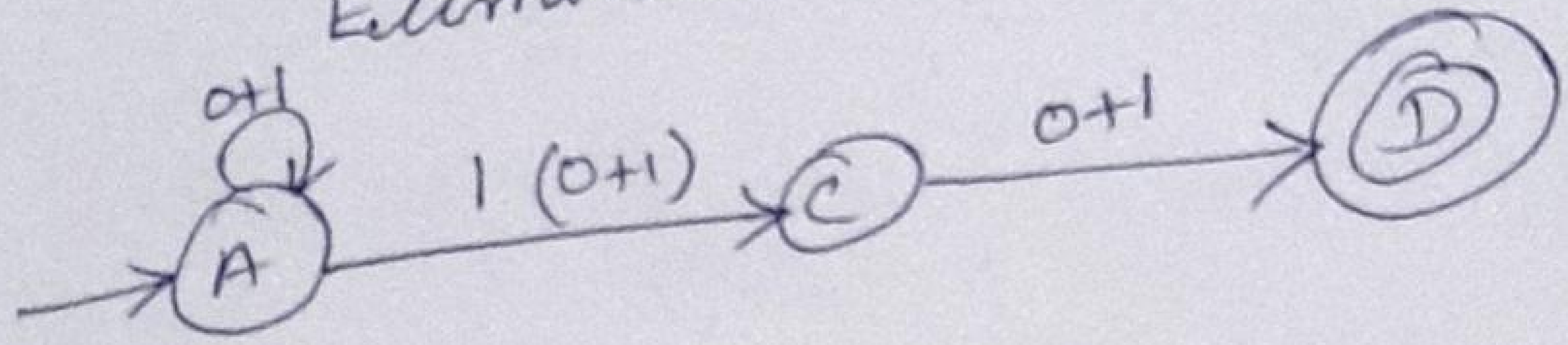
D as final state.

Step 1:

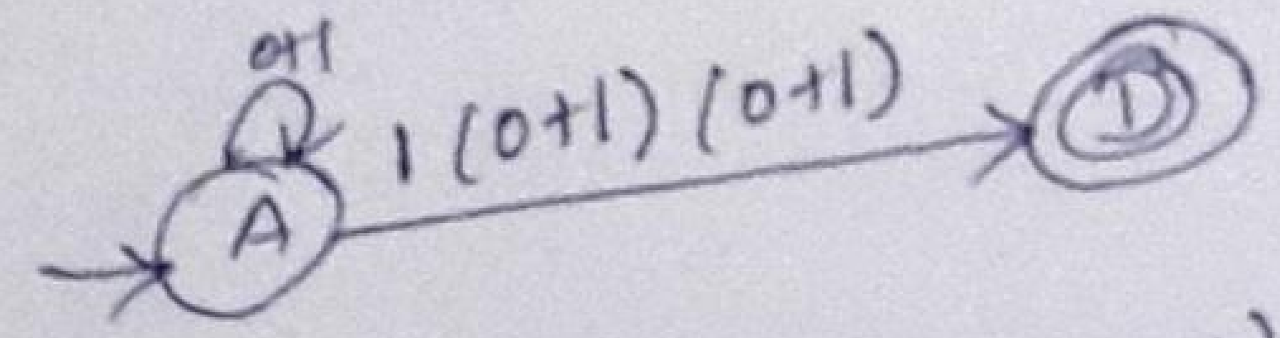


Step 2:

Eliminate B.



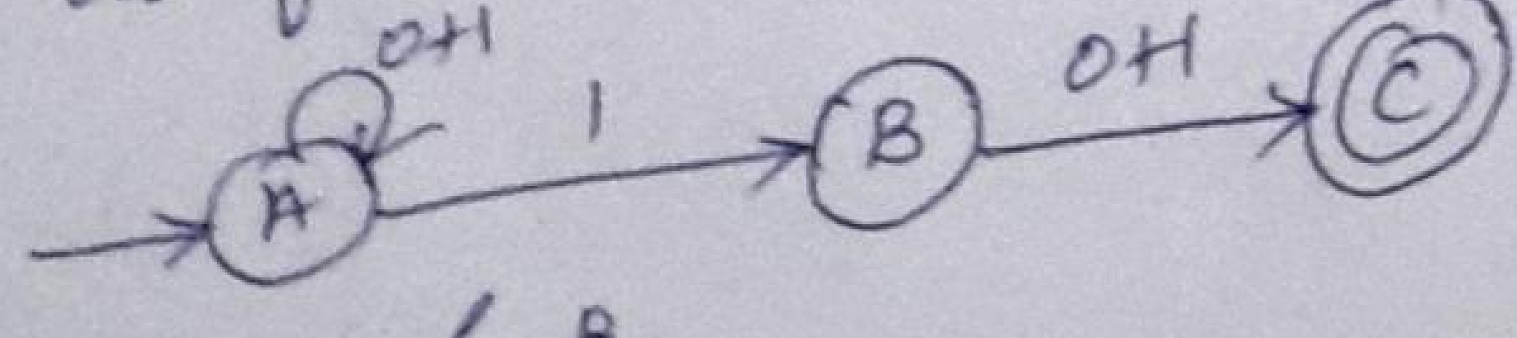
Step 3: Eliminate C.



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

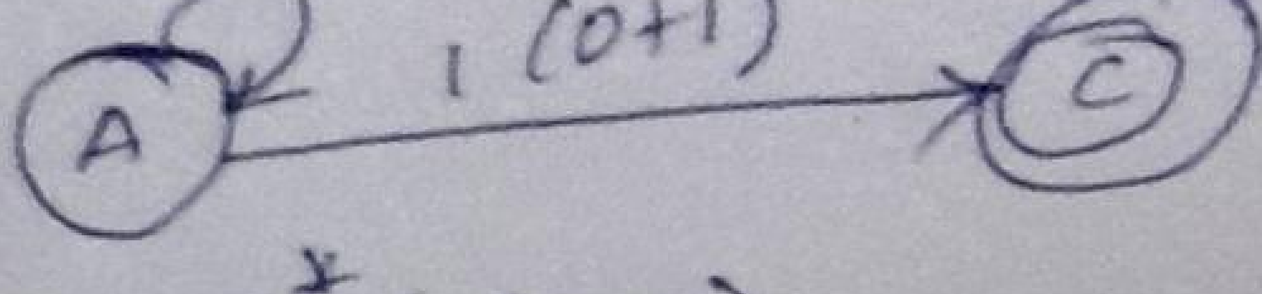
C as final state.

①.



eliminate B.

②.

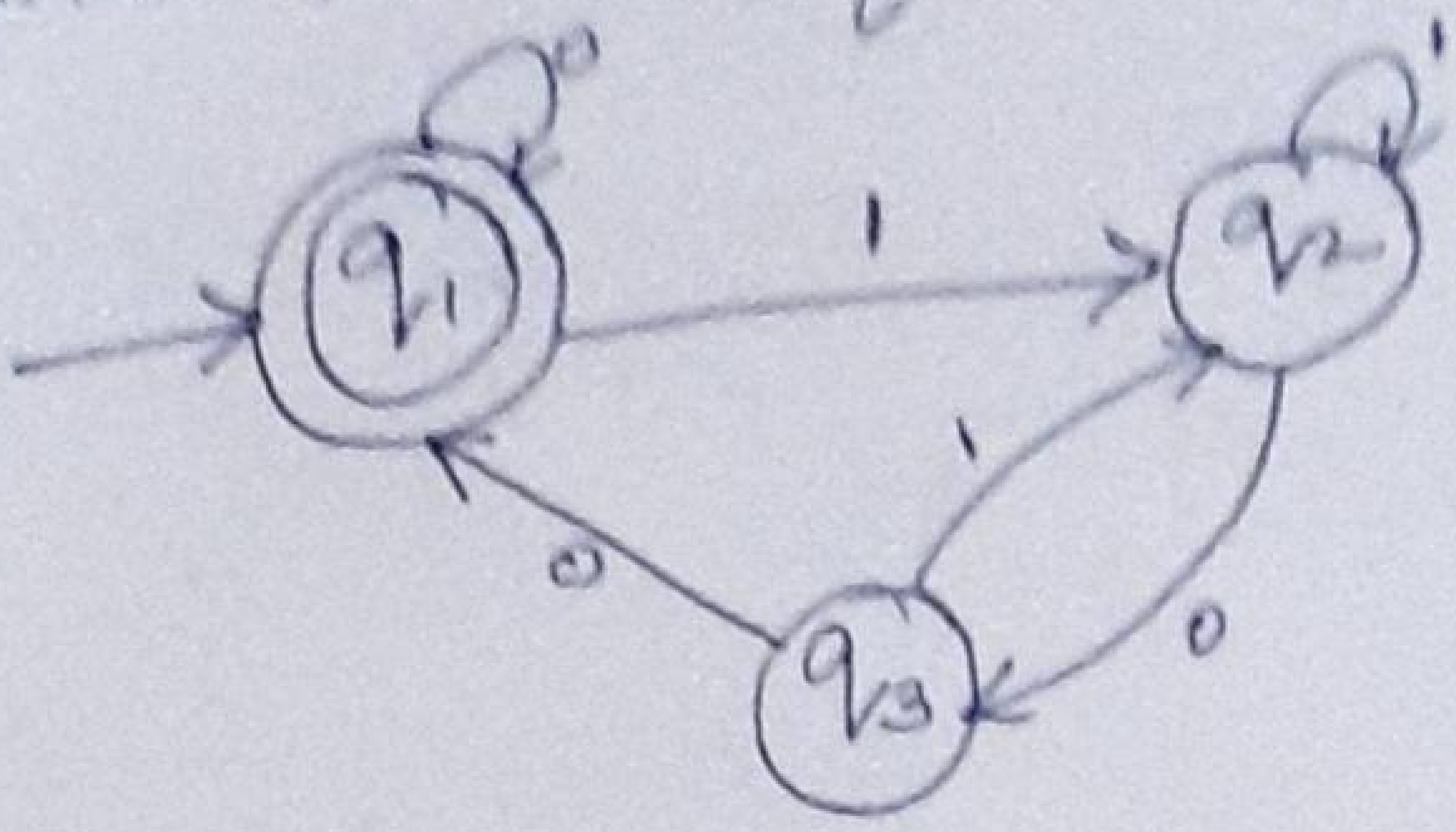


$(0+1)^* 1 (0+1)$

The RE is,

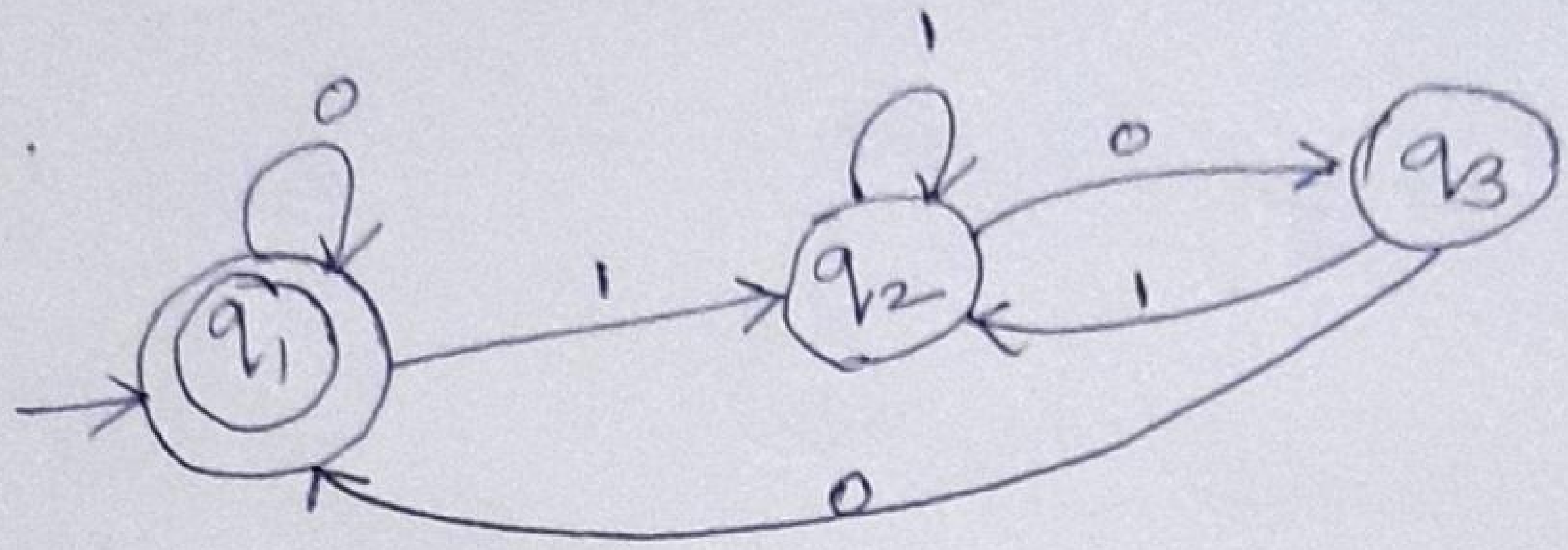
$(0+1)^* 1 (0+1)(0+1) + (0+1)^* 1 (0+1)$

Find the RE for the DFA.

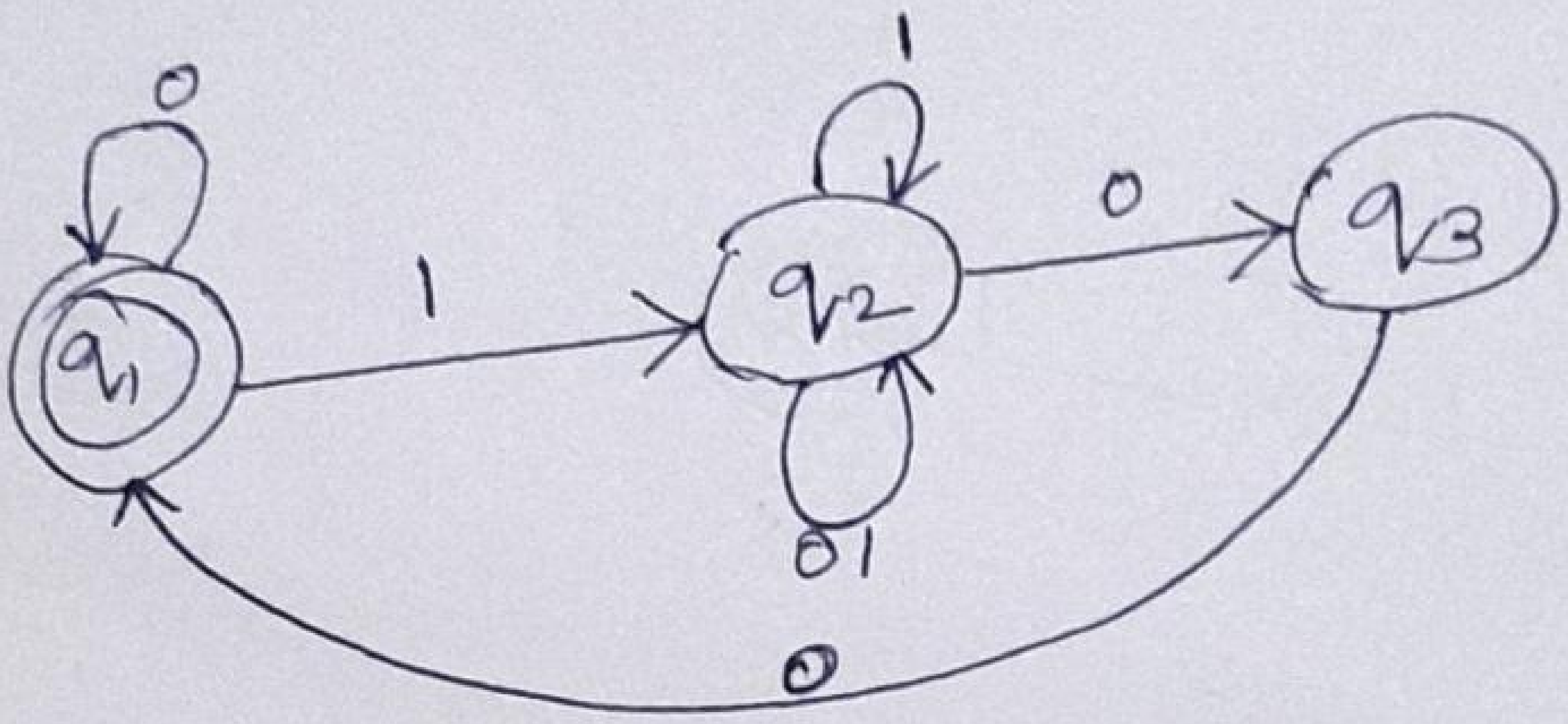


Ans.

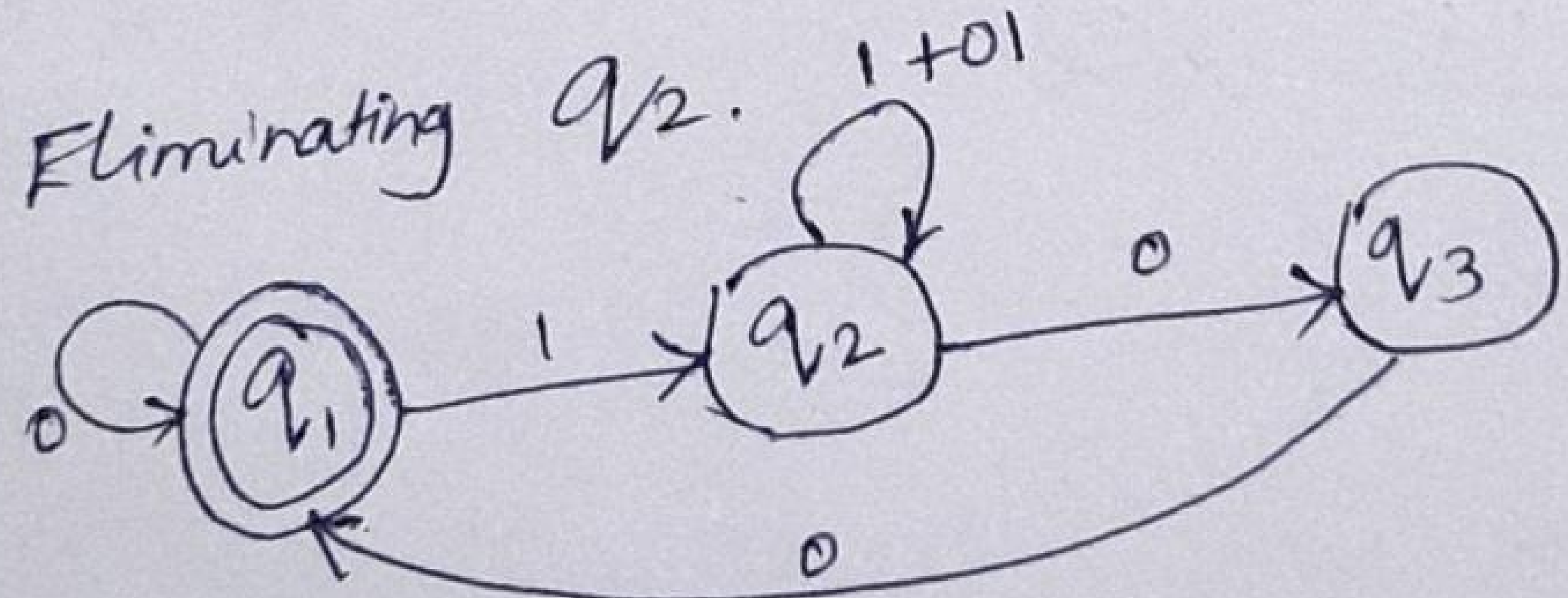
①



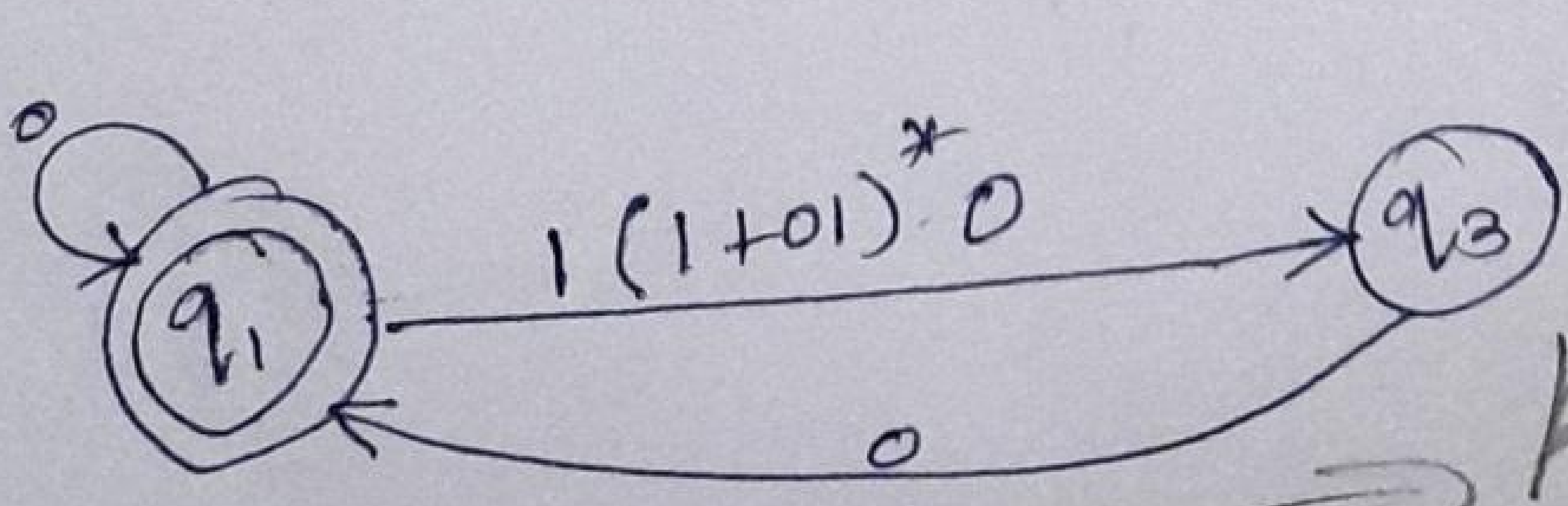
②.



③. Eliminating q_2.

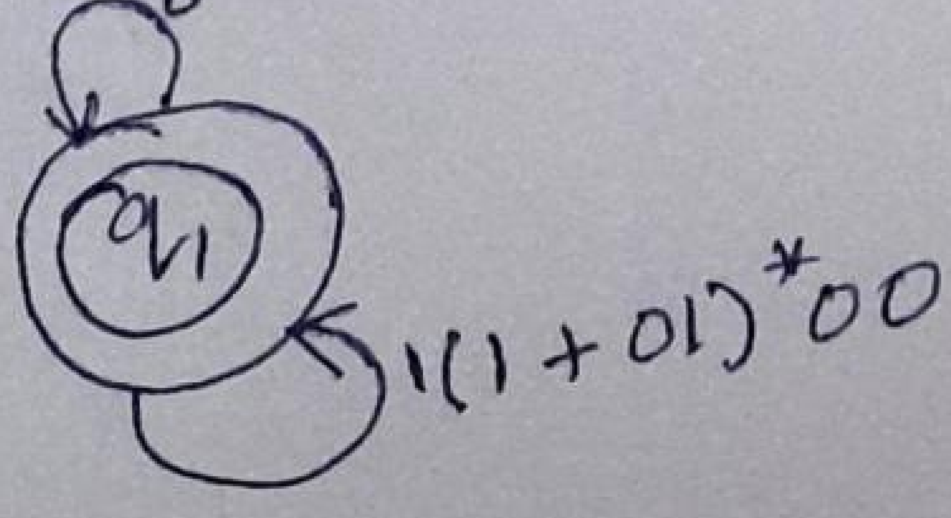


④.

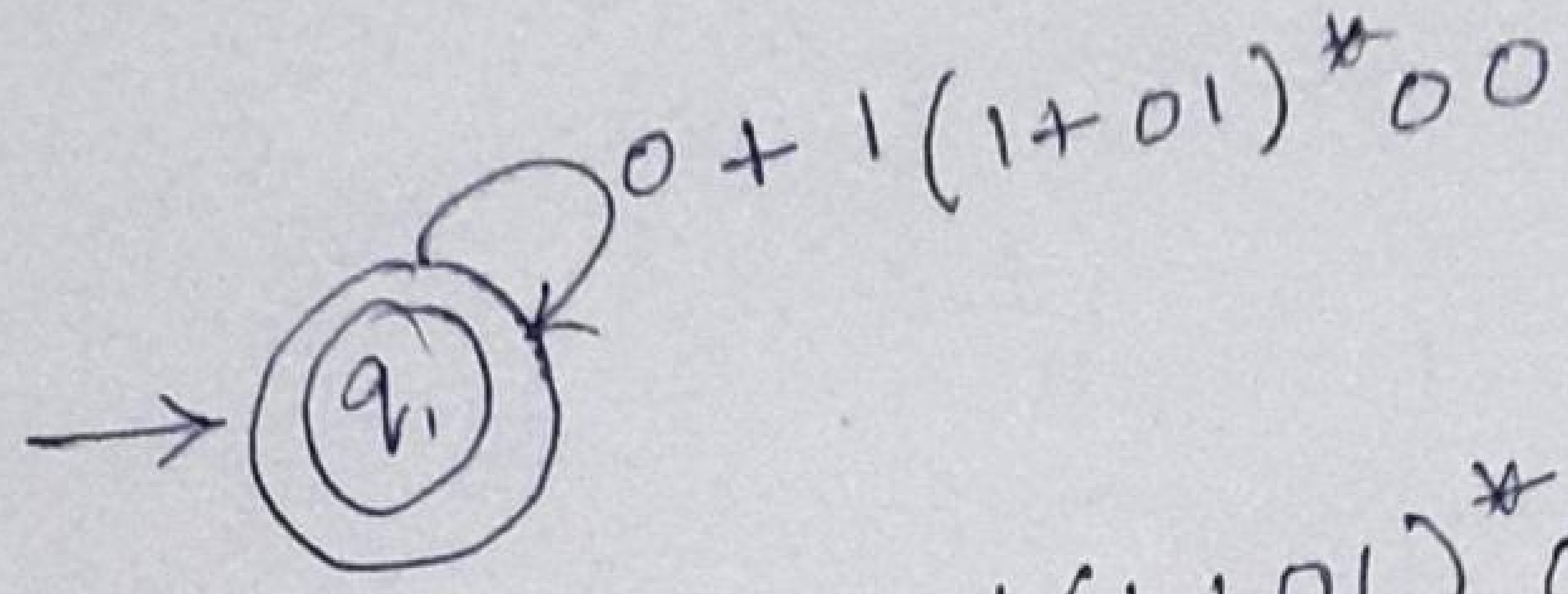


⇒ Rule 4 check this

⑤



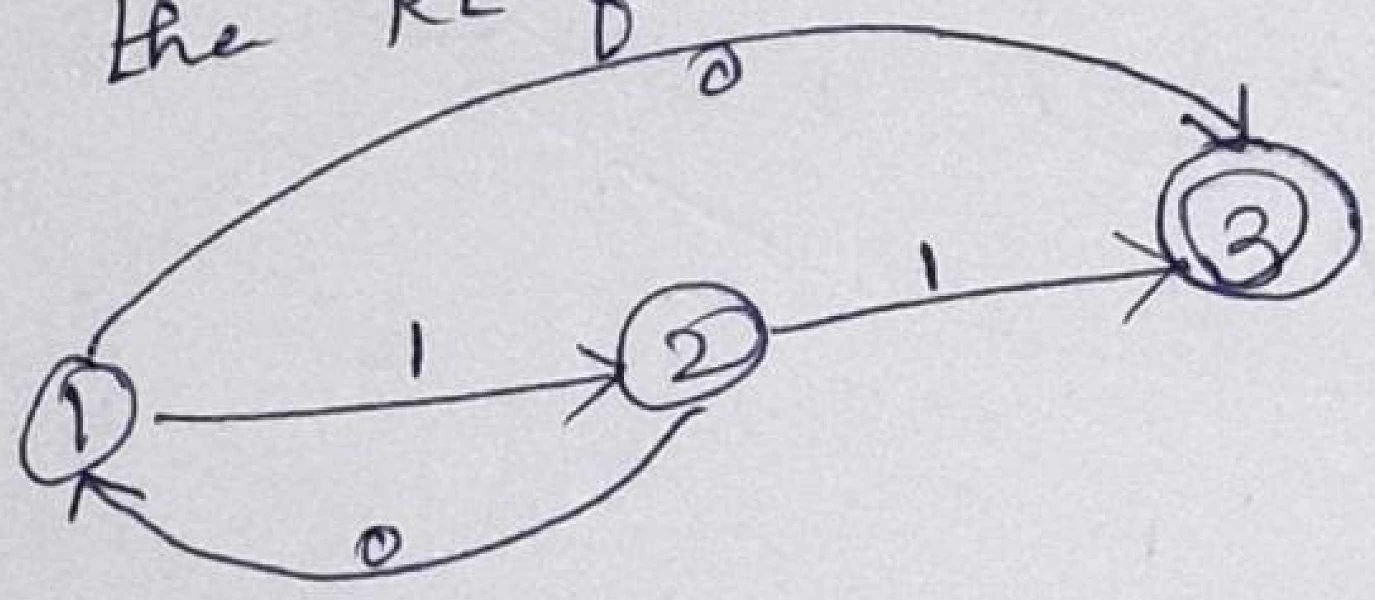
(6)



The RE is $(0+1(1+01)^*00)^*$

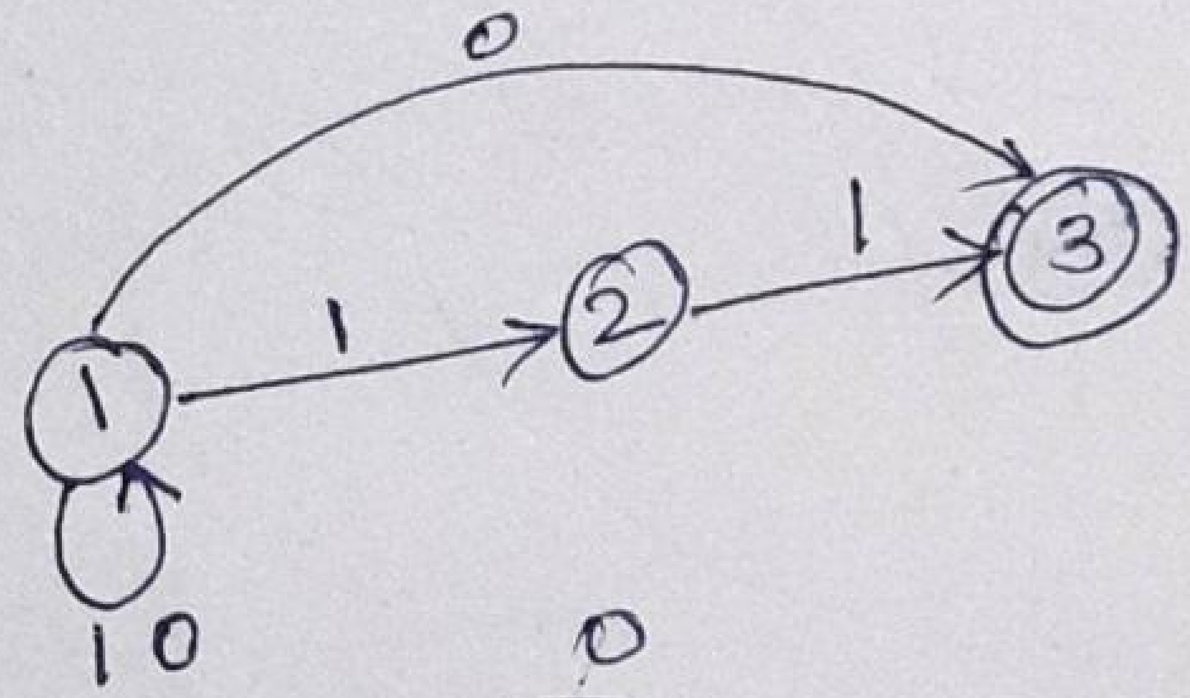
Plm.

Find the RE for the DFA.

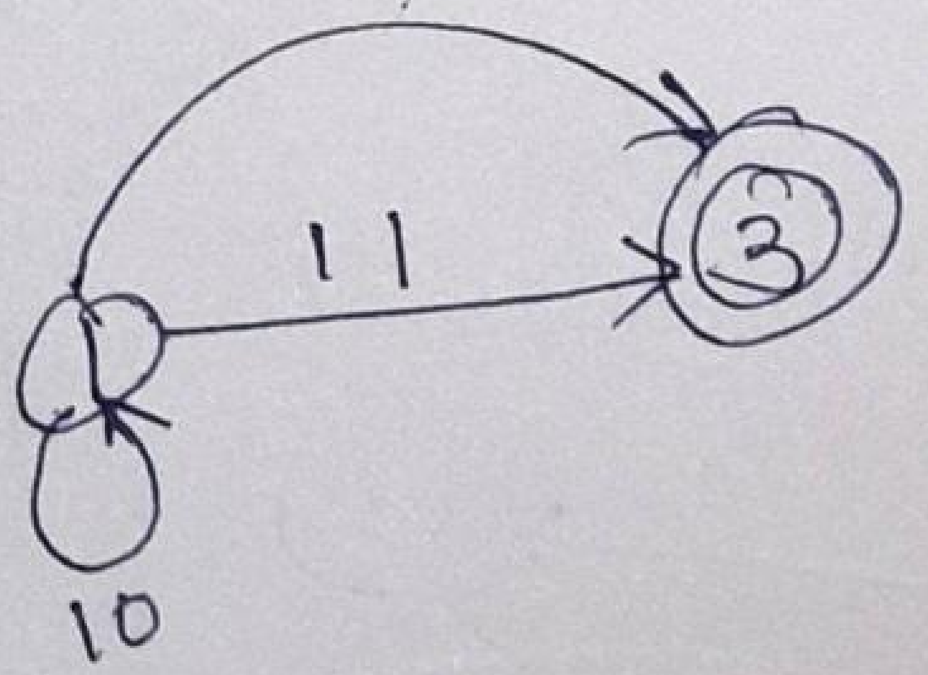


Sln.

(1).



(2).



The RE is

$0+(10)^*11$

Converting RE to Automata.

Theorem:

Every language defined by a RE is also defined by a finite automaton.

Proof:

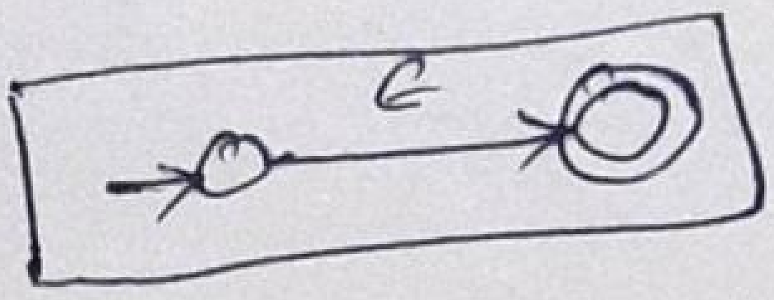
Suppose $L = L(R)$ for a RE

To prove that $L = L(E)$ for some ϵ-NFA E with

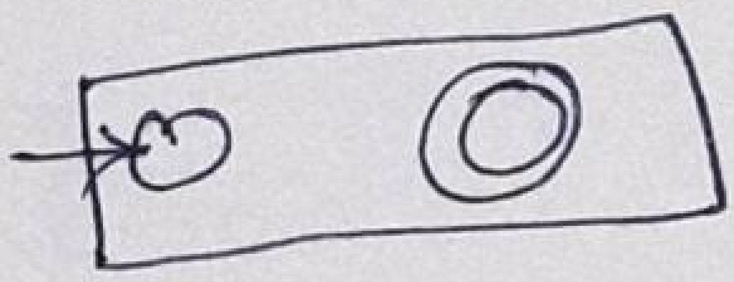
1) Exactly one accepting state
2) No arcs into the initial state
3) No arcs out of the accepting state.

Basis:

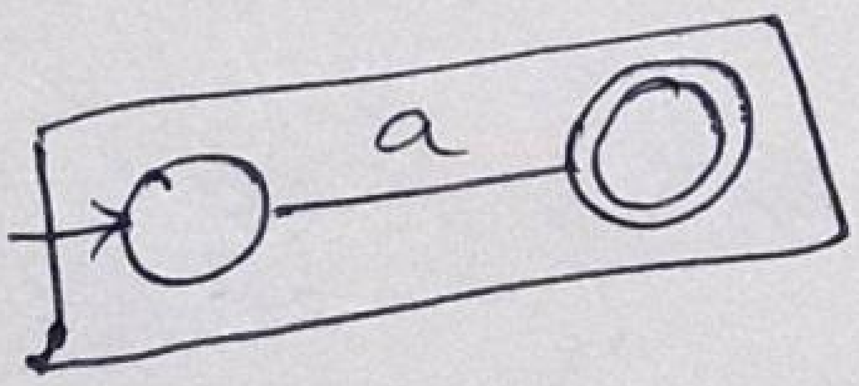
a) $r = \epsilon$



b) $r = \phi$



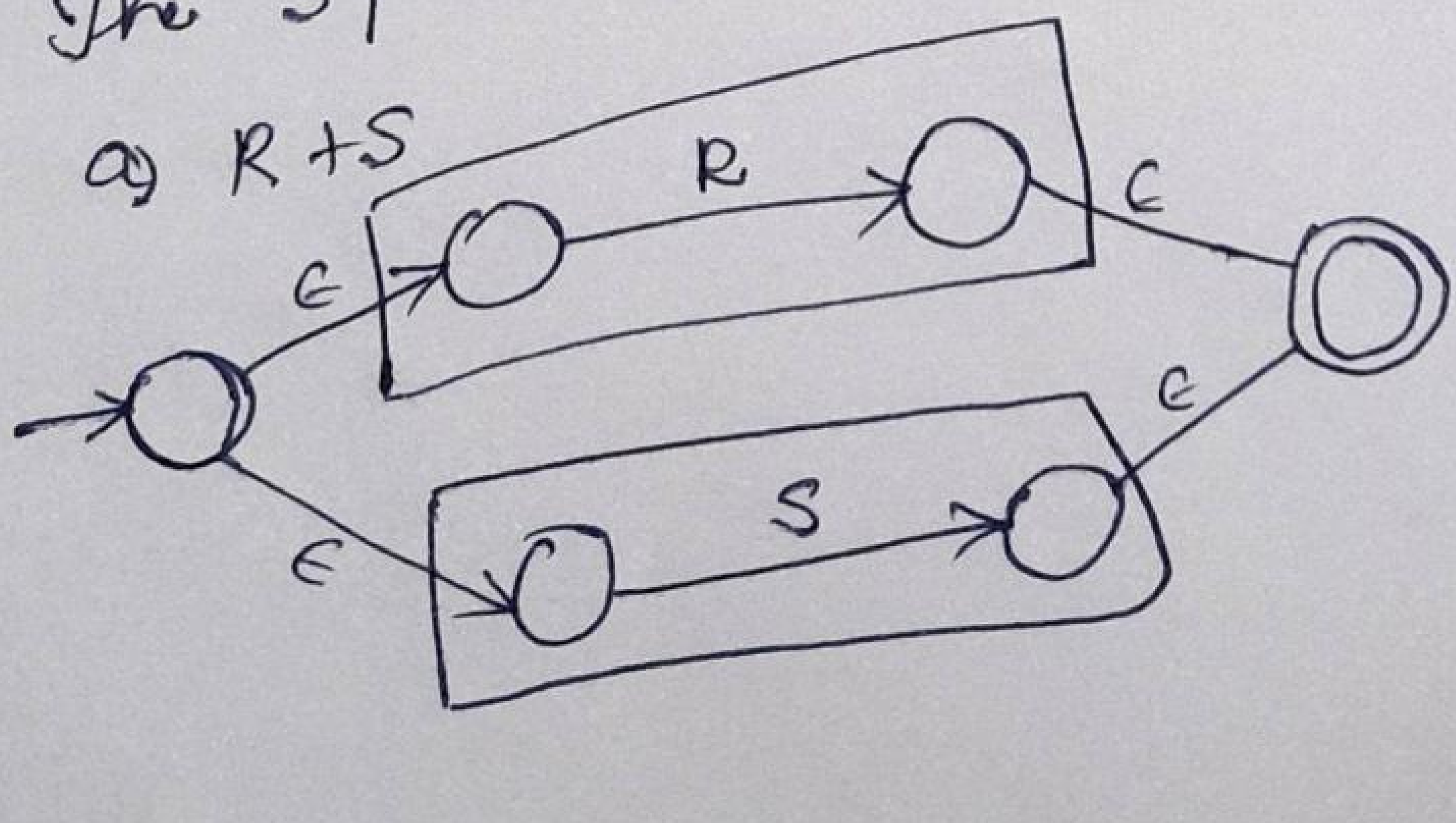
c) $r = a$



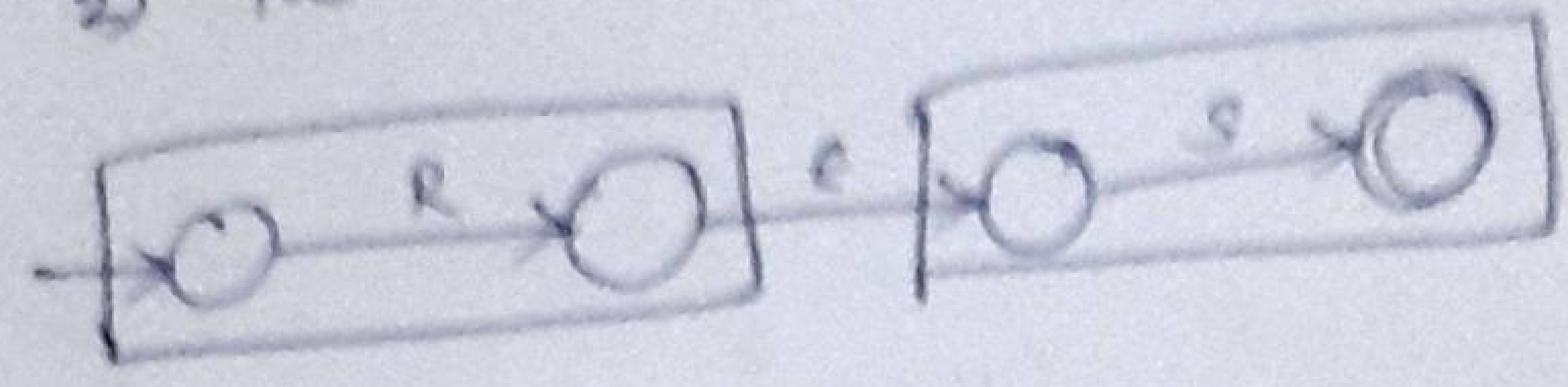
Induction:

The 3 parts of the induction

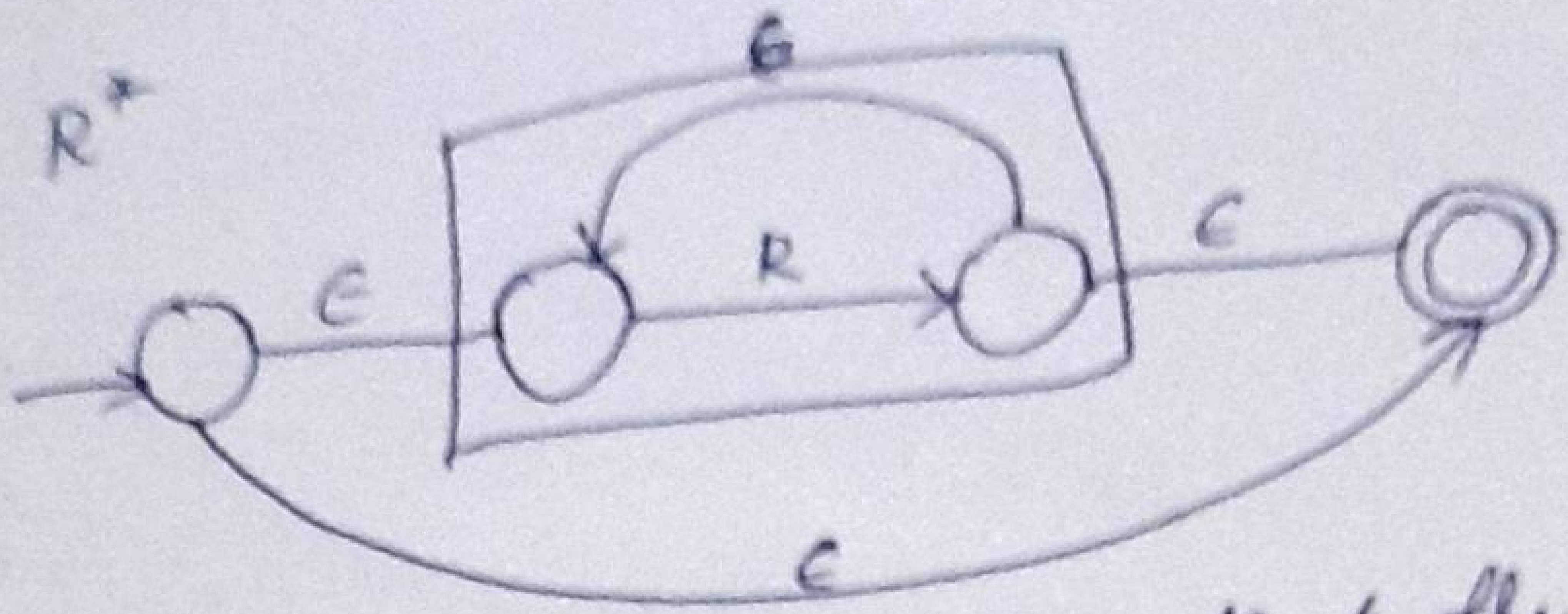
a) $R+S$



2) RS



3) R^*

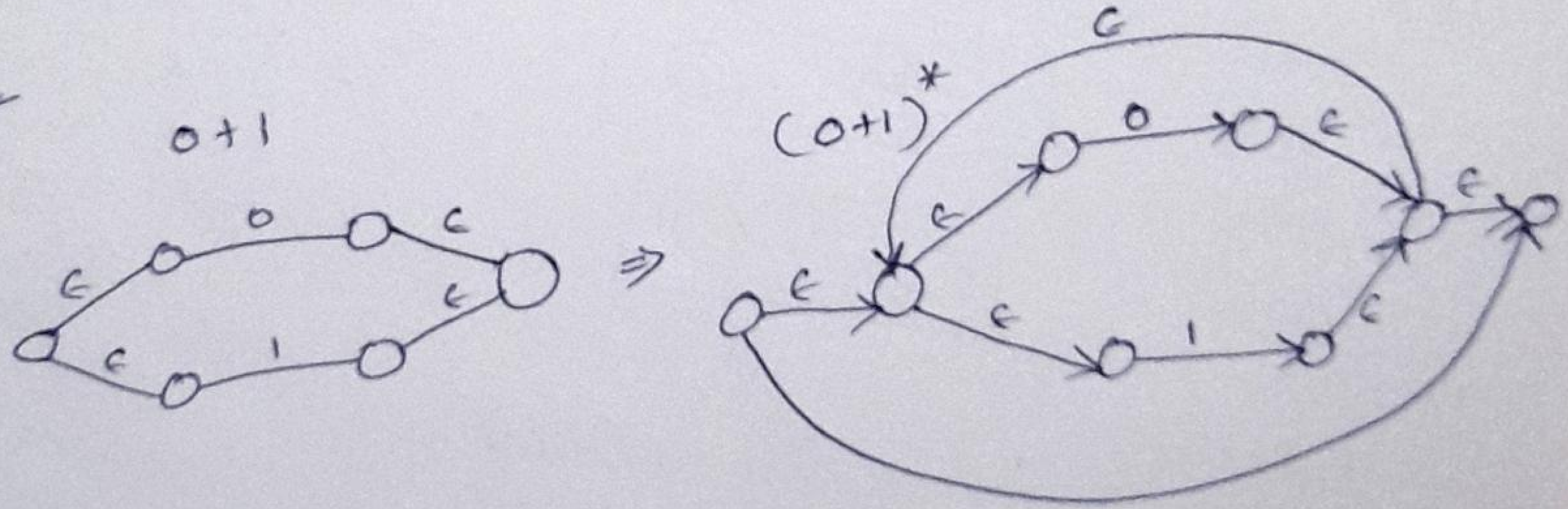


It is a simple observation that the constructed automata satisfy the 3 conditions given in the inductive hypothesis.

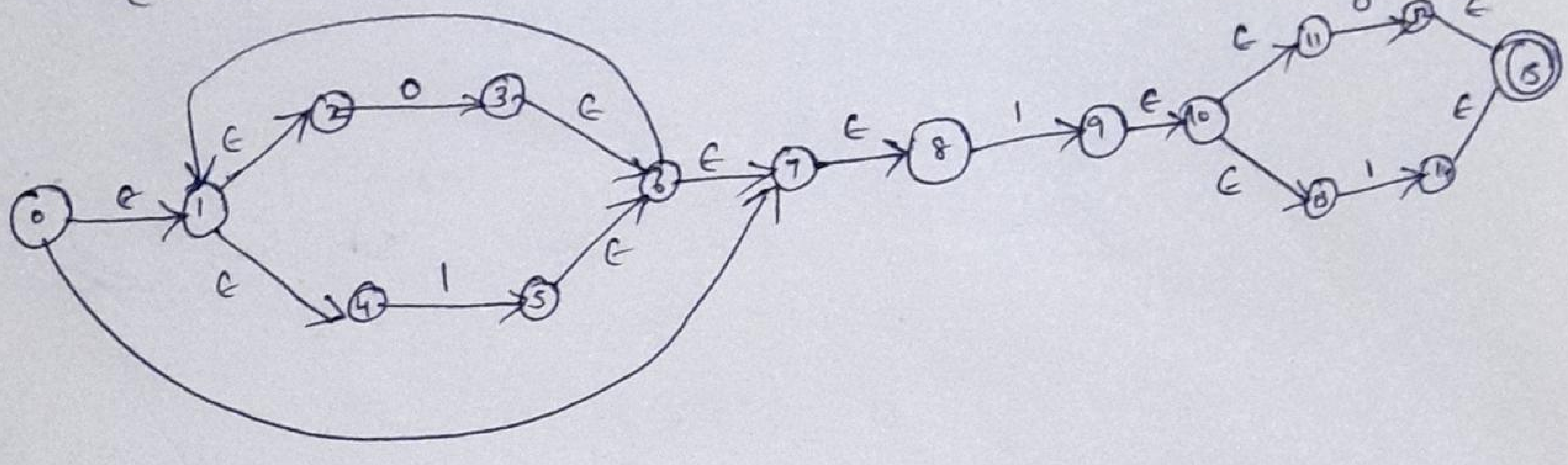
Hence proved.

Convert the RE $(0+1)^* 1(0+1)$ to an ε-NFA.

Ans



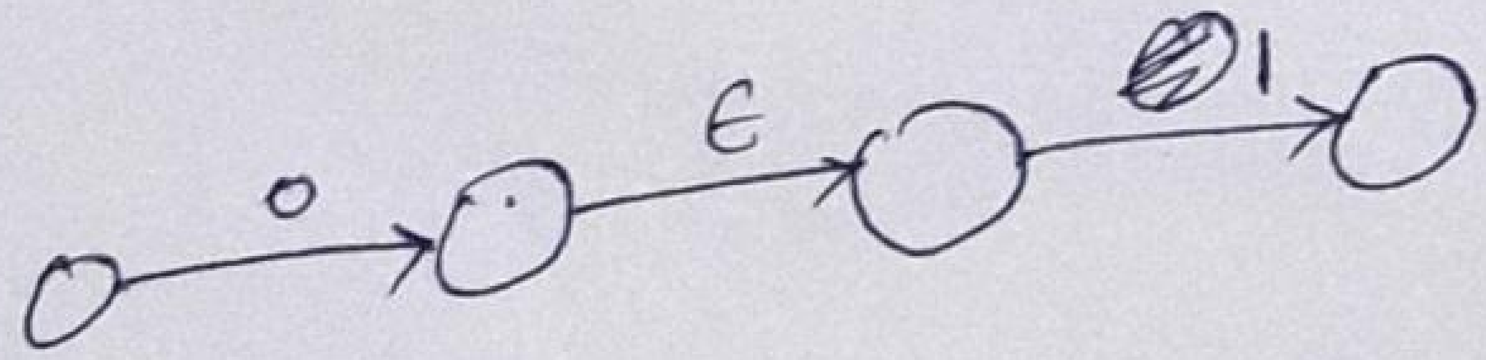
$(0+1)^* \, 1 \, (0+1)$



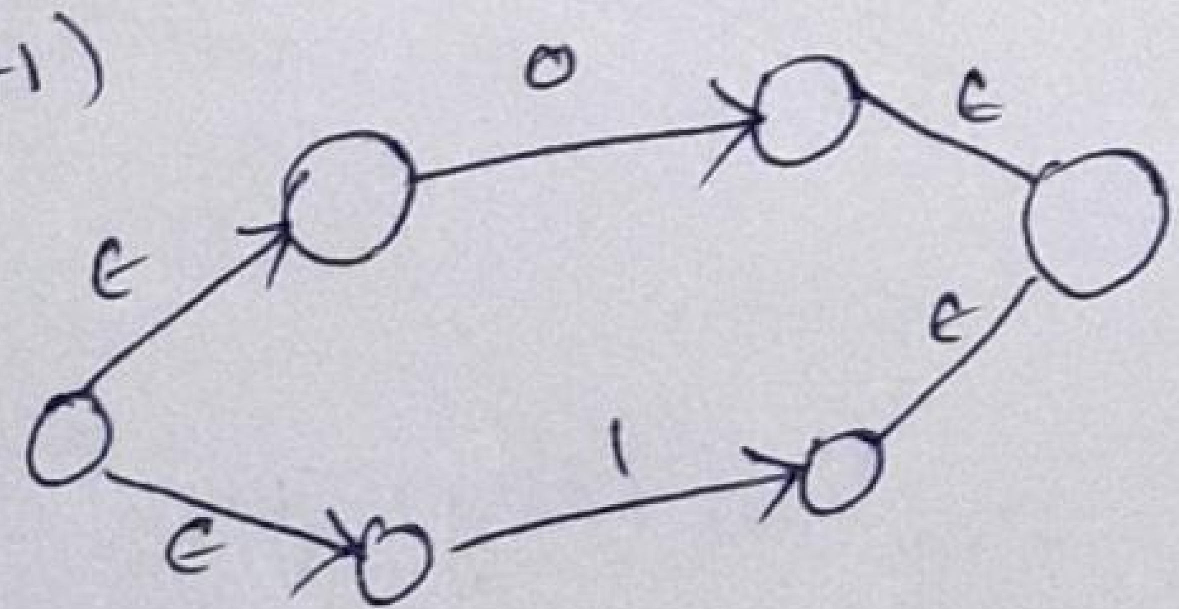
Convert the RE $((01) + (0+1))^*$ to an ε-NFA.

Sol.

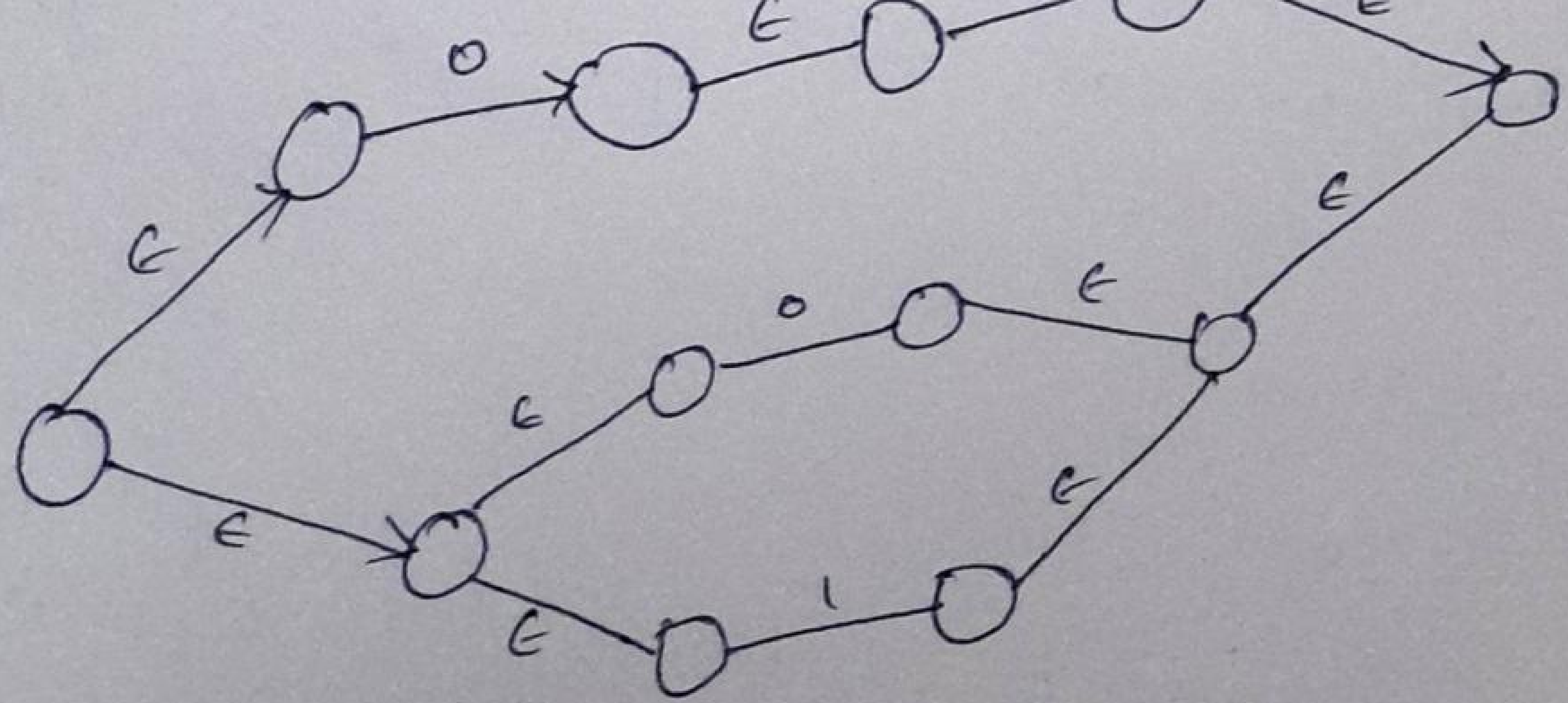
01



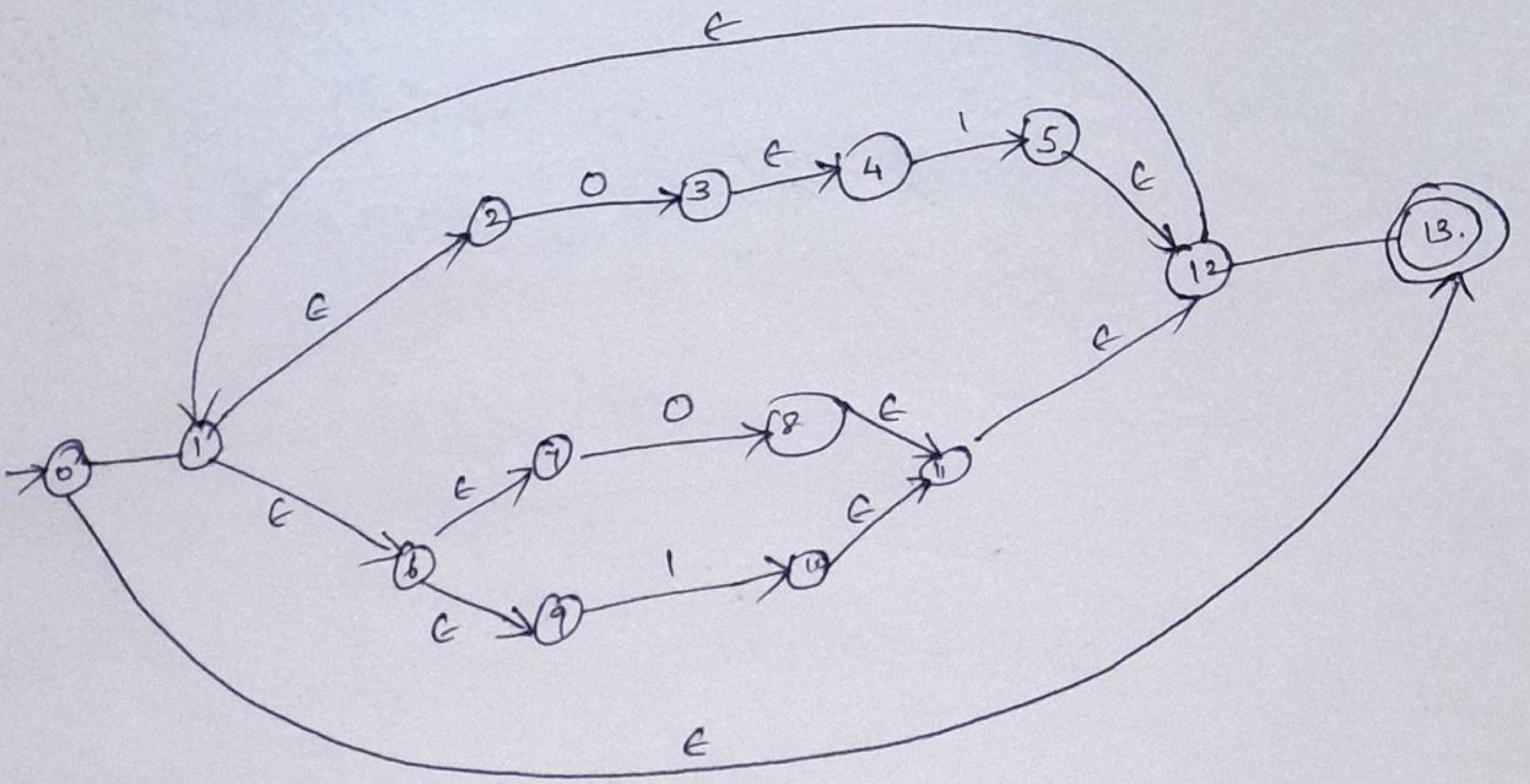
$(0+1)$



$(01) + (0+1)$



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



* ——————— *

PROVING LANGUAGES NOT TO BE REGULAR.

Theorem:

The Pumping Lemma for Regular Languages.

Let L be a Regular Language. Then there exists a constant n such that for every string ω in L such that $|\omega| \geq n$, we can break ω into 3 strings, $\omega = xyz$, such that.

1) $y \neq \epsilon$

2) $|xy| \leq n$

3. For all $k \geq 0$, the string xy^kz is also in L

Scanned by TapScanner

Proof:

Since L is regular, there exists a DFA, $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes it i.e., $L = L(M)$

Let no. of states in M is 'n'.

Now consider any string w of length 'n' or more say $w = a_1 a_2 \ldots a_m$ and each a_i is an i/p symbol

By Pigeonhole principle, we can find 2 different integers i & j with $0 \leq i \leq j \leq n$, such that

$P_i = P_j$.

Now we can break $w = xyz$ as follows.

1) $x = a_1 a_2 \ldots a_i$

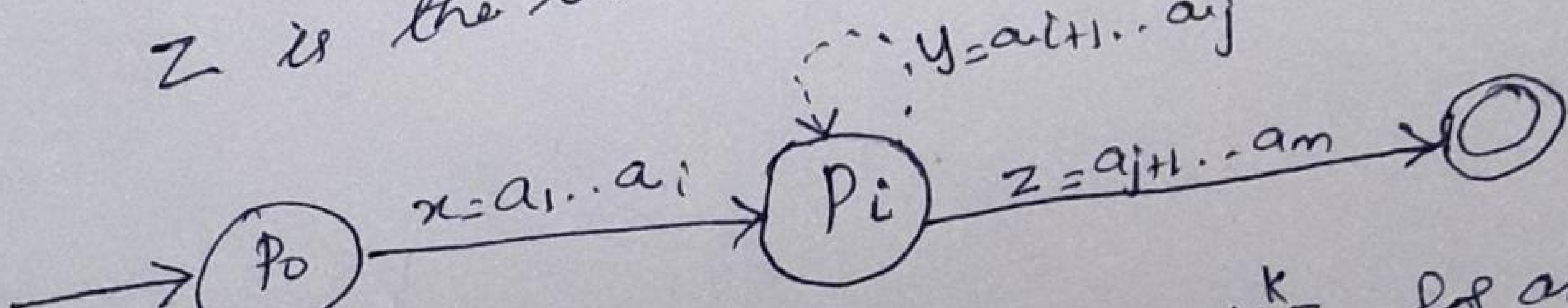
2) $y = a_{i+1} a_{i+2} \ldots a_j$

3) $z = a_{j+1} a_{j+2} \ldots a_m$

i.e., x takes us to P_i once

y takes us from P_i back to P_i,

z is the balance of w



Now when A receives the i/p $xy^k z$ for any $k \geq 0$,

if $K = 0$, then the automation goes from the start state P_0 to P_i an i/p x.

Since P_i is also P_j, it must be that A goes from P_i to the accepting state on input z.

Thus A accepts x z.

If $k>0$, then A goes from P_0 to P_i on input x, circles from P_i to P_i k times on i/p y^k and then goes to the accepting state on I/P z.

Thus for any $k \geq 0$, xy^kz is also accepted by A.

i.e., xy^kz is in L.

$\times \xrightarrow{\text{Hence proved.}} \times$

Show that $L = \{0^n 1^n \mid n \geq 1\}$ is not Regular.

Proof

Suppose L is regular,

then L will be accepted by FSA

By Pumping Lemma we write,

$w = xyz$ with $|xy| \leq n$ & $y \neq 0$

consider $0^m 1^n \in L$

$$0^m 1^m = \underbrace{\overbrace{\underbrace{000}_{x}\underbrace{0\ldots\ldots 00}_{y}}^{0^m}0}\overbrace{\underbrace{11\ldots\ldots1}_{z}}^{1^m}$$

$xy^kz = xyy^{k-1}z$

$xy = 0^p$

$y = 0^v$

$z = 0^{m-p}1^m$

$$xy^k z = xyy^{k-1}z$$

$$= 0^p 0^{q(k-1)} 0^{m-p} 1^m$$

for $k=1$

$$xy^k z = 0^p 0^{(0)} 0^{m-p} 1^m$$

$$= 0^{p+m-p} 1^m$$

$$= 0^m 1^m \in L$$

for $k=2$

$$xy^k z = 0^p 0^{q(1)} 0^{m-p} 1^m$$

$$= 0^{p+q+m-p} 1^m$$

$$= 0^{q+m} 1^m \notin L$$

In the given language the no of 0's and 1's are equal. So it is not regular

Hence proved

x ——————— x

Show that $L = \{a^{i^2} \mid i \geq 1\}$ is not regular.

Proof.

Suppose L is regular,

Then L will be accepted by FSA

By pumping lemma, we write,

$w = xy^kz$ with $|xy| \leq n$ & $y \neq 0$

Consider, $a^{i^2} \in L$

$$a^{i^2} = a^p \; [p = i^2]$$

$$\underbrace{\overbrace{a\,a\,a}^{x}\overbrace{\dots\; aaa \dots\; a}^{y}}_{p}$$

$$xy^kz = xyy^{k-1}z$$

$x = a^q$

$y = a^r$

$xy = a^s \rightarrow [q + r = s]$

$z = a^{p-s}$

$$xy^kz = a^s a^{r(k-1)} a^{p-s}$$

for $k = 1$;

$$xy^kz = a^s a^{r(0)} a^{p-s} \quad \text{? powers will be added}$$

$$= a^p = a^{i^2} \in L$$

for $k = 2$,

$$xy^kz = a^s a^{r(1)} a^{p-s}$$

$$= a^{p+r} = a^{i^2+r} \notin L$$

Since $i^2 + r$ is not a perfect square,

$L = \{a^{i^2} \mid i \geq 1\}$ is not a RL.

CLOSURE PROPERTIES OF REGULAR LANGUAGE (9 properties).

1. The Union of 2 regular set is regular.

Let us take 2 REs.

$RE1 = a(aa)^*$ & $RE2 = (aa)^*$

So $L1 = \{a, aaa, aaaaa, \ldots\}$ strings of odd length excluding NULL

$L2 = \{\epsilon, aa, aaaa, aaaaaa \ldots\}$ Strings of even length including NULL.

$L1 \cup L2 = \{\epsilon, a, aa, aaa, aaaa \ldots\}$ Strings of all possible lengthes including NULL.

$RE(L1 \cup L2) = a^*$ (which is a RE itself).

2. The intersection of 2 Regular set is regular.

$RE1 = a(a^*)$ & $RE2 = (aa)^*$

$L1 = \{a, aa, aaa, aaaa \ldots\}$ String of all possible lengths excluding NULL

$L2 = \{\epsilon, aa, aaaa, aaaaaa \ldots\}$ String of all even lengths including NULL.

$L1 \cap L2 = \{aa, aaaa, \ldots\}$ String of all even length excluding NULL.

$RE(L1 \cap L2) = aa(aa)^*$ which is RE itself.

3. Complement of Regular set is regular.

$RE = (aa)^*$

$L \{\epsilon, aa, aaaa \ldots\}$ String of even length including NULL

Complement of L is all string not in L.

$L' = \{a, aaa, aaaaa\}$ String of odd L. excluding NULL.

$RE(L') = a(aa)^*$ which is a RE.

4. The difference of 2 Regular set is regular.

$RE1 = a(a^*)$ & $RE2 = (aa)^*$

$L1 = \{a, aa, aaa, aaaa \ldots\}$ String of all possible L, excluding NULL.

$L2 = \{\epsilon, aa, aaaa, aaaaaa \ldots\}$ String of even L including NULL

$L1 - L2 = \{a, aaa, aaaaa \ldots\}$ String of odd L excluding NULL.

$RE(L1-L2) = a(aa)^*$ which is a RE.

5. The reverse of a regular set is regular

We have to prove LR is also regular if L is a regular set.

Let $L = \{01, 10, 11, 10\}$

$RE(L) = 01 + 10 + 11 + 10$

$LR = \{10, 01, 11, 01\}$

$RE(LR) = 01 + 10 + 11 + 10$ which is regular.

Scanned by TapScanner

6. The closure of a Regular set is regular

If $L = \{a, aaa, aaaaa, \ldots\}$ (String of odd length excluding NULL).

$RE(L) = a(aa)^*$

$L^* = \{a, aa, aaa, aaaa, \ldots\}$ (String of all length excluding NULL)

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

7. The concatenation of 2 regular set is regular

Let $RE1 = (0+1)^*0$ & $RE2 = 01(0+1)^*$

Here $L1 = \{0, 00, 10, 000, 010, \ldots\}$ (Set of string ending in 0)

$L2 = \{01, 010, 011 \ldots\}$ (Set of string beginning with 01)

Then $L1L2 = \{001, 00110, 0011, 0001, 00010, 00011, 1001, 10010, \ldots\}$

(Set of strings containing 001 as a substring which can be represented by a Regular expression.

$(0+1)^* 001 (0+1)^*$.

Hence proved.

Scanned by TapScanner

8. The homomorphism of regular language is regular.

Homomorphism → substitution of a string by some other symbols.

eg. String 'aabb' can be written as 0011

Let Σ is the set of i/p alphabets and Γ be the substitution symbols.

Then $\Sigma^* \rightarrow \Gamma^*$ is homomorphism

Let $w = a_1 a_2 \ldots a_n$

$h(w) = h(a_1) h(a_2) \ldots h(a_n)$

$h(L) = \{ h(w) : w \in L \}$

$h(L) \rightarrow$ homomorphic image of L.

9. The inverse homomorphism of regular language is regular.

Let $\Sigma^* \rightarrow \Gamma^*$ is homomorphism.

Let L be the RL where $L \in \Sigma$, the $h(L)$ be homomorphic language.

The inverse homomorphic language can be represented by $h^{-1}(L)$

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

Hence proved.

EQUIVALENCE AND MINIMIZATION OF AUTOMATA

Step 1:

Construct ε-NFA from the given regular expression.

Step 2:

Find the ε-closure of the state q_0 from the constructed ε-NFA.

Step 3:

Perform the following steps until there are no more new state has been constructed.

i) Find the transition of the given RE symbols over Σ from the new state, i.e., move (new state symbol)

ii) Find the ε-closure of move (new state symbol).

Step 4:

Draw the DFA transition table and diagram.

Step 5:

Split the states into final states & non final states

Step 6:

Combine the states that have same moves for all the input.

Step 7:

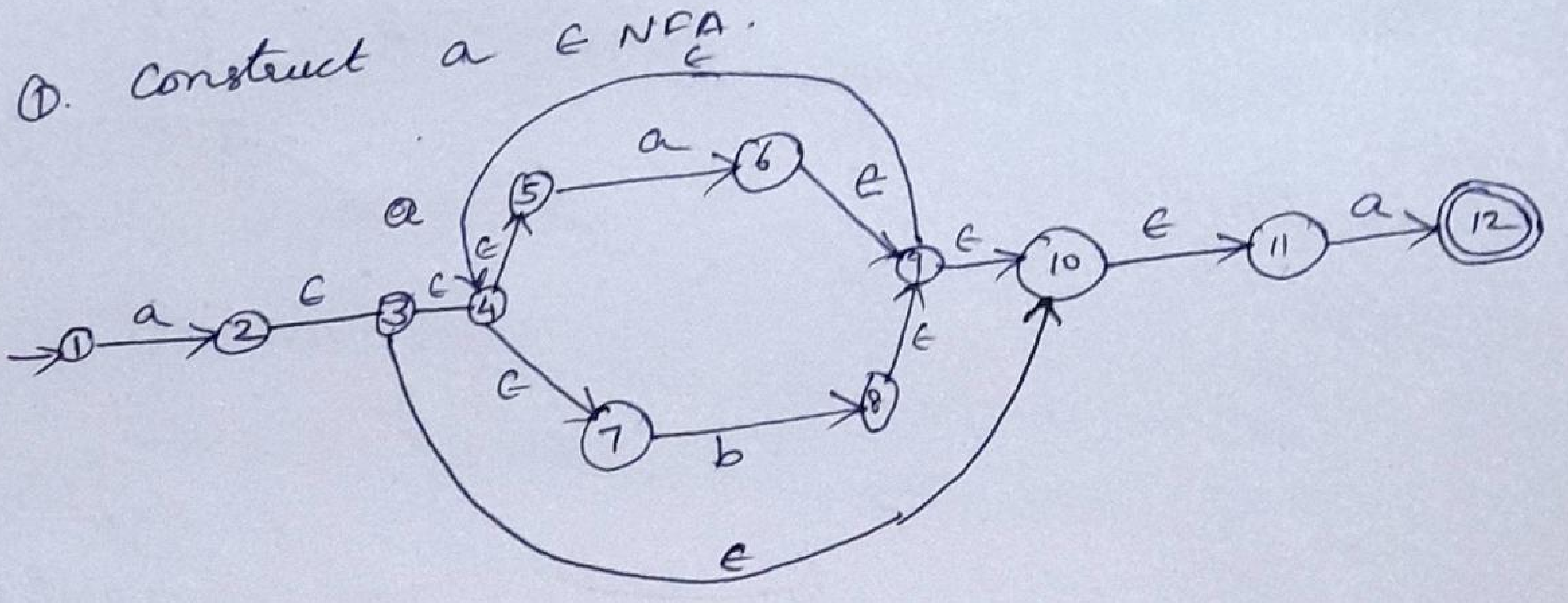
Now the DFA is minimized & draw the transition table & diagram for the minimized DFA

RE → ε NFA → DFA → Minimized DFA.

1 Construct a minimized DFA for the RE

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

① Construct a ε NFA.



② Conversion of ε-NFA to DFA.

ε-closure (1) = {1} ——— (A)

Move (A, a) = {2}.

ε-closure(Move (A, a)) = {2, 3, 4, 5, 7, 10, 11} — (B)

Move (A, b) = φ

Move (B, a) = {6, 12}

ε-closure (Move (B, a)) = {6, 9, 10, 11, 4, 5, 7, 12}

= {4, 5, 6, 7, 9, 10, 11, 12} ——— (C)

Move (B, b) = {8}

ε-closure (Move (B, b)) = {8, 9, 10, 11, 4, 5, 7}

= {4, 5, 7, 8, 9, 10, 11} ——— (D)

Scanned by TapScanner

Move (C,a) = {6, 12} —— C.

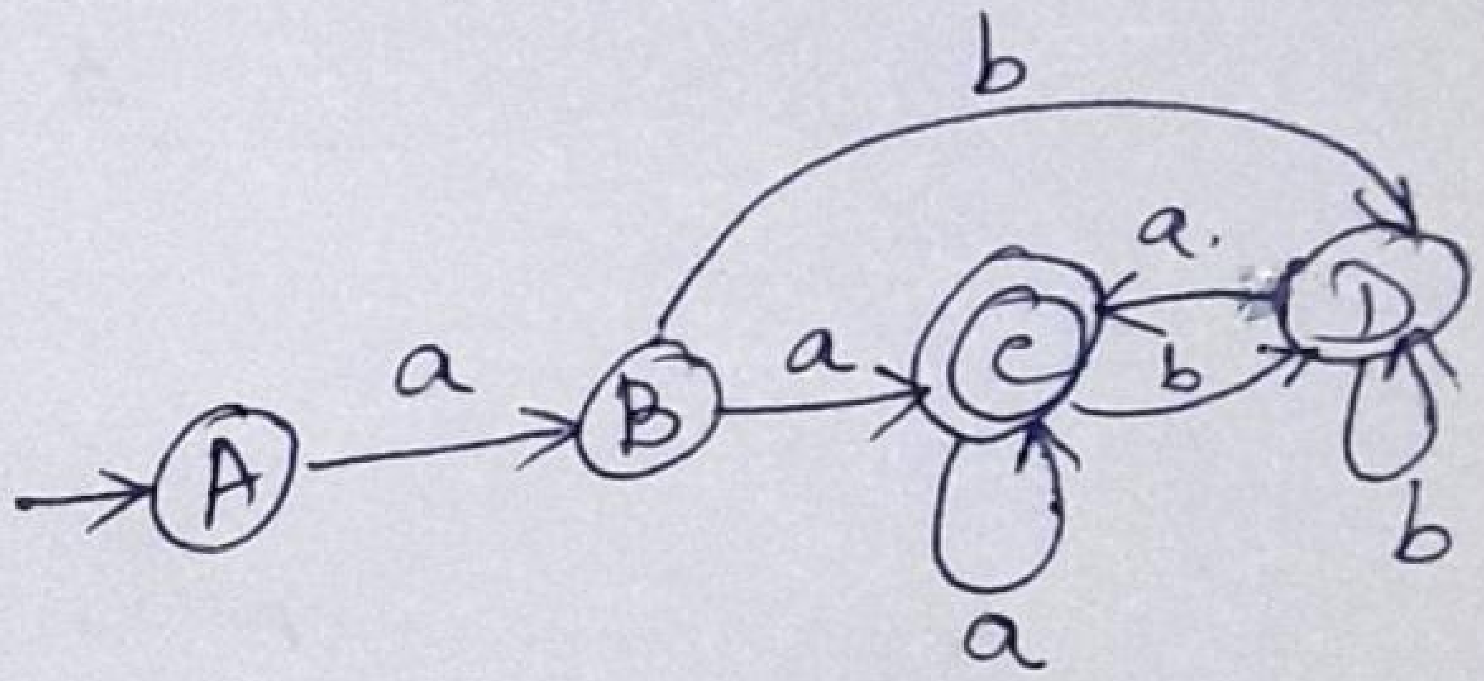
Move (C,b) = {8) —— D

Move (D,a) = {6,12} —— C

Move (D,b) = {8} —— D.

DFA transition table.

		a	b.
→	A	B	φ
	B	C	D
*	C	C	D
	D.	C	D.



③ Minimized DFA.

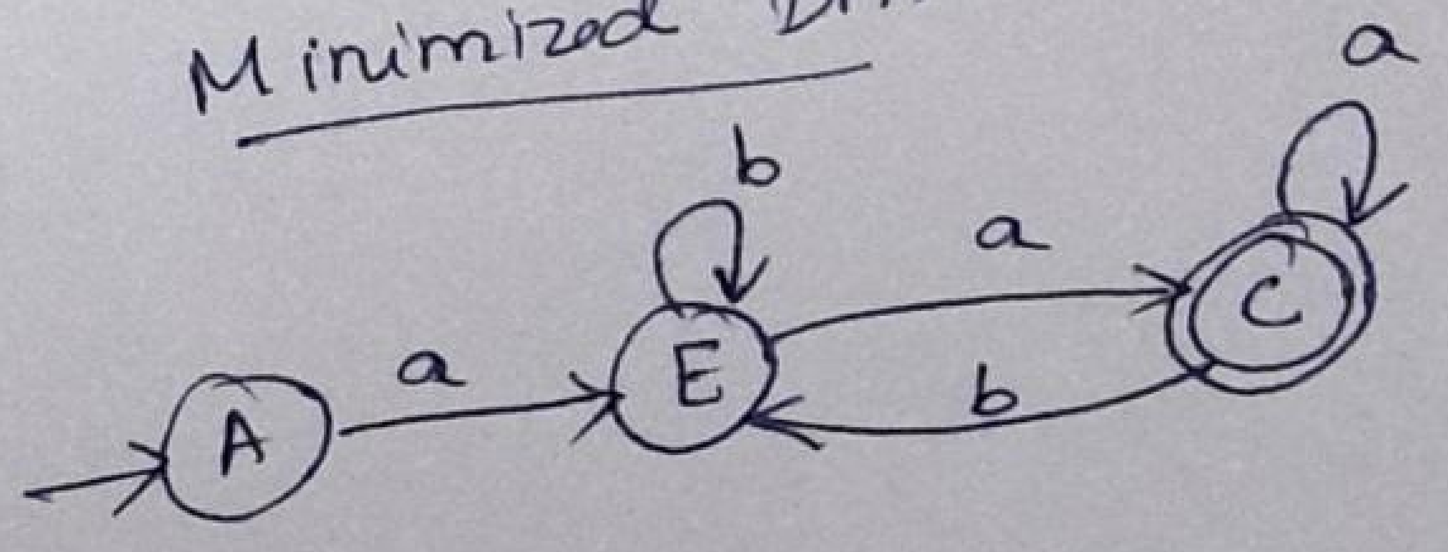
(ABD)
Non final states

(C)
final state.

	a	b.
A	B	φ
B	C	D ✓
D	C	D ✓

		a	b
→	A	E	φ
	E	C	E
*	C	C	E

Minimized DFA.

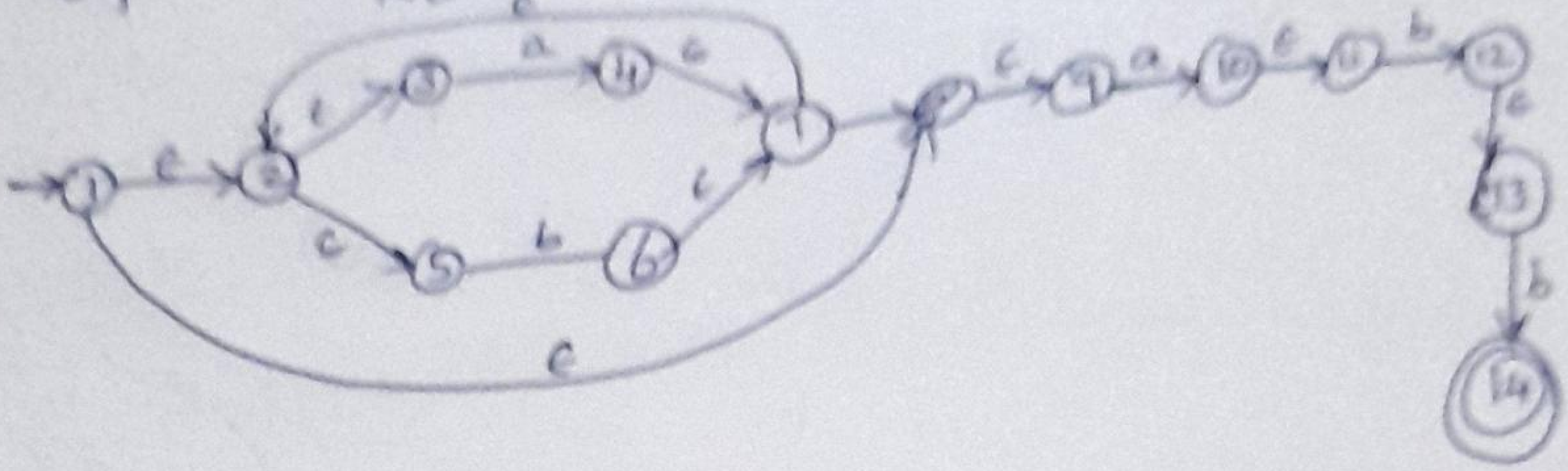


x ——————————— x

Convert the RE to minimized DFA.

$(a|b)^* abb.$

Step 1: RE to ε-NFA



Step 2: ε-NFA to DFA

ε-Closure (1) = {1, 2, 3, 5, 8, 9} ——— A

Move (A, a) = {4, 10}

ε-closure (Move (A, a)) = {4, 7, 8, 2, 3, 5, 10, 11, 9}

= {2, 3, 4, 5, 7, 8, 9, 10, 11} ——— B

Move (A, b) = {6}

ε-closure (Move (A, b)) = {6, 7, 8, 9, 2, 3, 5}

= {2, 3, 5, 6, 7, 8, 9} ——— C

Move (B, a) = {4, 10} ——— B

Move (B, b) = {6, 12}

ε-Closure (Move (B, b)) = {6, 7, 8, 9, 2, 3, 5, 12, 13}

= {2, 3, 5, 6, 7, 8, 9, 12, 13} ——— D

$Move(C, a) = \{4, 10\}$ —— B

$Move(C, b) = \{6\}$ —— C

$Move(D, a) = \{4, 10\}$ —— B

$Move(D, b) = \{6, 14\}$

$\epsilon\text{-closure}(Move(D, b)) = \{6, 7, 8, 9, 2, 3, 5, 14\}$

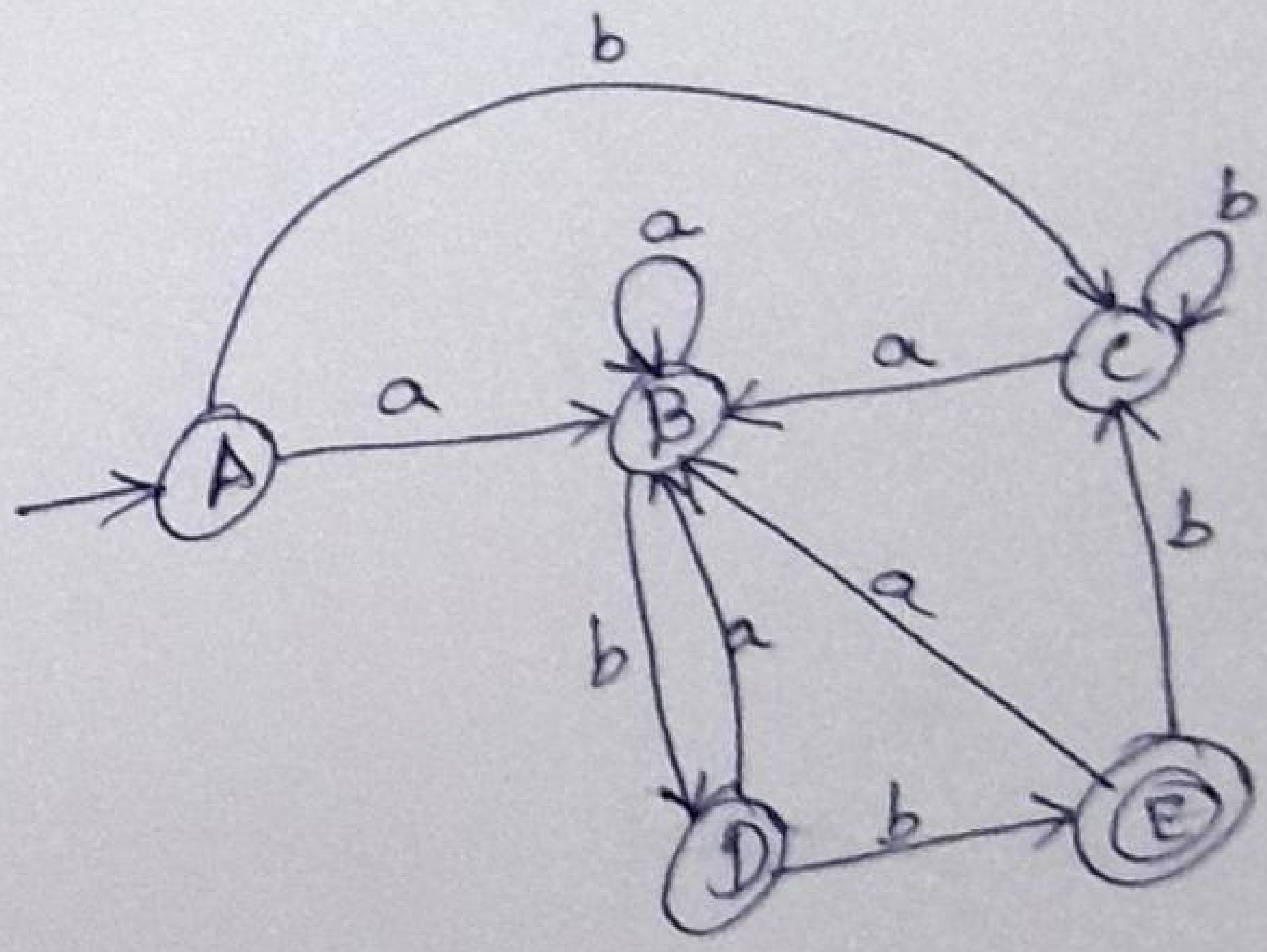
$= \{2, 3, 5, 6, 7, 8, 9, 14\}$ —— E

$Move(E, a) = \{4, 10\}$ —— B

$Move(E, b) = \{6\}$ —— C

Transition table.

	a	b.
→ A	B	C
B	B	D
C	B	C
D	B	E
* E	B	C.



Minimizing DFA. (ABCD) (E)

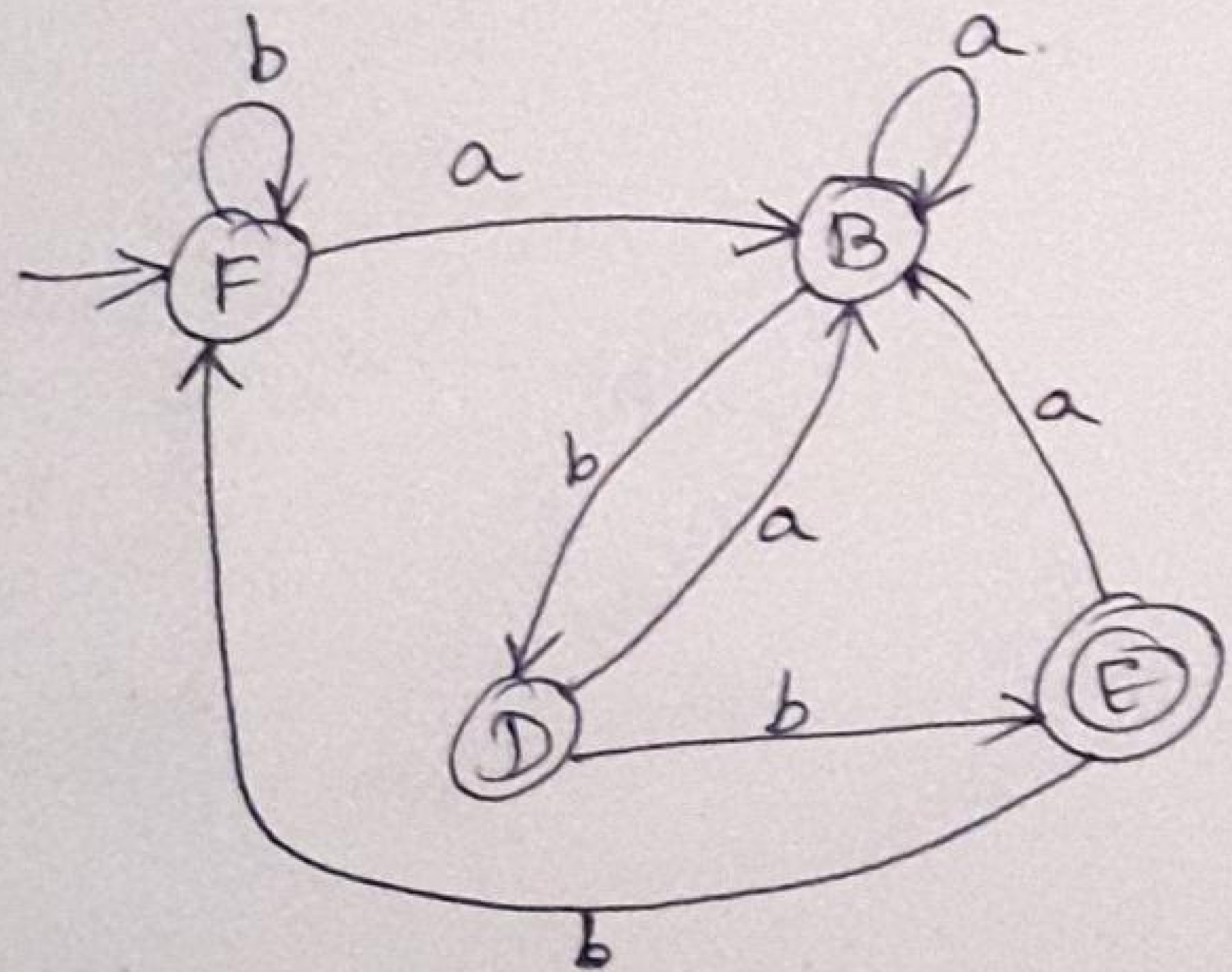
	a	b
A	B	C ✓
B	B	D
C	B	C ✓
D	B	E

((AC) BD) (E)

AC → F

Minimized DFA.

	a	b
→ F	B	F
B	B	D
D	B	E
* E	B	F



X ———————— X

1) $10 + (0+11)0^* 1$

2) $a(a/b)^* abb$

UNIT - III

CONTEXT FREE GRAMMAR AND LANGUAGES

CFG

A context free grammar (CFG) is one whose production rules are of the form

$$A \to \alpha$$

where A is any single non terminal and α is any combination of terminals and non terminals. A DFA/NFA cannot recognize strings from this type of language since we must be able to remember information somehow. Instead we use a push down automaton which is like a DFA except that stack is allowed to use.

A context free Grammar is a way of describing languages by recursive rules or substitution rules called production.

A CFG consists of quadruple (V, T, P, S).

V is a set of non terminal or variables

T is a set of terminals.

P is the set of production rules

S is the start symbol.

Q. The grammar $(\{A\}, \{a, b, c\}, P, A)$

$$P: A \to aA$$
$$A \to abc$$

Derivations using a grammar:

Derivation is a process of expanding the start symbol using one of its productions until a string is derived consisting entirely of terminals.

There are 2 types of derivations.

- leftmost derivation
- rightmost derivation.

Leftmost derivation:

In Leftmost derivation, the leftmost variable is replaced by one of its production bodies. It is indicated by using the relations $\underset{lm}{\Rightarrow}$ and $\overset{*}{\underset{lm}{\Rightarrow}}$, for one or many steps respectively.

Rightmost Derivation:

In rightmost derivation, the rightmost variable is replaced by one of its production bodies. It is indicated by using the relations $\underset{rm}{\Rightarrow}$ and $\overset{*}{\underset{rm}{\Rightarrow}}$. for one or many steps respectively.

Problem:

Consider G whose productions are $S \to aAS / a$
$A \to SbA / SS / ba$. For the string $\omega = aabbaa$
find i) Leftmost derivation ii) Right most derivation.

i) LMD

$S \underset{lm}{\Rightarrow} a\underline{A}S$

$\underset{lm}{\Rightarrow} a\underline{S}bAS$

$\underset{lm}{\Rightarrow} aab\underline{A}S$

$\underset{lm}{\Rightarrow} aabba\underline{S}$

$\underset{lm}{\Rightarrow} aabbaa$

ii) RMD.

$S \underset{rm}{\Rightarrow} aA\underline{S}$

$\underset{rm}{\Rightarrow} a\underline{A}a$

$\underset{rm}{\Rightarrow} aSb\underline{A}a$

$\underset{rm}{\Rightarrow} aSbbaa$

$\underset{rm}{\Rightarrow} aabbaa$

x ———————— x

The language of a Grammar.

If G(V, T, P, S) is a CFG, the language of G, denoted L(G), is the set of terminal strings that have derivations from the start symbol. That is,

$$L(G) = \{ \omega \text{ in } T^* / S \overset{*}{\underset{G}{\Rightarrow}} \omega \}$$

Pbm.

1. Find the language L(G) for the following grammar

$S \to aCa$
$C \to aCa / b$

Ans

$S \to aCa$

$\to aba.$

$S \to aCa.$

$\to aaCaa$

$\to aaaCaaa.$

$\to aaabaaa.$

$$L(G) = \{a^n b a^n / n \geq 1\}$$

2. Find the $L(G)$ for the following grammar.

$S \to 0S1$

$S \to \epsilon$

$S \to 0S1$

$\to 00S11$

$\to 000S111$

$\to 000111$

$S \to 0S1$

$\to 01$

$$L(G) = \{0^n 1^n / n \geq 0\}$$

3. Find the Language $L(G)$ for the grammar

$S \to aSb$

$S \to ab$

$S \to aSb$

$\to aabb$

$S \to ab.$

$$L(G) = \{a^n b^n / n \geq 1\}$$

Scanned by TapScanner

4. Find $L(G)$ for the grammar.

$$S \to aB$$
$$B \to b$$
$$B \to bA$$
$$A \to aB$$

Sln:

$S \to aB$
$\to ab$

$S \to aB$
$\to abA$
$\to abaB$
$\to abab$

$S \to aB$
$\to abA$
$\to abaB$
$\to ababA$
$\to ababaB$
$\to ababab$

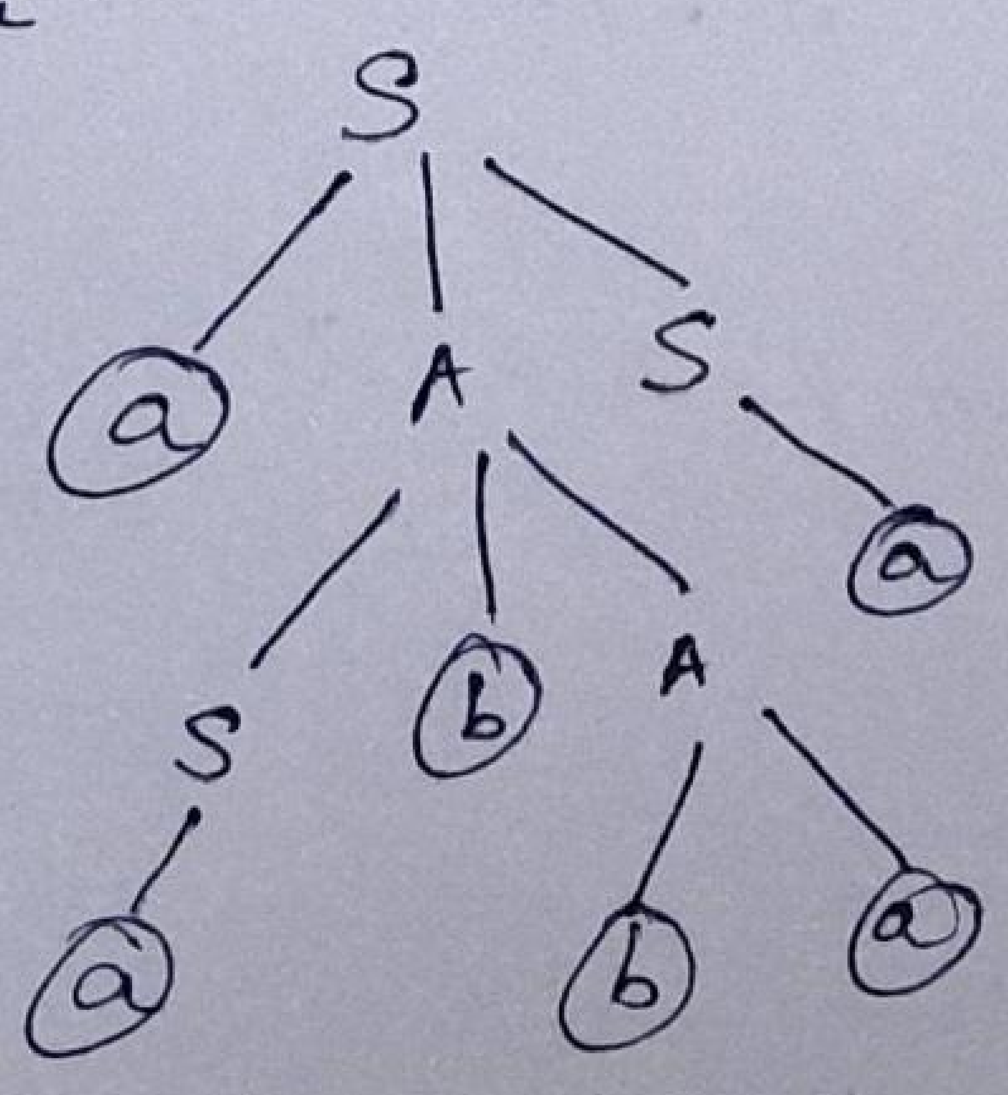
$$L(G) = \{(ab)^n \mid n \geq 1\}$$

Parse Trees:

Parse tree is a tree representation of derivations.

Pbm:

Consider G whose productions are $S \to aAS \mid a$, $A \to SbA \mid SS \mid ba$. For the string $w = aabbaa$, Construct a parse tree.



Derivation to trees : (from inferences to trees)

Theorem.

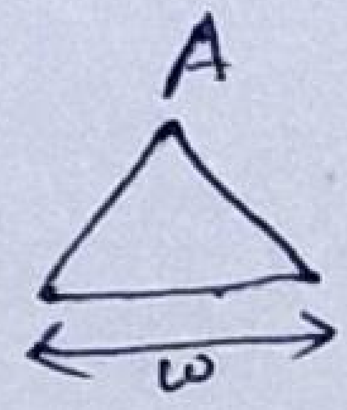
Let $G = (V, T, P, S)$ be a CFG, if the recursive inference procedure tell us that terminal string w is in language of variable A, then there is a parse tree with root A and yeild w.

Proof

Basis:

Here there must be a production $A \to w$

The desired parse tree is then



Induction:

w is inferred in $n+1$ steps

Suppose the last step was based on the production

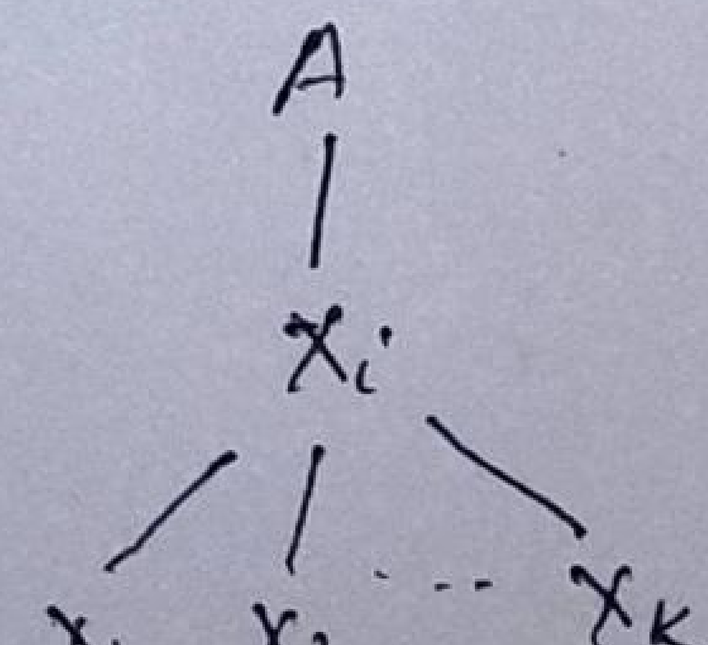
$$A \to X_1 X_2 \ldots X_K.$$

where X_i may be terminal or non terminal

Now we can break the string w as $w_1 w_2 w_3 \ldots w_k$ and two possible cases are,

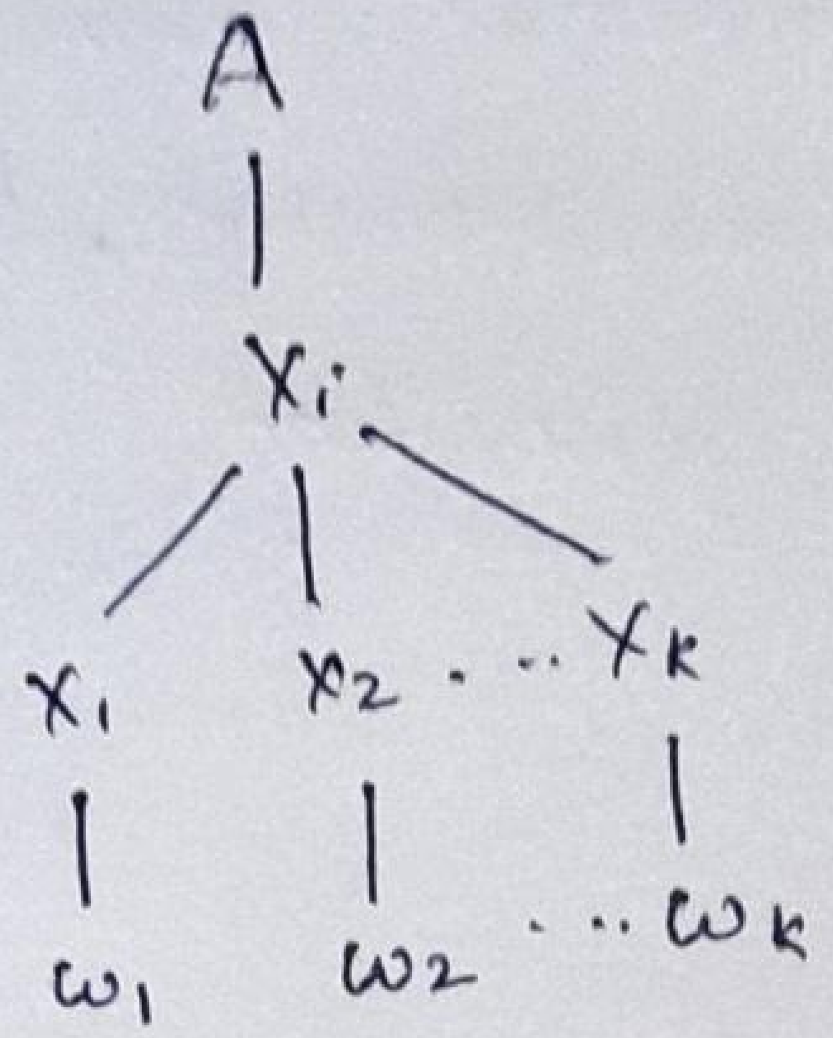
Case 1:

If $w_i = X_i$, then X_i is a terminal



Case 2:

If X_i is non terminal.



$w = w_1 w_2 w_3 \ldots w_k$

So if there is a recursive inference that yields the string w, then there exist a parse tree to yield w.

From parse tree to Derivation:

Theorem:

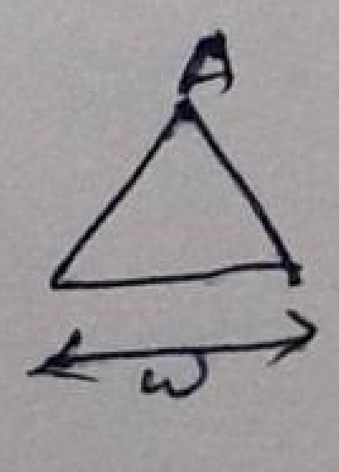
Let $G = (V, T, P, S)$ be a CFG and suppose there is a parse tree with root labeled by variable A and with yield w, where w is in T^*. Then there is a left most derivation $A \overset{*}{\underset{lm}{\Rightarrow}} w$ in grammar G.

Proof:

Basis:

Height is 1

The tree must look like



Consequently $A \to w \in P$, and $A \underset{lm}{\Rightarrow} w$

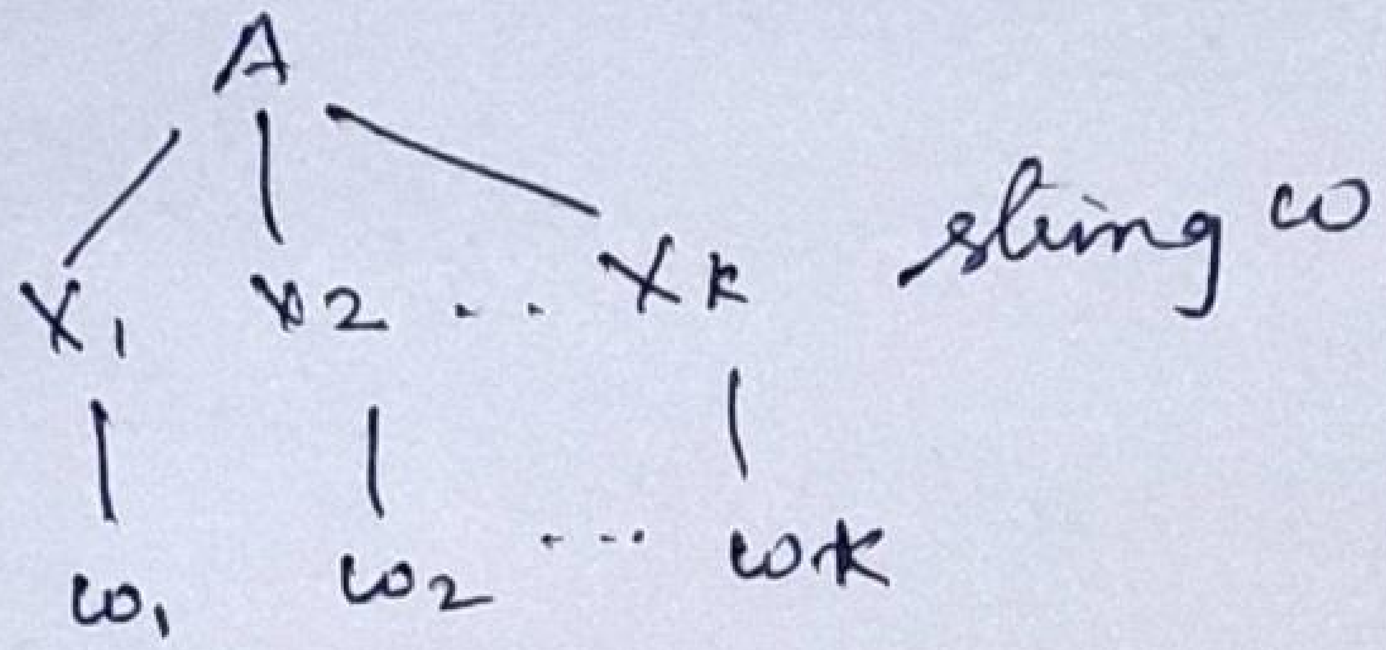
Induction:

If the height of the tree is n where $n > 1$

Case 1: If X_i is a terminal, then $X_i = w_i$

Case 2: If X_i is a ~~..~~ non terminal, then there will be left most derivation

$$X_i \underset{lm}{\overset{*}{\Rightarrow}} w_i$$



$$A \to X_1 X_2 \ldots X_k$$

LMD

$$A \underset{lm}{\overset{*}{\Rightarrow}} x_1 x_2 \ldots x_k$$

Then for each $i = 1, 2 \ldots k$, in order

$$A \underset{lm}{\overset{*}{\Rightarrow}} x_1 x_2 \ldots x_i \, x_{i+1} \ldots x_k$$

$$x_1 = w_1 w_2 \ldots w_k$$

then LMD of the form

$$A \underset{lm}{\overset{*}{\Rightarrow}} w_1 w_2 \ldots w_i \, X_{i+1} X_{i+2} \ldots X_k$$

1. If X_i is a terminal, then we can derive the string w straightly using LMD

$$A \underset{lm}{\overset{*}{\Rightarrow}} w_1 w_2 \ldots w_i \, X_{i+1} X_{i+2} \ldots X_k$$

2. If X_i is non terminal, we continue to derive the string w_i from X_i using LMD.

$$X_i \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \cdots \underset{lm}{\Rightarrow} w_i$$

We proceed with,

$$w_1 w_2 \ldots w_{i-1} X_i X_{i+1} \ldots X_k \underset{lm}{\Rightarrow}$$

$$w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} \ldots X_k \underset{lm}{\Rightarrow}$$

$$w_1 w_2 \ldots w_{i-1} \alpha_2 X_{i+1} \ldots X_k \underset{lm}{\Rightarrow}$$

$$w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$$

The result is a derivation $A \overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 \ldots w_i X_{i+1} \ldots X_k$.

When $i = k$, the result is a leftmost derivation of w from A.

* ———— *

Ambiguity in Grammars & Languages:

If a grammar has two distinct parse trees then that grammar is known as ambiguous grammar.

Scanned by TapScanner

Show that the grammar $E \rightarrow E+E \mid E*E \mid a \mid b$ is ambiguous. (take i/p as $a+a*b$)

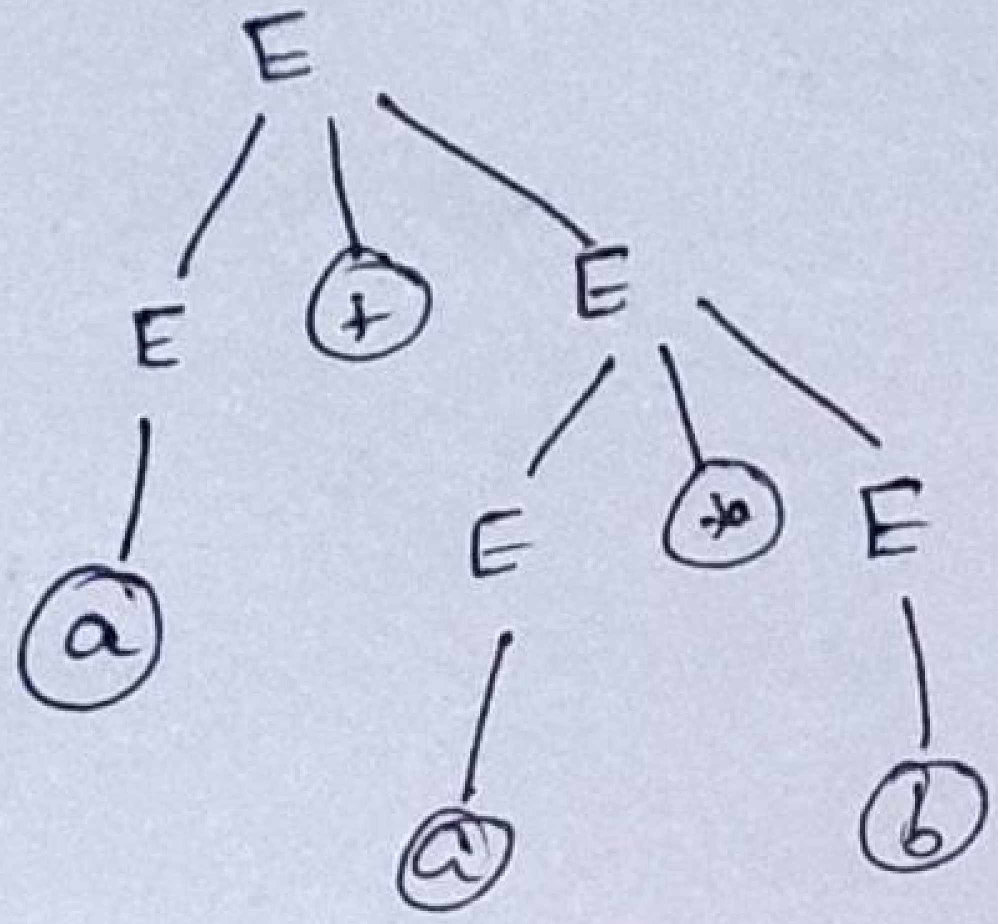
Sol.

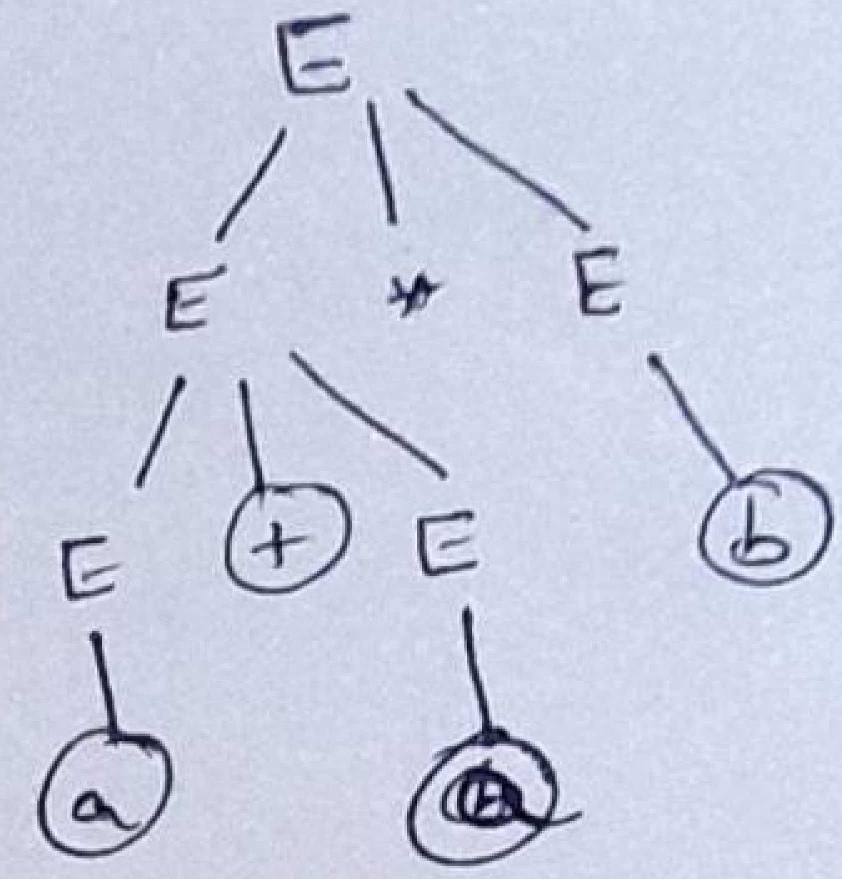
$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow a$

$E \rightarrow b.$





Here two parse trees are generated, hence the given grammar is ambiguous.

Show that the grammar $S \rightarrow aSbS \mid bSaS \mid \epsilon$

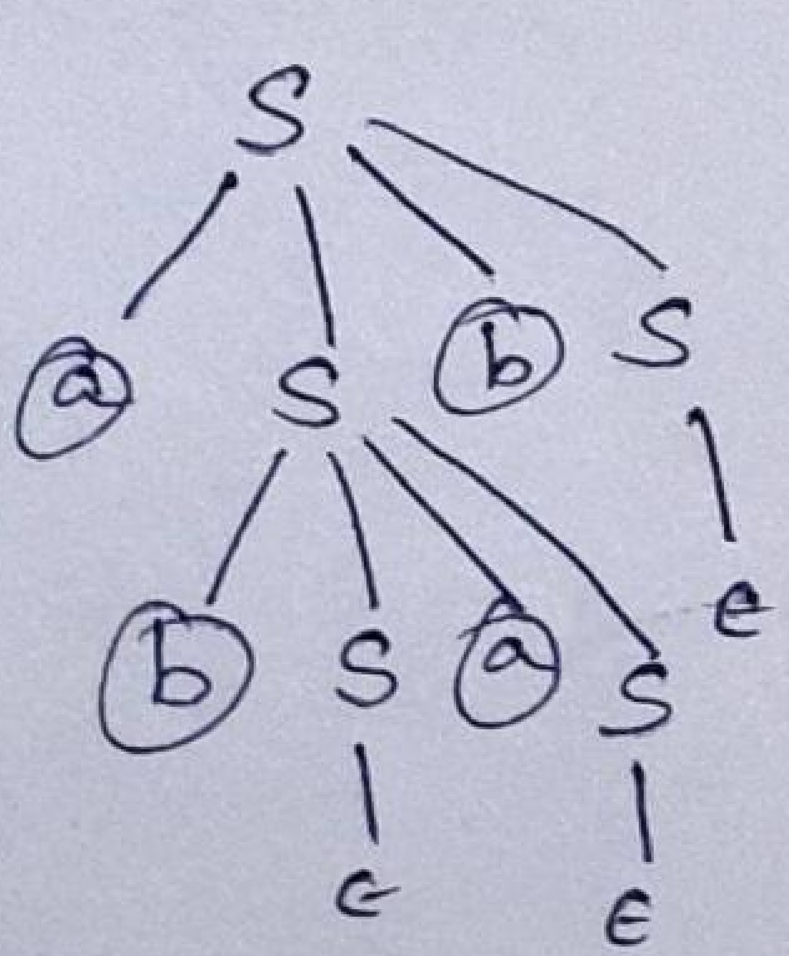
Take i/p as abab.

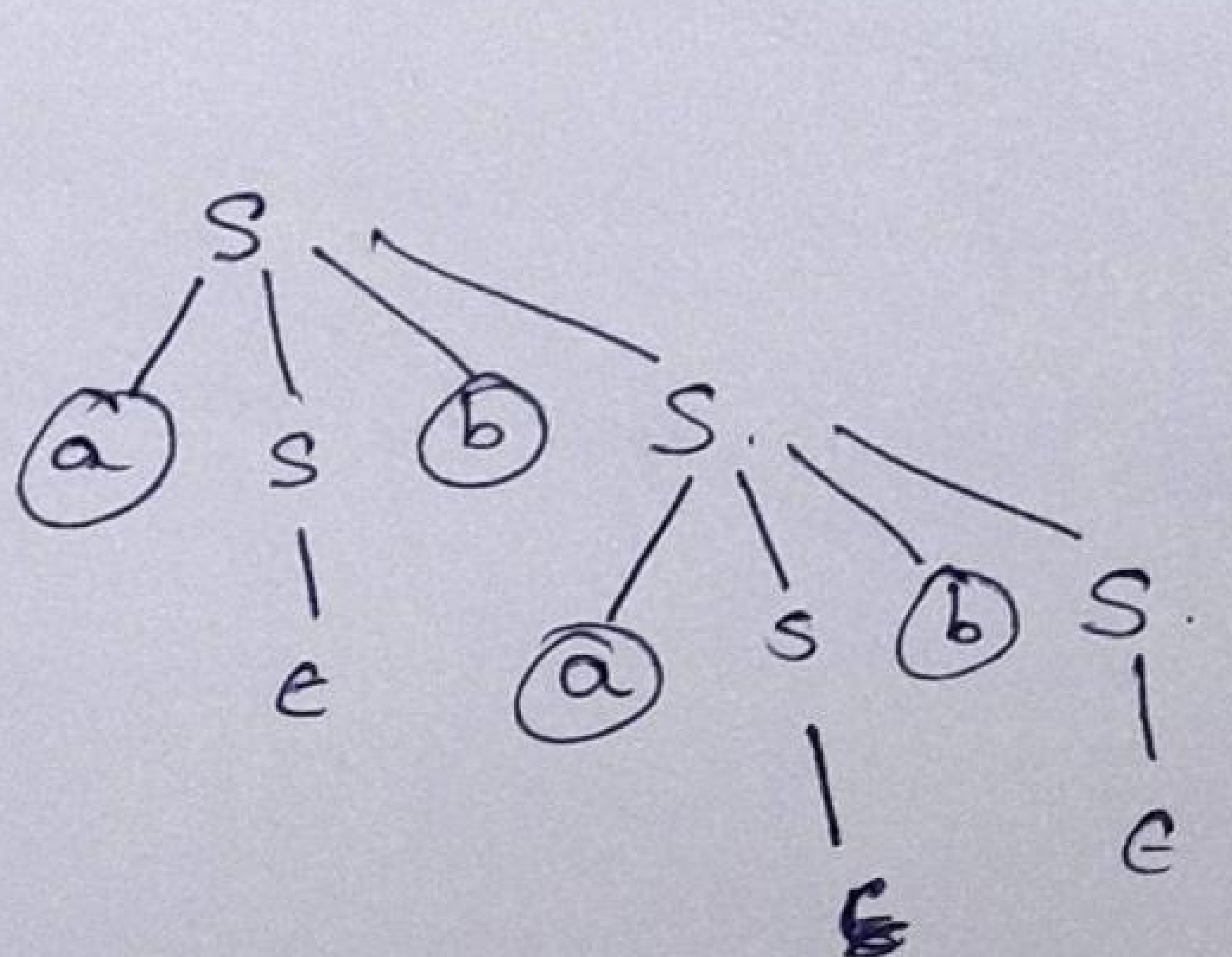
Sol.

$S \rightarrow aSbS$

$S \rightarrow bSaS$

$S \rightarrow \epsilon.$





Here two parse trees are generated. Hence the given grammar is ambiguous.

Removing ambiguity from Grammar.

The 2 causes of ambiguity in the grammar are,

i) The precedence of operators is not respected

ii) A sequence of identical operators can group either from the left or from the right.

Sn. → Introduce several different variables to the expressions that share a level of 'binding strength'.

Specifically:

1. A factor is an expression that cannot be broken
   a) Identifier
   b) Paranthesized expression.
2. A term is an expression that cannot be broken by the + operator.
3. An expression are those that can be broken.

Pbm

Convert the following grammar into unambiguous grammar.

$E \to E+E \mid E*E \mid (E) \mid I$

$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Sn.

$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$F \to I \mid (E)$

$T \to F \mid T*F$

$E \to T \mid E+T$

Scanned by TapScanner

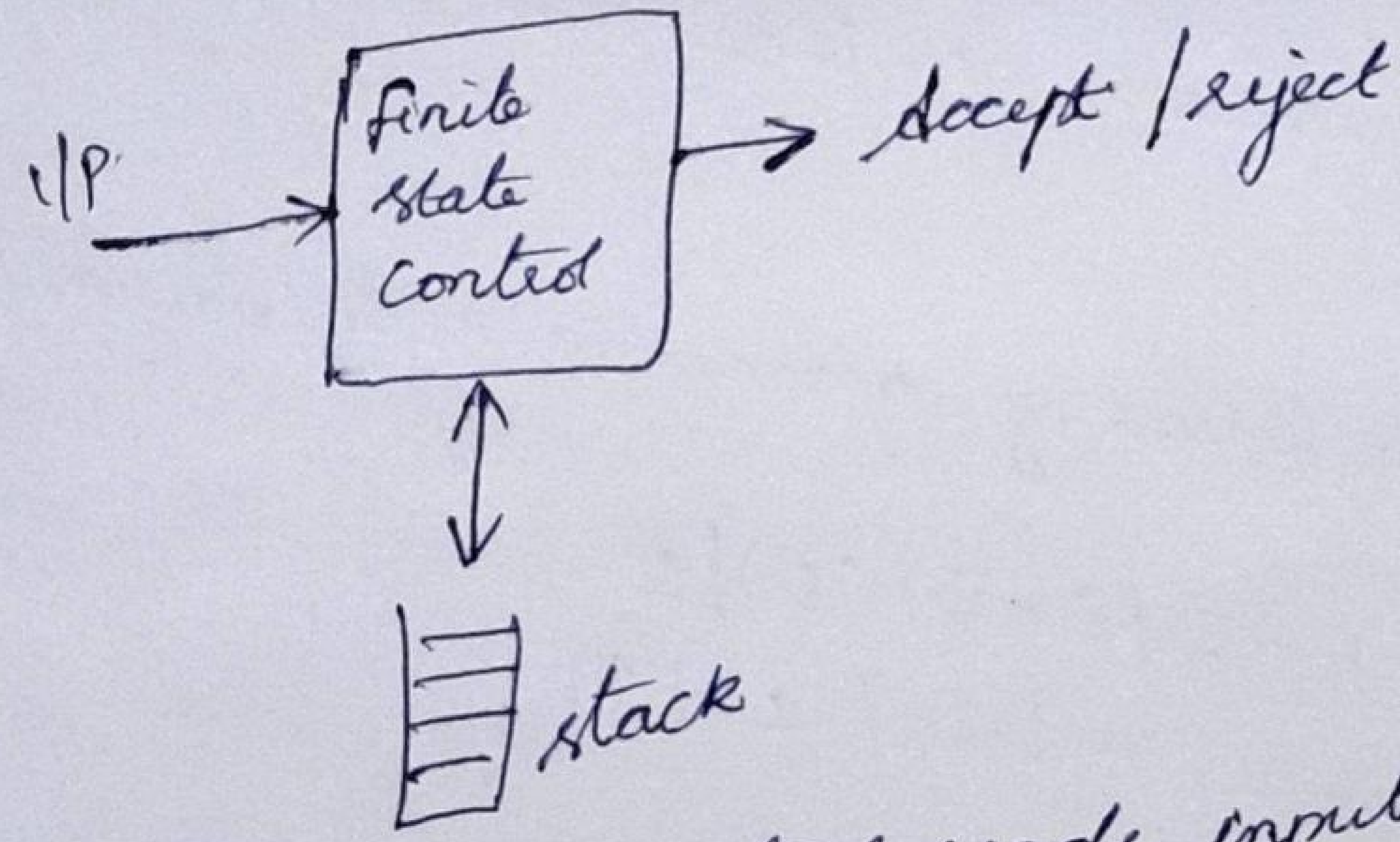
Push Down Automata:

Push down automata is an extension of the non deterministic finite automation with ϵ-transitions.

The push down automata is essentially an ϵ-NFA with the addition of a stack.

~~Determination~~ Definition of Push Down Automata:

The push down automaton is in essence a non deterministic finite automaton with ϵ-transitions permitted and one additional capability: a stack on which it can store a string of "stack symbols".



A finite state control reads inputs, one symbol at a time.

A push down automaton (PDA) involves seven components

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

Q : A finite set of states

Σ : A finite set of I/P symbols.

Γ : A finite stack alphabet.

δ : The transition function

q_0 : the start state

Z_0 : The start symbol

F : The set of accepting states, or final states.

Pbm.

Construct a PDA for the language $L = \{w \in \{a,b\}^* \; n_a(w) = n_b(w)\}$

Sln

$M = (\{q_0, q_1\}, \{a,b\}, \{Z_0, a, b\}, \delta, q_0, Z_0, q_1)$

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

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

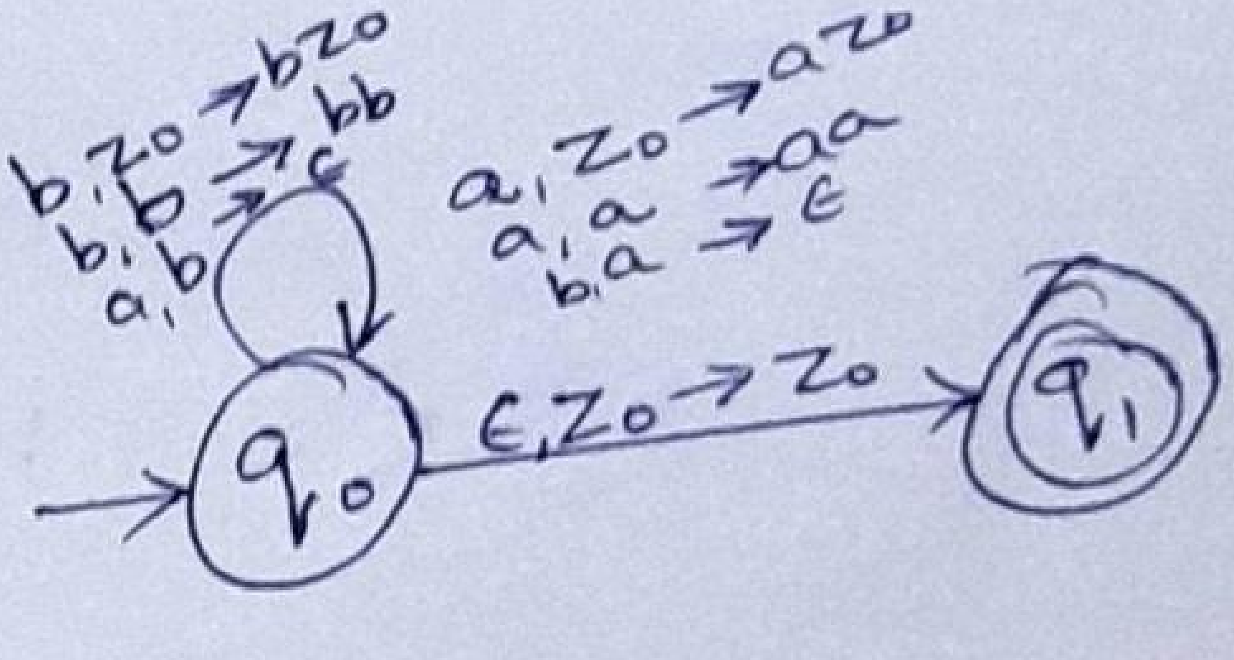
$\delta(q_0, b, a) = (q_0, \epsilon)$

$\delta(q_0, \epsilon, Z_0) = (q_1, Z_0)$

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

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

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



Input $w = baab$

$(q_0, baab, Z_0) \vdash (q_0, aab, bZ_0)$

$\vdash (q_0, ab, Z_0)$

$\vdash (q_0, b, aZ_0)$

$\vdash (q_0, \epsilon, Z_0)$

$\vdash (q_1, Z_0)$

2. Construct a PDA for the language $L = \{0^n 1^n \mid n \geq 1\}$

$0^n 1^n \Rightarrow 0011$

Sln: $M = (\{q_0, q_1, q_2\}, \{0,1\}, \{Z_0, 0, 1\}, \delta, q_0, Z_0, q_2)$

$\delta(q_0, 0, Z_0) = (q_1, 0Z_0)$

$\delta(q_1, 0, 0) = (q_1, 00)$

$\delta(q_1, 1, 0) = (q_2, \epsilon)$

$\delta(q_2, 1, 0) = (q_2, \epsilon)$

$\delta(q_2, \epsilon, Z_0) = (q_3, Z_0)$

Input $w = 0011$

$(q_0, 0011, Z_0) \vdash (q_1, 011, 0Z_0)$

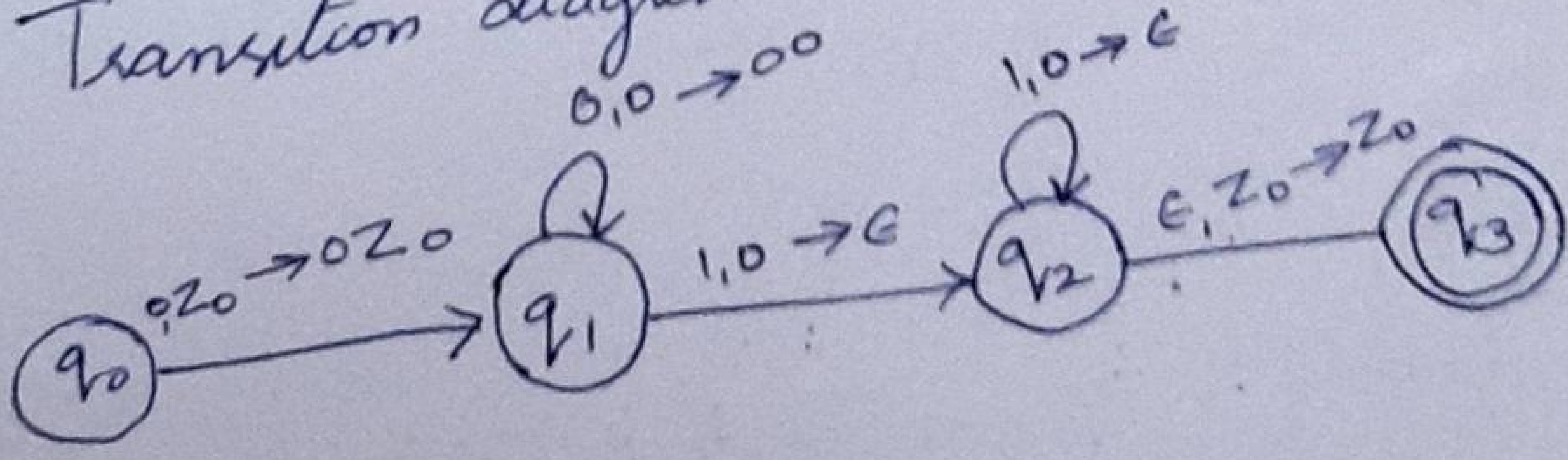
$\vdash (q_1, 11, 00Z_0)$

$\vdash (q_2, 1, 0Z_0)$

$\vdash (q_2, \epsilon, Z_0)$

$\vdash (q_3, Z_0)$

Transition diagram



x ———— x

Scanned by TapScanner

Languages Accepted by a PDA.

1. Acceptance by final state:

PDA accepts its input by consuming it and entering an accepting state.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then $L(P)$, ~~the~~ the language accepted by P by final state is,

$$\{w \mid (q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \alpha)\}$$

for some state q in F and any stack string α.

1. Construct a NPDA for accepting the language by final state $L = \{ww^R \mid w \in \{a, b\}^*\}$ [even length palindrome]

$L = \{aab\ baa,\ abbb\ bbba, \ldots\}$

Ans $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{Z_0, a, b\}, q_0, Z_0, q_2)$

$$\left.\begin{aligned} \delta(q_0, a, Z_0) &= (q_0, aZ_0) \\ \delta(q_0, b, Z_0) &= (q_0, bZ_0) \end{aligned}\right\}$$

$$\left.\begin{aligned} \delta(q_0, a, a) &= (q_0, aa) \\ \delta(q_0, a, b) &= (q_0, ab) \\ \delta(q_0, b, a) &= (q_0, ba) \\ \delta(q_0, b, b) &= (q_0, bb) \end{aligned}\right\} \text{Stay in same state } q_0 \text{ and push the symbol}$$

$\delta(q_0, \epsilon, Z_0) = (q_1, Z_0)$
$\delta(q_0, \epsilon, a) = (q_1, a)$
$\delta(q_0, \epsilon, b) = (q_1, b)$

} moves from q_0 to q_1 with i/p ϵ. i.e. middle position

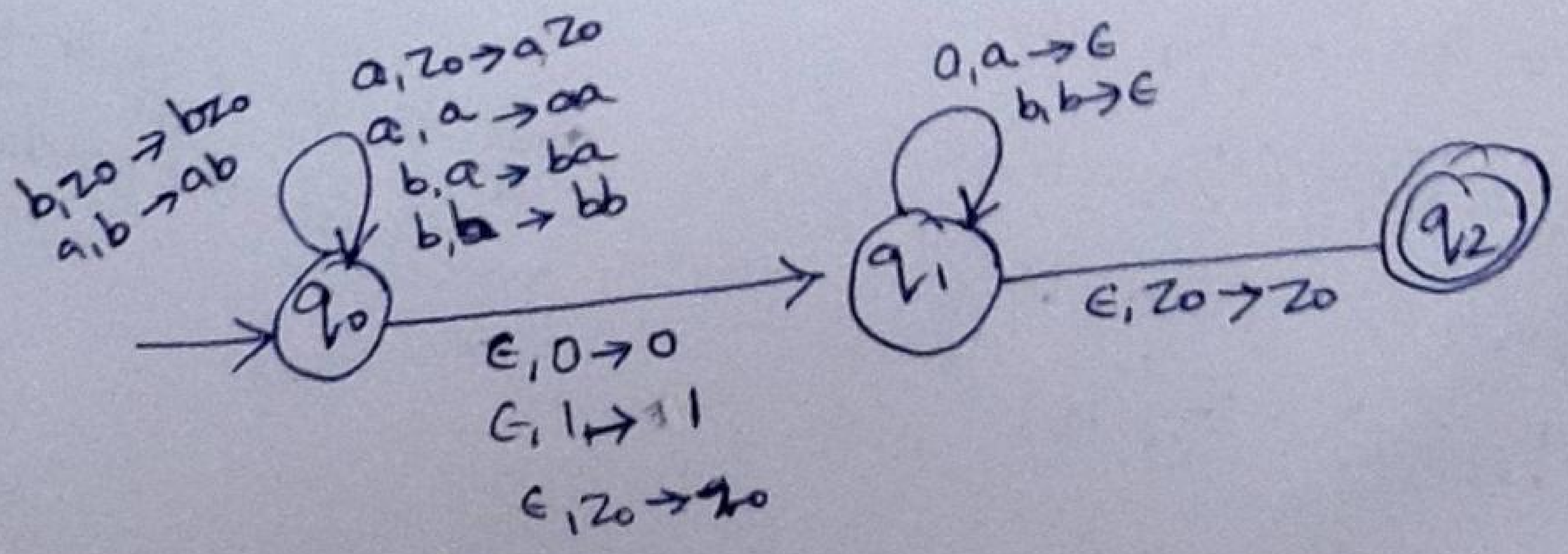
$\delta(q_1, a, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$

} pop the matched symbol.

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$ } Reached bottom of the stack moved to next state i.e. final state

Input. abbb bbba

$$
\begin{aligned}
(q_0, abbbbbba, Z_0) &\vdash (q_0, bbb\,bbba, aZ_0) \\
&\vdash (q_0, bb\,bbba, baZ_0) \\
&\vdash (q_0, b\,bbba, bbaZ_0) \\
&\vdash (q_0, bbba, bbbaZ_0) \\
&\vdash (q_1, bbba, bbbaZ_0) \\
&\vdash (q_1, bba, bbaZ_0) \\
&\vdash (q_1, ba, ba, Z_0) \\
&\vdash (q_1, a, aZ_0) \\
&\vdash (q_1, \epsilon, Z_0) \\
&\vdash (q_2, Z_0)
\end{aligned}
$$



2. Acceptance by Empty stack:

A PDA accepts its input and the set of strings that causes the PDA to empty its stack.

For each PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, $N(P)$ is the language accepted by P by empty stack. It is defined as.

$$N(P) = \{w \mid (q_0, w, Z_0) \vdash^{*} (q, \epsilon, \epsilon)\} \text{ for any state } q.$$

pbm.

Construct a PDA for accepting the language by empty stack.

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

sln.

$$L = \{\underset{mn=1}{abc}, \underset{m=1, n=2}{aabcc}, \underset{m=n=2}{aabbcc}, \ldots\}$$

no. of a's == no. of c's.

I/P.
a – Push(a)
b – unchange.
c. – pop(a)

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

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

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

$\delta(q_1, c, a) = (q_2, \epsilon)$

$\delta(q_2, c, a) = (q_2, \epsilon)$

$\delta(q_2, \epsilon, Z_0) = (q_3, Z_0)$

$P = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$

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

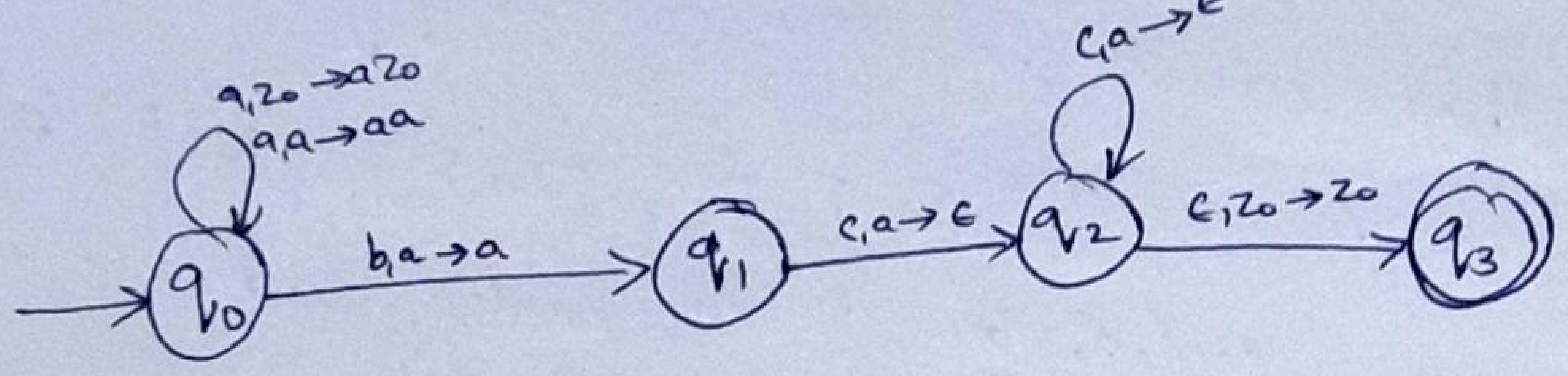
$\Sigma = \{a, b, c\}$

$\Gamma = \{a, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_3\}$



Input: aabcc.

$(q_0, aabcc, Z_0) \vdash (q_0, abcc, aZ_0)$

$\vdash (q_0, bcc, aaZ_0)$

$\vdash (q_1, cc, aaZ_0)$

$\vdash (q_2, c, aZ_0)$

$\vdash (q_2, \epsilon, Z_0)$

$\vdash (q_3, \epsilon)$

x ——————— x

From Empty stack to Final stack.

Theorem:

If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$ then there is a PDA P_F such that $L = L(P_F)$.

To prove:

If there exist a PDA P_N that accepts a language by empty stack then there exists PDA P_F that accepts a language by reaching final state.

Proof:

To prove this theorem, we use a new symbol x_0 which must not be a symbol in Γ i.e. $(x_0 \in \Gamma^*)$

- Here x_0 is used as the starting top symbol of the stack.
- And x_0 is the symbol marked on the bottom of the stack P_N.
- P_N goes on processing the input.
- If P_N sees x_0, then it finishes processing the string.
- Now construct P_F with a new starting state P_0 and final state P_F.

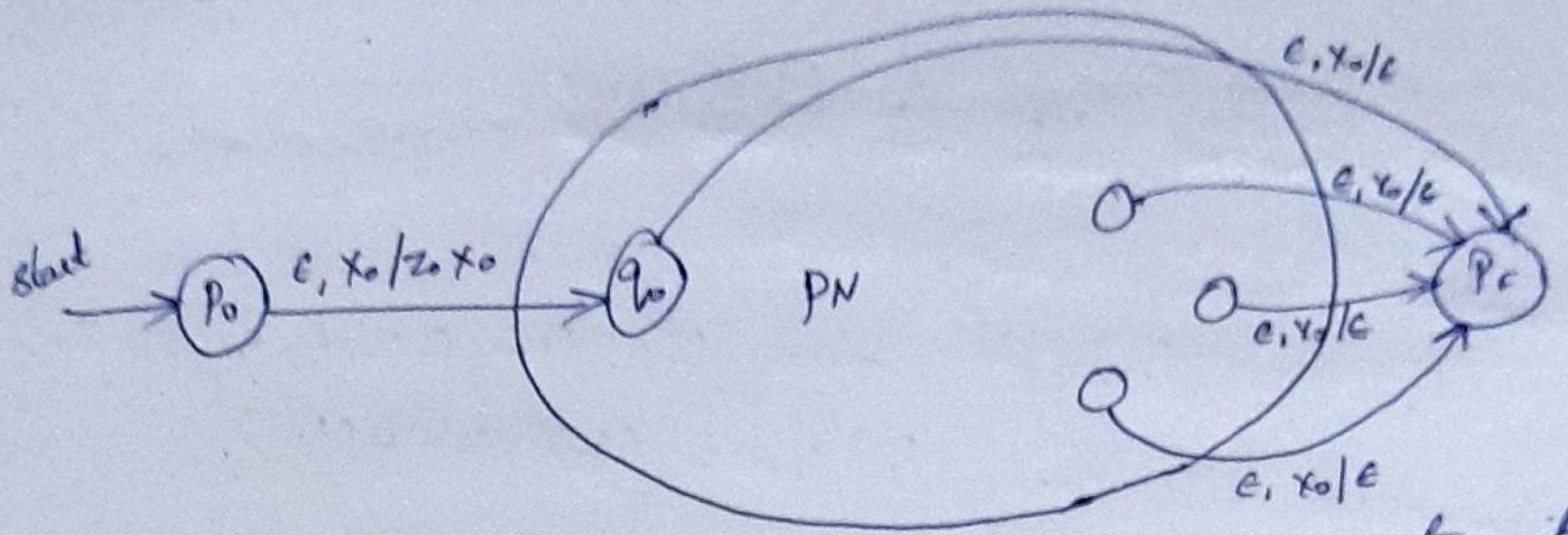


fig:- P_F simulates P_N & accepts if P_N empties its stack.

$P_F = (Q \cup (P_0, P_F), \Sigma, \Gamma \cup \{X_0\}, \delta_F, P_0 X_0, \{P_F\})$

where δ_F is defined as,

1. $\delta_F(P_0, \epsilon, X_0) = (q_0, Z_0, X_0) \Rightarrow P_F$
2. For all states q in Q, inputs a in Σ or $a = \epsilon$, and stack symbols Y in Γ, $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$
3. In addition to rule ②, $\delta_F(q, \epsilon, X_0)$ contains (P_F, ϵ) for every state q in Q.

$$(P_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \vdash^{*}_{P_F} (q, \epsilon, X_0) \vdash_{P_F} (P_F, \epsilon, \epsilon)$$

Thus the PDA P_F accepts the final state.

From Final state to Empty stack

Theorem:

Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$

Then there is a PDA P_N such that $L = N(P_N)$

Proof:

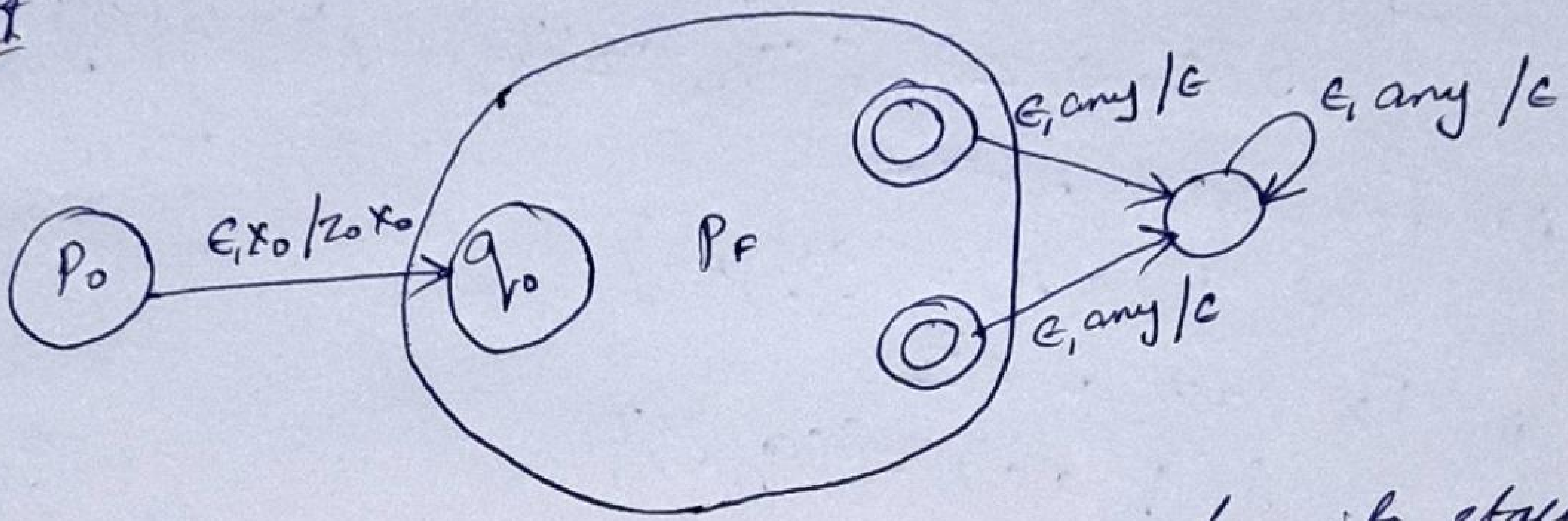


Fig:-

P_N simulates P_F and empties its stack when and only when P_N enters an accepting state.

Let $P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$

where δ_N is defined by,

1. $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$
2. For all states q in Q, input symbols 'a' in Σ or $a = \epsilon$, and Y in Γ, $\delta_N(q, a, Y)$ contains every pair that is in $\delta_F(q, a, Y)$. i.e., P_N simulates P_F.
3. For all accepting states q in F and stack symbol Y in Γ or $Y = X_0$, $\delta_N(q, \epsilon, Y)$ contains (p, ϵ)
4. For all stack symbols Y in Γ or $Y = X_0$, $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$

$$(p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0X_0) \vdash^{*}_{P_N} (q, \epsilon, \alpha X_0)$$
$$\vdash^{*}_{P_N} (p, \epsilon, \epsilon)$$

Thus PDA P_N accepts empty stack.

Equivalence of Pushdown Automata and CFG.

1) From Grammars to Pushdown Automata:

Convert CFG to Greiback Normal Form (GNF) [$A \to a\alpha$, $\to a$, A]
if needed.

i) To all transition function first include

$\delta(q_0, \epsilon, Z) = \{(q_1, SZ)\}$. (S - starting symbol)

ii) For each $A \to \alpha$

$\delta(q_1, \epsilon, A) = \{(q_1, \alpha)\}$

iii) For each $a \in \Sigma$

$\delta(q, a, a) = \{(q, \epsilon)\}$

iv) At end

$\delta(q, \epsilon, Z) = \{(q, \epsilon)\}$

Problem.

1. Construct the PDA for the grammar

$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$E \to I \mid E * E \mid E + E \mid (E)$

Sol.

$I \to a$ —— 1

$I \to b$ —— 2

$I \to Ia$ —— 3

$I \to Ib$ —— 4

$I \to I0$ —— 5

$I \to I1$ —— 6

$E \to I$ —— 7

$E \to E * E$ —— 8

$E \to E + E$ —— 9

$E \to (E)$ —— 10

12

$\delta(q_0, \epsilon, Z) = \{(q_1, EZ)\}$

Converting ① we get.

$\delta(q_1, a, a) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, I) = \{(q_1, a)\}$

Converting ② we get

$\delta(q_1, b, b) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, I) = \{(q_1, b)\}$

Converting ③, ④, ⑤, ⑥ we get

$\delta(q_1, \epsilon, I) = \{(q_1, Ia)\}$

$\delta(q_1, \epsilon, I) = \{(q_1, Ib)\}$

$\delta(q_1, \epsilon, I) = \{(q_1, I0)\}$

$\delta(q_1, \epsilon, I) = \{(q_1, I1)\}$

Converting ⑦, ⑧, ⑨, ⑩ we get.

$\delta(q_1, \epsilon, E) = \{(q_1, I)\}$

$\delta(q_1, \epsilon, E) = \{(q_1, E*E)\}$

$\delta(q_1, \epsilon, E) = \{(q_1, E+E)\}$

$\delta(q_1, \epsilon, E) = \{(q_1, (E))\}$

At last.

$\delta(q_1, \epsilon, Z) = \{(q_2, \epsilon)\}$

For other terminals

$\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$

$\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$

$\delta(q_1, (, () = \{(q_1, \epsilon)\}$

$\delta(q_1,),)) = \{(q_1, \epsilon)\}$

$\delta(q_1, +, +) = \{(q_1, \epsilon)\}$

$\delta(q_1, *, *) = \{(q_1, \epsilon)\}$

Take i/p $a*a+b$.

$\delta(q_0, a*a+b, z) \vdash (q_1, a*a+b, Ez)$

$\vdash (q_1, a*a+b, E*Ez)$

$\vdash (q_1, a*a+b, I*Ez)$

$\vdash (q_1, a*a+b, a*Ez)$

$\vdash (q_1, *a+b, *Ez)$

$\vdash (q_1, a+b, Ez)$

$\vdash (q_1, a+b, E+Ez)$

$\vdash (q_1, a+b, I+Ez)$

$\vdash (q_1, a+b, a+Ez)$

$\vdash (q_1, +b, +Ez)$

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

$\vdash (q_1, b, Iz)$

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

$\vdash (q_1, \epsilon, z)$

$\vdash (q_2, \epsilon)$.

2. Construct the PDA for the CFG.

$G = \{ \{S\}, \{a, b\}, S, P\}$ where $S \to aSbb \mid a$.

Sh.

Convert CFG into GNF

$S \to aSbb$

$S \to a$

$S \to aSbb \Rightarrow S \to aSA$

$A \to bb$

$A \to bb \Rightarrow A \to bB$

$B \to b$

Now we have,

$S \to aSA$ ——— 1

$A \to bB$ ——— 2

$B \to b$ ——— 3

$S \to a$ ——— 4

$\delta(q_0, \epsilon, Z) = \{(q_1, SZ)\}$

Converting 1 we get.

$\delta(q_1, \epsilon, S) = \{(q_1, aSA)\}$

2 $\delta(q_1, \epsilon, A) = \{(q_1, bB)\}$

3 $\delta(q_1, b, b) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, B) = \{(q_1, b)\}$

4 $\delta(q_1, a, a) = \{(q_1, \epsilon)\}$

$\delta(q_1, \epsilon, S) = \{(q_1, a)\}$

$\delta(q_1, \epsilon, Z) = \{(q_2, \epsilon)\}$

Take i/p ~ aabb

$$\delta(q_0, aabb, Z) \vdash \{(q_1, aabb, SZ)\}$$

$$\vdash \{(q_1, aabb, aSAZ)\}$$

$$\vdash \{(q_1, abb, SAZ)\}$$

$$\vdash \{(q_1, abb, aAZ)\}$$

$$\vdash \{(q_1, bb, AZ)\}$$

$$\vdash \{(q_1, bb, bBZ)\}$$

$$\vdash \{(q_1, b, BZ)\}$$

$$\vdash \{(q_1, b, bZ)\}$$

$$\vdash \{(q_1, \epsilon, Z)\}$$

$$\vdash \{(q_2, \epsilon)\}.$$

From PDA's to Grammars

Rules:

1. $\delta(q_i, a, A) = \{(q_j, \epsilon)\}$

$$[q_i, A, q_j] \longrightarrow a$$

2. $\delta(q_i, a, X) = \{(q_j, AX)\}$

$$[q_i, X, q_l] \rightarrow a[q_j, A, q_k][q_k, X, q_l]$$

pbm.

1) Construct a CFG for the following PDA

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0, \phi)$$

$$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$$

$$\delta(q_0, 0, X) = \{(q_0, XX)\}$$

$$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$$

$$\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, X) = \{q_1, \epsilon\}$$

$$\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$$

Sol:

next stack symbol.

$$\delta(q_0, 0, Z_0) = \{(q_0, \underbrace{XZ_0}_{\text{any one}})\}$$

option state.

diff states

$$[q_0, Z_0, q_0] \rightarrow 0[q_0, X, q_0][q_0, Z_0, q_0]$$

$$[q_0, Z_0, q_0] \rightarrow 0[q_0, X, q_1][q_1, Z_0, q_0]$$

$[q_0, Z_0, q_1] \rightarrow 0[q_0, X, q_0][q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \rightarrow 0[q_0, X, q_1][q_1, Z_0, q_1]$

$\delta(q_0, 0, X) = \{(q_0, XX)\}$

$[q_0, X, q_0] \rightarrow 0[q_0, X, q_0][q_0, X, q_0]$

$[q_0, X, q_0] \rightarrow 0[q_0, X, q_1][q_1, X, q_0]$

$[q_0, X, q_1] \rightarrow 0[q_0, X, q_0][q_0, X, q_1]$

$[q_0, X, q_1] \rightarrow 0[q_0, X, q_1][q_1, X, q_1]$

$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$

$[q_0, X, q_1] \rightarrow 1$

$\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$

$[q_1, X, q_1] \rightarrow 1$

$\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$

$[q_1, X, q_1] \rightarrow \epsilon$

$\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

$[q_1, Z_0, q_1] \rightarrow \epsilon$

After eliminating, the CFG is,

$S \rightarrow [q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \rightarrow 0[q_0, X, q_1][q_1, Z_0, q_1]$

$[q_0, X, q_1] \rightarrow 0[q_0, X, q_1][q_1, X, q_1]$

Scanned by TapScanner

$[q_0, X, q_1] \to 1$

$[q_1, X, q_1] \to 1$

$[q_1, X, q_1] \to \epsilon$

$[q_1, Z_0, q_1] \to \epsilon$

2. Construct the CFG for the following PDA.

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$

$\delta(q_0, \epsilon, Z_0) = \{(q_0, \epsilon)\}$

$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$

$\delta(q_0, a, Z) = \{(q_1, Z)\}$

$\delta(q_1, b, Z) = \{(q_1, \epsilon)\}$

$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$

Ans:

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$

$[q_0, Z_0, q_0] \to b[q_0, Z, q_0][q_0, Z_0, q_0]$

$[q_0, Z_0, q_0] \to b[q_0, Z, q_1][q_1, Z_0, q_0]$

$[q_0, Z_0, q_1] \to b[q_0, Z, q_0][q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \to b[q_0, Z, q_1][q_1, Z_0, q_1]$

$\delta(q_0, \epsilon, Z_0) = \{(q_0, \epsilon)\}$

$[q_0, Z_0, q_0] \to \epsilon$

$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$

$[q_0, Z, q_0] \to b\,[q_0, Z, q_0][q_0, Z, q_0]$

$[q_0, Z, q_0] \to b\,[q_0, Z, q_1][q_1, Z, q_0]$

$[q_0, Z, q_1] \to b\,[q_0, Z, q_0][q_0, Z, q_1]$

$[q_0, Z, q_1] \to b\,[q_0, Z, q_1][q_1, Z, q_1]$

$\delta(q_0, a, Z) = \{(q_1, Z)\}$

$[q_0, Z, q_1] \to a[q_1, Z, q_1]$

$[q_0, Z, q_0] \to a\,[q_1, Z, q_0]$

$\delta(q_1, b, Z) = \{(q_1, \epsilon)\}$

$[q_1, Z, q_1] \to b$

$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$

$[q_1, Z_0, q_0] \to a\,[q_0, Z_0, q_0]$

$[q_1, Z_0, q_1] \to a\,[q_0, Z_0, q_1]$

After eliminating the CFG is,

$S \to [q_0, Z_0, q_1]$

$[q_1, Z, q_1] \to b$

$[q_0, Z, q_1] \to a[q_1, Z, q_1]$

$[q_1, Z_0, q_1] \to a\,[q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \to b\,[q_0, Z, q_1][q_1, Z_0, q_1]$

$[q_0, Z, q_1] \to b\,[q_0, Z, q_1][q_1, Z, q_1]$

Extra problems on PDA.

1) Construct a PDA for accepting the language

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

Ans.

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

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

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

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

$\delta(q_0, b, a) = \{(q_1, a)\}$ — No change on stack

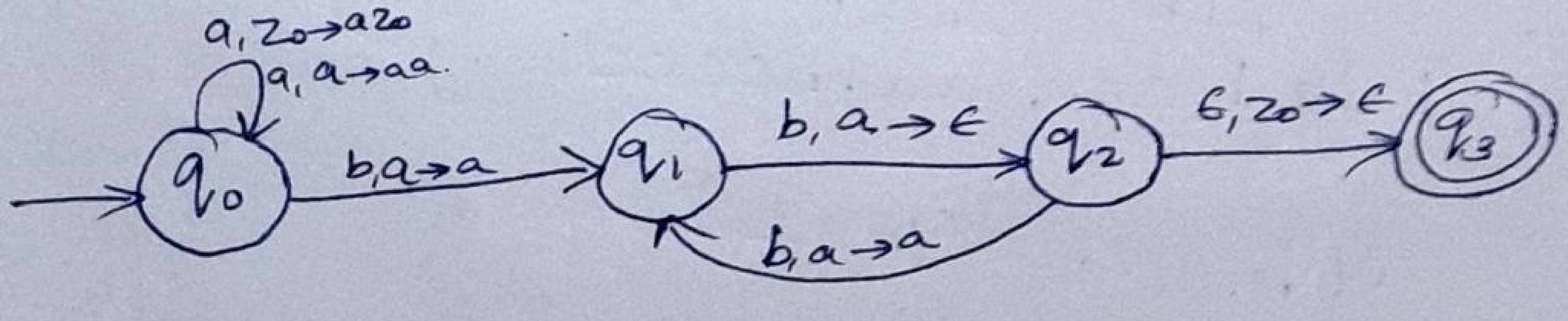
$\delta(q_1, b, a) = \{(q_2, \epsilon)\}$

odd b's goto state q_1
even b's goto state q_2

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

$\boxed{\delta(q_1, b, a) = \{(q_2, \epsilon)\}}$

$\delta(q_2, \epsilon, Z_0) \{(q_3, Z_0)\}$



Input: aaabbbbbb.

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

$\vdash (q_0, abbbbbb, aa z_0)$

$\vdash (q_0, bbbbbb, aaa z_0)$

$\vdash (q_1, bbbbb, aaa z_0)$

$\vdash (q_2, bbbb, aa z_0)$

$\vdash (q_1, bbb, aa, z_0)$

$\vdash (q_2, bb, a z_0)$

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

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

$\vdash (q_3, \epsilon)$

x ————→ x

2. Construct the PDA for accepting the language

$L = \{a^n b^m c^{n+m} \mid n \geq 0, m \geq 1$

Sln.

$L = \{\overset{n=1, m=1}{abcc}, \overset{n=2, m=2}{aabbcccc}, \ldots\}$

$n>0$. $\delta(q_0, a, z_0) = (q_0, a z_0)$

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

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

$\delta(q_1, b, b) = (q_1, bb)$

$\delta(q_1, c, b) = (q_2, \epsilon)$

$\delta(q_2, c, b) = (q_2, \epsilon)$

$\delta(q_2, c, a) = (q_2, \epsilon)$

$\delta(q_2, \epsilon, Z_0) = (q_3, Z_0)$

3. Construct the PDA for accepting the language

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

Sol

$L = \{abbccd, aabbbbbbdd, \ldots\}$

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

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

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

$\delta(q_1, b, b) = (q_1, bb)$

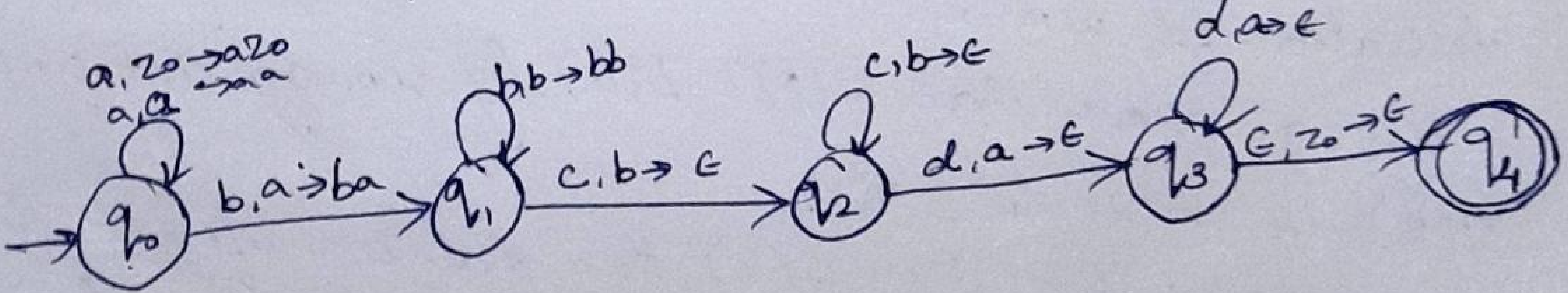
$\delta(q_1, c, b) = (q_2, \epsilon)$

$\delta(q_2, c, b) = (q_2, \epsilon)$

$\delta(q_2, d, a) = (q_3, \epsilon)$

$\delta(q_3, d, a) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, Z_0) = (q_4, Z_0)$



PDA = $\{\{q_0, q_1, q_2, q_3, q_4\}, \{a, b, c, d\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_4\}\}$

Theorem:

If PDF P is constructed from CFG. G then $N(P) = L(G)$

Proof:

Let $G = (V, T, P, S)$ be a grammar. Then exists a Greibach Normal form then we can construct PDA with simulates left most derivations in this grammar

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$

The transition function will include

$\delta(q_0, \epsilon, Z) = \{(q_1, SZ)\}$, so that after the first move of M, the stack contains the start symbol S of the derivation.

In addition, the set of transition rules is such that

i) $\delta(q_1, \epsilon, A) = \{(q_1, \alpha)\}$ for each $A \to \alpha$

ii) $\delta(q, a, a) = \{(q, \epsilon)\}$ for each $a \in \Sigma$

For a given input string w, the PDA simulates a leftmost derivation for w in G.

We can prove that $N(P) = L(G)$ by showing that w is in $N(P)$ iff w is in $L(G)$.

- If part: if w is in $L(G)$, then there is a leftmost derivation
- $S = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \gamma_n = w$ we show by induction on i that P simulates this leftmost derivation by the sequence of moves $(q, w, S) \vdash^* (q, y_i, \alpha_i)$ such that if $\gamma_i = x_i \alpha_i$, then $x_i y_i = w$.
- Only-if part: If $(q, x, A) \vdash^* (q, \epsilon, \epsilon)$, then $A \Rightarrow^* x$.
- We can prove this statement by induction on the no. of moves made by P.

To Complete the proof, w is in $N(P)$ & w is in $L(G)$, hence

$N(P) = L(G)$.

Theorem:

If PDF β is constructed from CFG. G then $N(P) = L(G)$

Proof.

Let $G = (V, T, P, S)$ be a grammar, Then exists a Greibach Normal form then we can construct PDA with simulates left most derivations in this grammar

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$$

The transition function will include

$\delta(q_0, \epsilon, Z) = \{(q_1, SZ)\}$, so that after the first move of M, the stack contains the start symbol S of the derivation.

In addition, the set of transition rules is such that

i) $\delta(q_1, \epsilon, A) = \{(q_1, \alpha)\}$ for each $A \to \alpha$

ii) $\delta(q, a, a) = \{(q, \epsilon)\}$ for each $a \in \Sigma$

For a given input string w, the PDA simulates a leftmost derivation for w in G.

We can prove that $N(P) = L(G)$ by showing that w is in $N(P)$ iff w is in $L(G)$.

- If part: if w is in $L(G)$, then there is a leftmost derivation
- $S = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \gamma_n = w$ we show by induction on i that

P simulates this leftmost derivation by the sequence of moves $(q, w, S) \vdash^* (q, y_i, \alpha_i)$ such that if $\gamma_i = x_i\alpha_i$, then $x_i y_i = w$.

- Only-if part: If $(q, x, A) \vdash^* (q, \epsilon, \epsilon)$, then $A \Rightarrow^* x$:
- We can prove this statement by induction on the no. of moves made by P.

To Complete the proof, w is in $N(P)$ & w is in $L(G)$, hence

$$N(P) = L(G).$$

Theorem:

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ be a PDA. Then there exists a CFG such that $L(G) = N(P)$.

Proof:

1. It has a single final state q_f iff the stack is empty.
2. All transitions must have the form

$\delta(q_i, a, A) = \{c_1, c_2, \dots c_n\}$ where

$\delta(q_i, a, A) = \{(q_j, \epsilon)\}$ —— ①

$\delta(q_i, a, A) = \{(q_j, BC)\}$ —— ②

i.e, each move either increases or decreases the stack content by a single symbol.

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{q_f\})$ satisfies the condition ① & ②

$G = (V, T, P, S)$

V - elements of the form $[q, A, p]$, q and p in Q & A in Γ

$T = \Sigma$

S - start symbol.

$S \to [q_0, Z_0, q]$ for each q in Q

P consists of: $u, v \in \Sigma^*$

$A, X \in \Gamma^*$

$q_i, q_j \in Q$

$(q_i, uv, AX) \vdash^* (q_f, v, X)$

implies that $[q_i, A, q_j] \to u$

Consider $[q_i, A, q_k] \to a[q_j, B, q_i][q_i, C, q_k]$

The corresponding transition for PDA is

$\delta(q_i, a, A) = \{(q_j, BC) \dots\}$

Similarly if $[q_i, A, q_j] \rightarrow a$ then the corresponding

transition is $\delta(q_i, a, A) = \{(q_j, \epsilon)\}$

For all sentential forms leading to a terminal string,

the argument holds true.

The conclusion is,

$\delta(q_0, w, Z_0) \vdash^* \{(q_f, \epsilon, \epsilon)\}$ is true iff $(q_0 Z_0 q_f) \xrightarrow{*} w$

consequently $L(M) = L(G)$.

Deterministic PushDown Automata.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then M is deterministic if and only if both the following conditions are satisfied.

1. $\delta(q, a, X)$ has atmost one element for any $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$ and $X \in \Gamma$

2. If $\delta(q, \epsilon, X) \neq \phi$ and $\delta(q, a, X) = \phi$ for every $a \in \Sigma$

For finite automata, the deterministic and non deterministic models were equivalent with respect to the languages accepted. The same is not true for PDA's. DPDAs accept only a subset of languages accepted NPDAs. That is NPDA is more powerful than DPDA. It is not always possible to convert non-deterministic pushdown automata to deterministic pushdown automata.

Non-Deterministic Pushdown Automata (NDPDA):

A PDA is called non-deterministic, if derivation generates more than one move in the designing of a particular task.

Pbm.

Check whether the language $L = \{a^n b^n \mid n > 0\}$ is deterministic CFL.

The PDA $M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{0, 1\}, \delta, q_0, Z_0 \{q_3\})$

with.

$\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$

$\delta(q_1, a, a) = \{(q_1, aa)\}$

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

$\delta(q_2, b, a) = \{(q_2, \lambda)\}$

$\delta(q_2, \lambda, Z_0) = \{(q_3, \lambda)\}$

It satisfies the DPDA conditions hence it is deterministic.

UNIT - IV
Properties of Context free Language

Normal Forms of CFG:

There are 2 normal forms for CFG.

1) Chomsky Normal Form (CNF)
2) Greiback Normal Form (GNF).

Simplifications of CFG:

1. Eliminate useless symbols
2. Eliminate ϵ production
3. Eliminate unit production

1. Eliminating useless symbols.

Useless symbols are those variables or terminals that do not appear in any derivation of a terminal string from the start symbol.

Eg1. Consider the grammar

$S \rightarrow AB / a$

$A \rightarrow b$

A generates B & S generates a. B does not generate any terminal so B can be eliminated.

After eliminating.

$\boxed{S \rightarrow a}$

$A \rightarrow b$

$S \rightarrow AB$.

$A \rightarrow b$. ×

Then. A cannot be replaced at $S \rightarrow AB$.

so remove $S \rightarrow AB$.

$\therefore S \rightarrow a$.

2. Eliminating ϵ production:

ϵ productions are of the form $A \to \epsilon$ for some variable A.

Eg) Grammar,

$S \to AB$

$A \to aAA / \epsilon$

$B \to bBB / \epsilon$

In the above grammar,

$A \to \epsilon$

$B \to \epsilon$ are ϵ productions.

After eliminating

$S \to AB / B / A$

$A \to aAA / aA / a$

$B \to bBB / bB / b$

3. Eliminating Unit productions.

Is of the form $A \to B$ where A & B are variables.

eg.

$I \to a / b / Ia / Ib / I0 / I1$

$F \to I / (E)$

$T \to F / T*F$

$E \to T / E+T$

In the above grammar,

$F \to I$, $T \to F$, $E \to T$ are unit productions

After eliminating we get

$F \to a / b / Ia / Ib / I0 / I1 / (E)$

$T \to a / b / Ia / Ib / I0 / I1 / (E) / T*F$

$E \to a / b / Ia / Ib / I0 / I1 / (E) / T*F / E+T$

$I \to a / b / Ia / Ib / I0 / I1$.

x ——————— x

Chomsky Normal Form (CNF):

Every Context free Language (CFL) is generated by a Context free Grammar (CFG) in which all productions are of the form $A \to BC$ or $A \to a$, $S \to \epsilon$ where A, B, C are variables and 'a' is a terminal. This form is called Chomsky Normal Form (CNF). [S is the start symbol]

Pbm.
Convert the grammar G to CNF

$S \to ASB \mid AC \mid \epsilon$
$A \to aAS \mid a$
$B \to SbS \mid A \mid bb$
$C \to AC \mid AB$

Ans Eliminate ϵ production.

$S \to \epsilon$
$S \to ASB \mid AB \mid AC$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid bS \mid Sb \mid b \mid A \mid bb$
$C \to AC \mid AB$

Eliminating unit productions.

$B \to A$

$S \to ASB \mid AB \mid AC$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid bS \mid Sb \mid b \mid aAS \mid aA \mid a \mid bb$
$C \to AC \mid AB$

Eliminating useless symbol.

C does not generate any terminal hence it is useless.

$S \to ASB / AB$

$A \to aAS / aA / a$

$B \to SbS / bS / Sb / b / aAS / aA / a / bb$

Converting the above grammar to CNF

$I \to a$

$J \to b$

$S \to ASB / AB$

$A \to IAS / IA / a$

$B \to SJS / JS / SJ / b / IAS / IA / a / JJ$

$K \to SB$

$L \to AS$

$M \to JS$

$S \to AK / AB$

$A \to IL / IA / a$

$B \to SM / JS / SJ / IL / IA / a / b / JJ$

The grammar in CNF is,

$S \to AK / AB$

$A \to IL / IA / a$

$B \to SM / JS / SJ / IL / IA / a / b / JJ$

$K \to SB$. $L \to AS$, $M \to JS$ $I \to a$ $J \to b$

2. Obtain the CNF for the grammar.

$S \rightarrow 0A0 / 1B1 / BB$

$A \rightarrow C$

$B \rightarrow S / A$

$C \rightarrow S / \epsilon$.

Sol:

Eliminate ϵ production.

$\boxed{C \rightarrow \epsilon}$

$S \rightarrow 0A0 / 1B1 / BB$

$A \rightarrow C / \epsilon$

$B \rightarrow S / A$

$C \rightarrow S$

$\boxed{A \rightarrow \epsilon}$

$S \rightarrow 0A0 / 00 / 1B1 / BB$

$A \rightarrow C$

$B \rightarrow S / A / \epsilon$

$C \rightarrow S$

$\boxed{B \rightarrow \epsilon}$

$S \rightarrow 0A0 / 00 / 1B1 / 11 / BB / B / \epsilon$

$A \rightarrow C$

$B \rightarrow S / A$

$C \rightarrow S$

$\boxed{S \rightarrow \epsilon}$

$S \rightarrow 0A0 / 00 / 1B1 / 11 / BB / B$

$A \rightarrow C$

$B \rightarrow AS / \epsilon / A$

$C \rightarrow S / \epsilon$

$$\boxed{\begin{array}{l} B \to \epsilon \\ C \to \epsilon \end{array}}$$

$S \to 0A0 / 00 / 1B1 / 11 / BB / B$

$A \to C$

$B \to S/A$

$C \to S$

Eliminate Unit production

$A \to C$

$B \to S$

$B \to A$

$C \to S$

$S \to 0A0 / 00 / 1B1 / 11 / BB / B$

$C \to 0A0 / 00 / 1B1 / 11 / BB / B$

$A \to 0A0 / 00 / 1B1 / 11 / BB / B$

$B \to 0A0 / 00 / 1B1 / 11 / BB / B$

Eliminate useless symbols.

There is no useless symbols.

Converting the above grammar into CNF.

$W \to 0$

$X \to 1$

$S \to WAW / WW / XBX / XX / BB / B$

$C \to WAW / WW / XBX / XX / BB / B$

$A \to WAW / WW / XBX / XX / BB / B$

$B \to WAW / WW / XBX / XX / BB / B$

$Y \to AW$

$Z \to BX$

$S \to WY / XZ / BB / WW / XX / B$

$A \to WY / XZ / BB / WW / XX / B$

$B \to WY / XZ / BB / WW / XX / B$

$C \to WY / XZ / BB / WW / XX / B$

The grammar in CNF form is,

$S \to WY / XZ / BB / WW / XX / B$

$A \to WY / XZ / BB / WW / XX / B$

$B \to WY / XZ / BB / WW / XX / B$

$C \to WY / XZ / BB / WW / XX / B$

$Y \to AW$

$Z \to BX$

$W \to 0$

$X \to 1$

x ———————— x

Greibach Normal Form (GNF)

A CFG G is in Greibach Normal Form (GNF) form if every production is of the form, $A \to a\alpha$ where $\alpha \in N^*$ and $a \in T$ and $S \to \lambda$ is in G.

- Start symbol can generate ϵ.
- Non Terminal generating single Terminal $A \to a$
- Non T. generating Terminal followed by any no. of symbols, or N.T.
  $A \to aAbBcC$

pbm.

Convert the following grammar in GNF

$S \rightarrow AB$
$A \rightarrow BS/b$
$B \rightarrow SA/a.$

Sln

$S \rightarrow AB$
$A \rightarrow BS/b$ ⊙ All productions are in CNF.
$B \rightarrow SA/a$

step 1: Renaming the variables S with A_1, A with A_2 & B with A_3.

$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1/b$
$A_3 \rightarrow A_1A_2/a$

step 2: Identify the grammar which does not satisfy the condition $A_i = A_j X_k$, $A_j > A_i$

$A_3 \rightarrow A_1A_2/a$ is identified.

step 3: Substitute A_1 in A_3 by substitution rule.

$A_3 \rightarrow A_2A_3A_2/a$

step 4: Again the condition is not satisfied so substitute A_2 in A_3.

$\underline{A_3} \rightarrow \underline{A_3}A_1A_3A_2/a/bA_3A_2$

step 5: Left recursive production is identified $A_3 \rightarrow A_3..$ So introduce a new variable Z and substitute A_3 in left recursion.

5

$A_3 \rightarrow bA_3A_2Z \,/\, aZ \,/\, bA_3A_2 \,/\, a$

$Z \rightarrow A_1A_3A_2 \,/\, A_1A_3A_2Z$

Step 6: Now the grammar is,

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \,/\, b$

$A_3 \rightarrow bA_3A_2Z \,/\, aZ \,/\, bA_3A_2 \,/\, a$

$Z \rightarrow A_1A_3A_2 \,/\, A_1A_3A_2Z$

Step 7: To convert into GNF substitute A_2 and A_3.

$A_3 \rightarrow bA_3A_2Z \,/\, aZ \,/\, bA_3A_2 \,/\, a$

$A_2 \rightarrow bA_3A_2ZA_1 \,/\, aZA_1 \,/\, bA_3A_2A_1 \,/\, aA_1 \,/\, b$

$A_1 \rightarrow bA_3A_2ZA_1A_3 \,/\, aZA_1A_3 \,/\, bA_3A_2A_1A_3 \,/\, aA_1A_3 \,/\, bA_3$

$Z \rightarrow bA_3A_2ZA_1A_3A_3A_2 \,/\, aZA_1A_3A_3A_2 \,/\, bA_3A_2A_1A_3A_3A_2 \,/\, aA_1A_3A_3A_2 \,/\, bA_3A_3A_2 \,/\, bA_3A_2ZA_1A_3A_3A_2Z \,/\, aZA_1A_3A_3A_2Z \,/\, bA_3A_2A_1A_3A_3A_2Z \,/\, aA_1A_3A_3A_2Z \,/\, bA_3A_3A_2Z$

x ———————— x

Left recursion procedure.

$A \rightarrow A\alpha \,/\, \beta$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \,/\, \epsilon$

pbm: Convert the grammar to GNF.

$S \to aSa$, $S \to bSb$, $S \to aa$, $S \to bb$.

Sr:

$S \to aSa$ $A \to a$

$S \to bSb$ $B \to b$

$S \to aa$ $\Rightarrow$ $S \to ASA$

$S \to bb$ $S \to BSB$

$S \to AA$

$S \to BB$

Step 1: Renaming the variables S with A_1, $A \to A_2$, $B - A_3$

$A_2 \to a$

$A_3 \to b$

$A_1 \to A_2 A_1 A_2$

$A_1 \to A_3 A_1 A_3$

$A_1 \to A_2 A_2$

$A_1 \to A_3 A_3$

Step 2: Identify the variables which does not satisfy the condition.

$A_i \to A_j X_k$, $A_j > A_i$.

Here all variables satisfy the condition.

Step 3: Converting the grammar to GNF.

$A_1 \to a A_1 A_2$

$A_1 \to b A_1 A_3$

$A_1 \to a A_2$

$A_1 \to b A_3$

$A_2 \to a$

$A_3 \to b$.

Scanned by TapScanner

Pumping Lemma for context Free Language (CFL)

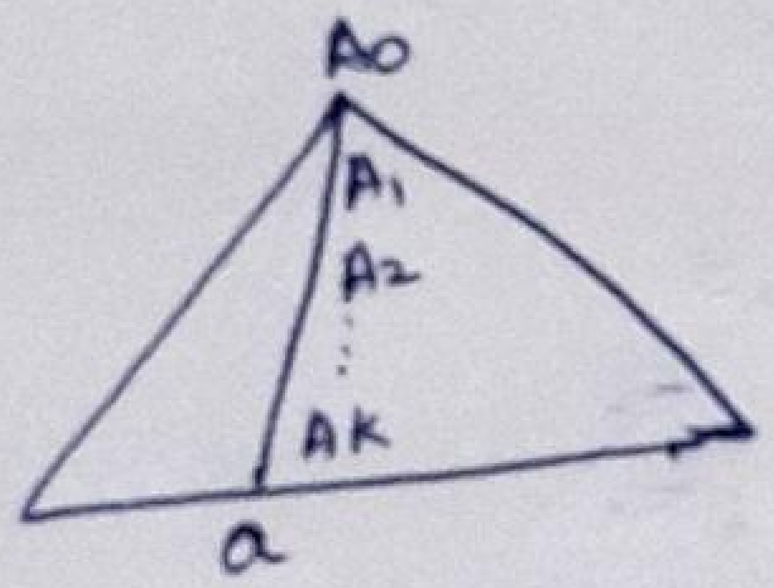
Let L be a CFL. Then there exists a constant n such that, if z is any string in L such that $|z|$ is at least n, then we can write $z = uvwxy$, subject to the following condition.

1. $|vwx| \leq n$
2. $vx \neq \epsilon$
3. For all $i \geq 0$, uv^iwx^iy is in L.

Proof

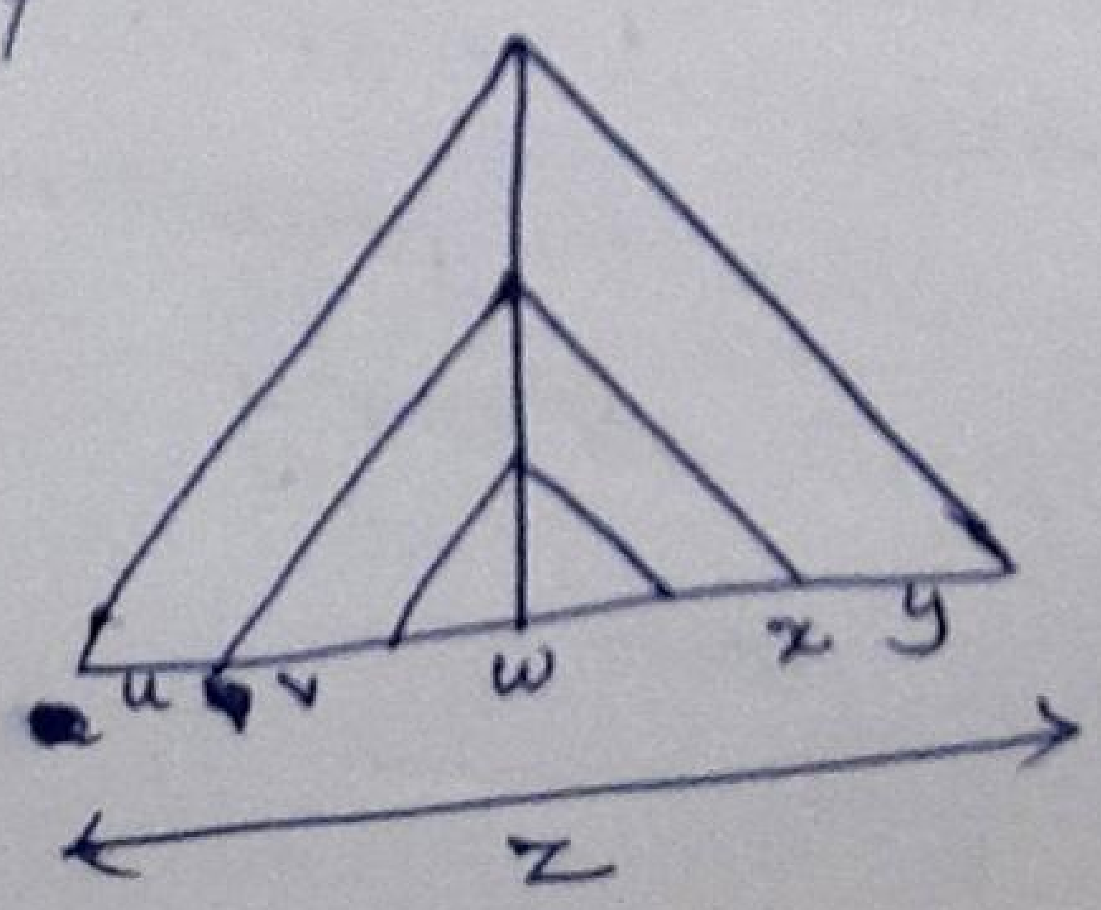
First step is to find a chomsky normal form grammar G for L.

Now starting with a CNF grammar $G = (V, T, P, S)$ let G have 'm' variables such that $L(G) = L - \{\epsilon\}$, let
choose $n = 2^m$

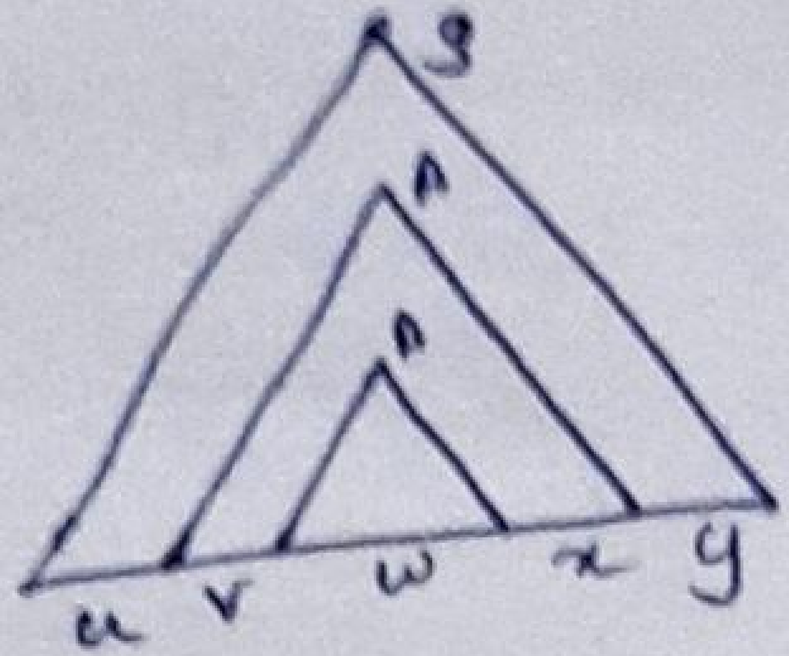


The above figure suggests the longest path in the tree for z, where k is atleast m and the path is of length $k+1$. since $k \geq m$, there are atleast $m+1$ accuracy of variables $A_0, A_1, \ldots A_k$ on the path."

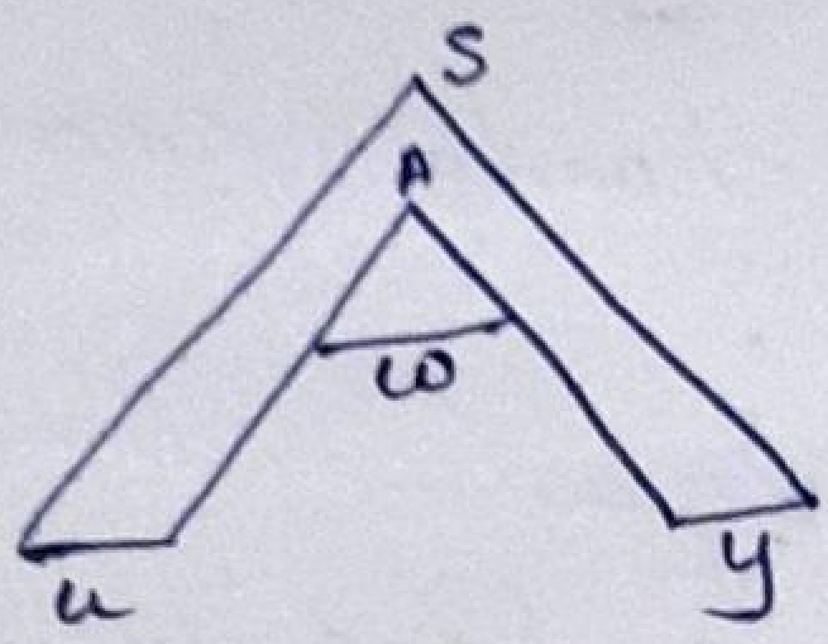
It is possible to divide the tree as.



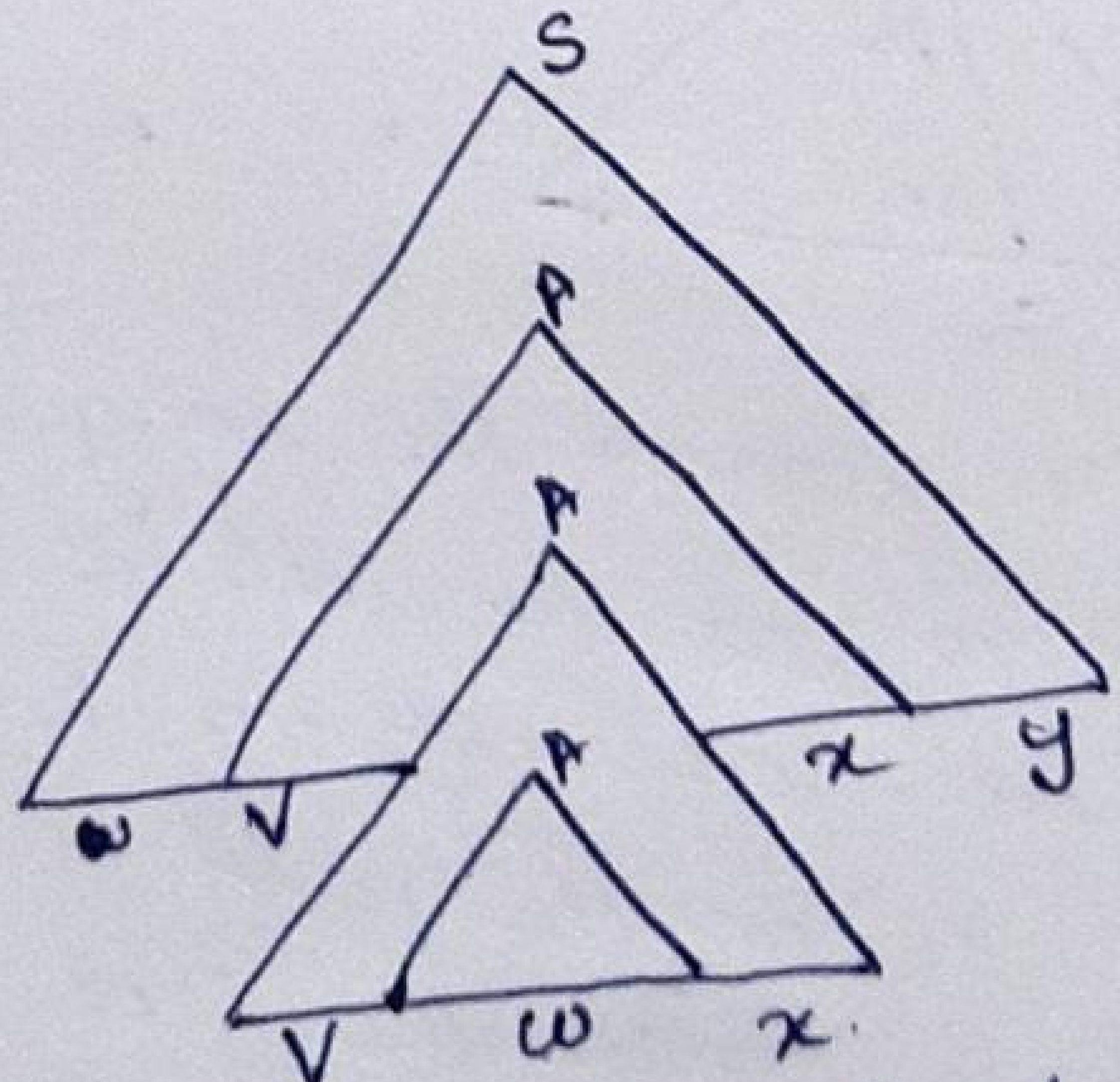
If $A_i = A_j = A$, then we can construct new parse tree as follows



First we may replace the subtree rooted at A_i, by the ~~subtree~~ subtree rooted at A_j



It has yield uwy and corresponds to the case $i=0$ in the pattern of strings uv^iwx^iy



In the above figure, we have replaced the ~~substring~~ subtree rooted at A_j by the entire subtree rooted at A_i. The yield of this tree is uv^2wx^2y. Thus the parse tree in G for all strings of the form uv^iwx^iy.

Hence proved

Pbm.

1. Prove that $L = \{a^n b^n c^n / n \geq 1\}$ is not context free.

An. Assume L is a context free language.

Let $z = a^p b^p c^p$.

$$\underbrace{aaa\ldots a}_{u}\,\underbrace{bbb\ldots b}_{vwx}\,\underbrace{ccc\ldots c}_{y}$$

$u = a^p$, $vwx = b^p$, $y = c^p$

$w = b^q$ $vx = b^{p-q}$

By using pumping lemma for CFL,

$$uv^iwx^iy = uvwxy\,v^{i-1}x^{i-1}$$
$$= uvwxy(vx)^{i-1}$$
$$= a^p b^p c^p (b^{p-q})^{i-1}$$
$$= a^p b^p c^p (b)^{(p-q)(i-1)}$$

for $i = 1$

$$uvwxy = a^p b^p c^p (b)^{(p-q)(0)}$$
$$= a^p b^p c^p \in L.$$

for $i = 2$.

$$uv^2wx^2y = a^p b^p c^p b^{(p-q)(2-1)}$$
$$= a^p b^p c^p b^{(p-q)} \notin L$$

The no. of a's, b's & c's are not equal. Hence the language is not context free

Show that $L = \{a^k b^j c^k d^j \mid j \geq 1, k \geq 1\}$ is not context free.

Ans:

Assume L is a context free language.

Let $z = a^p b^q c^p d^q$

$$\underbrace{aa\ldots a}_{u}\ \underbrace{bbbb\ldots b\ ccc\ldots c}_{vwx}\ \underbrace{dddd\ldots d}_{y}$$

$u = a^p \quad vwx = b^q c^p \quad y = d^q$

$w = b^r c^s$

$vx = b^{(q-r)} c^{(p-s)}$

By using pumping lemma for CFL,

$uv^iwx^iy = uvwxy\, v^{(i-1)} x^{(i-1)}$

$= q^p b^q c^p d^q \left(b^{(q-r)} c^{(p-s)}\right)^{(i-1)}$

$= a^p b^q c^p d^q\, b^{(q-r)(i-1)} c^{(p-s)(i-1)}$

for $i = 1$

$uvwxy = a^p b^q c^p d^q \in L$

for $i = 2$.

$uv^2wx^2y = a^p b^q c^p d^q\, b^{(q-r)} c^{(p-s)} \notin L.$

The no. of a's and c's are not equal & b's & d's are not equal. Hence the language is not context free.

Scanned by TapScanner

8

3 Show that $L = \{0^{2^i} \mid i \geq 1\}$ is not a context free

Sol:

Assume L is a context free Language.

$$0^{2^i} = 0^p \, [\because p = 2^i]$$

$$\underbrace{0}_{u}\,\underbrace{0\,0\,0\,0\,0\,\cdots\,0\,0}_{vwx}\,\underbrace{0}_{y}$$

$u = 0^r$ $\quad vwx = 0^s$ $\quad y = 0^{p-(r+s)}$

$vx = 0^t$

By using pumping lemma for CFL,

$$uv^iwx^iy = uvwxy\,(vx)^{i-1}$$

$$= 0^r 0^s 0^{p-(r+s)} 0^{t(i-1)}$$

for $i = 1$

$$uvwxy = 0^r 0^s 0^{p-(r+s)}$$

$$= 0^p = 0^{2^i} \in L$$

for $i = 2$.

$$uv^2wx^2y = 0^r 0^s 0^{p-(r+s)} 0^{t(1)}$$

$$= 0^{p+t} = 0^{t+2^i} \notin L$$

$t + 2^i$ is not a perfect square, hence L is not a Context free.

Closure Properties of CFL:

The family of CFL's is closed under

* Union
* Concatenation
* Star Closure (*) and positive closure (+)
* homomorphisms and inverse homomorphisms.

The family of CFL's is not closed under

* intersection
* complementation
* difference.

Theorem:

The family of context-free language is closed under union, concatenation, and star-closure.

Proof of Closure under Union

• Assume that L1 and L2 are generated by the context-free grammars G1 = (V1 , T1 , S1 , P1) and G2 = (V2 , T2 , S2 , P2)

• Without loss of generality, assume that the sets V1 and V2 are disjoint

• Create a new variable S3 which is not in V1 ∪ V2

• Construct a new grammar G3 = (V3 , T3 , S3 , P3) so that

– V3 = V1 ∪ V2 ∪ { S3 }

– T3 = T1 ∪ T2

– P3 = P1 ∪ P2

• Add to P3 a production that allows the new start symbol to derive either of the start symbols for L1 and L2 – S3 → S1 | S2

• Clearly, G3 is context-free and generates the union of L1 and L2 , thus completing the proof

Proof of Closure under Concatenation

• Assume that L1 and L2 are generated by the context-free grammars G1 = (V1 , T1 , S1 , P1) and G2 = (V2 , T2 , S2 , P2)

• Without loss of generality, assume that the sets V1 and V2 are disjoint

• Create a new variable S4 which is not in V1 ∪ V2

• Construct a new grammar G4 = (V4 , T4 , S4 , P4) so that

– V4 = V1 ∪ V2 ∪ { S4 }

– T4 = T1 ∪ T2

– P4 = P1 ∪ P2

• Add to P4 a production that allows the new start symbol to derive the concatenation of the start symbols for L1 and L2 – S4 → S1S2

• Clearly, G4 is context-free and generates the concatenation of L1 and L2 , thus completing the proof

Proof of Closure under Star-Closure

• Assume that L1 is generated by the context-free grammars G1 = (V1 , T1 , S1 , P1)

• Create a new variable S5 which is not in V1

• Construct a new grammar G5 = (V5 , T5 , S5 , P5) so that

– V5 = V1 ∪ { S5 }

– T5 = T1

– P5 = P1

• Add to P5 a production that allows the new start symbol S5 to derive the repetition of the start symbol for L1 any number of times – S5 → S1S5 | λ

• Clearly, G5 is context-free and generates the star-closure of L1 , thus completing the proof

Theorem:

The family of context-free language is not closed under intersection and complementation.

No Closure under Intersection

• Unlike regular languages, the intersection of two context-free languages L1 and L2 does not necessarily produce a contextfree language

• As a counterexample, consider the context-free languages L1 = { $a^n b\ ^n c^m$: n ≥ 0, m ≥ 0 } L2 = { $a^n b^m c^m$: n ≥ 0, m ≥ 0 }

• However, the intersection L1 and L2 is the language L3 = { $a^n b^n c^n$: n ≥ 0 }

• L3 can be shown not be context-free by applying the pumping lemma for context-free languages

Not Closed under Complementation:(By contradiction)

- Suppose that context-free languages *are* closed under complementation.
- Then if L_1 and L_2 are context-free languages, so are L_1' and L_2'. Since we have proved closure under union, $(L_1' \cup L_2')$ must also be context-free, and, by our assumption, so must its complement $(L_1' \cup L_2')'$.
- However, by de Morgan's laws (for sets), $(L_1' \cup L_2')' \equiv (L_1 \cap L_2)$, so this must also be a context-free language.
- Since our choice of L_1 and L_2 was arbitarary, we have contradicted the non-closure of intersection, and have thus proved the lemma.

TURING MACHINES:

Turing machine is a simple mathematical model of a computer. It was proposed by the mathematician "Alan Turing" in 1936.

It is similar to a finite automaton but with unlimited and unrestricted memory. Turing machine is a much more accurate model of a general purpose computer.

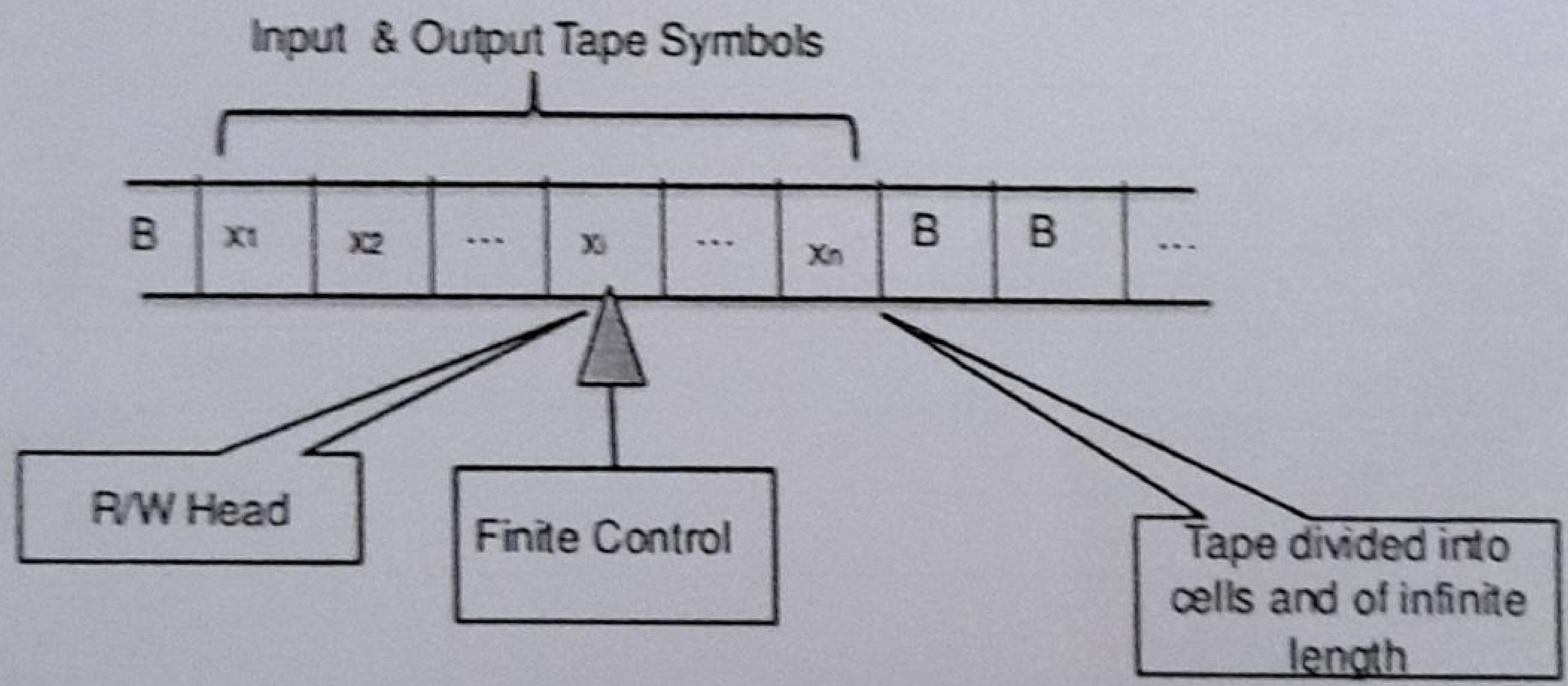
Formal Definition :

Formally, a deterministic turing machine (DTM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where

- Q is a finite nonempty set of states.
- Γ is a finite non-empty set of tape symbols, callled the tape alphabet of M.
- $\Sigma \subseteq \Gamma$ is a finite non-empty set of input symbols, called the input alphabet of M.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L \times R\}$ is the transition function of M.
- $q_0 \in Q$ is the initial or start state.
- $B \in \Gamma \setminus \Sigma$ is the blank symbol
- $F \subseteq Q$ is the set of final state.

So, given the current state and tape symbol being read, the transition function describes the next state, symbol to be written on the tape, and the direction in which to move the tape head (L and R denote left and right, respectively).

MODELS:



The turing machine can be thought of as a finite state automaton connected to a R/W (Read/Write) head. It has an infinite tape which is divided into number of cells.

Each cell stores one symbol. The input to and the output from the finite state automata (or) control unit are affected by R/W head which can examine one cell at a time.

In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automata to determine.

1. A new symbol to be written on the tape in the cell under the R/W head.
2. A motion of the R/W head along the tape: either the head moves one cell left or one cell right.
3. The next state of automaton
4. whether to halt or not.

PROBLEMS

1. Design a turing machine that accept the language $L=\{0^n1^n : n\geq 1\}$ compute 0011.

The formal sepcification of the TM M is.

$$M=(\{q_0,q_1,q_2,q_3,q_4\},\{0,1\},\{0,1,X,Y,B\},\delta,\ q_0,B,\{q_4\})$$

The transitions are as follows:

$\delta(q_0,0)=(q_1,X,R)$

$\delta(q_1,0)=(q_1,0,R)$
$\delta(q_1,1)=(q_2,Y,L)$
$\delta(q_2,0)=(q_2,0,L)$
$\delta(q_2,X)=(q_0,X,R)$
$\delta(q_1,Y)=(q_1,Y,R)$
$\delta(q_2,Y)=(q_2,Y,L)$
$\delta(q_3,Y)=(q_3,Y,R)$
$\delta(q_0,Y)=(q_3,Y,R)$
$\delta(q_3,B)=(q_4,B,R)$

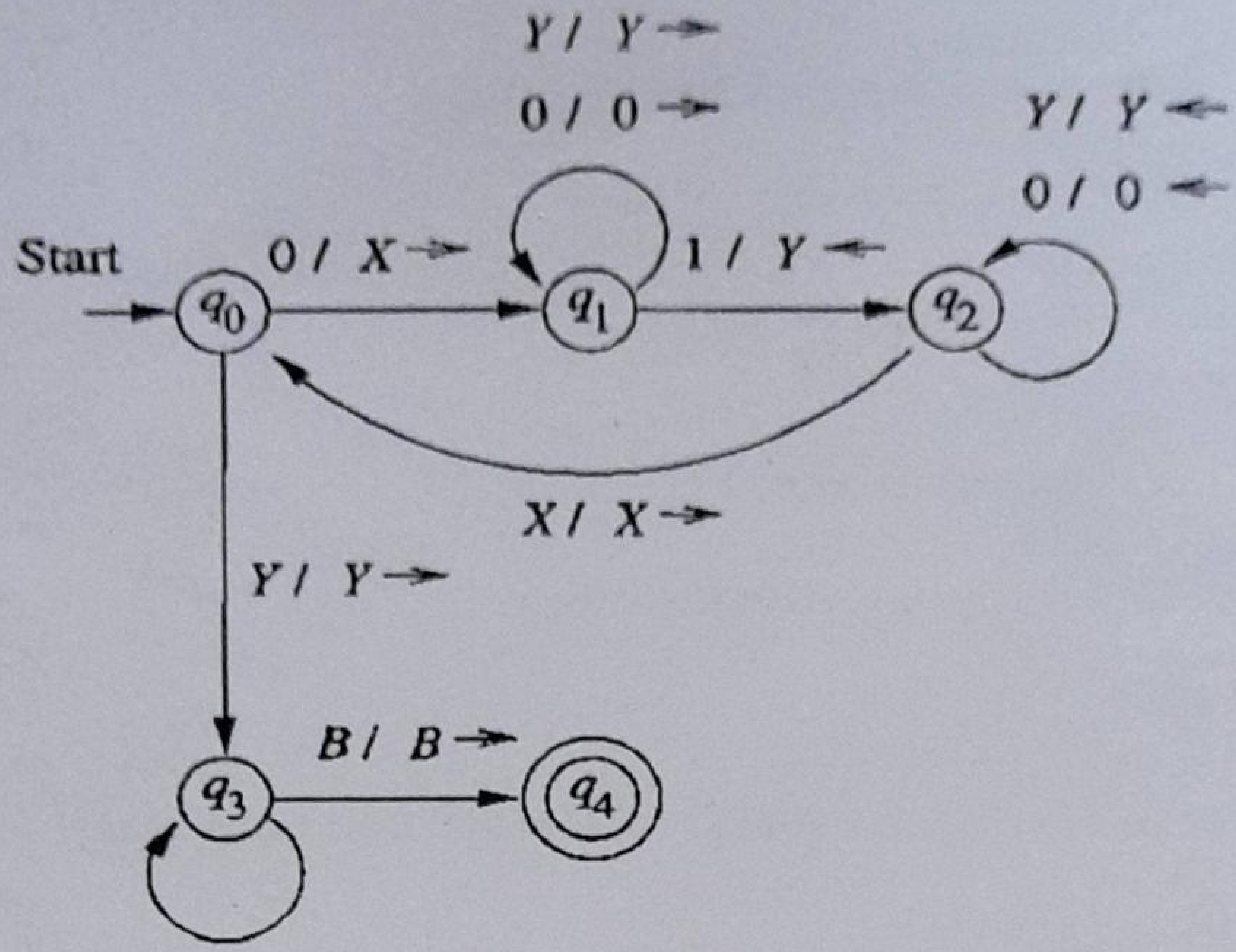
Transition Table:

State	0	1	X	Y	B
$\rightarrow q_0$	(q_1,X,R)	-	-	(q_3,Y,R)	-
q_1	$(q_1,0,R)$	(q_2,Y,L)	-	(q_1,Y,R)	-
q_2	$(q_2,0,L)$	-	(q_0,X,R)	(q_2,Y,L)	-
q_3	-	-	-	(q_3,Y,R)	(q_4,B,R)
$*q_4$	-	-	-	-	-

0011

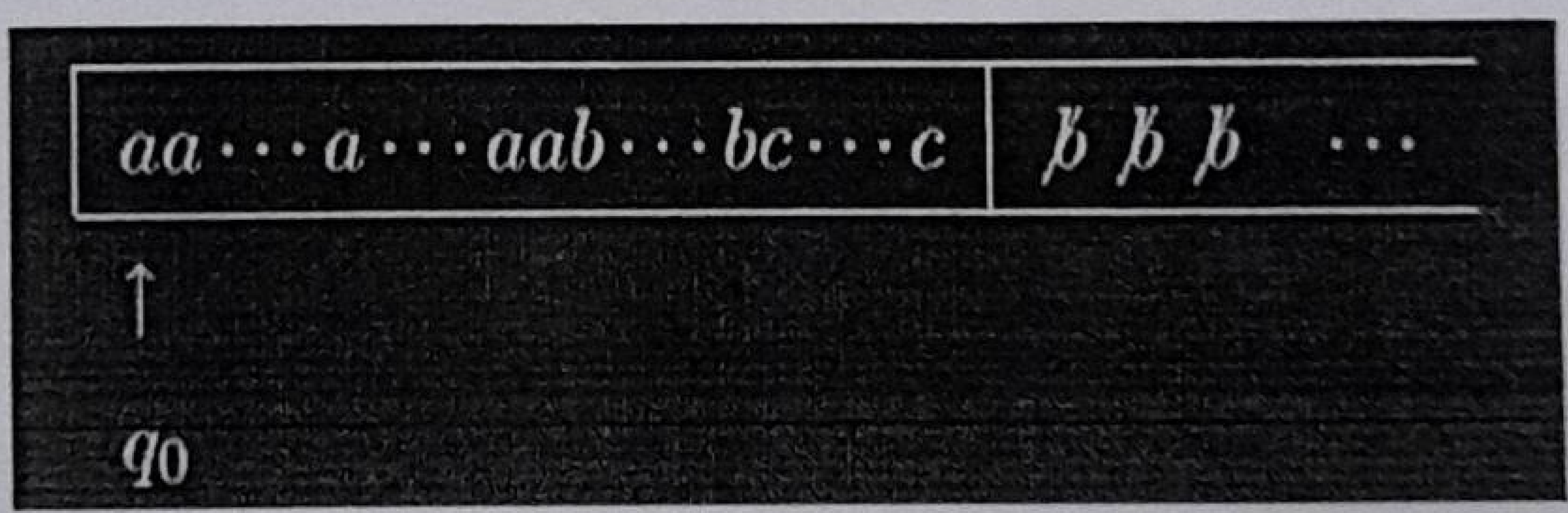
$q_0 0011 \vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1 \vdash Xq_00Y1 \vdash XXq_1Y1 \vdash XXYq_11$
$\vdash XXq_2YY \vdash Xq_2XYY \vdash XXq_0YY \vdash XXYq_3Y \vdash XXYYq_3B \vdash XXYYBq_4B$

Transition Diagram:



2. Construct a TM for accepting $\{a^i b^j c^k | i,j,k \geq 1, i = j + k\}$.

The informal description of the TM is as follows. Consider the figure which shows the initial ID.



The machine starts reading a ‘a’ and changing it to a X; it moves right; when it sees a ‘b`, it converts it into a Y and then starts moving left. It matches a’s and b’s. After that, it matches a’s with c’s. The machine accepts when the number of a’s is equal to the sum of the number of b’s and the number of c’s.

Formally M = (K,Σ,Γ,δ,q0,F)

K = {q0,q1,q2,q3,q4,q5,q6,q7,q8}

F = {q8}

Σ = {a,b,c}

Γ = {a,b,c,X,Y,Z,6 b}

δ is defined as follows:

$\delta(q0,a) = (q1,X,R)$

In state q0, it reads a 'a' and changes it to X and moves right in q1.

$\delta(q1,a) = (q1,a,R)$

In state q1it moves right through the 'a's.

$\delta(q1,b) = (q2,Y,L)$

When it sees a 'b' it changes it into a Y .

$\delta(q2,a) = (q2,a,L)$

$\delta(q2,Y) = (q2,Y,L)$

In state q2it moves left through the 'a's and Y s.

$\delta(q2,X) = (q0,X,R)$

When it sees a X it moves right in q0and the process repeats.

$\delta(q1,Y) = (q3,Y,R)$

$\delta(q3,Y) = (q3,Y,R)$

$\delta(q3,b) = (q2,Y,L)$

After scanning the 'a's it moves through the Y s still it sees a 'b', then it converts it into a Y and moves left.

$\delta(q3,c) = (q4,Z,L)$

When no more 'b's remain it sees a 'c' in state q3, changes that it into Z and starts moving left in state q4. The process repeats. After matching 'a's and 'b's, the TM tries to match 'a's and 'c's.

$\delta(q4,Y) = (q4,Y,L)$

$\delta(q4,a) = (q4,a,L)$

$\delta(q4,X) = (q0,X,R)$

$\delta(q3,Z) = (q5,Z,R)$

$\delta(q5,c) = (q4,Z,L)$

$\delta(q5,Z) = (q5,Z,R)$

$\delta(q4,Z) = (q4,Z,L)$

When no more 'a's remain it sees a Y in state q0 checks that all 'b's and 'c's have been matched and reaches the final state q8.

$\delta(q0,Y) = (q6,Y,R)$

$\delta(q6,Y) = (q6,Y,R)$

$\delta(q6,Z) = (q7,Z,R)$

$\delta(q7,Z) = (q7,Z,R)$

$\delta(q7,6\ b) = (q8,b,halt)$

Ex.1: Construct a TM that performs addition. **[Nov/Dec11, 12]**

Soln:

Procedure:

- The function is defined as f(x, y) = x+y.
  'x' is given by 0^x.
  'y' is given by 0^y.
- The input is placed on tape as $0^x|0^y$, where '|' is the separator.
- Then the output will be 0^{x+y}.
- Starting from the first zero in the 0^x, the tape head moves till it finds a separator '|' and replaces it by '0', move right to find the blank symbol.
- Then moves left one cell and replace the zero in that cell by a blank symbol.

Let M = (Q, Σ, Γ, δ, q_0, B, F), assume the set of states Q = {q_0, q_1, q_2, q_3}, Σ = {0, 1}, Γ = {0,1,B}, q_0 = {q_0}, F = {q_3}.

Assume x = 3, y = 2 then, input string is: $0^3|0^2$ ==> 000|00BBB

000|00BBB ⊢ 000|00BBB ⊢ 000|00BBB ⊢ 000|00BBB ⊢ 000000BBB ⊢ 000000BBB ⊢ 000000BBB

$\uparrow_{q_0}$ $\uparrow_{q_0}$ $\uparrow_{q_0}$ $\uparrow_{q_0}$ $\uparrow_{q_1}$ $\uparrow_{q_1}$ $\uparrow_{q_1}$

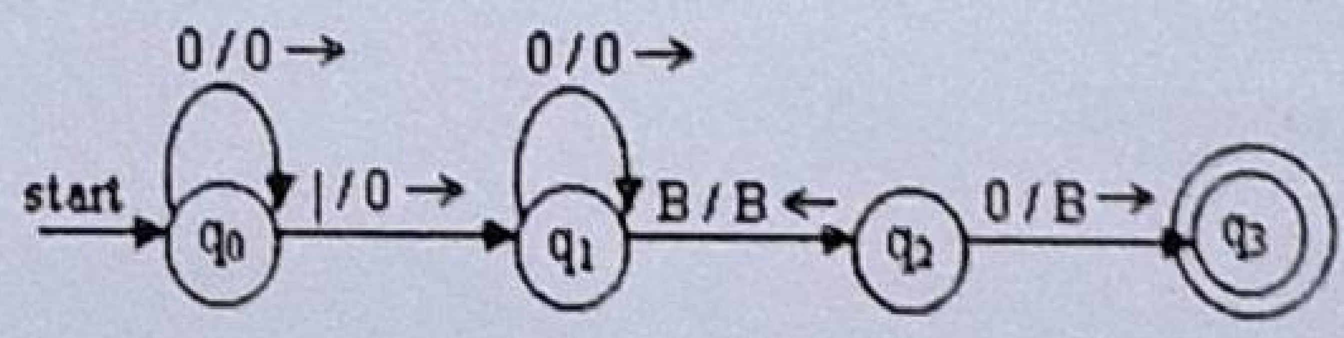
⊢ 000000BBB ⊢ 00000BBBB

$\uparrow_{q_2}$ $\uparrow_{q_3}$

Transition Table:

State	0	|	B
→q_0	(q_0,0,R)	(q_1,0,R)	-
q_1	(q_1,0,R)	-	(q_2,B,L)
q_2	(q_3,B,R)	-	-
*q_3	-	-	-

Transition Diagram:



Ex.2: Construct a TM to compute the function, $f(x) = x+1$

Soln:

- 'x' is given by 0^x.
- $\therefore f(x) = x+1 = 0^{x+1}$.
- The output contains one more '0' than the input.
- Initially the TM is at q_0.
- At q_0 if it reads a blank symbol by skipping 0's, replace it with '0' and enters final state.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, assume the set of states $Q = \{q_0, q_1\}$, $\Sigma = \{0\}$, $\Gamma = \{0, B\}$, $q_0 = \{q_0\}$, $F = \{q_1\}$.

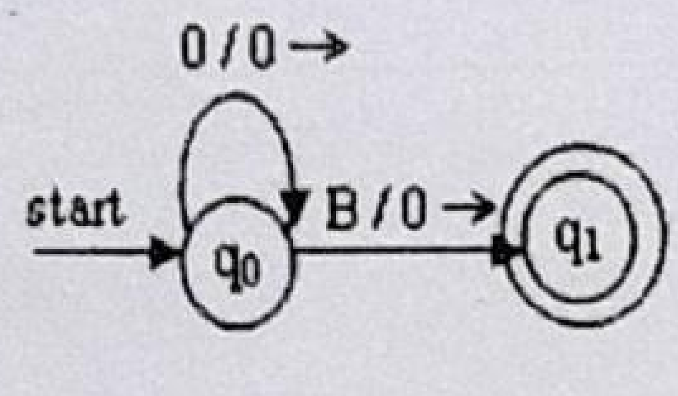
Assume x = 3 then, input string is: $0^3 \Longrightarrow$ 000BBB

$$\underset{\uparrow q_0}{000BBB} \vdash \underset{\uparrow q_0}{000BBB} \vdash \underset{\uparrow q_0}{000BBB} \vdash \underset{\uparrow q_0}{000BBB} \vdash \underset{\uparrow q_1}{0000BB}$$

Transition Table:

State	0	B
$\rightarrow q_0$	$(q_0,0,R)$	$(q_1,0,R)$
$*q_1$	-	-

Transition Diagram:



Ex.3: Design a TM to compute proper subtraction.

Soln: Proper subtraction is defined by $m \dot{-} n$.

ie) $m \dot{-} n = \max(m - n, 0)$

$$m \dot{-} n \text{ is } \begin{cases} m - n & \text{if} \quad m > n \\ 0 & \text{if} \quad m < n \end{cases}$$

Procedure:

- The TM starts its operation with $0^m|0^n$ on its input tape.
- Initially the TM is at state q_0.
- At q_0, it replaces the leading '0' by blank and search right looking for first '|'.
- After finding it, the TM searches right for '0' and change it to '|'.
- Then move the tape head to left till reaches the blank symbol. And then enter state q_0 to repeat the cycle.

The repetition ends if:

(1) Searching right for a '0', TM encounters a blank. Then n 0's in $0^m|0^n$ have all been changed to B. Replace the $(n+1)^{th}$ '|' by '0' and n B's. Leaving m-n 0's on its tape.

(2) TM cannot find a '0' to change it to blank during the beginning of the cycle. Change all zero's and 1's to blank and the result in zero.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, assume the set of states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0,1,B\}$, $q_0 = \{q_0\}$, $F = \{q_6\}$.

(i) Assume m=2, n=1 then, Input string is: 00|0

$00|0BBB \vdash B0|0BBB \vdash B0|0BBB \vdash B0|0BBB \vdash B0|1BBB \vdash B0|1BBB \vdash B0|1BBB \vdash B0|1BBB$

$\uparrow q_0 \quad \uparrow q_1 \quad \uparrow q_1 \quad \uparrow q_2 \quad \uparrow q_3 \quad \uparrow q_3 \quad \uparrow q_3 \quad \uparrow q_0$

$\vdash BB|1BBB \vdash BB|1BBB \vdash BB|1BBB \vdash BB|1BBB \vdash BB|BBBB \vdash BBBBBBB$

$\uparrow q_1 \quad \uparrow q_2 \quad \uparrow q_2 \quad \uparrow q_4 \quad \uparrow q_4 \quad \uparrow q_4$

$\vdash B0BBBBB$

$\uparrow q_4$

(ii) Assume m=1, n=2 then, input string is: 0|00

$0|00BB \vdash B|00BB \vdash B|00BB \vdash B|10BB \vdash B|10BB \vdash B|10BB \vdash BB10BB \vdash BBB0BB$

$\uparrow q_0 \quad \uparrow q_1 \quad \uparrow q_2 \quad \uparrow q_3 \quad \uparrow q_3 \quad \uparrow q_0 \quad \uparrow q_5 \quad \uparrow q_5$

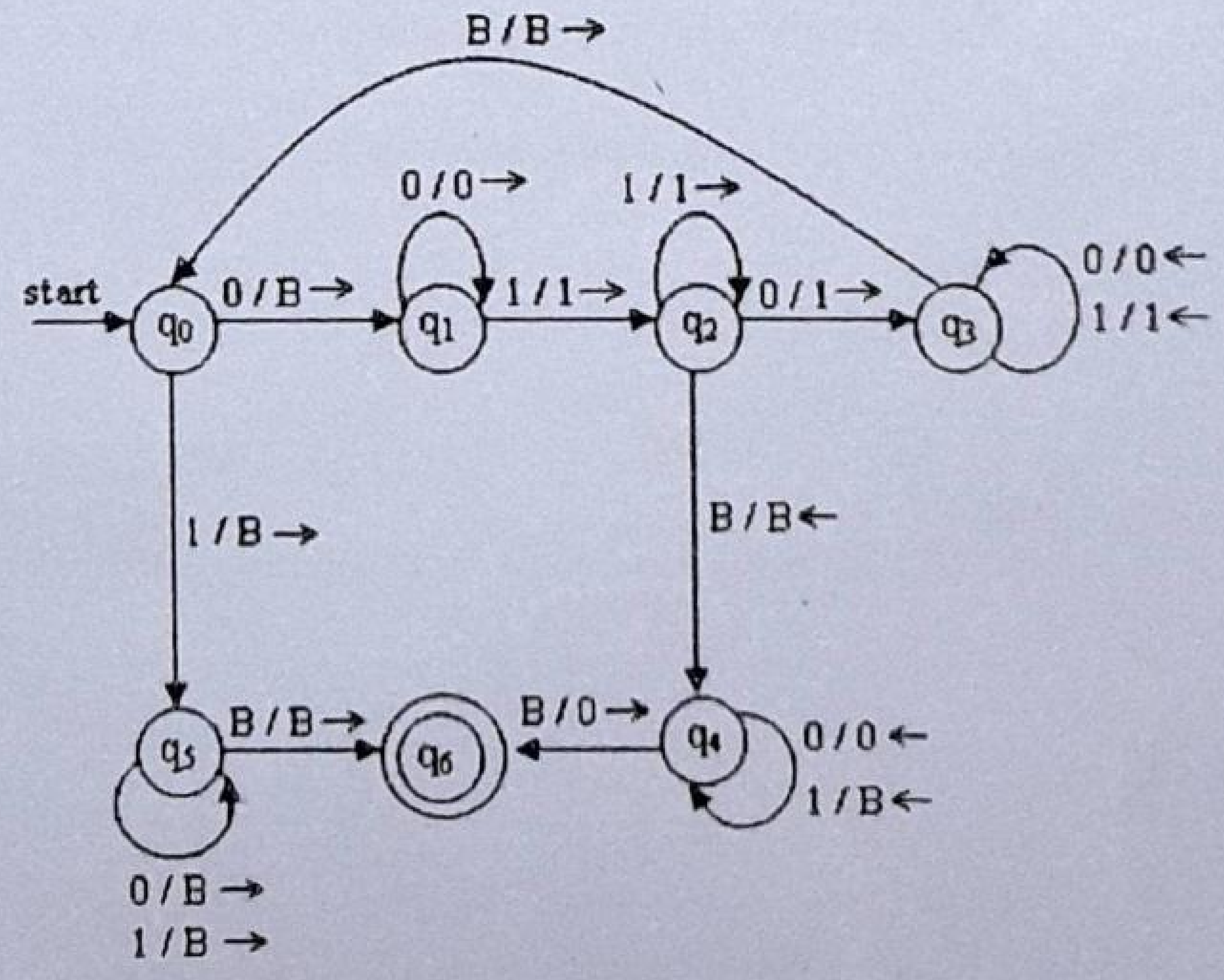
$\vdash BBBBBB \vdash BBBBBB$

$\uparrow q_5 \quad \uparrow q_6$

Transition Table:

State	0	1	B
$\rightarrow q_0$	(q_1,B,R)	(q_5,B,R)	-
q_1	$(q_1,0,R)$	$(q_2,1,R)$	-
q_2	$(q_3,1,L)$	$(q_2,1,R)$	(q_4,B,L)
q_3	$(q_3,0,L)$	$(q_3,1,L)$	(q_0,B,R)
q_4	$(q_4,0,L)$	(q_4,B,L)	$(q_6,0,R)$
q_5	(q_5,B,R)	(q_5,B,R)	(q_6,B,R)
$*q_6$	-	-	-

Transition Diagram:



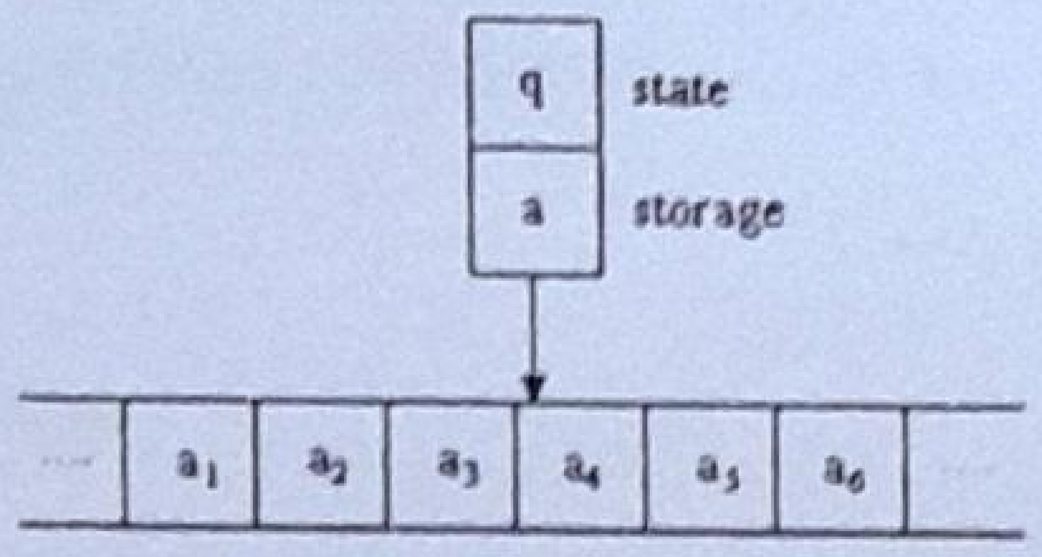
PROGRAMMING TECHNIQUES FOR TM:

There are different techniques are used for constructing Turing Machine. They are,

(1) Storage in the state.
(2) Multiple Tracks
(3) Subroutines

(1) Storage in the state:

The finite control holds a finite amount of information. Then the state of the finite control is represented as a **pair of elements**. The first element represents the **state** and the second element represents **storing a symbol**.



Ex.1: Construct a TM, M=(Q, {0,1}, {0,1,B}, δ, [q_0,B], Z0, [q_1,B]), that look at the first input symbol records in the finite control and checks that symbol does not appear elsewhere on its input.

Soln: For the states of Q as, $Q \times \{0, 1, B\} = \{q_0, q_1\} \times \{0, 1, B\}$

$$Q = \{[q_0, 0], [q_0, 1], [q_0, B], [q_1, 0], [q_1, 1], [q_1, B]\}$$

In this, the finite control holds a pair of symbol, that is, both the state and the symbol.

(i) $\delta([q_0, B], a) = ([q_1, a], a, R)$; where, a=0 (or) 1

At 'q_0', the TM reads the first symbol 'a' and goes to state 'q_1'. The input symbol is coped into the second component of the state and moves right.

(ii) $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, R)$; where, $\bar{a}$ is the complement of 'a'.

ie) if $a = 0$ then $\bar{a} = 1$

if $a = 1$ then $\bar{a} = 0$

At q_1, if the TM reads the other symbols, M skips over and moves right.

(iii) $\delta([q_1, a], a) = ([q_1, B], B, R)$

If M reaches the same symbol, it halts without enters accepting.

(iv) $\delta([q_1, a], B) = ([q_2, B], B, R)$

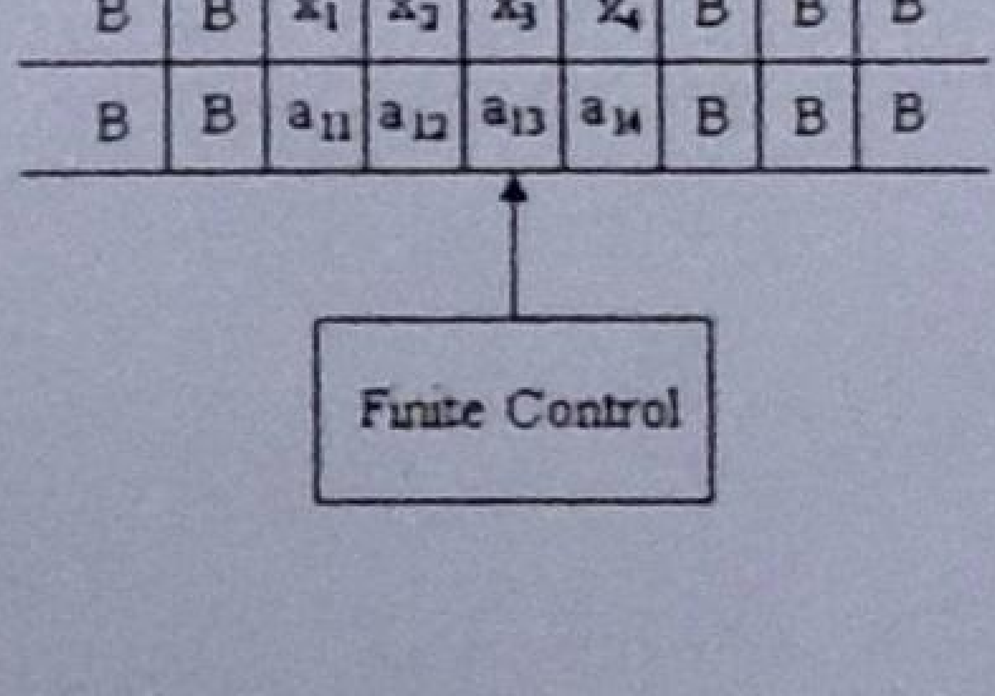
If M reaches the first blank, then it enters the accepting state.

Input String: 011BBB

$$\underset{[q_0, B]}{\underset{\uparrow}{0}11BBB} \vdash \underset{[q_1,0]}{0\underset{\uparrow}{1}1BBB} \vdash \underset{[q_1,0]}{01\underset{\uparrow}{1}BBB} \vdash \underset{[q_1,0]}{011\underset{\uparrow}{B}BB} \vdash \underset{[q_2, B]}{011B\underset{\uparrow}{B}B}$$

(2) Multiple Tracks:

It is possible that a Turing Machine's input tape can be divided into several tracks. Each track can hold symbols.



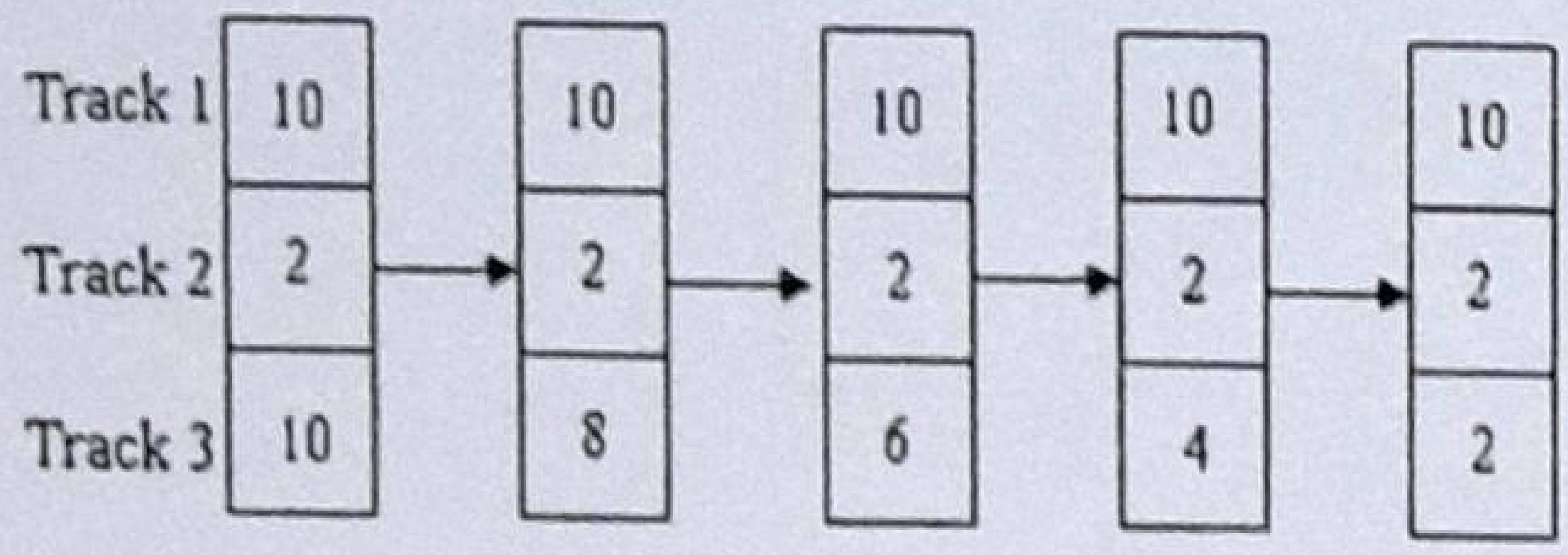
Ex.1: Construct a TM that takes an input greater than 2 and checks whether it is even or odd.

Soln:

Procedure:

(1) The input is placed into first tape or track.
(2) The integer 2 is placed on the second track.
(3) The input on the first track is copied into third track.
(4) The number on the second track is subtracted from the third track.
(5) If the remainder is same as the number in the second track then the number on the first track is even.
(6) If it is greater than 2, then continue this process until the remainder in the third track is <= 2, if it is equal to 2 then the number is even otherwise it is odd.

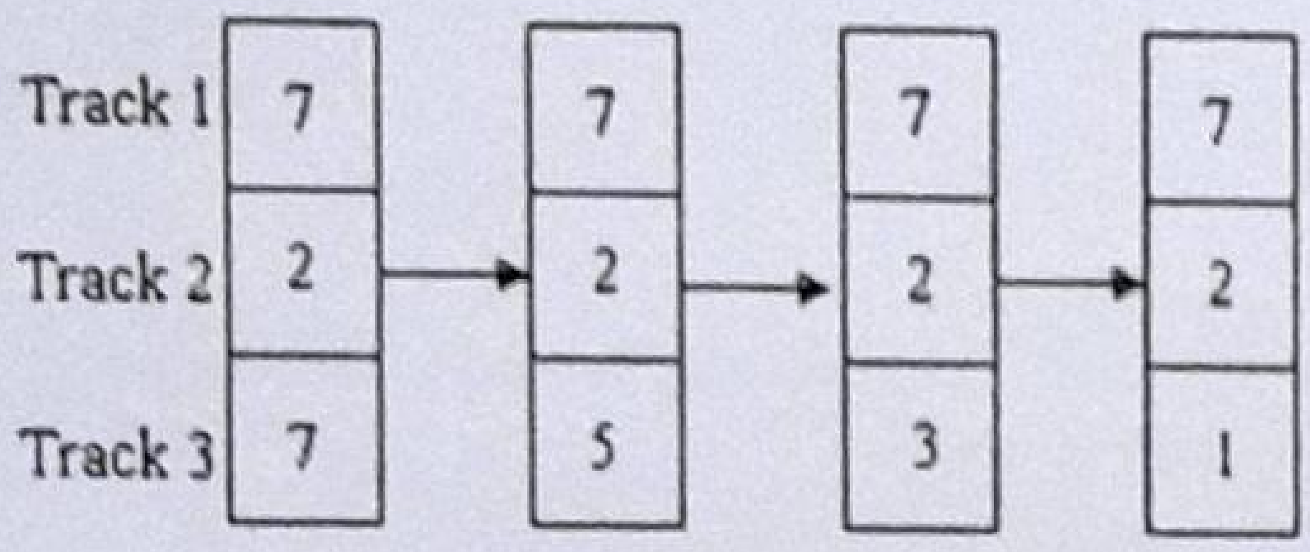
(i) Take the input ==> 10



Finally second track number and the third track number is equal.

∴ The given number is even.

(ii) Assume the input ==> 7



∴ The given number is odd.

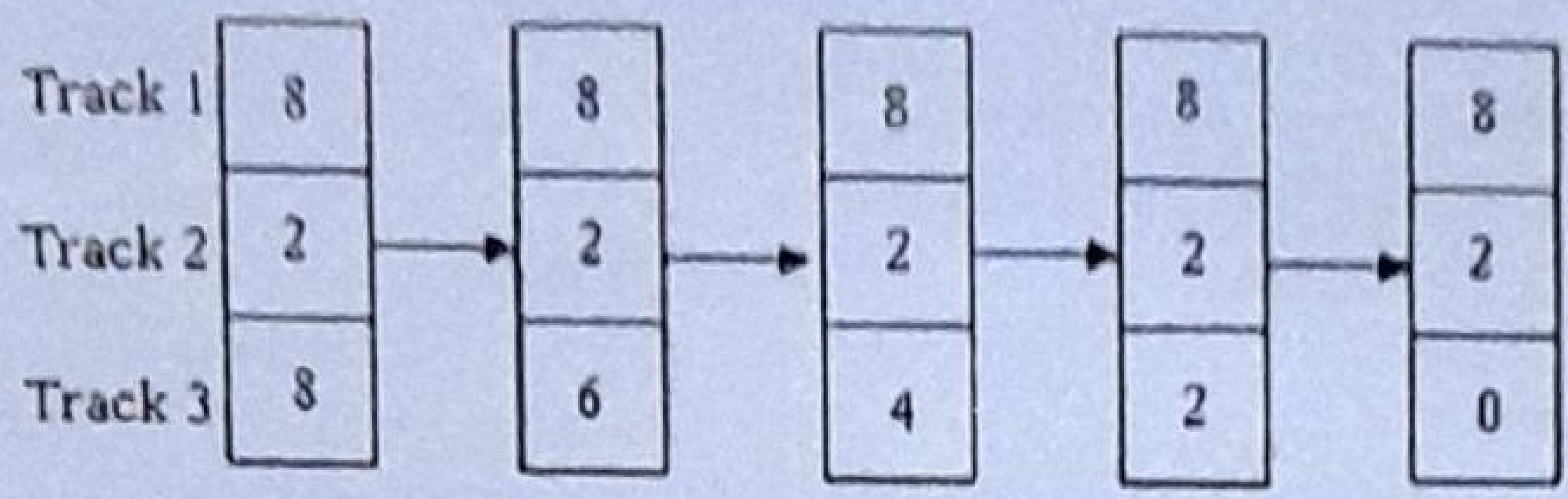
Ex.2: Design a TM that takes an input greater than 2 and checks whether the given input is prime or not.

Soln: **Procedure:**

(1) The input is placed into first track.
(2) The integer 2 is placed on the second track.
(3) The input on the first track is copied into third track.
(4) The number on the second track is subtracted from the third track.
(5) If the remainder is zero, then the number on the first track is not a prime.
(6) If the remainder is non – zero, then increase the number on the second track by one.
(7) If the second track equals the first track, then the given number is prime.

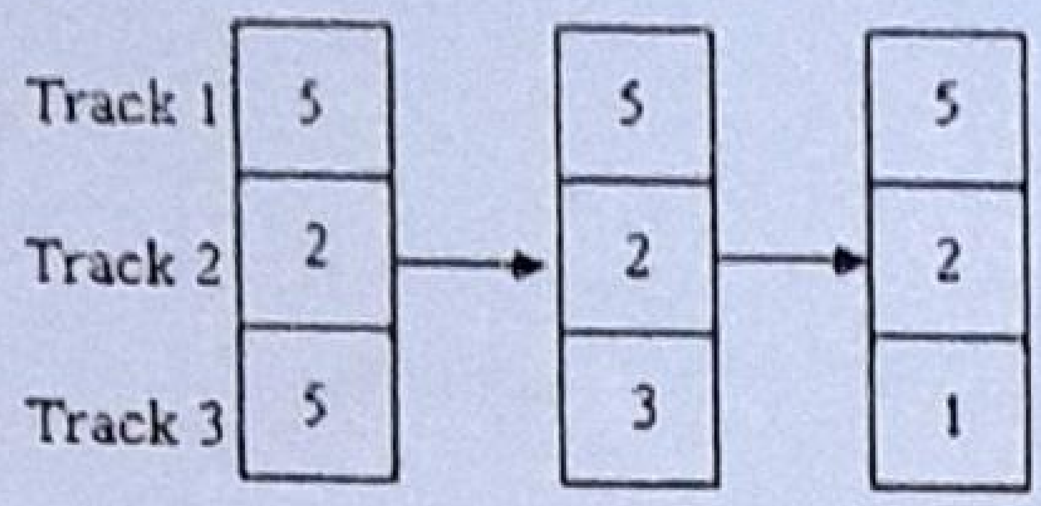
Scanned by TapScanner

(i) Assume the input ==> 8

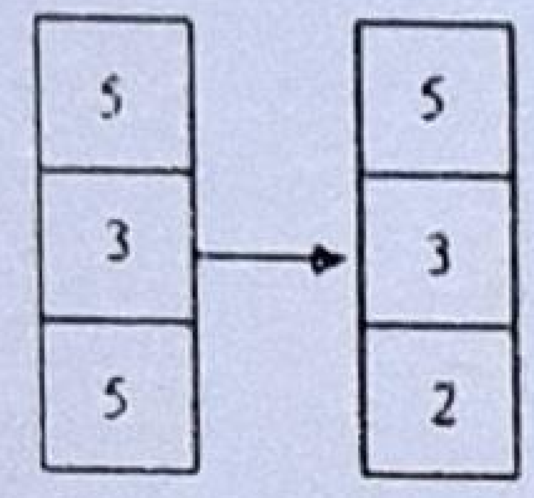


∴ The given number is not a prime number.

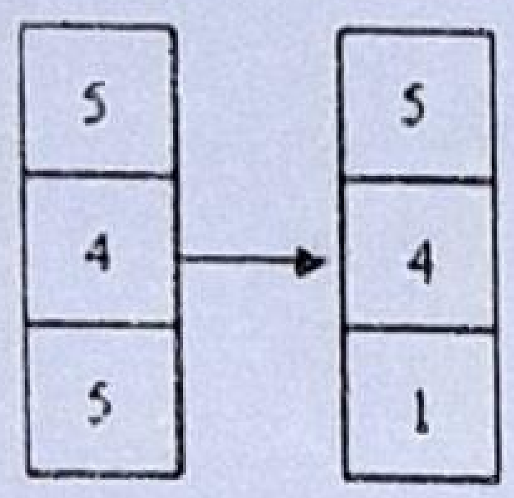
(ii) Assume the input ==> 5



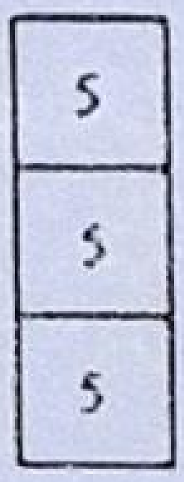
Increase the second track value by 1



Increase the second track value by 1



Increase the second track value by 1



Here the number on second track is equal to the number on the first track

∴ The given number is prime.

(3) <u>Subroutines:</u>

Subroutines are used in computer languages, which perform some task repeatedly. A turing machine can simulate any type of subroutine found in programming languages. A part of the TM program can be used as subroutine. This subroutine can be called for any number of times in the main TM program.

Ex: Design a TM to implement multiplication function, $f(m,n) = m*n$ **[Nov /Dec 2014, Apr/ May 2015]**

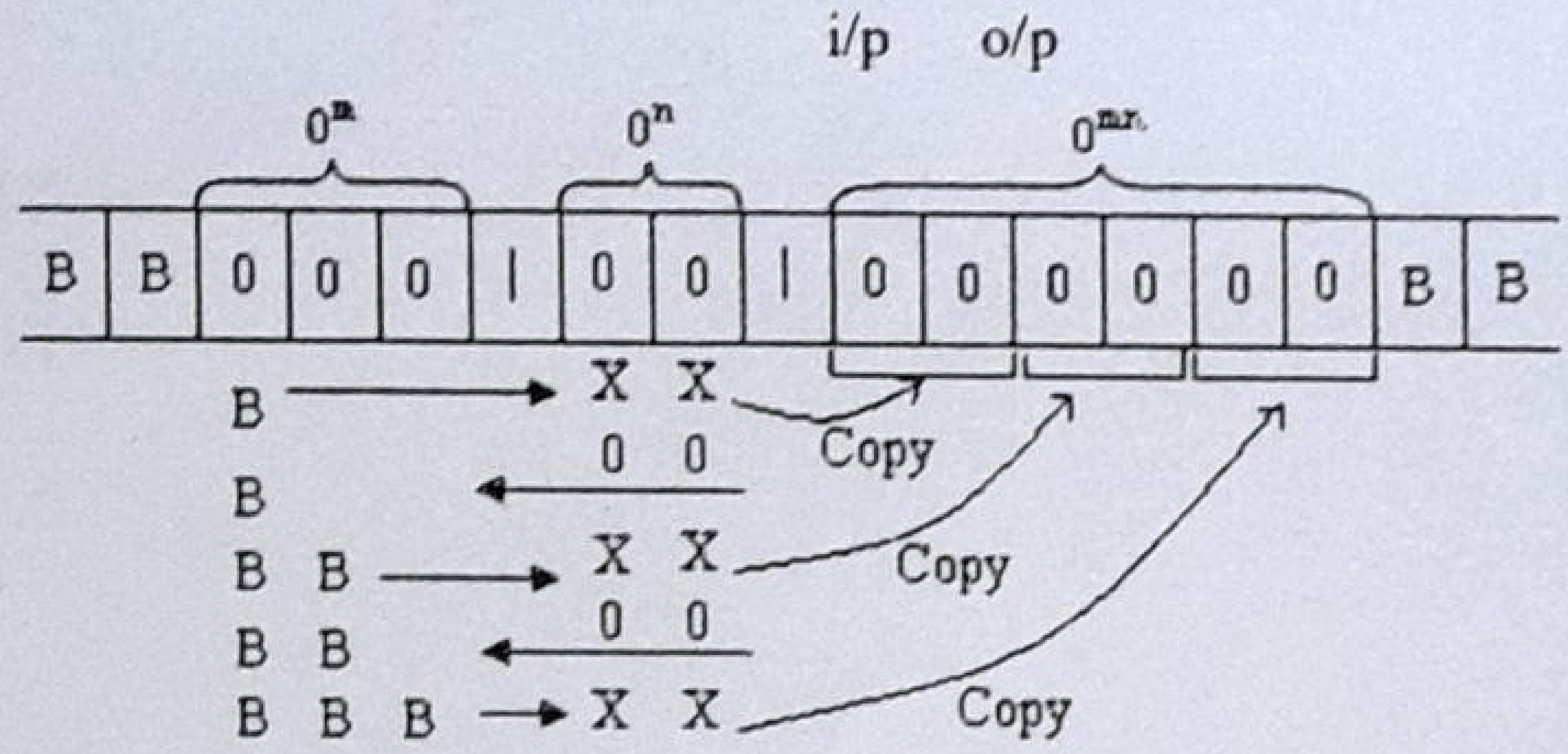
Soln: 'm' is given by 0^m

'n' is given by 0^n

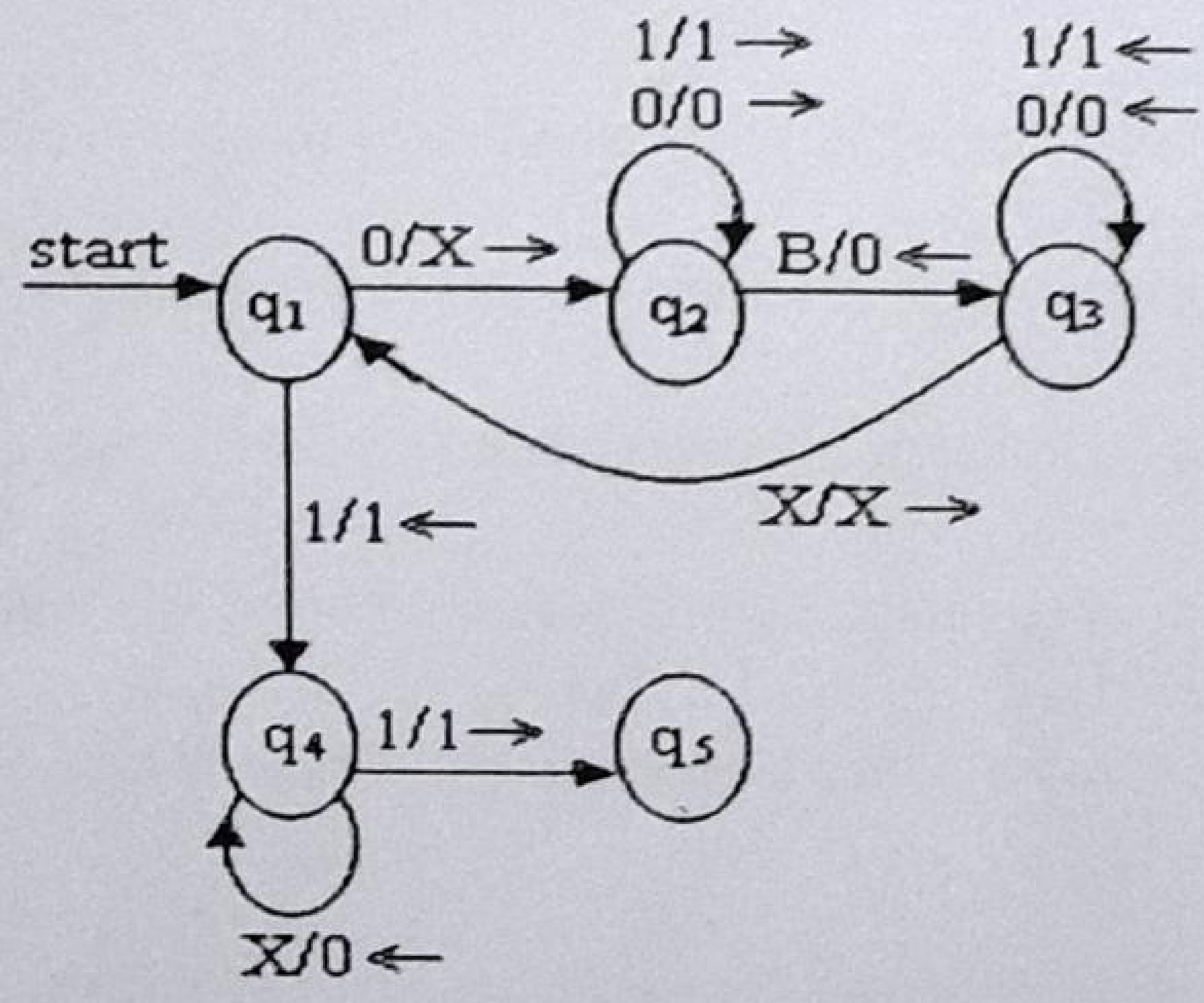
Input is: $o^m \mid o^n$

Output is: o^{m*n}

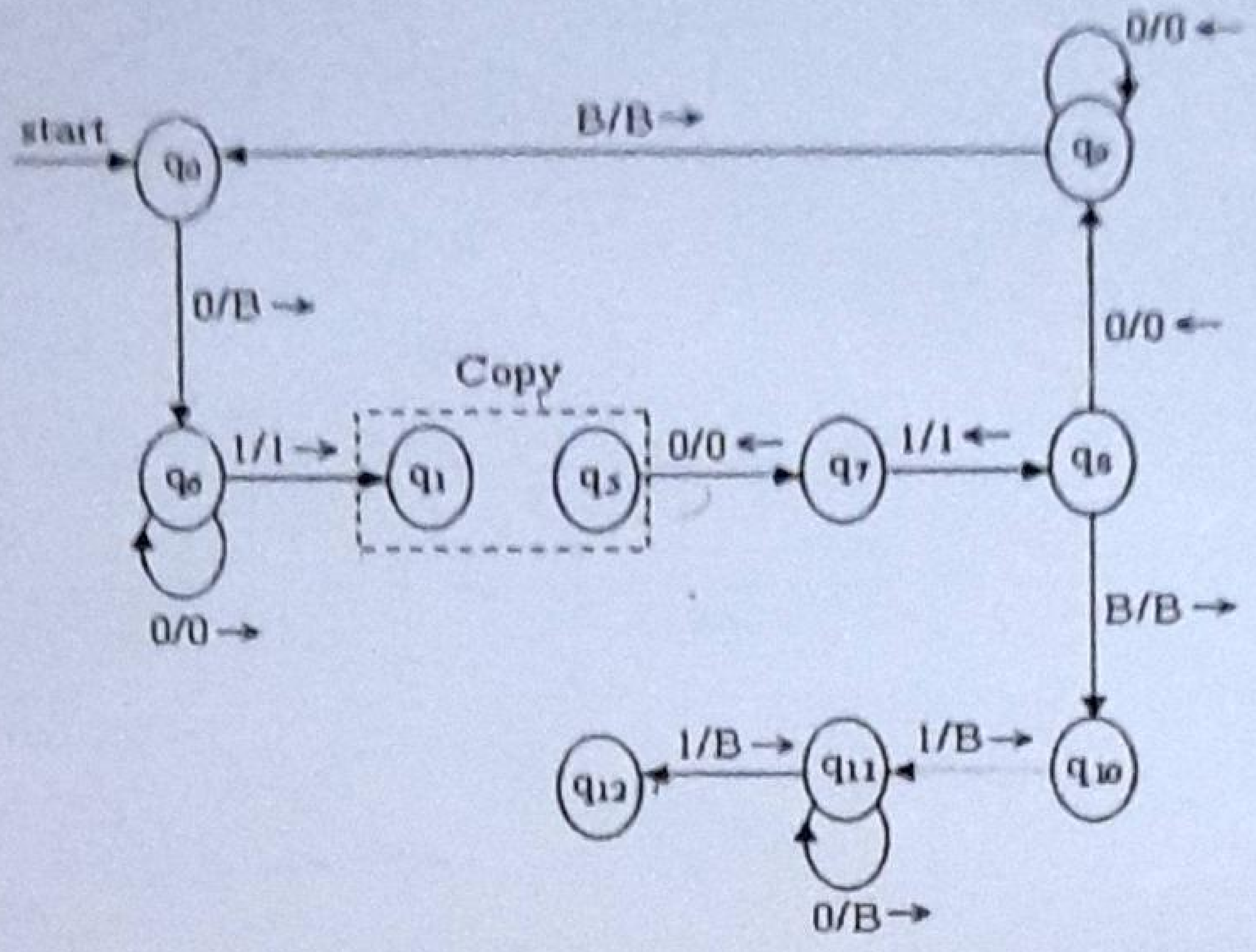
Input and output is placed into the tape, that is, $\underbrace{0^m \mid 0^n}\mid\underbrace{0^{mn}}$



The main concept is, it copy 'n' zero's 'm' times.



The Subroutine Copy



The Complete Multiplication Program Uses in Subroutine Copy

Assume m = 3, n = 2 then, input is $0^3|0^2$, that is placed on the input tape as,

000|00|BBBBBBB ⊢ B00|00|BBBBBBB ⊢ B00|00|BBBBBBB ⊢ B00|00|BBBBBBB ⊢ B00|00|BBBBBBB
↑q_0 ↑q_6 ↑q_6 ↑q_6 ↑q_1

⊢ B00|X0|BBBBBBB ⊢ B00|X0|BBBBBBB ⊢ B00|X0|BBBBBBB ⊢ B00|X0|0BBBBBB
↑q_2 ↑q_2 ↑q_2 ↑q_3

⊢ B00|X0|0BBBBBB ⊢ B00|X0|0BBBBBB ⊢ B00|X0|0BBBBBB ⊢ B00|XX|0BBBBBB
↑q_3 ↑q_3 ↑q_1 ↑q_2

⊢ B00|XX|0BBBBBB ⊢ B00|XX|0BBBBBB ⊢ B00|XX|00BBBBB ⊢ B00|XX|00BBBBB
↑q_2 ↑q_2 ↑q_3 ↑q_3

⊢ B00|XX|00BBBBB ⊢ B00|XX|00BBBBB ⊢ B00|XX|00BBBBB ⊢ B00|X0|00BBBBB
↑q_3 ↑q_1 ↑q_4 ↑q_4

⊢ B00|00|00BBBBB ⊢ B00|00|00BBBBB ⊢ B00|00|00BBBBB ⊢ B00|00|00BBBBB
↑q_4 ↑q_5 ↑q_7 ↑q_8

⊢ B00|00|00BBBBB ⊢ B00|00|00BBBBB ⊢ B00|00|00BBBBB ⊢ BB0|00|00BBBBB
↑q_9 ↑q_9 ↑q_0 ↑q_6

⊢ BB0|00|00BBBBB ⊢ BB0|00|00BBBBB ⊢ BB0|X0|00BBBBB ⊢ BB0|X0|00BBBBB
↑q_6 ↑q_1 ↑q_2 ↑q_2

⊢ BB0|X0|00BBBBB ⊢ BB0|X0|00BBBBB ⊢ BB0|X0|00BBBBB ⊢ BB0|X0|000BBBB
↑q_2 ↑q_2 ↑q_2 ↑q_3

⊢ BB0|X0|000BBBB ⊢ BB0|X0|000BBBB ⊢ BB0|X0|000BBBB ⊢ BB0|X0|000BBBB
↑q_3 ↑q_3 ↑q_3 ↑q_3

⊢ BB0|X0|000BBBB ⊢ BB0|XX|000BBBB ⊢ BB0|XX|000BBBB ⊢ BB0|XX|000BBBB
↑q_1 ↑q_2 ↑q_2 ↑q_2

⊢ BB0|XX|000BBBB ⊢ BB0|XX|000BBBB ⊢ BB0|XX|0000BBB ⊢ BB0|XX|0000BBB
↑q_2 ↑q_2 ↑q_3 ↑q_3

⊢ BB0|XX|0000BBB ⊢ BB0|XX|0000BBB ⊢ BB0|XX|0000BBB ⊢ BB0|XX|0000BBB

q_5 q_3 q_3 q_1

⊢ BB0|XX|0000BBB ⊢ BB0|X0|0000BBB ⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB

q_4 q_4 q_4 q_5

⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB ⊢ BB0|00|0000BBB

q_7 q_8 q_9 q_0

⊢ BBB|00|0000BBB ⊢ BBB|00|0000BBB ⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB

q_6 q_1 q_2 q_2

⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB ⊢ BBB|X0|0000BBB

q_2 q_2 q_2 q_2

⊢ BBB|X0|0000BBB ⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB

q_2 q_3 q_3 q_3

⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB ⊢ BBB|X0|00000BB

q_3 q_3 q_3 q_3

⊢ BBB|X0|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB

q_1 q_2 q_2 q_2

⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB ⊢ BBB|XX|00000BB

q_2 q_2 q_2 q_2

⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B

q_3 q_3 q_3 q_3

⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B ⊢ BBB|XX|000000B

q_3 q_3 q_3 q_1

⊢ BBB|XX|000000B ⊢ BBB|X0|000000B ⊢ BBB|00|000000B ⊢ BBB|00|000000B

q_4 q_4 q_4 q_5

⊢ BBB|00|000000B ⊢ BBB|00|000000B ⊢ BBB|00|000000B ⊢ BBBB00|000000B

q_7 q_8 q_{10} q_{11}

⊢ BBBBB0|000000B ⊢ BBBBBB|000000B ⊢ BBBBBBB000000B

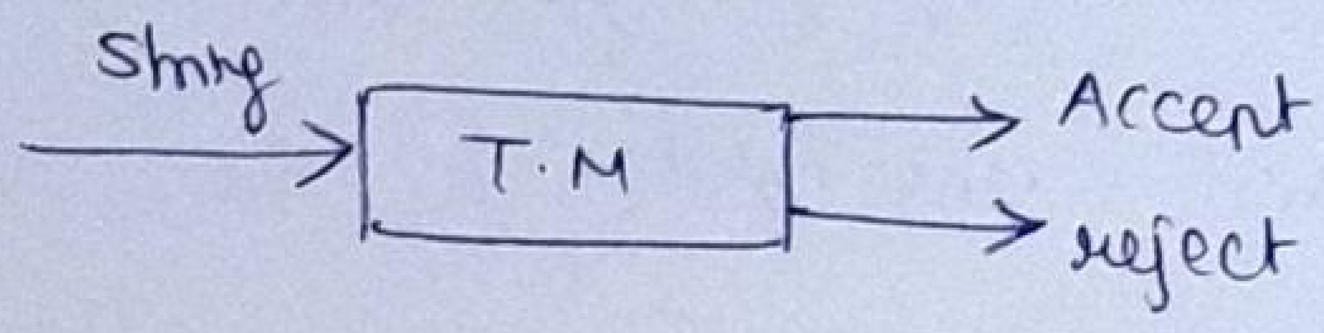
q_{11} q_{11} q_{12}

Unit-5

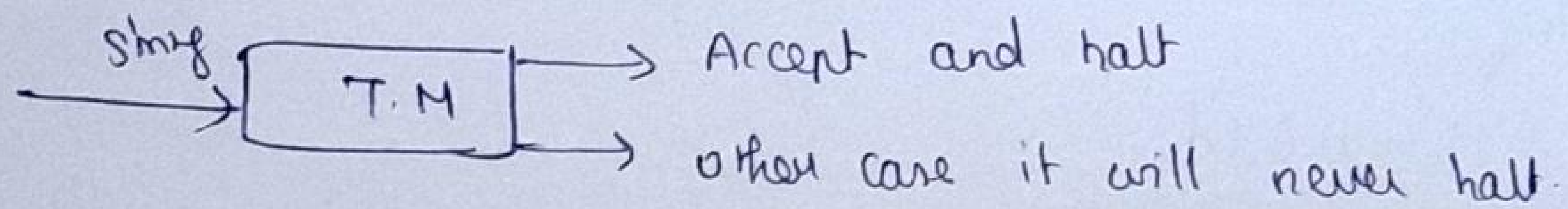
Undecidability

① Decidability & Undecidability:

Recursive language:



Recursive Enumerable language:



Decidable language:

Recursive language [accept, reject] [halt]

Partially decidable language:

Recursive Enumerable language. [Sometimes halt or not]

Undecidable language:

⇒ If it is not decidable then, it is undecidable.

⇒ Sometimes partially decidable language but not decidable.

⇒ If a language is not even partially deciable, then there exists no Turing machine for that language.

Diagnolization language Ld : a - non REL

$Ld = \{ w_i \in (0+1)^* \mid w_i \notin L(M_i) \}$

Such that : i : integer → $base_2$ → binary string → M_i (Machine)
↘ w_i (word)

Ld : it is a set of binary strings which are not accepted by a machine represented by the same string.

Is Ld a REL ?

Proof :

If $w \in Ld$

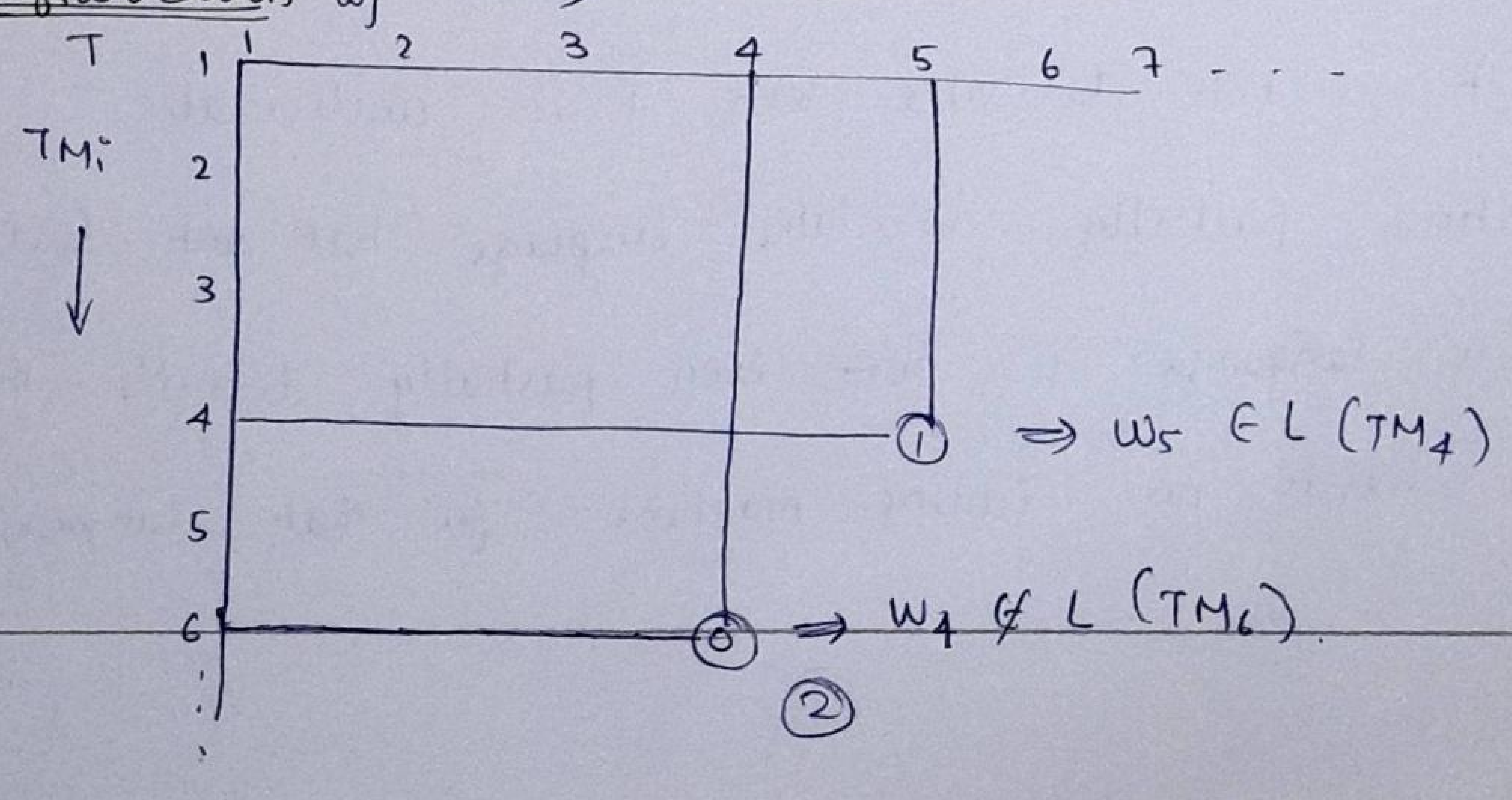
then $\exists$ TM M such that

$w \in L(M)$

$w \in Ld$ - Halt + accept

$w \notin Ld$ - Halt & Reject
↘ & loop

Diagnolization w_j → Binary matrix



<vector> $d = diag(T)$

1,1	2,2	3,3	4,4	.	.	.	i,i	
1	0	1					$\downarrow$	

1 if $w_i \in L(M_i)$

0 if $w_i \in L(M_i)$

→ d' is the complement of d

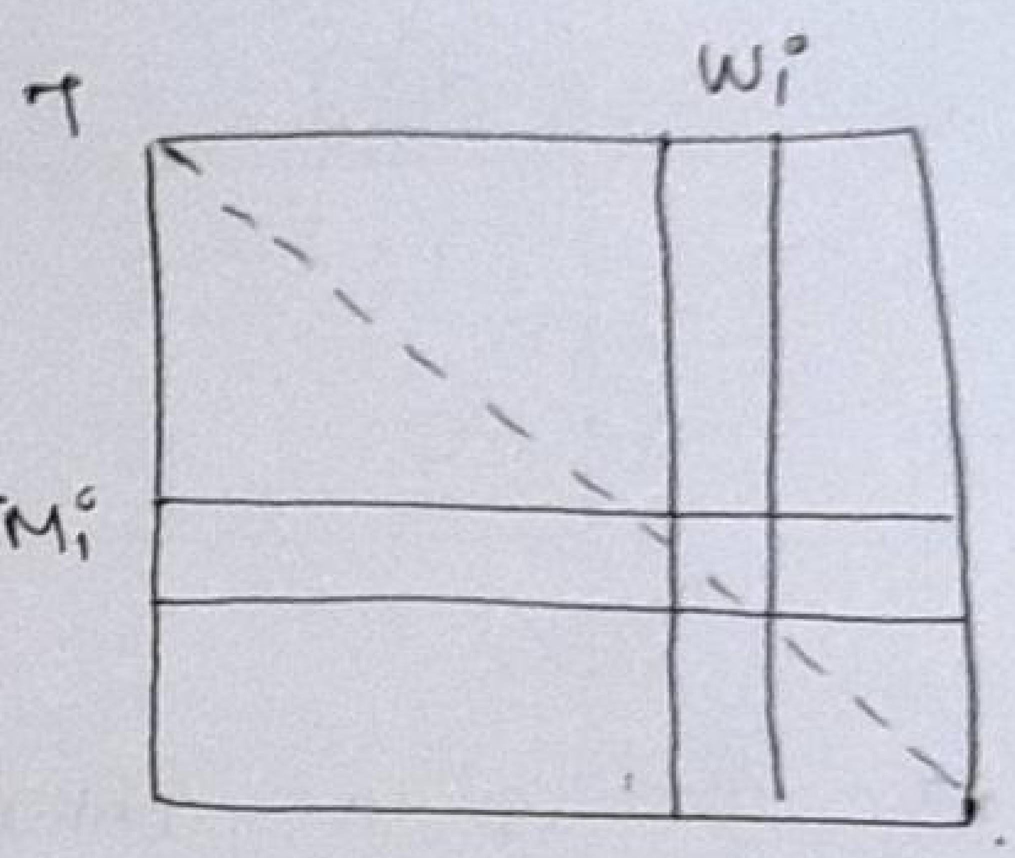
1	2	3				i	
$d' =$ 0	1	0					..

$d' =$

				i	
				$\downarrow$	

Ld $\cancel{=}$ 1 if $w_i \notin L(M_i)$ TM_i

0 if $w_i \in L(M_i)$



$d'[i] = 1 \Leftrightarrow w_i \notin L(M_i)$

$Ld = w_i / d'[i] = 1$

Is Ld a REL ?

⇔ Is there a TM that accepts

⇔ Is there a Row in T which is equal to d' vector.

proof: Lets Assume i^{th} row be the row in T which is equal to d' vector.

$\rightarrow T_i = d'$

$\rightarrow T_i[j] = d'[j] \ \forall \ j = 1, 2, 3 \ldots$

$\Rightarrow T_i[j] = (d[j])'$

$\rightarrow$ for $j = i$,

$\Rightarrow T_j[j] = d[j] = (d[j])'$

$d[j]$	$(d[j])'$
1	0
0	1

$\therefore$ It is a contradiction $\Rightarrow\Leftarrow$

$\therefore$ ~~#~~ Ld is a non - REL.

And the word Ld $\rightarrow$ does not have the TM.

And the 1^{st} language is Ld //

AN UNDECIDABLE PROBLEM WITH RE:

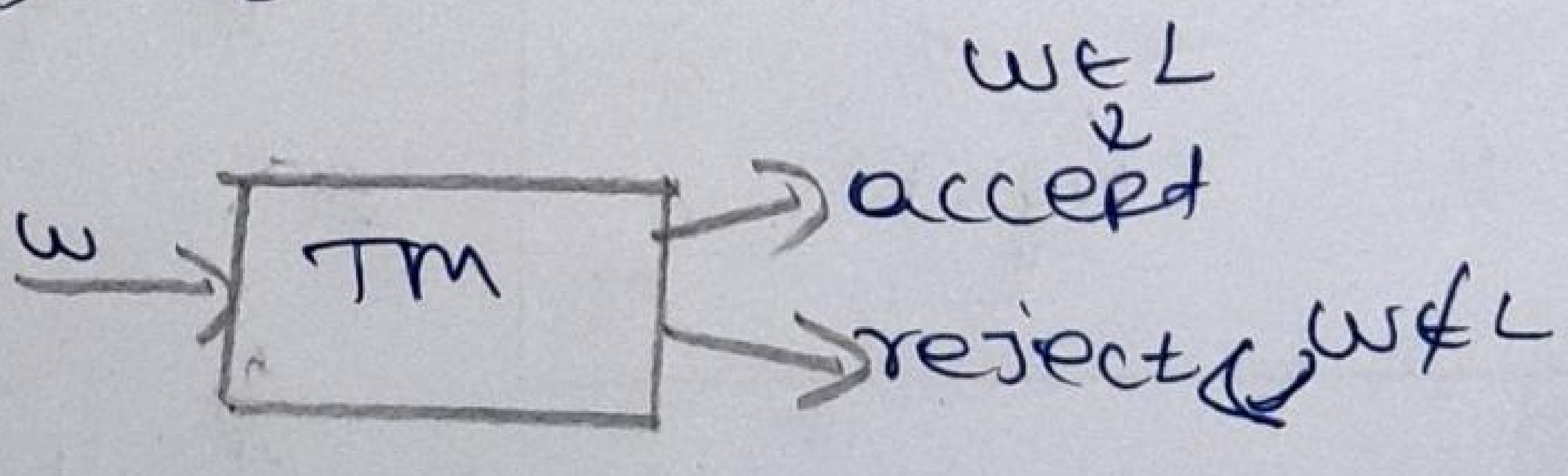
The recursively enumerable language has two categories.

(i) All the languages that has some algorithm and an algorithm for language L can be involved by a Turing machine. Turing Machine always halts on ropid input and enters in accept State, but on invalid input and thats not entering in access state thus languages of this Category, are particularly called a recursive languages.

(ii) All languages can be modeled by Turing Machine but three is no guarantee that the TM will eventually halts. In the Case that we cannot predict that turing Machine will halt (or) will enter in an infinite loop for certain input. Such type of languages can be denoted by pair(μ, w) where μ is a turing machine and w is an input String. These languages of these category are called as recursive enumerable languages.

Recursive language:

*) These is a TM for a languge which halt on every string:

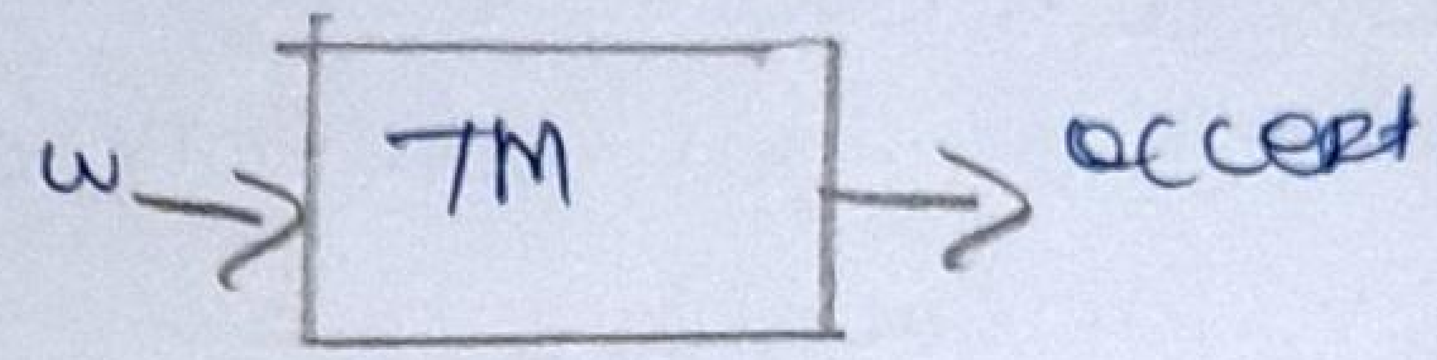


L={w {a,b} start with ab}

ab

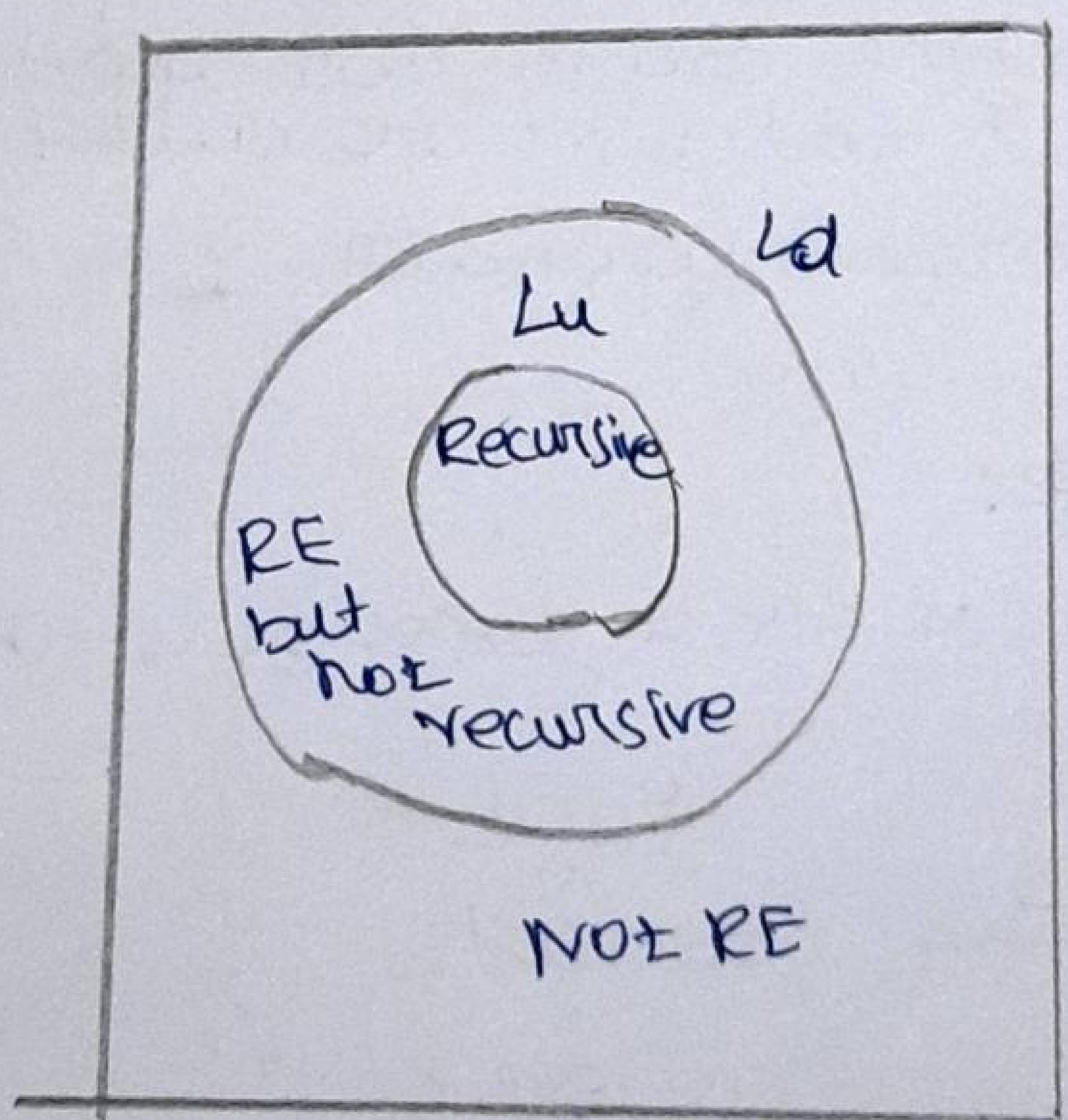
Recursively Enumerable language:

There is a T.M for a language which accepts Every String otherwise not.



$w \notin L \rightarrow \begin{cases} \text{reject} \\ \text{infinite loop.} \end{cases}$

Relationship between RE and non-RE Languages



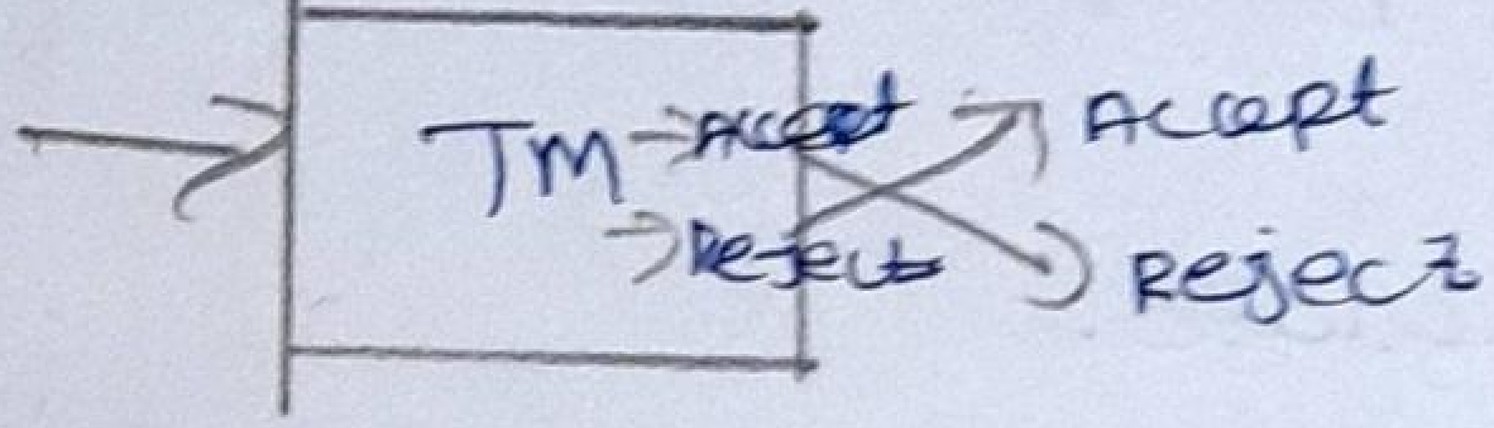
The above figure shows the relationship among three languages.

1. The recursive languages.

2. The language that are recursively enumerable but not recursive.

3. The non recursively enumerable languages.

Proof: Let $L = L(M)$ for some TM M that always halts. We construct a TM $\bar{M}$ such that $\bar{L} = L(\bar{M})$ by the construction



Then is, $\bar{M}$ behaves just like M. However, M is modified as follows to create $\bar{M}$.

(1) The accepting states of M are made non accepting states of $\bar{M}$ with no transitions. ie, in these states $\bar{M}$ will each halts without accepting.

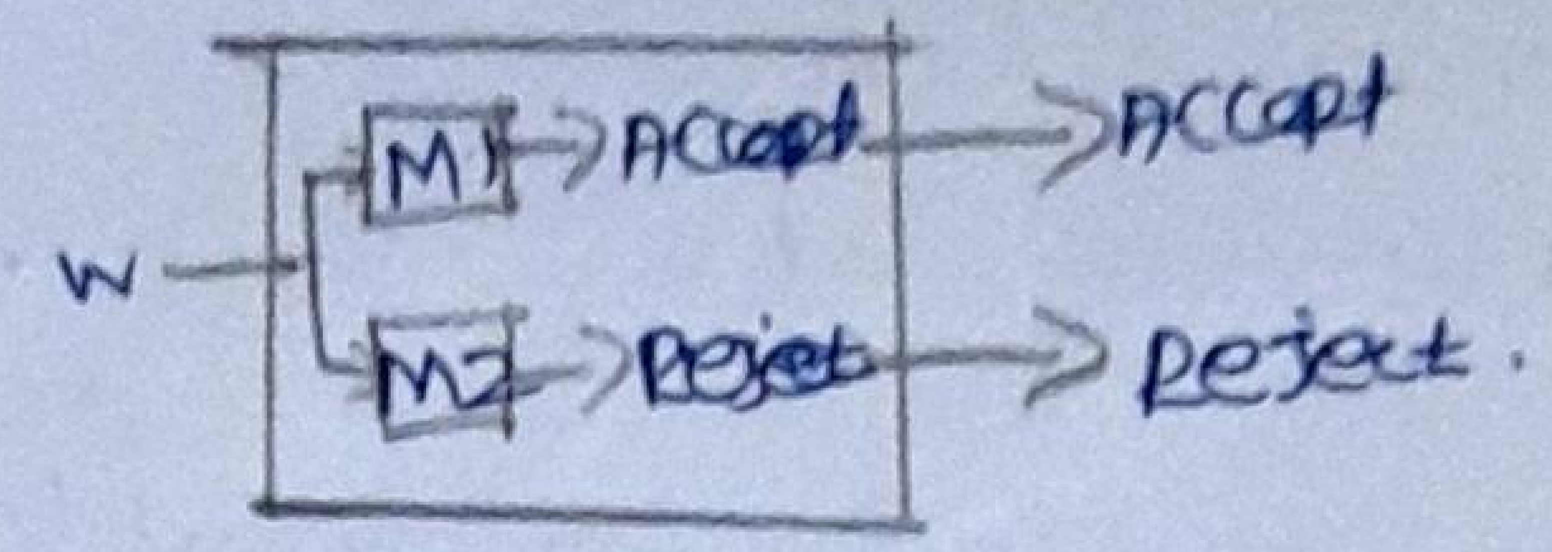
(2) $\bar{M}$ has a new accepting state 'r', there are no transitions from 'r'.

(3) For each combination of a non accepting state of M and a tape symbol of M such that M has no transition, add a transition to the accepting state 'r'.

Theorem: If L and $\bar{L}$ are recursively enumerable then L is recursive.

Proof: Let M1 be the L and M2 be the $\bar{L}$.

Construct the M1 and M2 simultaneously.



(1) Both L and $\bar{L}$ are recursive.

(2) Neither L nor $\bar{L}$ is recursive enumerable.

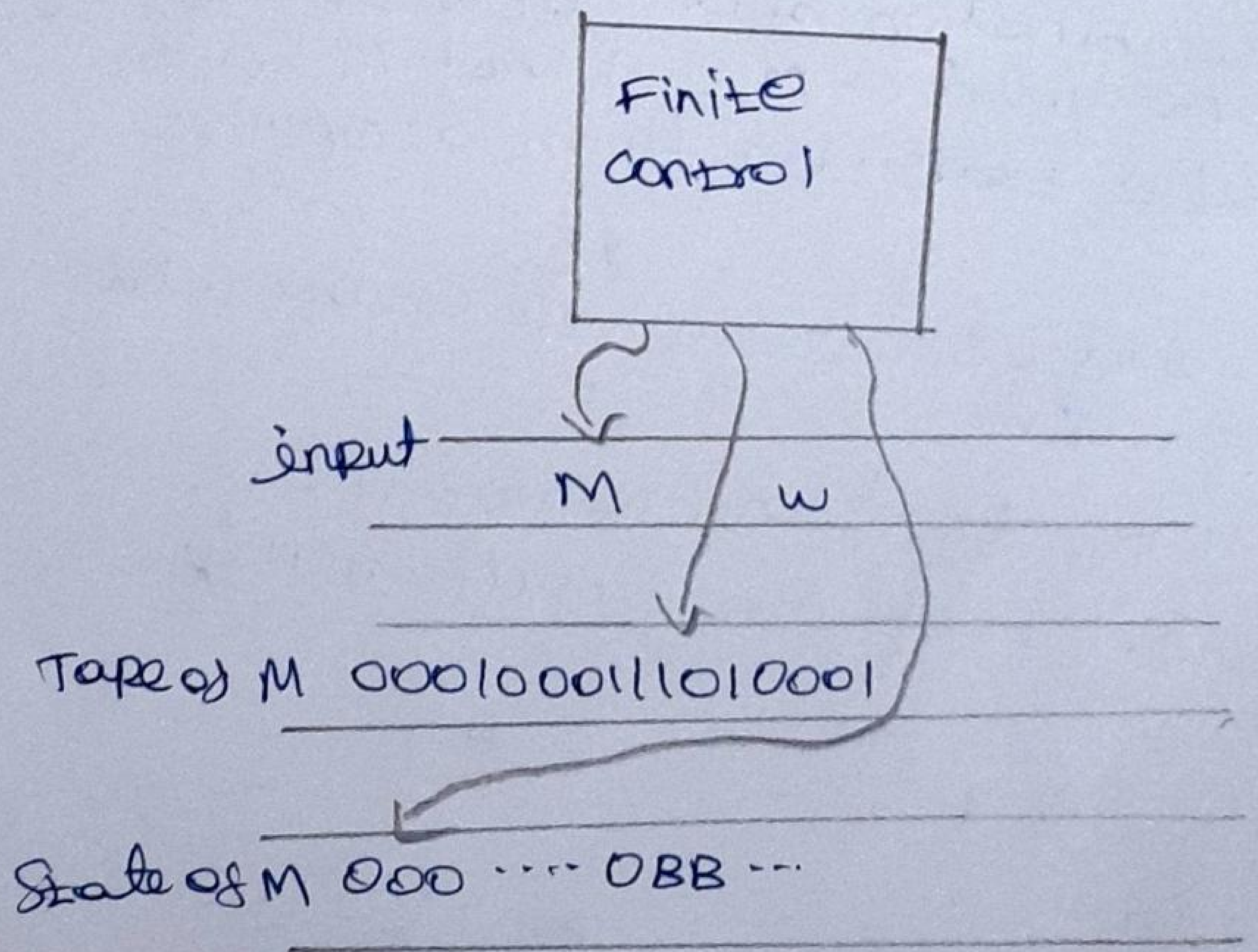
(3) one of the L is recursively enumerable and other is not recursively enumerable.

Universal Language:

The universal language L_u as a set of binary strings that is encoded in pair (M, w)

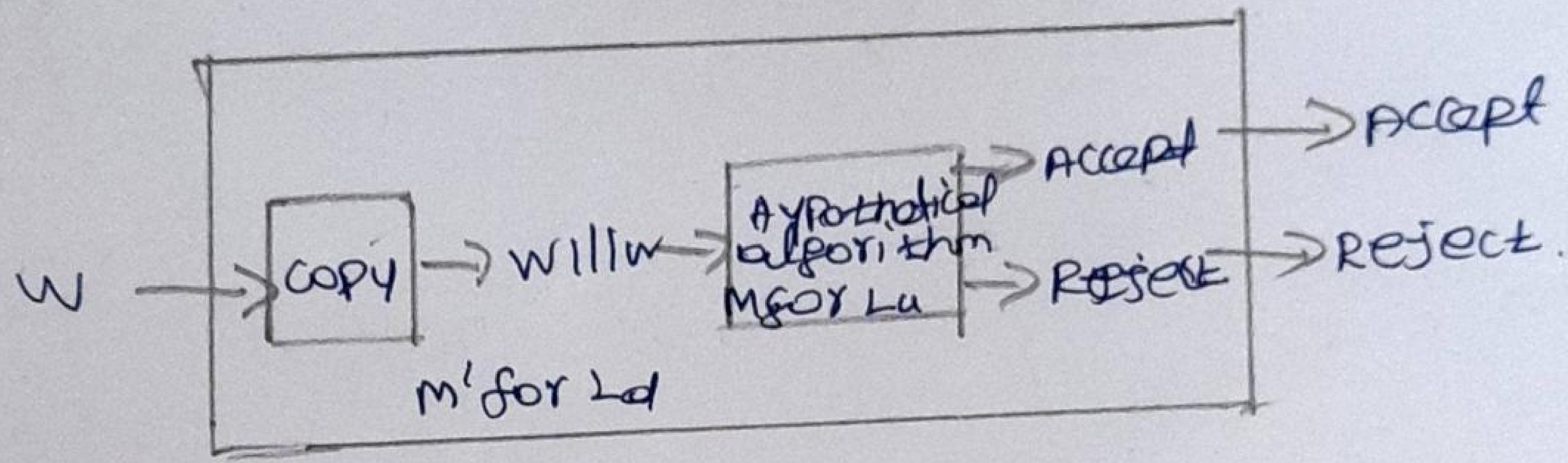
Prove that L_u is recursively enumerable but not recursive.

Proof: L_u is recursively enumerable



Prove: Lu is recursively enumerable but not recursive.

Proof by contradiction. Assume that is recursive. Hence, there exists a TM in the following format



Undecidable problem about TM

Turing Machines that Accepts the Empty language:

In this, we are using two languages, called L_e and L_{ne}. Each consist of binary strings. If 'w' is a binary string, then it represent some TM, M_i.

If $L(M_i) = \Phi$, that is, M_i does not Accept any input, then 'w' is in L_e. Thus, L_e is the Language Consisting of all those encoded TM's whose language is empty. On the other hand, if $L(M_i)$ is not the empty Language, then 'w' is in L_{ne}. Thus, L_{ne} is the Language of all Codes for Turing Machines that accept at least one input string. Define the two languages as,

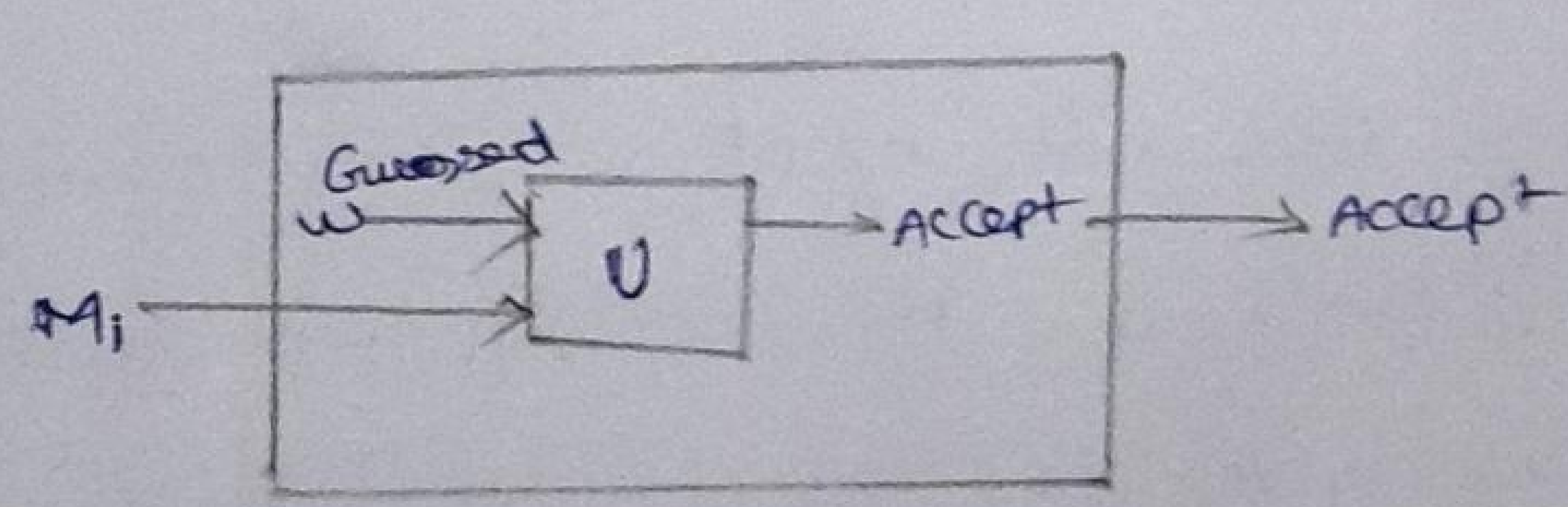
① $L_e = \{M \mid L(M) = \Phi\}$

② $L_{ne} = \{M \mid L(M) \neq \Phi\}$

Theorem: L_{ne} is recursively Enumerable

Proof:

In this, a TM that Accepts L_{ne}. It is easiest to describe a non deterministic TM M.



Operations:

① M takes as input a TM code M_i.

② Using its nondeterministic capability, M guesses an input 'w', that M_i might accept.

③ M test whether M_i accepts 'w'. For this part, M can simulate the universal TM U that accepts L_u.

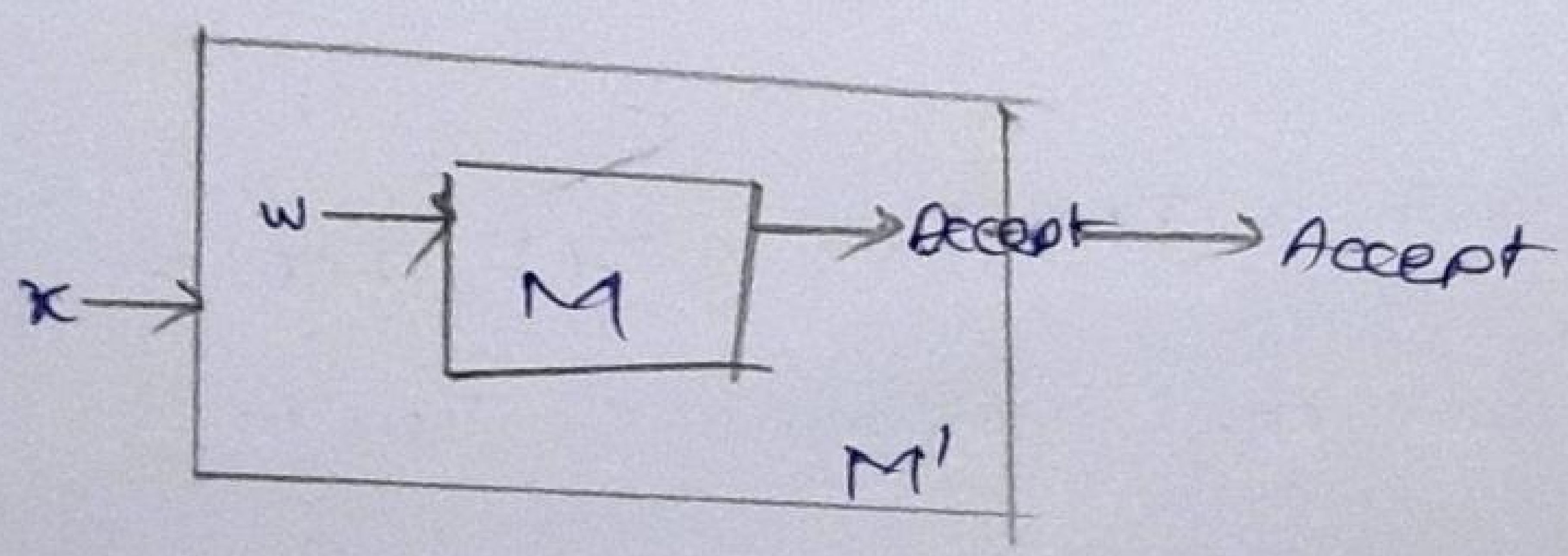
④ If M_i accepts 'w', then M accepts its own input, which is M_i.

If M_i accepts even one string M will guess that string and accept M_i. Howeverver, if $L(M_i) = \phi$, then no guess 'w' leads to acceptance by M_i, so M does not accept M_i. Thus $L(M) = L_{ne}$.

Theorem: L_{ne} is not recursive.

Proof:

In this, we must design an algorithm that converts an input that is a binary coded pair (M, w) into a TM M' such that $L(M') \neq \phi$ if and only if M accepts input 'w'. The construction of M' is shown in Diagram.



If M does not accept 'w', then M' accept none of its input's. ie, $L(M') = \phi$. However, if M accepts w. then M' accept every input, and thus $L(M')$ surely is not ϕ. M' is designed to do the following.

① M' ignores its own input 'x'. Rather it replaces its input by the String that represent TM M and input String 'w'. Since M' is designed for a specific pair (M, w) which has some length n, we may construct M' to have a sequence of States $q_0, q_1, \dots q_n$. where q_0 is the start State.

ⓐ In State q_i, for $i = 0, 1 \dots n-1$, M' writes the $(i+1)$ bit of the code for (M, w) goes to State q_{i+1}, and moves right.

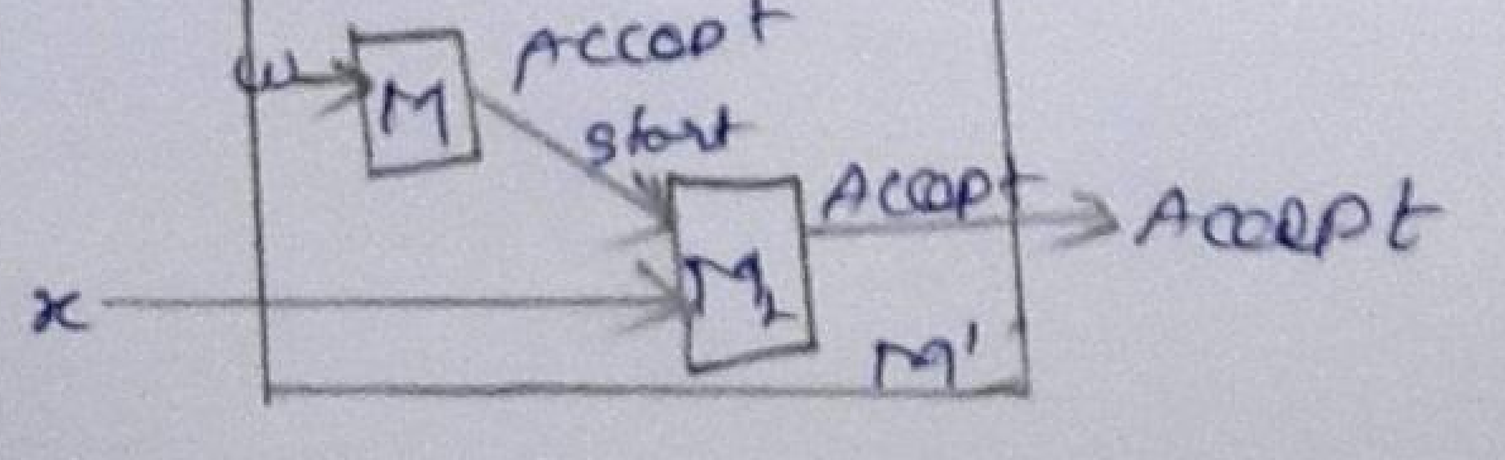
ⓑ In State q_n M' moves right, if necessary replacing any nonblanks by blanks.

② when M' reaches a blank in State q_n, it uses a similar collection of States to reposition its head at the left end of the tape.

③ Now, using additional State, M' simulates a universal TM U on its present tape.

④ If U accepts, then M' accepts. If U never accepts, then M' never accepts either.

<u>Rice Theorem:</u>



* Every non-trivial properties of REL is undecidable.

<u>Def</u>

• If P is a non-trivial property, and the language holding the property, L_P, is recognised by TM M, then $L_P = \{M \mid L(M) \in P\}$

is decidable.

* Property of languages, P, is simply a set of languages. If any language belongs to P ($L \in P$), it is said that L satisfies the property P.

<u>Properties</u>

① Trivial, if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages (REL).

② Non-Trivial, it is satisfied by some REL and are not satisfied by other.

* <u>Property 1</u> - There exists Turing Machines, M_1 and M_2 that ~~re~~recognized the same language, ie, either ($M_1, M_2 \in L$) or ($M_1, M_2 \notin L$).

* <u>Property 2</u> - There exists Turing Machines, M_1 and M_2, that recogniz the ~~same~~ ~~L~~ where M_1 recognizes the language while M_2 does not, ie, $M_1 \in L$ and $M_2 \notin L$.

The Post Correspondence Problem

PCP is an undecidable problem. That was introduced by Emil post in 1946

Two sequence of strings

$A = w_1 w_2, w_3 \ldots w_n$

$B = z_1, z_2, z_3 \ldots z_n$ over Σ

Instance of PCP has solution. if there is any sequence of integers $i, i_2 \ldots i_m$, $m \geq 1$

such that $w_i, w_{i_2}, w_{i_3} \ldots w_{i_m} = x_i, x_{i_2} \ldots x_{im}$

Dominos:

$\frac{B}{CA}$ $\frac{A}{AB}$ $\frac{CA}{A}$ $\frac{ABC}{C}$

we need to find a sequence of dominos such that the top and bottom strings are the same.

$\frac{A}{AB}$	$\frac{B}{CA}$	$\frac{CA}{A}$	$\frac{A}{AB}$	$\frac{ABC}{C}$

Ex: Tables

	A	B	
1	1	111	$\longrightarrow \frac{1}{111}$
2	10111	10	$\longrightarrow \frac{10111}{10}$
3	10	0	$\longrightarrow \frac{10}{0}$

A: 10111 1 1 10

B: 10 111 111 0

Proof:

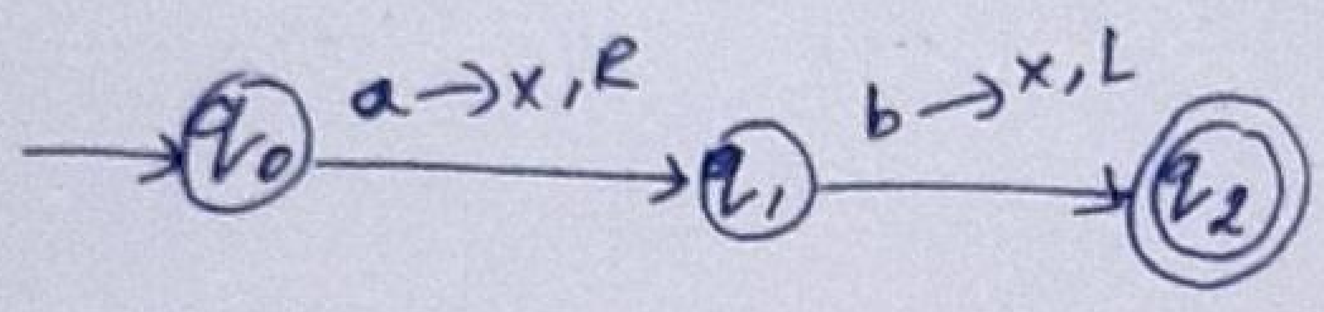
Take a problem that is already proven to be undecidable. Try to convert it to PCP

If we can successfully convert it to an equivalent PCP Then we prove the PCP is undecidable.

undecidable PCP

$\Downarrow$

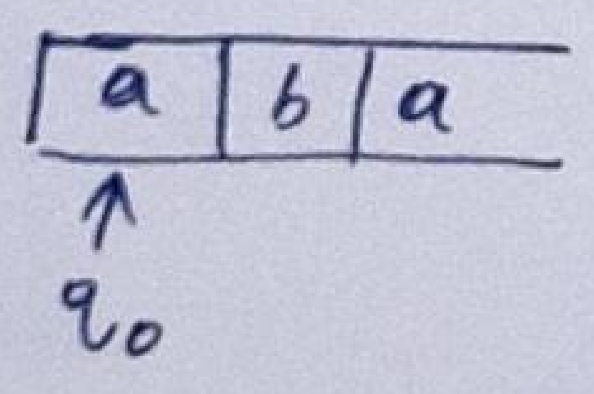
Modified PCP - MPCP



$\Sigma = \{a, b\}$ $\quad \Gamma = \{a, b, x, B\}$

i/p $\to$ $w = aba$

Step 1:

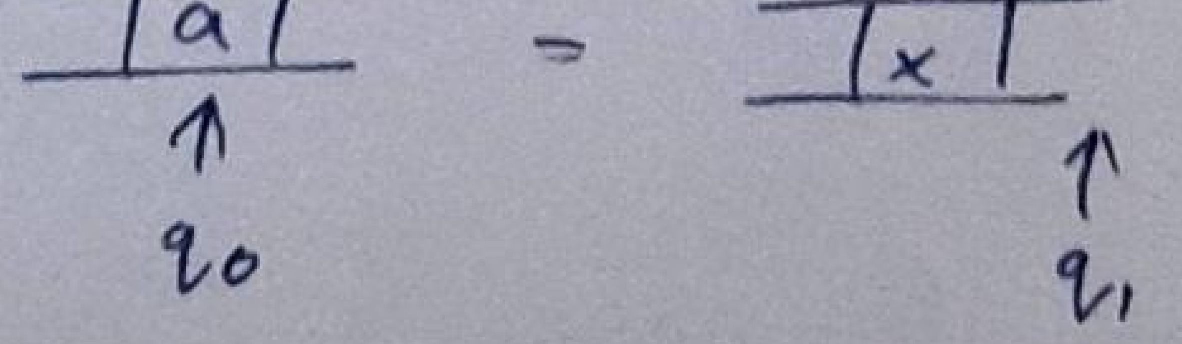


$q_0 aba \Rightarrow \left[\dfrac{\#}{\# q_0 aba \#}\right]$

Step 2:

Making dominos for a Right transition

$\delta(q_0, a) = (q_1, x, R)$

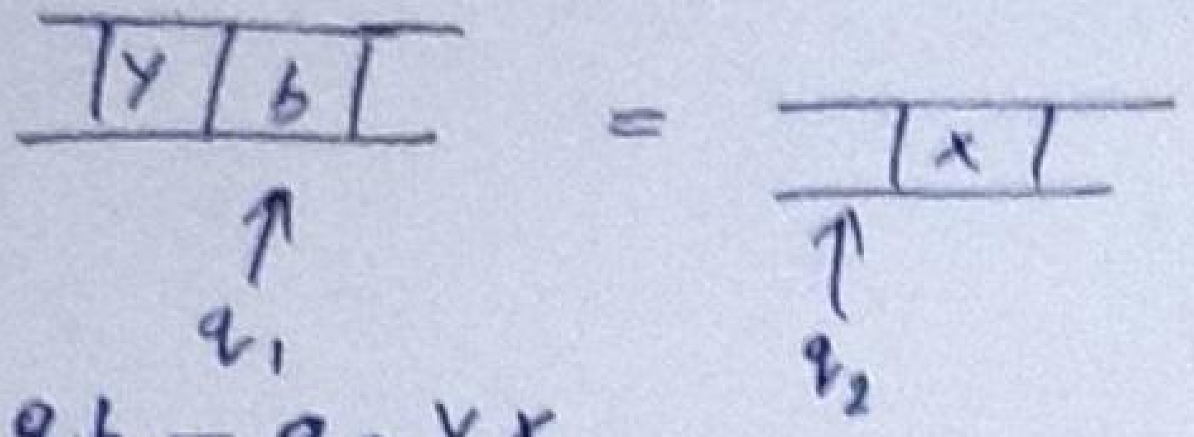


$q_0 a = x q_1 \implies \left[\dfrac{q_0 a}{x q_1}\right]$

Step 3:

Making dominos for Left transition

$\delta(q_1, b) = (q_2, x, L)$



$Y \in \Gamma$

$Y q_1 b = q_2 Y x$

$\left[\dfrac{a q_1 b}{q_2 a x}\right], \left[\dfrac{b\, q_1 b}{q_2\, b\, x}\right], \left[\dfrac{x\, q_1 b}{q_2\, x x}\right], \left[\dfrac{B\, q_1 b}{q_2\, B x}\right]$

Step 4

dominos for all possible tape symbols

$\Gamma = \{a, b, x, B\}$

$\left[\dfrac{a}{a}\right], \left[\dfrac{b}{b}\right], \left[\dfrac{x}{x}\right], \left[\dfrac{B}{B}\right]$

Step 5:

For all possible tape symbols after reaching the accepting state

$\left[\dfrac{a q_2}{q_2}\right], \left[\dfrac{q_2 a}{q_2}\right] \left[\dfrac{b q_2}{q_2}\right] \left[\dfrac{q_2 b}{q_2}\right] \left[\dfrac{x q_2}{q_2}\right] \left[\dfrac{q_2 x}{q_2}\right] \left[\dfrac{B q_2}{q_2}\right] \left[\dfrac{q_2 B}{q_2}\right]$

Step 6:

Dominos for the blank and # symbols

$\left[\dfrac{\#}{\#}\right] \quad \left[\dfrac{\#}{B\#}\right]$

Scanned by TapScanner

Step 7:

$$\left[\frac{q_2\#\#}{\#}\right]$$

Solution:

$$\left[\frac{\#}{\#q_0aba\#}\right]\left[\frac{q_0a}{xq_1}\right]\left[\frac{b}{b}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#xq_1ba\#}\right]\left[\frac{xq_1b}{q_2xx}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#q_2xxa\#}\right]\left[\frac{q_2x}{q_2}\right]\left[\frac{x}{x}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#q_2xa\#}\right]\left[\frac{q_2x}{q_2}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#q_2a\#}\right]\left[\frac{q_2a}{q_2}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#q_2\#}\right]\left[\frac{q_2\#\#}{\#}\right]$$

$\therefore$ pcp is undecidable