INDEX

NAME _____ SUBJECT _____

STD. _____ DIV. _____ ROLL NO. _____ SCHOOL _____

Finite Automata

Unit - 1

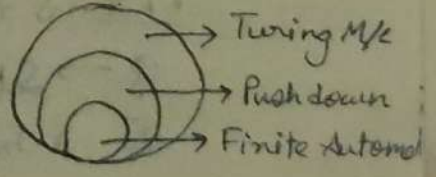
★) Topics:

1.1) Automata Theory:

- A machine that is relatively self operating.
- Automata: Developing a machine.
- Computability: where a machine can accept the input or not.
- Complexity: Every problem will have n. no. of solutions which has to choose.

1.1.1) Types of Automata:

- Finite Automata → Non deterministic F.A (NFA)
  → Deterministic FA (DFA)
- Pushdown automata.
- Turing machine.



1.2) Representation of Automata:

- **Symbols:** A user-defined entity
- **Alphabet:** A finite set of symbols denoted by Σ
- **String:** A sequence of symbols of finite length.
- **Language:** It is a set of strings of symbols drawn from a finite alphabet. A formal language can be specified either by a set of rules (such as regular expression or context free grammar) that generates the language or by a formal machine that accepts the language.

- **Kleene Star:** (Σ^*) It is a unary operator on a set of symbols or strings, that gives the infinite set of all possible strings of all possible length including ε

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

eg: $\{"ab", "c"\}^* = \{\varepsilon, "ab", "c", "abab", "abc", "cab", "cc", "ababab" \ldots\}$

- **Kleene closure:** Same as Kleene Star but excluding the empty set or Σ^0

$$\Sigma^+ = \Sigma^* - \Sigma^0 = \Sigma^1 \cup \Sigma^2 \cup \ldots \infty$$

eg: $\{"ab", "c"\}^+ = \{"ab", "c", "abab", "abc", "cab", "cc", "ababab"\}$

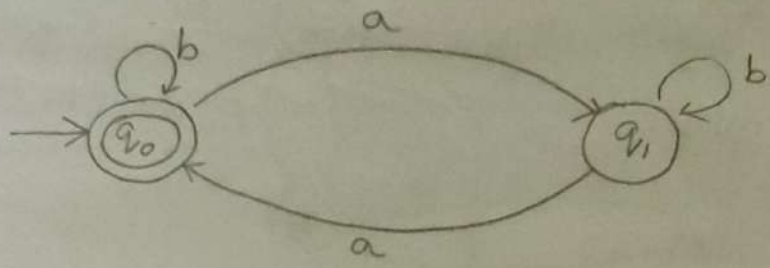
- **FA:** 5 tuples $\rightarrow \{Q, \Sigma, \delta, q_0, F\}$

$Q \rightarrow$ Set of all states, $\delta \rightarrow$ Transition function
$\Sigma \rightarrow$ Input symbols $q_0 \rightarrow$ Initial state
$F \rightarrow$ Final states

1.3) Designing F.A:

Q. Design a machine which accept even no. of a's. i.e inputs aab, aba, baa, abaa etc.

A:



$Q = \{q_0, q_1\}$ $\quad \Sigma = \{a, b\}$

$\delta =$

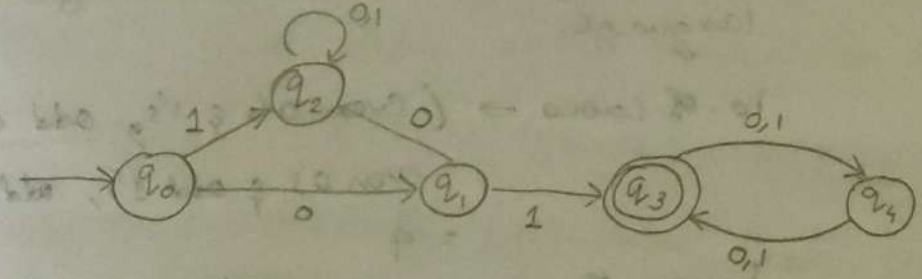
	q_0	q_1
a	q_1	q_0
b	q_0	q_1

$F = \{q_0\}$

$q_0 = \{q_0\}$

Q. Design a DFA which starts with 0,1 & it should have even no. of length. check for 0110

A:



$Q = \{q_0, q_1, q_2, q_3, q_4\}$ $\quad \Sigma = \{0, 1\}$

$\delta =$

	q_0	q_1	q_2	q_3	q_4
0	q_1	q_2	q_2	q_4	q_3
1	q_2	q_3	q_2	q_4	q_3

$F = \{q_3\}$

$q_0 = q_0$

To check weather 0110 is accepted

$\delta(q_0, \varepsilon) = q_0$

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

$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1) = \delta(q_1, 1) = q_3$

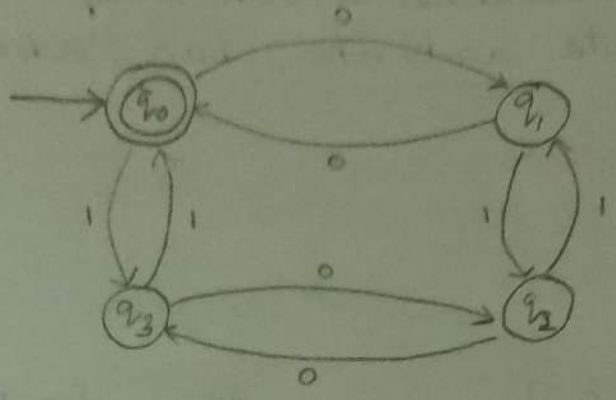
$\delta(q_0, 011) = \delta(\delta(q_0, 0), 11) = \delta(\delta(q_1, 1), 1) = \delta(q_3, 1) = q_4$

$\delta(q_0, 0110) = \delta(\delta(q_0, 0), 110) = \delta(\delta(q_1, 1), 10) = \delta(\delta(q_3, 1), 0) = \delta(q_4, 0)$

$= q_3$ //

★Q. Design a F.A that accepts even no. of 0's & 1's.

A:



$Q = \{q_0, q_1, q_2, q_3\}$ $\quad \Sigma = \{0, 1\}$

$F = \{q_0\}$

$q_0 = q_0$

δ	q_0	q_1	q_2	q_3
0	q_1	q_0	q_3	q_2
1	q_3	q_2	q_1	q_0

Note: changing the Final State changes the language.

no. of cases → (even 0's & 1's, odd 0's & 1's, even 0's & odd 1's, odd 0's & even 1's)

= 4

hence there are 4 states.

Final states → q_0 → even 0's & 1's

q_2 → odd 0's & 1's

q_1 → odd 0's & even 1's

q_3 → even 0's & odd 1's.

Q. Write

1.4) Mathematical Inductive proof

Q. Prove that $1+4+7+\ldots(3n-2) = \frac{n(3n-1)}{2}$

A: Step-1: Find value with $n=1$ on both side

$n=1$,

LHS $= 1$ | RHS $= \frac{1(3(1)-1)}{2} = 1$

$\therefore$ LHS = RHS

Step-2: Find value with $n=k$ on both side

$n=k$,

$$1+4+7+\ldots 3k-2 = \frac{k(3k-1)}{2}$$

Step-3: Find value with $n=k+1$ on both side.

$$1+4+7+\ldots+(3k-2)+3(k+1)-2 = \frac{k(3(k+1)-1)}{2}$$

Step-4: Substitute the LHS value from step-2 & simplify to prove.

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

$$\frac{k(3k-1)+6(k+1)-4}{2} = \frac{k(3k+2)}{2}$$

$$\frac{3k^2+5k+2}{2} = \frac{3k^2+5k+2}{2}$$

Hence proved.

Q. P.T. $1^2+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6}$

A: Let $n=1$

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

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

$$\Rightarrow 1 = \frac{6}{6}$$

$$\Rightarrow 1 = 1$$

Let $n = k$

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

Let $n = k+1$

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

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

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

Hence proved.

Note: Deductive proof:

hypothesis $2^x \geq x^2$

Sequence of steps:

$x=1 : 2 \geq 1$

$x=2 : 4 \geq 4$

$x=3 : 8 \geq 9$ X

$x = 1, 2$

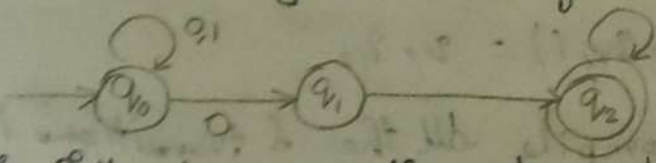
Debunked

1.5) NFA Vs DFA

DFA	NFA
1) Empty transaction is not possible	1) Empty transaction is possible
2) Dead state is possible	2) Deadstate is not possible
3) Difficult to construct	3) Construction is very easy
4) Every state with input will have a transaction to single state.	4) Every state with input will have transactions to multiple states.

1.6) NFA to DFA Conversion

The following is the given NFA



The following are the steps to convert the given NFA to DFA:

Step-1: Create the transition table

	0	1
q_0	$q_0 q_1$	q_0
q_1	ϕ	q_2
q_2	q_2	q_2

Step-2: Start at $\delta(q_0, 0)$ & $\delta(q_0, 1)$ Find the transitions, any new states which show up should be found next, on getting two states, say $q_1 q_2$ let $q_1 q_2$ be the new state.

$\delta(q_0, 0) = q_0 q_1$ (New State)

$\delta(q_0, 1) = q_0$

$\delta(q_0q_1, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$

$= q_0q_1 \cup \emptyset$

$= q_0q_1$

$\delta(q_0q_1, 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$

$= q_0 \cup q_2$

$= q_0q_2$ (New state)

$\delta(q_0q_2, 0) = \delta(q_0, 0) \cup \delta(q_2, 0)$

$= q_0q_1 \cup q_2$

$= q_0q_1q_2$

$\delta(q_0q_2, 1) = \delta(q_0, 1) \cup \delta(q_2, 1)$

$= q_0 \cup q_2$

$= q_0q_2$

$\delta(q_0q_1q_2, 0) = q_0q_1q_2$

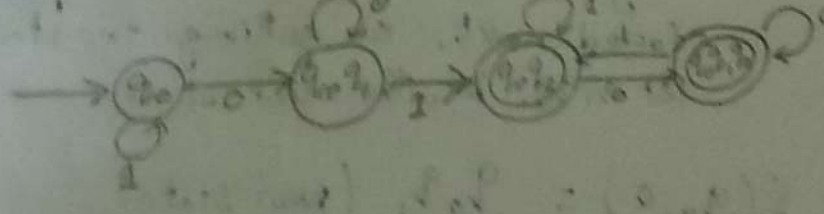
$\delta(q_0q_1q_2, 1) = q_0q_2$

Step 3: Compile all the δ transitions into a table

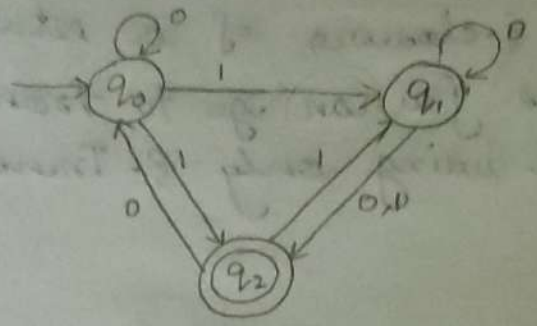
	0	1
q_0	q_0q_1	q_0
q_0q_1	q_0q_1	q_0q_2
* q_0q_2	$q_0q_1q_2$	q_0q_2
* $q_0q_1q_2$	$q_0q_1q_2$	q_0q_2

Note: Since q_2 is final state, all states with q_2 are final.

Step 4: Draw the DFA from the table.



Q. Convert the following NFA to DFA.



A:

δ	0	1
q_0	q_0	$q_1 q_2$
q_1	$q_1 q_2$	q_2
* q_2	q_0	q_1

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1 q_2$ (N.S)

$\delta[q_1 q_2, 0) = \delta(q_0, 0) \cup \delta(q_2, 0)$

$= q_1 q_2 \cup q_0$

$= q_0 q_1 q_2$ (N.S).

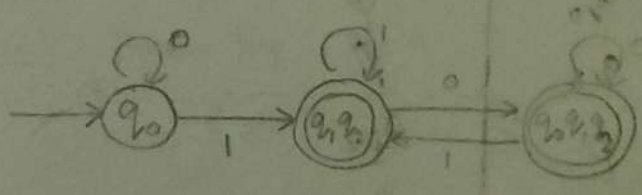
$\delta[q_1 q_2, 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$

$= q_1 q_2$

$\delta[q_0 q_1 q_2, 0) = q_0 q_1 q_2$

$\delta[q_0 q_1 q_2, 1) = q_1 q_2$

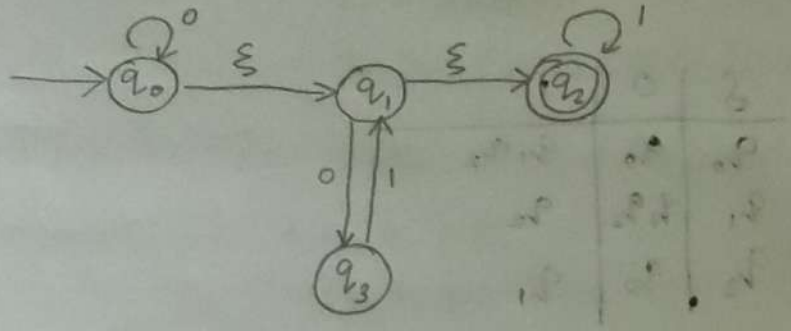
δ	0	1
q_0	q_0	$q_1 q_2$
* $q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$
* $q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$



1.7) ξ-closure :

In ξ-NFA, ξ-closure of a state is the set of states you can go to from the current state using only ξ-Transitions.

eg:



$\xi(q_0) = \{q_0, q_1, q_2\}$

$\xi(q_1) = \{q_1, q_2\}$

$\xi(q_2) = \{q_2\}$

$\xi(q_3) = \{q_3\}$

1.8) ξ-NFA to NFA :

Formula : $\xi(\delta(\xi(q_n, i)))$

Continuing with the same NFA from 1.7, First find the ξ-closure of each state.

step-2 : Write the transition table of the ξ-NFA

	0	1
q_0	q_0	ϕ
q_1	q_3	ϕ
q_2	ϕ	q_2
q_3	ϕ	q_1

Step-3 : Apply the formula to each state using the gi table from prev step

$\xi(\delta(\xi(q_0), 0)) = \xi(\delta(q_0 q_1 q_2, 0))$

$= \xi(q_0 q_3)$

$= q_0 q_1 q_2 \cup q_3$

$= q_0 q_1 q_2 q_3$

$\xi(\delta(\xi(q_0), 1)) = \xi(\delta(q_0 q_1 q_2, 1))$

$= \xi(q_2)$

$= q_2$.

$\xi(\delta(\xi(q_1), 0)) = \xi(\delta(q_1 q_2), 0) = \xi(q_3) = q_3$

$\xi(\delta(\xi(q_1), 1)) = \xi(\delta(q_1 q_2), 1) = \xi(q_2) = q_2$.

$\xi(\delta(\xi(q_2), 0)) = \xi(\delta(q_2), 0) = \xi(\phi) = \phi$

$\xi(\delta(\xi(q_2), 1)) = \xi(\delta(q_2), 1) = \xi(q_2) = q_2$.

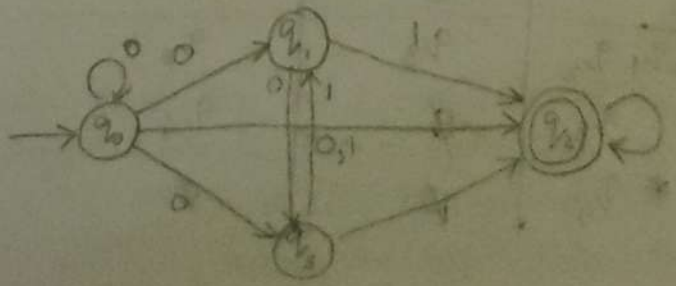
$\xi(\delta(\xi(q_3), 0)) = \xi(\delta(q_3), 0) = \xi(\phi) = \phi$

$\xi(\delta(\xi(q_3), 1)) = \xi(\delta(q_3, 1)) = \xi(q_1) = q_1 q_2$.

Step-4 : Compile the results in a transition table

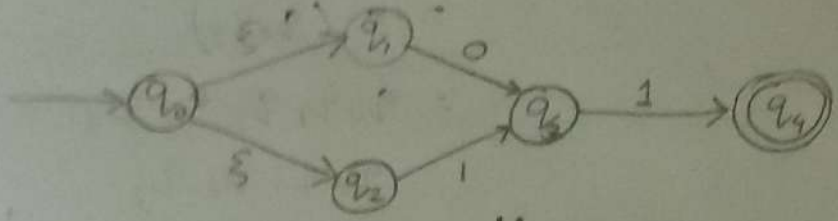
	0	1
q_0	$q_0 q_1 q_2 q_3$	q_2
q_1	q_3	q_2
q_2	ϕ	q_2
q_3	ϕ	$q_1 q_2$

Step-5: Draw the NFA



1.9) ε-NFA to DFA:

Convert the ε-NFA given below to DFA:



Step-1: Find ε closure of each state

$\varepsilon(q_0) = \{q_0, q_1, q_2\}$

$\varepsilon(q_1) = q_1$

$\varepsilon(q_2) = q_2$

$\varepsilon(q_3) = q_3$

$\varepsilon(q_4) = q_4$

Step-2: start at starting state, find $\varepsilon(\delta(\varepsilon(q_0), 0))$ & $\varepsilon(\delta(\varepsilon(q_0), 1))$, and continue the same for any new state that emerge.

$\varepsilon(\delta(\varepsilon(q_0), 0)) = \varepsilon(\delta(q_0 q_1 q_2, 0)) = \varepsilon(q_3) = q_3$ (1)

$\varepsilon(\delta(\varepsilon(q_0), 1)) = \varepsilon(\delta(q_0 q_1 q_2, 1)) = \varepsilon(q_3) = q_3$.

$\varepsilon(\delta(\varepsilon(q_3), 0)) = \varepsilon(\delta(q_3, 0)) = \varepsilon(\phi) = \phi$

$\varepsilon(\delta(\varepsilon(q_3), 1)) = \varepsilon(\delta(q_3, 1)) = \varepsilon(q_4) = q_4$

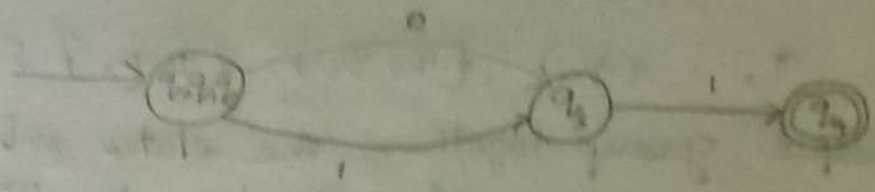
$\varepsilon(\delta(\varepsilon(q_4), 0)) = \phi$

$\varepsilon(\delta(\varepsilon(q_4), 1)) = \phi$

Step-3: Compile the results in a transition table

	0	1
$\rightarrow q_0 q_1 q_2$	q_3	q_3
q_3	ϕ	q_4
$*q_4$	ϕ	ϕ

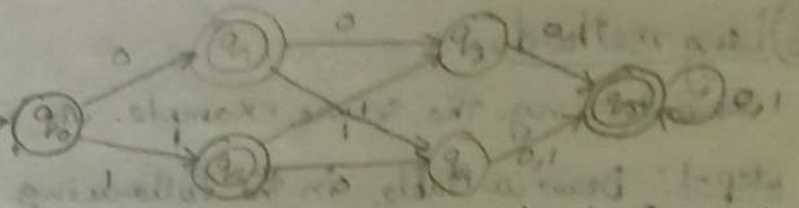
Step-1 : Draw the DFA



1.10) Minimization of DFA :

(a) Shortcut method:

given is a DFA to be minimised.



Step-1: make the transition table for all states.

	0	1
q_0	q_1	q_2
* q_1	q_3	q_4
* q_2	q_4	q_3
q_3	q_5	q_5
q_4	q_5	q_5
* q_5	q_5	q_5

Step-2: group together all the final states & non-final states in different sets as π_0

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

Step-3: check transition of each element in both set if the transition of states in same set are not member of same set, then split the states into new sets.

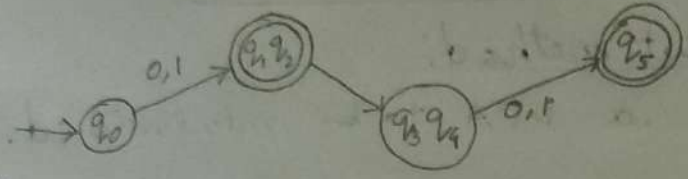
$$\pi_1 = \{q_0, q_3, q_4\} \{q_1, q_2\} \{q_5\}$$

here, q_0, q_3 & q_4 have transitions to q_1, q_2 & q_5 which are all from same set, however q_1 & q_2 transition to q_3 & q_4 where as q_5 transition to q_5 which are in different sets, hence we split $q_1 q_2$ & q_5

Step-4: Repeat step-3 until no longer splits are needed.

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

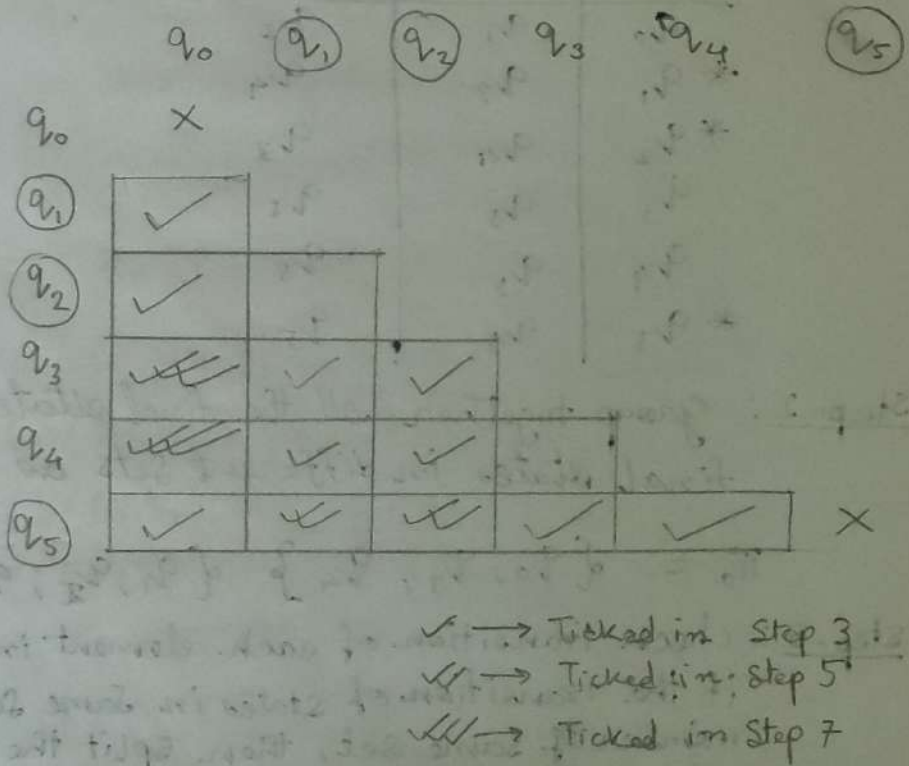
Step-5: Group together the states which are in same set & re-draw the DFA.



(b) Long method:

considering the same example as previous step

Step-1: Draw a table in the following format [lower triangular region of a matrix of combination of all states]



✓ → Ticked in Step 3
✓✓ → Ticked in Step 5
✓✓✓ → Ticked in Step 7

Step-2: circle the final states in the Table.

Step-3: Tick off the intersections of all final & non final state (i.e. one final & one non final)

Step-4: Write down all unmarked positions

unmarked: $(q_2 q_1), (q_3 q_0), (q_4 q_0), (q_4 q_3), (q_5 q_1), (q_5 q_2)$

Step 5: Find the δ-Transition of the set of states which are unmarked from step-4, If the result in either δ+ Transition is a marked state, then mark the selected state in Table.

$\delta(q_2 q_0, 0) = (q_1, q_3)$ X
$\delta(q_2 q_1, 1) = (q_3 q_4)$ X
$\delta(q_3 q_0, 0) = (q_5 q_1)$ X
$\delta(q_3 q_0, 1) = (q_5, q_2)$ X
$\delta(q_4 q_0, 0) = (q_5 q_1)$ X
$\delta(q_4 q_0, 1) = (q_5 q_2)$ X
$\delta(q_5 q_2, 0) = (q_5 q_3)$ ✓
$\delta(q_5 q_2, 1) = (q_5 q_4)$ ✓
$\delta(q_4 q_3, 0) = (q_5 q_5)$ X
$\delta(q_4 q_3, 1) = (q_5 q_5)$ X

X | unmarked in Table
✓ | Marked in Table - as of step-4

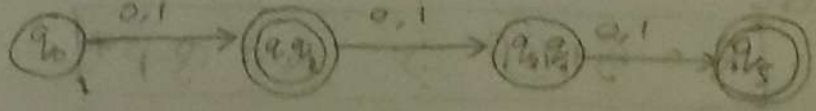
Step-6: remove the states marked in step-5 from the list of unmarked states

unmarked: $(q_2 q_1)$ $(q_3 q_0)$ $(q_4 q_0)$ $(q_4 q_3)$

Step-7: repeat the process of step-5 & step-6 till no more unmarked states can be removed.

$\delta(q_3 q_0, 0) = (q_5 q_1)$ ✓
$\delta(q_4 q_0, 0) = (q_5 q_2)$ ✓
$\delta(q_2 q_1, 0) = (q_4 q_3)$ X
$\delta(q_2 q_1, 1) = (q_3 q_4)$ X
$\delta(q_4 q_3, 0) = (q_5 q_5)$ X
$\delta(q_4 q_3, 1) = (q_5 q_5)$ X

Step-8: The unmarked groups can be grouped together and then the DFA can be redrawn.



1.11) Regular Expression : (Regex)

A Regular Expression describes a set of strings that matched the pattern. In other words, it accepts a certain set of strings & rejects the rest

eg:

i) The set {010, 100, 111} → RE = 010 + 100 + 111

ii) The set {ε, 0, 00, 000 ...} → RE = 0*

iii) All string start with 0 → RE = 0 (0+1)*

Q. Define Regular Expression for the following

(i) Even no. of zeroes.

(ii) At least two zeroes

(iii) 0's followed by one or more 1's followed by 0's

A: (i) RE = (1*01*01*)*

(ii) (1+0)* 0 (1+0)* 0 (1+0)*

(iii) 0* 1+ 0*

1.12) DFA's to Regular Expression:

Two methods of Conversion:

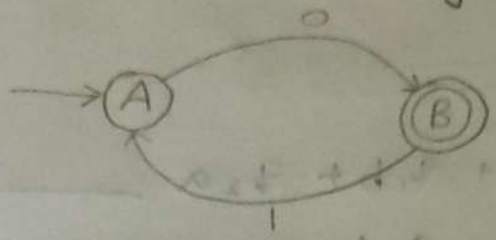
(i) Arden's Method

(ii) State Elimination method.

Let P and Q be two regular Expression over Σ, if P does not contain a null string. then

$$R = Q + RP \Rightarrow R = QP^*$$

Q. Convert the Following DFA to RE.



A:

Step-1 : Find the path to each state of DFA

$$A = \varepsilon + B \cdot 1$$
$$B = A \cdot 0$$

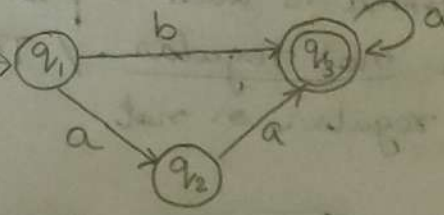
Step-2: Bring this to a form of $R = Q + RP$

$$B = A \cdot 0$$
$$B = (\varepsilon + B \cdot 1) 0$$

Compare $\begin{cases} B = 0 + B(1 \cdot 0) \\ R = Q + RP \end{cases}$

$\therefore B = 0(10)^*$ | Note: R is the final state.

Q. Convert the Following DFA to RE



A: $q_1 = \varepsilon \longrightarrow (1)$

$q_2 = a \cdot q_1 \longrightarrow (2)$

$q_3 = b \cdot q_1 + a \cdot q_2 + a \cdot q_3 \longrightarrow (3)$

$\therefore q_3 = b + a q_2 + a q_3$ | from (1) $\because q_1 = \varepsilon$

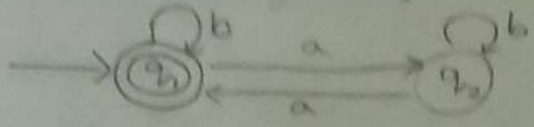
$\Rightarrow q_3 = b + a(a q_1) + a q_3$ | from (2)

$\Rightarrow q_3 = (b + aa) + a q_3$ | from (1) $\because q_1 = \varepsilon$

$R = Q + PR$

$\therefore q_3 = (b + aa) a^*$ $\therefore R.E = (b + aa) a^*$

Q) Convert the DFA given below to RE.



A: $q_1 = \varepsilon + q_1 b + q_2 a \longrightarrow ①$

$q_2 = q_1 a + q_2 b \longrightarrow ②$.

$q_1 = \varepsilon + q_1 b + q_2 a a^*$

$\begin{bmatrix} q_2 = (q_1 a) + q_2 b \\ R = Q + RP \end{bmatrix}$

$\therefore q_2 = (q_1 a) b^*$

$\therefore q_1 = \varepsilon + q_1 b + (q_1 a) b^* a$

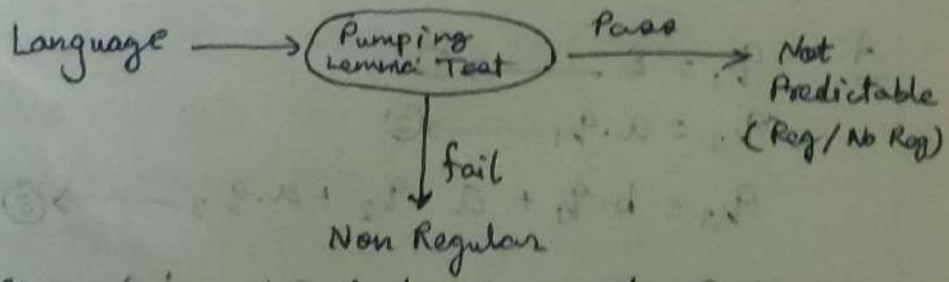
$q_1 = \varepsilon + (b + aab^*) q_1$

$q_1 = \varepsilon + (b + aab^*)^*$

R.E $= \varepsilon + (b + aab^*)^*$

1.13) Pumping Lemma:

Pumping Lemma is used to prove that given equation is not regular. / To check if a language is regular or not.



It is a negative test, since only fail gives a decidable outcome.

If L is an infinite language, then there exists some +ve integer 'n' (Pumping length) such that any string $w \in L$ has length greater than equal to 'n'. i.e. $|w| \geq n$, then w can be divided into three parts, $w = xyz$ satisfy the following conditions

(i) For each $i \geq 0$, $xy^iz \in L$
(ii) $|y| > 0$
(iii) $|xy| \leq n$

i.e
$xyz \in L$
$xyyz \in L$
$xyyyz \in L$ ...

eg: $L = a^n b^{2n}$, $n \geq 0$

Step 1: lets take a sample $n = 2$

$\therefore$ aa bb bb $\in L$

Step 2: Divide it into 3 parts (can divide in any way)

$\underset{x}{aa}\ \underset{y^2}{bb}\ \underset{z}{bb} \in L$

$\underset{x}{aa}\ \underset{y^2}{bb\,bb}\ \underset{z}{bb} \notin L$

Hence, on pumping y i.e. increasing power of y $xy^iz \notin L$, hence Pumping Lamma test fails.

can do this with any xyz, such as

$\underset{x}{a}\ \underset{y}{ab}\ \underset{z}{bbb} \in L$

$\underset{x}{a}\ \underset{y^2}{ab\,ab}\ \underset{z}{bbb} \notin L$

Context Free Grammars (CFGs)

Unit - 2

★ Topics:

2.1) Context Free Grammar:

A context free grammar is a set of recursive rules used to generate patterns of string (language).

It is a set of 4-Tuples (V, T, P, S)

CFG: (V, T, P, S)

V: Finite set of variables (Non Terminal)

T: finite set of Terminals ($V \cap T = \phi$)

P: Production rules (Substitution Rules)

S: Start variable / state.

Note: Variable → capital Leter → Can be substituted by var or terminal

Terminal → Small letter → no substitution

Production rule → $(\alpha \rightarrow \beta)$ → variables/Terminals $(V \cup T)^*$

(α: only one variable)

Start state → First LHS of Production / given otherwise

eg: $S \rightarrow 0S1$
$S \rightarrow \varepsilon$

This can also be written as $S \rightarrow 0S1 \mid \varepsilon$

here $V = \{S\}$ | All capital letter symbols can be substituted

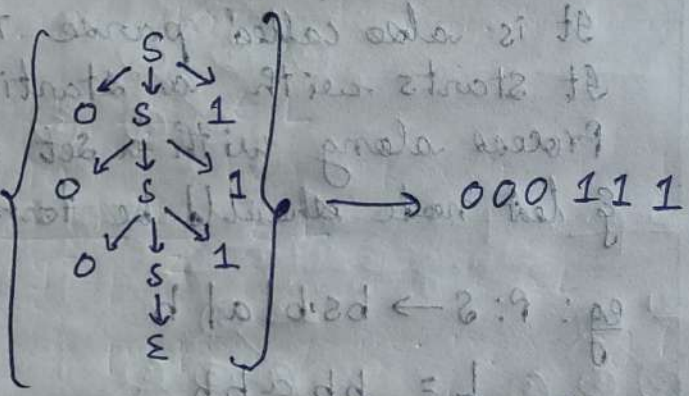
$T = \{0, 1, \varepsilon\}$ | final values of variables

$P = S \rightarrow 0S1 \mid \varepsilon$

$S = \{S\}$

This means we can either substitute 0S1 or ε in place of S, this can be substituted as many times & ε can be substituted to terminate.

eg: i.e.

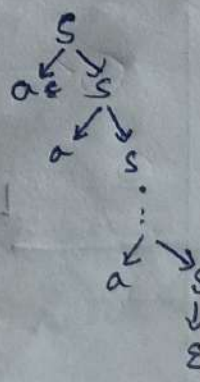


{ i.e. Language is ~~$0^* 1^*$~~ $0^n 1^n \; \forall \; n \geq 0$ }

Language = $0^n 1^n \; \forall \; n \geq 0$.

Q. Design a CFG which accepts $L = a^*$

A: P: $S \rightarrow aS \mid \varepsilon$
$V \rightarrow \{S\}$
$T \rightarrow \{a, \varepsilon\}$
$S \rightarrow \{S\}$



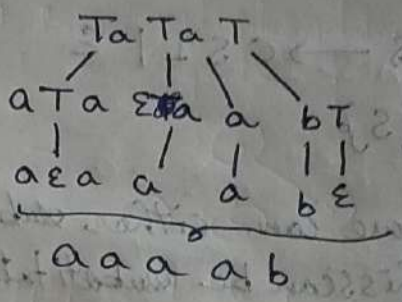
Q. Construct a CFG which accepts atleast 2a's over {a, b}

A: R.E = (a+b)* a (a+b)* a (a+b)*

P: S → TaTaT

T → aT | bT | ε

eg:

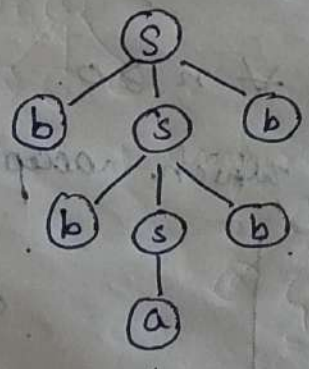


2.2) Derivation tree:

It is also called parse tree.
It starts with a starting symbol & Process along with a set of non-terminals & leaf node should be terminal symbols.

eg: P: S → bSb | a | b

L = bbabb



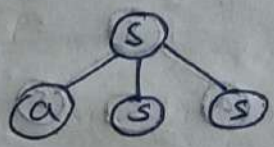
L = bbabb

2.2.1) **Left Derivation Tree:**

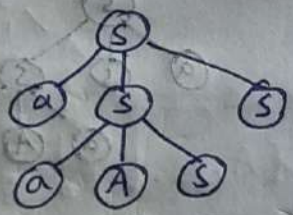
A Left Derivation Tree is obtained by applying production to the left most variable in each step.

Eg: Draw D.T. to generate the string aabaa from the Grammar $S \rightarrow aAS \mid ass \mid \epsilon$
$A \rightarrow SbA \mid ba$

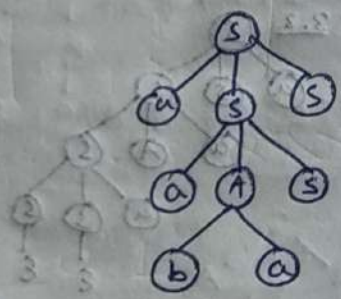
Step1: Start from starting state, then choose a production suitable to generate given string



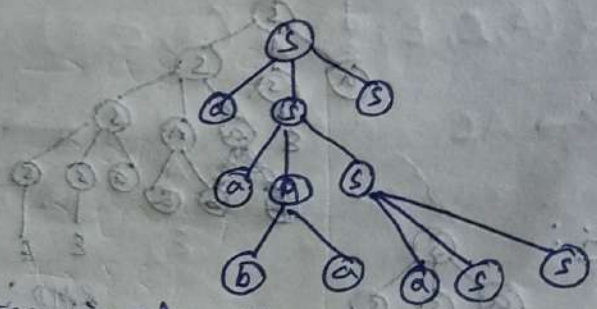
Step 2: Now apply production to the left most node.



Step 3: Again apply production from left to right

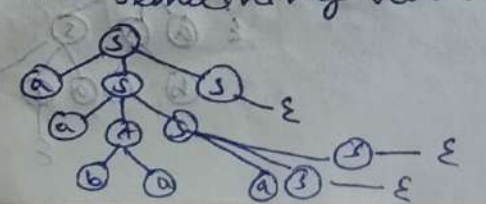


Step 4: Continue till you get your String



Step 5: terminate the remaining variable states

LMD = $S \rightarrow ass$
$S \rightarrow aAS$
$A \rightarrow ba$
$S \rightarrow ass$
$S \rightarrow \epsilon$

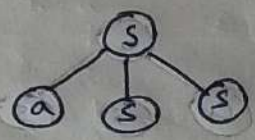


2.2.2) Right Derivation Tree:

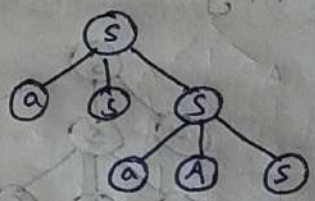
A Right Derivation Tree is obtained by applying Production to the rightmost variable in each step.

eg: Draw D.T to generate the string aabaa from the Grammar $S \rightarrow aAS|ass|\epsilon$
$A \rightarrow SbA|ba$

Step 1: Start from starting node, then choose the production suitable to generate the given string.

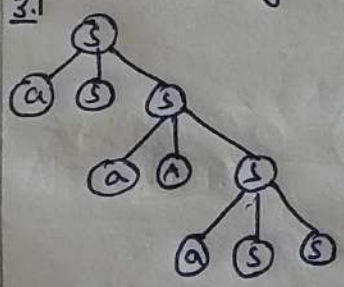


Step 2: Now apply suitable production to the Right most node

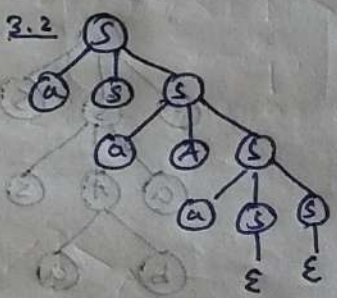


Step 3: Repeat same from Right to left till required string is obtained.

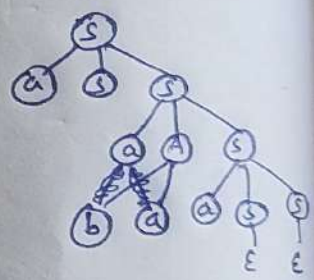
3.1



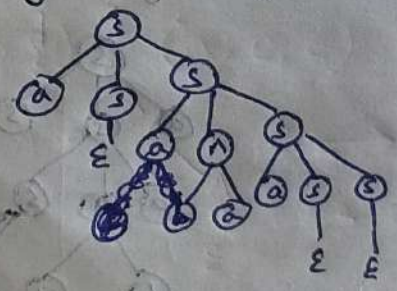
3.2



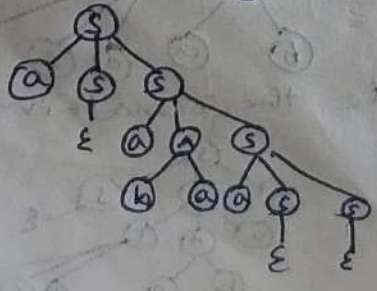
3.3



Step 4: Terminate any remaining variable state



Redraw ::



RMD: $S \rightarrow aSS$
$S \rightarrow aAS$
$S \rightarrow ass$
$S \rightarrow \epsilon$
$S \rightarrow \epsilon$
$A \rightarrow ba$
$S \rightarrow \epsilon$

Q. Design a CFG which accepts $a^n b^{m+n} c^m$, $m, n > 0$. Draw the LDT & RDT.

A: $L = \{a^n b^{m+n} c^m \mid m, n > 0\}$

$= \{abbc, aabbbbcc, \ldots\}$

Note: Lay out Language, it'll help to design CFG, identify repeting patterns.

P: $S \rightarrow aSbbTc \mid asb \mid \varepsilon$

$T \rightarrow bTc \mid \varepsilon$

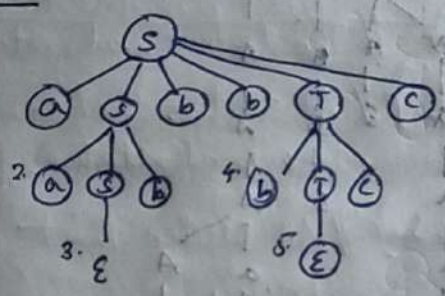
V: {S, T}

T: {a, b, c}

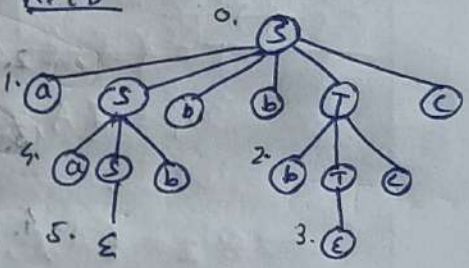
S: {S}

Lets take case aabbbbcc

LMD



RMD



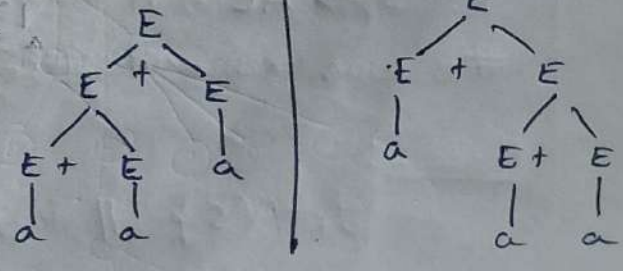
2.3) Ambiguity:

A CFG is said to be ambiguous if there exists more than one derivative tree. It is also called ambiguous grammar.

- more than one left most parse tree
- More than one right most parse tree
- More than one parse tree.

eg: $E \rightarrow E+E \mid a$

$L \rightarrow a+a+a$



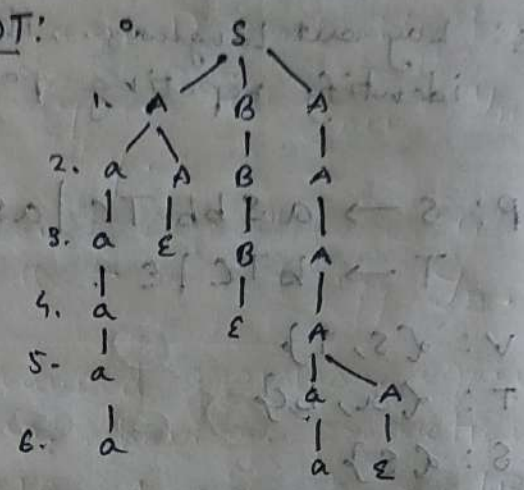
eg: $L = \{a\ a\}$

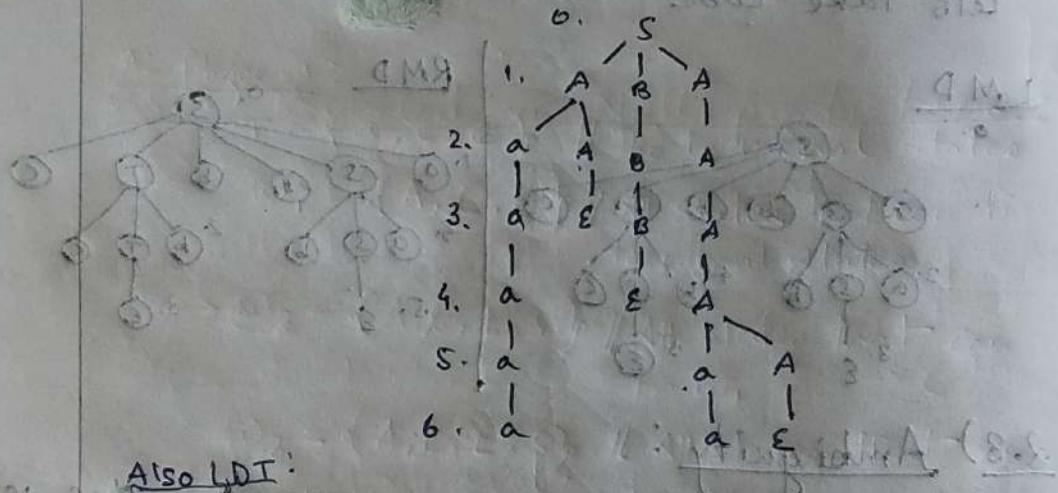
$P: \{ S \rightarrow ABA$

$A \rightarrow aA \mid \varepsilon$

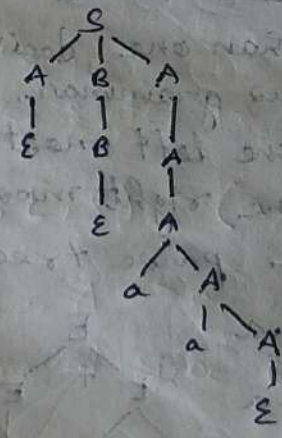
$B \rightarrow bB \mid \varepsilon \}$

LDT:





Also LDT:



2.5) Simplification of CFG :

In CFG, sometimes all production rules & symbols are not needed for the derivation of string. Besides this there may also be some NULL production and unit productions. Elimination of these productions and symbols is called simplication of CFG.

The following are the steps of simplification :

(i) Reduction of CFG.
(ii) Removal of unit Production.
(iii) Removal of Null production.

2.5.1) Reduction of CFG :

CFG are reduced in two phases. in phase one we go from terminal to variable, in phase two we go from start to terminal.

a) Phase - 1 :

Derivation of an equivalent grammar G', from the CFG G, such that each variable derive some terminal string.

Procedure :

Step-1 : Set W_1 : Include all symbol that derive any terminal.

Step-2 : Set W_2 : Include all symbol that derives the symbols in W_1, then add those to symbols of W_1

Step-3 : Keep repeating step-2 till $W_n = W_{n+1}$ i.e. no change after one step.

Step-4 : reconstruct the new CFG G' by removing any symbol which is not present in W_n.

b) Phase - 2 :

Derivation of an equivalent grammar G", from the CFG G', such that each symbol appears in a sentential form. i.e. starting from starting state

all symbols should be reachable in G''.

Procedure:

Step-1: include the starting state in Set Y_1

Step-2: Set Y_2: include all symbols (both terminal & non terminal) reachable from states in Y_1

Step-3: repeat step-2 till $Y_{n+1} = Y_n$ i.e. no change after a step.

Step-4: reconstruct CFG G'' from G' removing all symbol that is not in Y_n.

eg: Find a reduced grammar equivalent to the grammar G, having production rules:

P: $S \to AC|B$, $A \to a$, $C \to c|BC$, $E \to aAe$

| c → small
| C → capital

A: Phase-1:

$T = \{a, c, e\}$ // The terminal states

$\omega_1 = \{A, C, E\}$ // All states that can directly reach terminal states.

$\omega_2 = \{A, C, E, S\}$ // All states that can reach states in ω_1

$\omega_3 = \{A, C, E, S\}$ // All states that can reach states in ω_2

$\therefore \omega_2 = \omega_3$.

We take states in ω_3 only.

$G' = \{(A, C, E, S), (a, c, e), P, \{S\}\}$

P: $S \to AC$, $A \to a$, $C \to c$, $E \to aAe$

#B was removed since it was not present in ω_3.

Phase 2:

$Y_1 = \{S\}$ // initializing with starting state.

$Y_2 = \{S, A, C\}$ // include states reachable from states in Y_1

$Y_3 = \{S, A, C, a, c\}$ // include states reachable from states in Y_2

$Y_4 = \{S, A, C, a, c\}$ // include states reachable from states in Y_3.

$\because Y_4 = Y_3$
we take states in Y_4 only.

$\therefore G'' = \{(A, C, S), (a, c), P, (S)\}$

E & e get removed as they do not appear in Y_4

$\therefore$ P: $S \rightarrow AC, A \rightarrow a, C \rightarrow c$.
↳ reduced CFG.

2.5.2) Removal of Unit Production:

Any production Rule of the form $A \rightarrow B$ where $A, B \in$ Non Terminals, is called Unit Production.

Procedure for removal:

Step-1: To remove $A \rightarrow B$, add production $A \rightarrow X$ to the production rule whereever $B \rightarrow X$ occurs in the grammar. [$X \in$ Terminal, X can be Null]

Step-2: Delete $A \rightarrow B$ from the grammar.

Step 3: Repeat from step 1 until all unit Productions are removed.

Eg: Remove unit productions from the Grammar whose production rule is given by

P: $S \rightarrow Xy, X \rightarrow a, Y \rightarrow Z|b, Z \rightarrow M, M \rightarrow N, N \rightarrow a$.

A: Step-1: Identify the unit productions

$Y \rightarrow Z, Z \rightarrow M, M \rightarrow N$

P:

Step-2: Since $N \to a$, we add $M \to a$.

P: $S \to XY$, $X \to a$, $Y \to z|b$, $Z \to M$, $M \to a$, $N \to a$

Step-3: since $M \to a$, we add $Z \to a$

P: $S \to XY$, $X \to a$, $Y \to z|b$, $Z \to a$, $M \to a$, $N \to a$

Step-4: Since $Z \to a$, we add $Y \to a$

P: $S \to XY$, $X \to a$, $Y \to a|b$, $Z \to a$, $M \to a$, $N \to a$

Step-5: Remove the unreachable symbols

P: $S \to XY$, $X \to a$, $Y \to a|b$.

Q.5.3) Removal of Null productions:

In a CFG, a Non-Terminal Symbol 'A' is a nullable variable if there is a production $A \to \epsilon$ or there is a derivation that starts with 'A' & leads to ϵ (like $A \to \ldots \to \epsilon$)

Procedure of removal:

Step-1: To remove $A \to \epsilon$, select all productions with A on RHS.

Step-2: Replace each occurence of A with ϵ and include all possible combinations of after replace A.

Step-3: Add the resultant production to the grammar.

eg: Remove Null productions from the following Grammar

$S \to ABAC,\ A \to aA|\epsilon,\ B \to bB|\epsilon,\ C \to c$

A: Step-1: Identify the nullable variables

$A \to \epsilon,\ B \to \epsilon$

Step-2: To eleminate $A \to \epsilon$ Null production, Select all production with A on RHS, replace each A with ϵ in every possible combination.

$S \to ABAC$	# ABAC
$S \to ABC$	# ABεC
$S \to BAC$	# εBAC
$S \to BC$	# εBεC.

$\therefore S \to ABAC|ABC|BAC|BC$

$A \to aA$	#aA.
$A \to a$	#aε

Step 3: Create the new production by taking the previous step & make it in grammar.

New Production: $S \to ABAC|ABC|BAC|BC$

$A \to aA|a$

$B \to bB|\epsilon$

$C \to c$

Step-4: repeat steps 1, 2 & 3 for B as well.

$S \to ABAC|ABC|BAC|BC$ # $B \to B$

$S \to AAC|AC|AC|C$. # $B \to \epsilon$

$B \to BbB$ # $B \to B$

$B \to b$ # $B \to \epsilon$

$\therefore$ New production: $S \to ABAC|ABC|BAC|BC|AAC|AC|C$

$A \to aA|a$

$B \to bB|b$

$C \to c$

2.6) Chomsky Normal Form (CNF):

In Chomsky Normal Form, we have a restriction on the length of RHS; which is, Elements in RHS should be either two variables or one Terminal.
A CFG in chomsky Normal Form is in the following forms:

$A \longrightarrow a$
$A \longrightarrow BC$

| A, B, C are non-terminals
| a is a terminal.

★

2.7) Conversion of CFG to CNF:

The aim of conversion is that all production are in form $A \rightarrow a$ [one terminal in RHS] or $A \rightarrow BC$ [two variable in RHS]

The following are the steps of conversion:

Step-1 : If the start symbol S occurs on some right hand side, create a new symbol S' and a new production $S' \longrightarrow S$

Step-2 : Remove Null productions (2.5.3)

Step-3 : Remove unit productions (2.5.2)

Step-4 : If any production has more than 2 variable on RHS, i.e. $A \rightarrow B_1 B_2 \ldots B_n$,
Replace with $A \rightarrow B_1 C$ where $C \longrightarrow B_2 \ldots B_n$
(i.e. Bring down to two variable in RHS)
Repeat this step for all production having more than two variables on RHS.

Step-5 : If the RHS of any production has both terminal & variable, i.e. $A \longrightarrow aB$,
then replace it by $A \longrightarrow XB$ and $X \rightarrow a$.
Repeat this for every production of the form $A \rightarrow aB$

eg: Convert the following CFG to CNF:

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

Step-1: Check if starting state S appears in any RHS, if yes create a new $S' \to S$

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

Step-2: • check if any null production exists

$B \to \epsilon$

• Removing null production B using (2.5.3)

$S \to ASA \mid aB \mid a$

$A \to B \mid S \mid \epsilon$

$B \to b$

• check if any new null production arises

$A \to \epsilon$

• Remove any new null production using (2.5.3)

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

$A \to B \mid S$

$B \to b$

• Compile the new production

$S' \to S$

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

$A \to B \mid S$

$B \to b$

Step-3: • Identify all unit productions in the grammar

$S \to S$

$S' \to S$

$A \to B$

$A \to S$.

• Remove all unit production using (2.5.2)

• Removing $S \to S$:

$S' \to S$

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

$A \to B \mid S$.

$B \to b$

- Removing $S' \rightarrow S$:

$S' \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow B \mid S$

$B \rightarrow b$

- Removing $A \rightarrow B$:

$S' \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid S$

$B \rightarrow b$

- Removing $A \rightarrow S$.

$S' \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA$

$B \rightarrow b$

Step-4 : • Now find out the productions that have more than TWO variables in RHS

$S' \rightarrow ASA$ $S \rightarrow ASA$ $A \rightarrow ASA$.

- Remove by adding $X \rightarrow SA$

$S' \rightarrow AX \mid aB \mid a \mid AS \mid SA$

$S \rightarrow AX \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid AX \mid aB \mid a \mid AS \mid SA$

$B \rightarrow b$

$X \rightarrow SA$

Step-5 : • Lastly find productions with both variable & terminal in RHS :

$S' \rightarrow aB$ $S \rightarrow aB$ $A \rightarrow aB$

- Remove by adding $Y \rightarrow a$

$S' \rightarrow AX \mid YB \mid a \mid AS \mid SA$

$S \rightarrow AX \mid YB \mid a \mid AS \mid SA$

$A \rightarrow b \mid AX \mid YB \mid a \mid AS \mid SA$

$B \rightarrow b$

$X \rightarrow SA$, $Y \rightarrow a$

→ CNF

2.8) Greibach Normal Form (GNF):

A CFG is in Greibach Normal Form if the productions are in the following forms:

$A \rightarrow b$ | only one Terminal

$A \rightarrow bC_1C_2 \ldots C_n$ | one Terminal & n variables. (must have terminal at beginning)

Here $A, C_1, C_2 \ldots C_n$ are Non-Terminals. and b is Terminal.

[Explained & mentioned steps directly with example]

* 2.9) CFG to GNF Conversion:

Lets Consider The following CFG

P: $S \rightarrow CA \mid BB$
$B \rightarrow b \mid SB$
$C \rightarrow b$
$A \rightarrow a$

The following are the steps to Convert the given CFG to GNF

Step-1: check if the given CFG has any unit Production or Null production. If yes then remove them (2.5.2/2.5.3)

- No unit or null production found.

Step-2: check if the given CFG is already in CNF form, if not, convert to CNF (2.7)

- Already in CNF form

Step-3: change the names of Non-Terminal Symbols to A_i where i is 1,2,3.... based on order of occurence.

- $S \rightarrow CA \mid BB$
  $B \rightarrow b \mid SB$
  $C \rightarrow b$
  $A \rightarrow a$

  Replace: S with A_1
  C with A_2
  A with A_3
  B with A_4

- Hence, we get:

$A_1 \rightarrow A_2\ A_3 \mid A_4\ A_4$

$A_4 \rightarrow b \mid A_1\ A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step-4: For each production in form $A_i \rightarrow A_j x$

[x can be $A_m\ A_m\ A_o$ ... i.e. anything]

if $i \geq j$ substitute value of A_j to make $i<j$ [i.e. $i<j$ should be there]

- $A_1 \rightarrow A_2\ A_3 \mid A_4\ A_4$

$1<2$ ✓ } follows $i<j$

$1<4$ ✓ }

- $A_4 \rightarrow b \mid A_1 A_4$

$4<1$ X } doesn't follow $i<j$

Substitute the value of A_1

$A_4 \rightarrow b \mid A_2 A_3\ A_4 \mid A_4\ A_4\ A_4$

still $4<2$ X } still does not follow

Substitute value of A_2

$A_4 \rightarrow b \mid b\ A_3 A_4 \mid A_4\ A_4\ A_4$ (Left Recursion)

↳ bA_iA_j is allowed as it satisfies GNF ✓

Now $4<4$ X | Not satisfying in $A_4 \rightarrow A_4 A_4 A_4$

Ignore the Left Recursion for now, we solve them in Step 5.

We've

$A_1 \rightarrow A_2\ A_3 \mid A_4\ A_4$

$A_4 \rightarrow b \mid b\ A_3\ A_4 \mid A_4\ A_4\ A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step-5 : Remove Left Recursion

$A_4 \rightarrow b \mid b A_3 A_4 \mid A_4 A_4 A_4$

- Introduce a New variable to remove the left Recursion.

here $A_i \rightarrow A_4$

$A_j \rightarrow A_4$

$A_i \rightarrow A_j X$

$A_4 \rightarrow A_4, A_4 A_4$

A_j is the problamatic variable & $A_4 A_4$ follows the problamatic variable i.e. X

make the new variable $Z \rightarrow XZ \mid X$

i.e. $Z \rightarrow A_4 A_4 Z \mid A_4 A_4$

- Now we take the other states mapped to A_i & add new variable Z after them & map them to A_i

i.e. $A_4 \rightarrow b \mid b A_3 A_4 \mid A_4 A_4 A_4$ becomes

$Z \rightarrow A_4 A_4 Z \mid A_4 A_4$

$A_4 \rightarrow b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$.

- The grammar becomes:

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$

$Z \rightarrow A_4 A_4 \mid A_4 A_4 Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Step-6 : Now check for any production that still doesn't obey GNF rules $A \rightarrow b$ / $A \rightarrow bC_1C_2 \ldots C_n$ & substute the value of problematic variable A_j in them till they follow

- $A_1 \rightarrow A_2 A_3 \mid A_4 A_4$ doesn't obey GNF

$\therefore A_1 \rightarrow b A_3 \mid A_4 A_4$ # replace $A_2 \rightarrow b$

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

Substituting value of A_4, i.e. each possible value of A_4.

- $Z \rightarrow \underline{A_4}A_4 \mid \underline{A_4}A_4Z$

doesn't follow rule of GNF.

∴ Substituting value of A_4 in underlined problematic positions. (i.e. the positions that effect the rule)

$Z \rightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid bA_4Z \mid bA_3A_4A_4Z$
$\mid bZA_4Z \mid bA_3A_4ZA_4Z$

∴ New Grammar is:

P: $A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

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

$Z \rightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

The above obtained grammar follows GNF

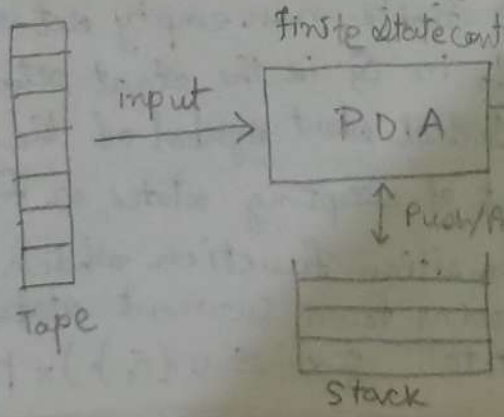
Push Down Automata

Unit-3

Pushdown Automata Topics:

3.1) Push Down Automata:

- A Pushdown automata (PDA) is a way to implement a Context Free Grammar, in a similar way we design Finite Automata for Regular Grammar.
- A Pushdown automata is essentially a finite automata with control of input tape on addition to a stack on which it can store a string
- with the help of a stack, the pushdown automata can remember an infinite amount of information



- PDA consists of a finite set of states, a finite set of input symbols and a finite set of pushdown symbols.
- In one transition of the pushdown automaton,
  - The control head reads the input symbol, then goto new state.
  - Replace the symbol at the top of the stack by any string.

3.1.1) PDA components:

- **Input tape**: This is devided in many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.
- **Finite control**: The finite control has some pointer which points the current symbol which is to be read.
- **Stack**: The stack is a structure in which we can push & remove the items from one end only. It has an infinite size. In PDA, the stack is used to store the items temporarily.

3.1.2) Defination of PDA:

A pushdown automata consists of seven tuples.

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

where,

Q - A finite non empty set of states

Σ - A finite set of input symbols.

Γ - A finite non empty set of stack symbols.

q_0 - q_0 in Q is the start state

Z_0 - Initial start symbol of the stack.

F - set of accepting states or final states.

δ - transition function which is used for moving from current state to next state. $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow Q \times \Gamma^*$

δ takes as arguement a triple $\delta(q, a, X)$ where:

- q is a state in Q
- a is either an Input Symbol in Σ or $a = \epsilon$
- X is a Stack symbol, that is a member of Γ

The output of δ is finite set of pairs (P, γ) where:

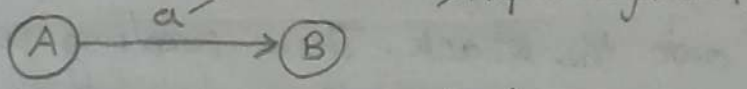
P is a new state from set of states Q.

γ is a string of stack symbols that replaces X at the top of the stack.

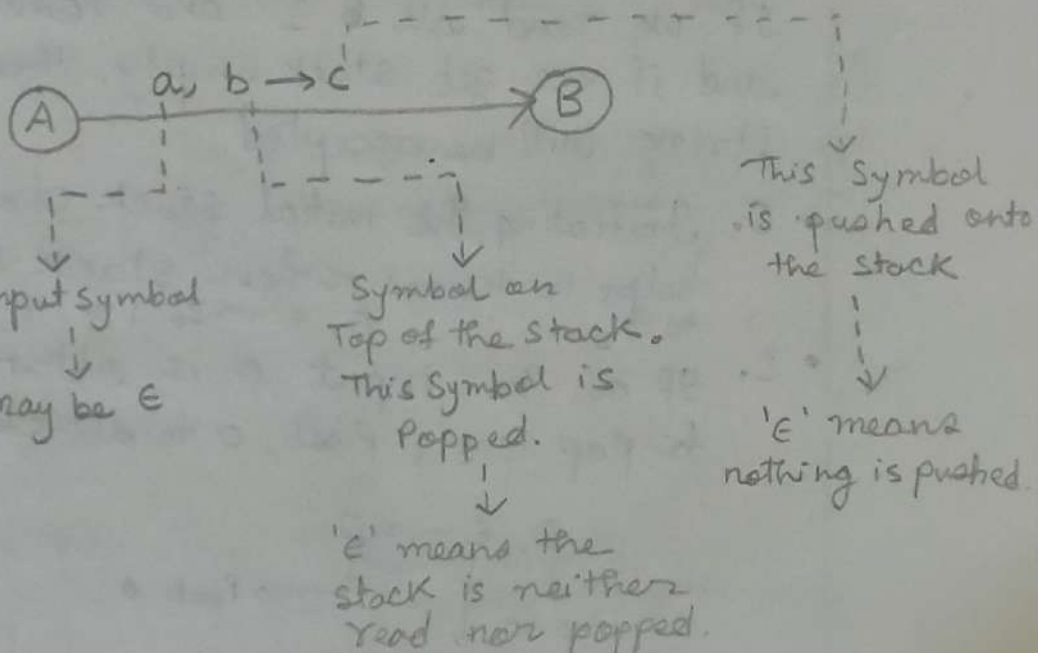
- If $\gamma = \epsilon$, then the stack is popped.
- If $\gamma = X$, then the stack is unchanged.
- If $\gamma = YZ$, then X is replaced by Z and Y is pushed onto the stack.

3.1.3) Graphical Notation of P.D.A:

- finite state machine:



- Push Down Automata:

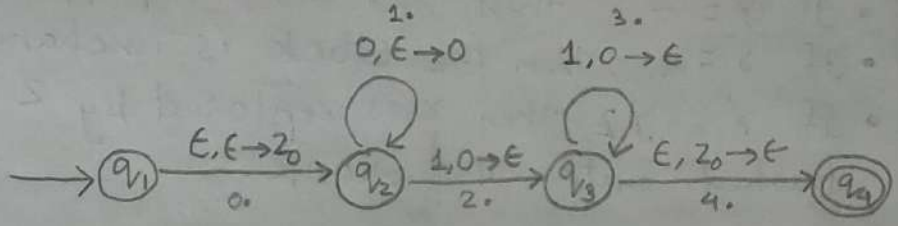


3.2) Construction of P.D.A :

Ex: Construct a PDA that accepts
$L = \{0^n 1^n \mid n \geq 0\}$

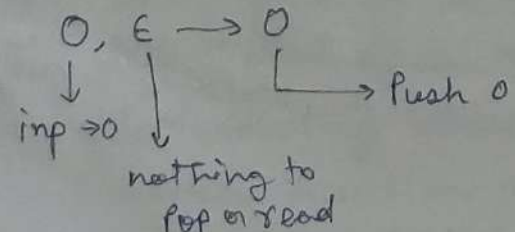
A: $L = \{0^n 1^n \mid n \geq 0\}$
i.e. equal no. of 0s & 1s {0011, 01, 000111...}

*Note: Z_0 is bottom most element of stack, it lets machine know when bottom of stack is reached. Also denoted by $.

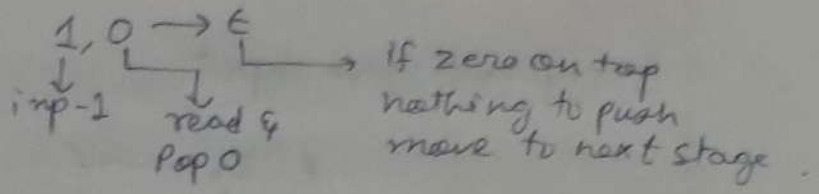


Explaination:

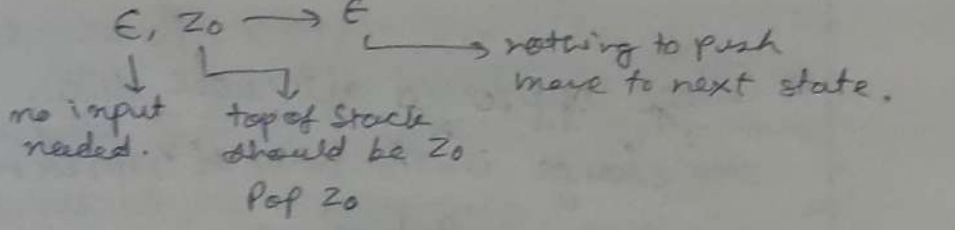
- The machine's logic is; First we will push all 0's onto the stack. Then reading every single 1 each 0 is popped from the stack.
  If we read all 1 and remove all 0's and if we get stack empty, then that string will be accepted.
- 0. Initiating the initial stack character Z_0, this helps to know when stack is empty. move to q_2 $\epsilon, \epsilon \rightarrow Z_0$ | no inp, no read, just Push Z_0.
- 1. If on q_2, input 0 is obtained, nothing to pop ∴ ϵ, Push 0 to stack.



- 2. If on q_2, input is 1, and top of stack is 0, then pop top of stack, nothing to push move to q_3.

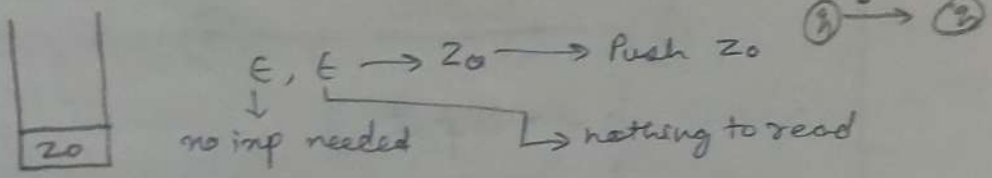


- 3. Same as 2, but move from q_3 to q_3.
- 4. No input required, if top of stack is Z_0, Pop Z_0, nothing to push to stack, move to q_4

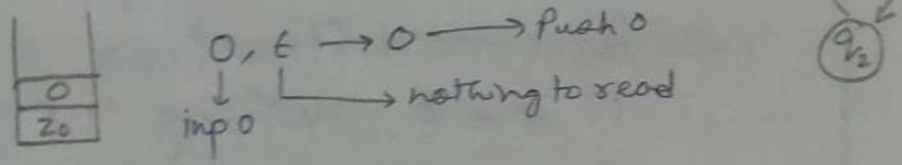


- Lets take example 0011.

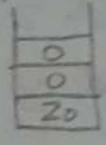
0.



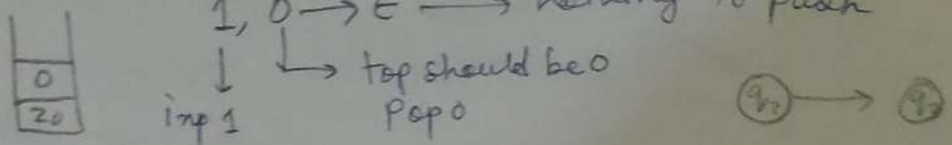
1. inp → 0



1. inp → 0



2. inp → 1

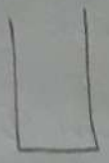


3. inp → 1

$1, 0 \rightarrow \epsilon$

Same as last step

4. inp → ϵ | No input



$\epsilon, Z_0 \rightarrow \epsilon$ → nothing to push

↓ no inp needed

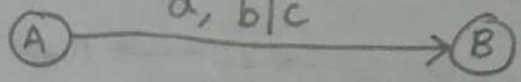
↓ Top should be Z_0 remove Z_0

→ Stack is empty

↓ accepted.

★) College approved alternate way of drawing

Only change is the way of representing transition here representation is



a → input

b → top element to be read & popped

c → element to be pushed

rules for c are different.

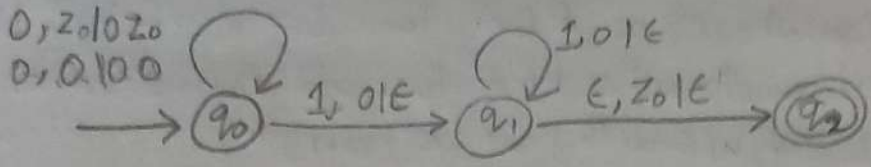
if $c = \epsilon$, nothing to push

if $c = b$, no change, just detect

if $c = yz$, pop b, push z, push y

⊗ if $c = y$, pop b, push y.

The following is the alternate diagram:



Explaination:

0, Z0 | 0 Z0 means for inp 0, if top is Z0 Pop Z0, Push Z0, Push 0.

0, 0 | 0 0 means, for inp 0, if top is 0 Pop 0, Push 0, Push 0.

(follows rules of 3.1.2 Last points) ← *

Transitions for 0011 : (considering initialized)

$\delta(q_0, 0, Z_0) = \{(q_0, 0\,Z_0)\}$ } Pushing 0's

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

$\delta(q_0, 1, 0) = \{(q_1, \epsilon)\}$ } Popping 0's.

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

$\delta(q_1, \epsilon, Z_0) = \{(q_2, \epsilon)\}$ } emptying of stack.

$\vdash$ for 0011 : (Stack symbols)

$\delta(q_0, 0011, Z_0) \rightarrow \vdash (q_0, 011, 0Z_0)$ ←

$\vdash (q_0, 11, 00Z_0)$ ←

$\vdash (q_1, 1, 0Z_0)$ ←

$\vdash (q_1, \epsilon, Z_0)$ ←

$\vdash (q_2, \epsilon, \epsilon)$ ←

→ accepted state.

remaining i/p

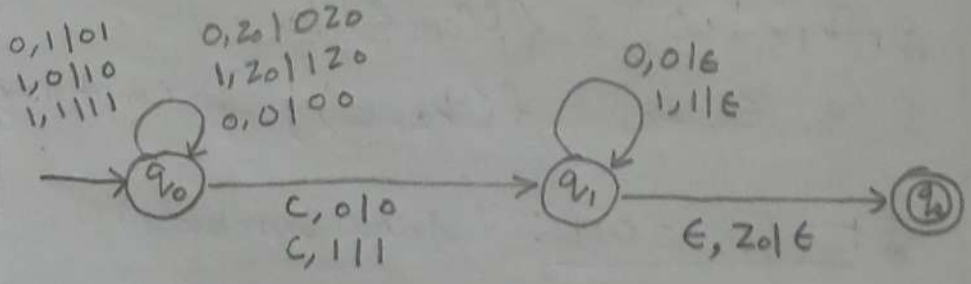
$\vdash (q_0, 0011, Z_0)$

current state, current stack

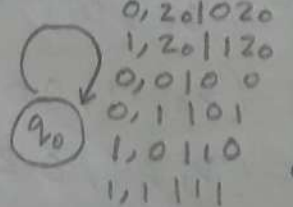
Q. Construct a PDA for $L = \{WC W^R \mid W \text{ in } (0+1)^*\}$
(or) Construct a PDA for odd Palindrome of form
$L = \{WC W^R \mid W \text{ in } (0+1)^*\}$

A:

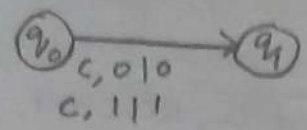
- We need to check if its a palindrome or not
  i.e. $\underbrace{xyz}_{W} C \underbrace{zyx}_{W^R}$
- To check this, we push everything from input to stack till C is obtained.
- when C is obtained, we move to next state.
- In next state, if input matches top of stack, we pop till stack is empty.
- If at any point input & top doesn't match, no transaction is available for such case so it stops & never reaches final state.
- on pop of Z_0, we reach final state.



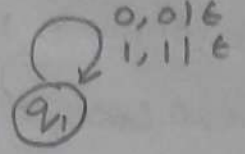
Explaination:



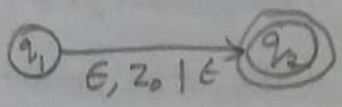
Simply pushes i/p symbol on top of stack
doesn't replace previous top
eg: $0, Z_0 | 0 Z_0$
if ip is 0 & Top is Z_0
Pop Z_0, Push Z_0, Push 0.



} if I/P is c, move to next step keep stack the same



} if I/P matches top of stack pop.



} If stack gets empty move to acceptance state.

Note : Rules of transitions in diagram

1) i, t | ε → if input is i & top is t pop t.
2) i, t | t → if input is i & top is t, no change in stack, move to next state.
3) i, t | yz → if input is i & top is t, replace t by z & push y.

Transition functions:

$\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$
$\delta(q_0, 1, z_0) = \{(q_0, 1z_0)\}$
$\delta(q_0, 0, 0) = \{(q_0, 00)\}$
$\delta(q_0, 0, 1) = \{(q_0, 01)\}$
$\delta(q_0, 1, 0) = \{(q_0, 10)\}$
$\delta(q_0, 1, 1) = \{(q_0, 11)\}$

} Push

Here,
PDA = $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \{q_2\})$
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, z_0\}$

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

} Accept the separator c.

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

} Pop

$\delta(q_1, \varepsilon, z_0) = \{(q_2, \varepsilon)\}$ } final state Acceptance.

Test input 100C001

$\delta(q_0, 100C001, Z_0) \rightarrow \vdash(q_0, 100C001, Z_0)$

$\vdash(q_0, 00C001, 1Z_0)$

$\vdash(q_0, 0C001, 01Z_0)$

$\vdash(q_0, C001, 001Z_0)$

$\vdash(q_1, 001, 001Z_0)$

$\vdash(q_1, 01, 01Z_0)$

$\vdash(q_1, 1, 1Z_0)$

$\vdash(q_1, \epsilon, Z_0)$

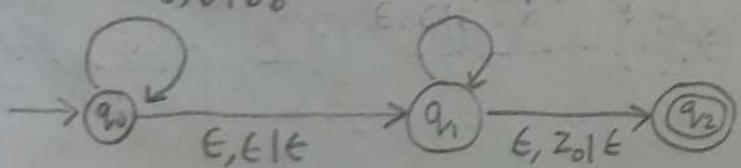
$\vdash(q_2, \epsilon) \rightarrow$ Accepted state

Q. Construct a PDA that accepts Even Palindromes of the form

$$L = \{WW^R \mid W = (a+b)^+\}$$

a, Z0 | a Z0
b, Z0 | b Z0
a, a | aa
a, b | ab
b, a | ba
b, b | bb

a, a | ∈
b, b | ∈



P.D.A = $(Q, \Sigma, \Gamma, \delta, \{q_0\}, \{Z_0\}, \{q_2\})$

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

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

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

(NDPDA)

$\delta(q_0, a, z_0) = \{(q_0, a z_0)\}$
$\delta(q_0, b, z_0) = \{(q_0, b z_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, b, a) = \{(q_0, ba)\}$
$\delta(q_0, a, b) = \{(q_0, ab)\}$
$\delta(q_0, b, b) = \{(q_0, bb)\}$
} Push

$\delta(q_0, \epsilon, \epsilon) = \{(q_1, \epsilon)\}$ } ϵ transition

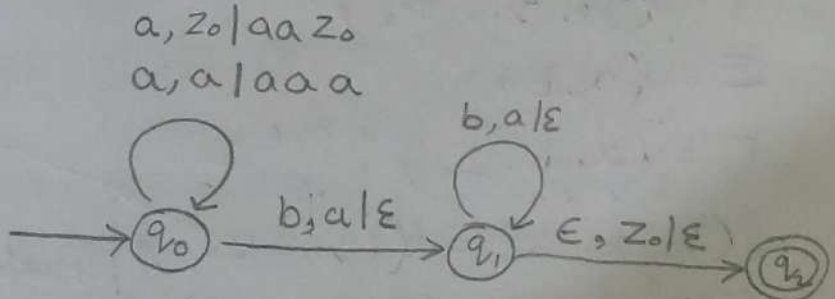
$\delta(q_1, a, a) = \{(q_1, \epsilon)\}$
$\delta(q_1, b, b) = \{(q_1, \epsilon)\}$
} POP

$\delta(q_1, \epsilon, z_0) = \{(q_2, \epsilon)\}$ } acceptance

Q. Construct PDA for the language $L = \{a^n b^{2n} / n \geq 1\}$

A: $L = \{$'n' no. of 'a' followed by '2n' no. of 'b'$\}$

Logic: If we read 'a' we push 2 'a's
If we read 'b' we pop 1 'a'.
If we are left with empty stack, accept



PDA = $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \{q_2\})$

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

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

$\Gamma = \{a, b, z_0\}$

$\delta(q_0, a, z_0) = \{(q_0, aa z_0)\}$
$\delta(q_0, a, a) = \{(q_0, aaa)\}$
] Push

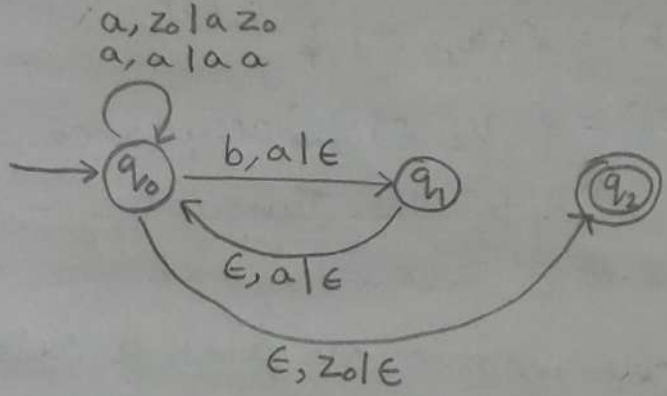
$\delta(q_0, b, a) = \{(q_1, \epsilon)\}$
$\delta(q_1, b, a) = \{(q_1, \epsilon)\}$
] Pop

$\delta(q_1, \epsilon, z_0) = \{(q_2, \epsilon)\}$] acceptance.

Q. Construct a PDA for the language $L=\{a^{2n}b^n / n\geq 1\}$
Trace your PDA for the input with $n=2$

A: $L = \{$ '2n' no. of 'a's & 'n' no. of bs $\}$

Logic: We add all 2n a's to Stack
for each b read, we pop a twice by poping a once while transition $q_0 \rightarrow q_1$ & once while the ϵ transition $q_1 \rightarrow q_0$.



$PDA = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \{q_2\})$

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

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

$\Gamma = \{a, b, z_0\}$

$\left.\begin{array}{l}\delta(q_0, a, z_0) = \{(q_0, a z_0)\} \\ \delta(q_0, a, a) = \{(q_0, aa)\}\end{array}\right]$ Push

$\left.\begin{array}{l}\delta(q_0, b, a) = \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, a) = \{(q_0, \epsilon)\}\end{array}\right]$ Pop

$\delta(q_0, \epsilon, z_0) = \{(q_2, \epsilon)\}$] acceptance.

for $n = 2$

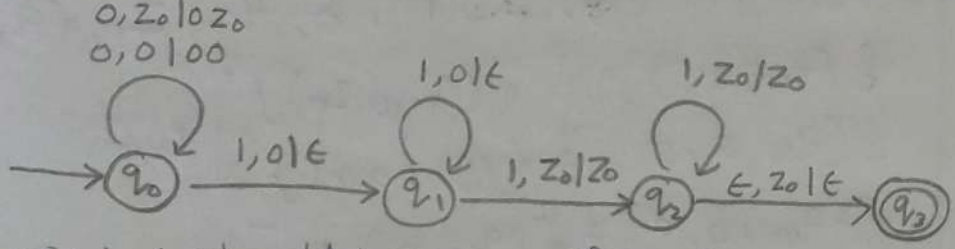
$L_n = aaaabb$

$\therefore$ I/P = aaaabb

$\vdash (q_0, aaaabb, Z_0)$
$\vdash (q_0, aaabb, aZ_0)$
$\vdash (q_0, aabb, aaZ_0)$
$\vdash (q_0, abb, aaaZ_0)$
$\vdash (q_0, bb, aaaaZ_0)$
$\vdash (q_1, b, aaaZ_0)$
$\vdash (q_0, b, aaZ_0)$
$\vdash (q_1, \epsilon, aZ_0)$
$\vdash (q_0, \epsilon, Z_0)$
$\vdash (q_2, \epsilon)$ Accepted state.

Q. Construct the PDA for the language
$L = \{0^n 1^m \mid n < m \text{ and } n, m \geq 1\}$

A:



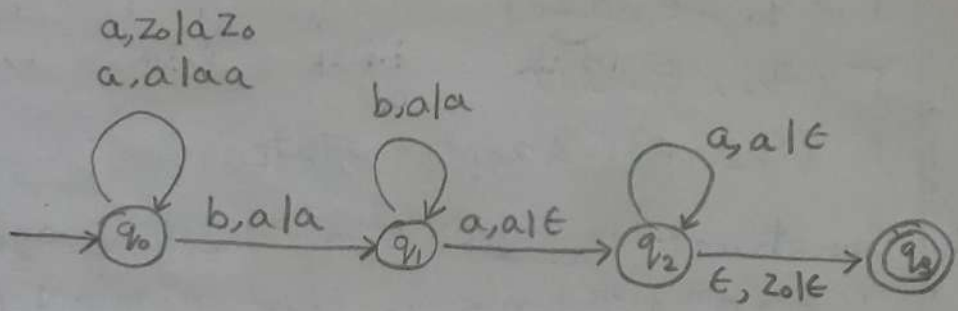
Logic: 1 should be more than 0.
first push all 0
then for each 1, Pop 0 if the top is 0
don't pop if the top is Z_0
for state to reach q_2, there must be an input of 1 when all 0's are popped, i.e. more 1's than 0's is needed for acceptance.

PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{q_3\})$
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, Z_0\}$
$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$
$\delta(q_0, 0, 0) = \{(q_0, 00)\}$] Push

$\delta(q_0, 1, 0) = \{(q_1, \epsilon)\}$]
$\delta(q_1, 1, 0) = \{(q_1, \epsilon)\}$] Pop
$\delta(q_1, 1, Z_0) = \{(q_2, Z_0)\}$]
$\delta(q_2, 1, Z_0) = \{(q_2, Z_0)\}$] make sure $m > n$ (i.e. overflow check)
$\delta(q_2, \epsilon, Z_0) = \{(q_3, \epsilon)\}$] acceptance

Q. Construct the PDA for the language $L = \{a^n b^m a^n \mid m, n \geq 1\}$

A:



PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{q_3\})$

$\delta(q_0, a, Z_0) = \{(q_0, a Z_0)\}$]
$\delta(q_0, a, a) = \{(q_0, aa)\}$] Push
$\delta(q_0, b, a) = \{(q_1, a)\}$]
$\delta(q_1, b, a) = \{(q_1, a)\}$] ignore
$\delta(q_1, a, a) = \{(q_2, \epsilon)\}$]
$\delta(q_2, a, a) = \{(q_2, \epsilon)\}$] Pop
$\delta(q_2, \epsilon, Z_0) = \{(q_3, \epsilon)\}$] acceptance.

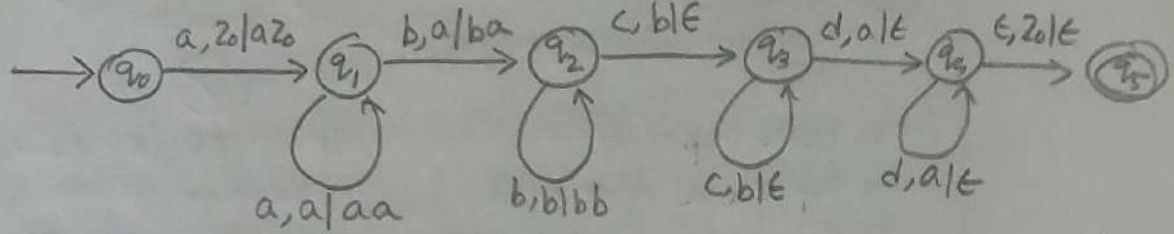
$\vdash (q_0, aabbaa, Z_0)$
$\vdash (q_0, abbaa, aZ_0)$
$\vdash (q_0, bbaa, aaZ_0)$
$\vdash (q_1, baa, aaZ_0)$
$\vdash (q_1, aa, aaZ_0)$

$\vdash (q_2, a, aZ_0)$

$\vdash (q_3, \theta\epsilon, Z_0)$

$\vdash (q_4, \epsilon)$ Accepted.

Q. Construct the PDA for the language
$L = \{a^n b^m c^m d^n \mid m, n \geq 1\}$



PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{q_3\})$

$\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$] Push a

$\delta(q_1, a, a) = \{(q_1, aa)\}$]

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

$\delta(q_2, b, b) = \{(q_2, bb)\}$]

$\delta(q_2, c, b) = \{(q_3, \epsilon)\}$] Pop b for each c

$\delta(q_3, c, b) = \{(q_3, \epsilon)\}$]

$\delta(q_3, d, a) = \{(q_4, \epsilon)\}$] Pop a for each d

$\delta(q_4, d, a) = \{(q_4, \epsilon)\}$]

$\delta(q_4, \epsilon, Z_0) = \{(q_5, \epsilon)\}$] acceptance

$\vdash (q_0, abbccd, Z_0)$

$\vdash (q_1, bbccd, aZ_0)$

$\vdash (q_2, bccd, baZ_0)$

$\vdash (q_2, ccd, bbaZ_0)$

$\vdash (q_3, cd, baZ_0)$

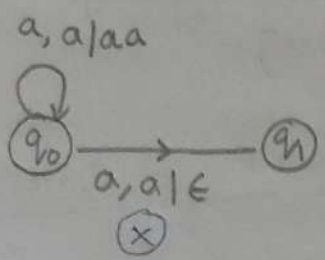
$\vdash (q_3, d, aZ_0)$

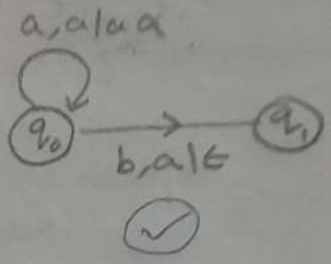
$\vdash (q_4, \epsilon, Z_0)$

$\vdash (q_5, \epsilon)$ Accepted.

3.3) Deterministic Pushdown Automata:

A PDA is said to be deterministic if all derivations in the design give only single move.





here Same inp a & Same top ele a give diff move

i.e. $\delta(q_0, a, a) = \{(q_0, aa), (q_1, \epsilon)\}$

It is DPDA since all derivation give single move

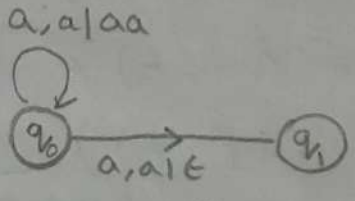
i.e. $\delta(q_0, a, a) = \{(q_0, aa)\}$ $\delta(q_0, b, a) = \{(q_1, \epsilon)\}$

Hence it is not DPDA
it is ND PDA

3.4) Non Deterministic Push down Automata:

A PDA is Non deterministic if with one transition we can take more than one move.

eg:

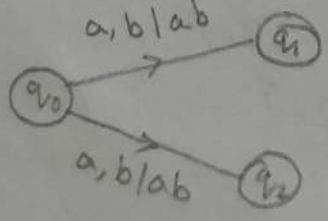


$\delta(q_0, a, a) = \{(q_0, aa), (q_1, \epsilon)\}$

here same delta transition/derivation given more than one move.

There are two ways in which PDA can be Non-Deterministic

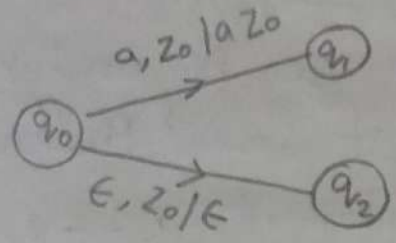
i) If two or more edges are labeled with same input and stack symbol.



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

Prev eg falls under this case.

ii) when a stack state have two edges with same stack symbol and one input symbol is ϵ.



here $\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$.
$\delta(q_0, \epsilon, Z_0) = \{(q_2, \epsilon)\}$
both derivation give single move, still it is NDPDA, since ϵ-move is present.

3.5) Comparision NFA Vs PDA :

NFA	PDA
• NFA stands for Non-deterministic Finite Autometa.	• PDA stands for Push Down Autometa.
• This model doesnot have memory to remember input symbols.	• This model has stack memory to remember input symbols.
• This model accepts Regular Language	• This model accepts Regular & context free language.
• It is always Non Deterministic	• It can be both deterministic & non deterministic.
• It has two types : - NFA with ϵ - NFA without ϵ	• It has two types : - Deterministic PDA - Non Deterministic PDA

3.6) Equivalance, CFG to PDA

3.6.1) Equivalance of CFG & PDA:

Theorem: A language is Context Free if & only if some Pushdown Automata recognises it.

Two different proofs:

(i) Given a CFG, show how to construct a PDA that recognises it

(ii) Given a PDA, show how to construct a CFG that regognises the same language

3.6.2) Constructing PDA from given CFG (CFG to PDA)

★) **Algorithm 1 : (via GNF)**

Step-1: check if given CFG is in GNF (Greibach Normal Form), if not, convert to GNF (2.9)

Step-2: For every Production rule of CFG in form $A \to aB$, add a derivation

$$\delta(q, a, A) = \{(q, B)\}$$

Step-3: For every production rule of CFG in form $A \to a$, add a derivation

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

Step-4: Finally add the acceptance rule (no need if $Z_0 \to S$)

$$\delta(q, \epsilon, Z_0) = \{(q, \epsilon)\}$$ # empty stack gives Z_0 i.e. $Z_0 \to \phi$

Step-5: PDA: $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q \to q$ (only one state) $q_0 \to q$

$\Sigma \to$ i/p symbols (Terminals) $Z_0 \to$ Starting state of CFG

$\Gamma \to$ Variables of CFG $F \to \phi$ (acceptance by empty stack)

Q. Construct PDA for the following grammar
$S \rightarrow AB$, $B \rightarrow b$, $A \rightarrow CD$, $C \rightarrow a$, $D \rightarrow a$

A: Step-1 : • given CFG is not in GNF, so we convert it to GNF.
• $S \rightarrow AB$, $A \rightarrow CD$ do not obey GNF rule
• Substitute the problamatic variable in $A \rightarrow CD$
$A \rightarrow aD$ ∵ $C \rightarrow a$
• Now substitute the problamatic variable in S
$S \rightarrow aDB$ ∵
• New GNF is
$S \rightarrow aDB$, $B \rightarrow b$, $A \rightarrow aD$, $C \rightarrow a$, $D \rightarrow a$

Step-2 : let PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$P = (q, \{a, b\}, \{S, A, B, C, D\}, \delta, q, S, \phi)$

Step-3 : Create the δ transitions by following the algorithm:

$\delta(q, a, S) = \{(q, DB)\}$
$\delta(q, b, B) = \{(q, \epsilon)\}$
$\delta(q, a, A) = \{(q, D)\}$
$\delta(q, a, C) = \{(q, \epsilon)\}$
$\delta(q, a, D) = \{(q, \epsilon)\}$

Step-4 : Take an example

eg : $\Gamma(q, aaab, S)$
$\Gamma(q, aab, DB)$
$\Gamma(q, ab, B)$
$\Gamma(q, b, \phi)$
$\Gamma(q, \epsilon, \phi)$ Accepted.

Q. Construct an unrestricted PDA equivalent of the grammar given below

$S \rightarrow aAA$, $A \rightarrow aS \mid bS \mid a$

A: 1) grammar is already in GNF

2) let PDA $P = (q, \{a, b\}, \{S, A\}, \delta, q, S, \phi)$

$\delta(q, a, S) = \{(q, AA)\}$

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

$\delta(q, b, A) = \{(q, S)\}$

3) The simulation of a baaaaa is

$\Gamma(q, abaaaa, S)$

$\Gamma(q, baaaa, AA)$

$\Gamma(q, aaaa, SA)$

$\Gamma(q, aaa, AAA)$

$\Gamma(q, aa, AA)$

$\Gamma(q, a, A)$

$\Gamma(q, \epsilon, \phi)$ Accepted.

rejected Part (∵ NB PDA)

$\Gamma(q, aa, SA)$

$\Gamma(q, a, AAA)$

$\Gamma(q, \epsilon, AA)$ ✗ $\Gamma(q, \epsilon, SAA)$ ✗

★) Algorithm 2: (Non GNF CFG) (S.C.)

Step-1: check if given CFG is not GNF. If its GNF, use algorithm-1.

Step-2: PDA $(Q, \Sigma, \Gamma, \delta, Z_0, q_0, F)$ is

$(q,$ Terminals (Σ), Terminals ∪ Variables (Γ), $\delta, S, q, \phi)$

Step-3: for any production $A \rightarrow aBc$

add drivation $\delta(q, \epsilon, A) = \{(q, aBc)\}$

or for $A \to x \quad \forall \; x \in (T \cup V)$ | T → terminals, V → variables

$\delta(q, \epsilon, A) = \{(q, x)\}$

Step-4 : for all Terminals t, add derivations

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

ie. for t in Terminals:

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

eg: Convert the CFG with P: $S \to aSb \mid ab$ to PDA

A: PDA P: $(q, \{a, b\}, \{a, b, S\}, \delta, S, q, \phi)$

$\delta(q, \epsilon, S) = \{(q, aSb)\}$
$\delta(q, \epsilon, S) = \{(q, ab)\}$
] Step 3 ε transitions.

$\delta(q, a, a) = \{(q, \epsilon)\}$
$\delta(q, b, b) = \{(q, \epsilon)\}$
] Step 4.

eg: $\Gamma(q, aaabbb, S)$

$\Gamma(q, aaabbb, aSb)$ | ε transition

$\Gamma(q, aabbb, Sb)$

$\Gamma(q, abbbb, aSbb)$ | ε transition

$\Gamma(q, abbb, Sbb)$

$\Gamma(q, abbb, aSbbb)$ | ε transition.

$\Gamma(q, bbb, bbb)$

$\Gamma(q, bb, bb)$

$\Gamma(q, b, b)$

$\Gamma(q, \epsilon, \phi)$ Accepted.

Q. Consider the CFG $G = (\{S, T, C, D\}, \{a, b, c, d\}, S, P)$ where P is

$S \to cCD \mid dTC \mid \epsilon$ $\quad C \to aTD \mid \epsilon$

$T \to cDC \mid cST \mid a$ $\quad D \to dC \mid d$

Present a PDA that accepts the language generated by this grammar.

A: Solving using Algorithm-2

$P = (q, \{a, b, c, d\}, \{a, b, c, d, S, T, C, D\}, \delta, q, S, \phi)$

$\delta(q, \epsilon, S) = \{(q, cCD), (q, dTC), (q, \epsilon)\}$

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

$\delta(q, \epsilon, C) = \{(q, aTD), (q, \epsilon)\}$

$\delta(q, \epsilon, D) = \{(q, dC), (q, d)\}$

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

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

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

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

Simulating for "caadd"

$\Gamma(q, caadd, S)$
$\Gamma(q, caadd, cCD)$
$\Gamma(q, aadd, CD)$
$\Gamma(q, aadd, aTDD)$
$\Gamma(q, add, TDD)$
$\Gamma(q, add, aDD)$
$\Gamma(q, dd, DD)$
$\Gamma(q, dd, dD)$
$\Gamma(q, d, D)$
$\Gamma(q, d, d)$
$\Gamma(q, \epsilon, \phi)$

Accepted.

★ 3.7) Conversion PDA to CFG :

Note : While process of conversion, we will be dealing with triplets of form $[q_i, X, q_j]$ where $q_i, q_j \in Q$ & $X \in \Gamma$, these triplets will later be renamed as variables such as A, B, X, Y, Σ symbols will become the terminals.

Algorithm :

Step-1 : add production $S \rightarrow [q_0, Z_0, q_j]$ where $q_j \in Q$ i.e. substitute every possible states (Q) in $[q_0, Z_0, q_j]$

eg: if $Q = \{q_0, q_1, q_2\}$

$S \rightarrow [q_0, Z_0, q_0]$
$S \rightarrow [q_0, Z_0, q_1]$
$S \rightarrow [q_0, Z_0, q_2]$

] q_0, Z_0 is fixed in this triplet.

Step-2 : for each derivation $\delta(q_m, A, X) = (q_n, B_1 B_2 \dots B_i)$ (i/p Σ terminal; $B_1 B_2 \dots \neq \epsilon$) we add a production

$$[q_m, X, q_{i+1}] \rightarrow A\,[q_n, B_1, q_2]\,[q_2, B_2, q_3]\,[q_3, B_3, q_4] \dots [q_i, B_i, q_{i+1}]$$

This is a complicated step, so let's break it down.

- $[q_m, X, \_] \rightarrow A\,[q_n, B_1, \_]\,[\_, B_2, \_] \dots [\_, B_3, \_] \dots [\_, B_i, \_]$

  - First we write the values which are fixed. Here q_m is the current state. X is top of stack, A is i/p, $B_1 B_2 \dots B_i$ are the new value to push in stack.
  - The no. of triplet in RHS depend on no. of symbols being pushed in stack i.e. i.
  - The blank spaces are to be filled with every possible combination of states from Q, with a pattern which is the last value of each triplet is first value of next, and the last element of last triplet is same as last ele of LHS.

- So, we start by leaving the blank spaces & writing the fixed values. Then we copy the production with blanks & fill them with every possible value of from Q.

eg: Let's say $Q = \{q_0, q_1\}$

we've $\delta = (q_0, a, Z_0) = (q_0, X_1 X_2 Z_0)$

∴ we add the following production first

$$[q_0, Z_0, \_] \rightarrow a\,[q_0, X_1, \_]\,[\_, Z_0, \_]$$

here B_1, B_2 are X, Z_0

only two triplets in RHS since only 2 values to be pushed on i=2.

Now we put every possible combination of $\{q_0, q_1\}$ in the blank spaces, such that same values are in — — & == (it should follow the rule last of prev triplet is first of current, last of last triplet is last of LHS)

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

Step-3: for every derivation of form $\delta(q_m, A, X) = \{(q_n, \epsilon)\}$ add a production

$$[q_m, X, q_n] \rightarrow A$$

Step-4: Remove the useless productions from step-2.

- Some triplet are present only in RHS & never in LHS. remove all production having those triplets.
- Some triplet in LHS repeat multiple times in RHS and are not possible to terminate, remove

all production with these triplets.

Step-5: rename the triplets as capital alphabets & re-write the production rules.

Q. Let $M = (\{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, \delta, q_0, Z_0, \phi)$ where δ is given by

$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$
$\delta(q_0, 0, X) = \{(q_0, XX)\}$
$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$
$\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$
$\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$
$\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

Construct CFG $G = (V, T, P, S)$ generating $N(M)$

A: 1. $S \longrightarrow [q_0, Z_0, q_0]$ X
$S \longrightarrow [q_0, Z_0, q_1]$] Step-1

2. $\delta(q_0, 0, Z_0) = (q_0, XZ_0)$ # Step 2

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

2. $\delta(q_0, 0, X) = (q_0, XX)$ # Step 2

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

4. $\delta(q_0, 1, X) = (q_1, \epsilon)$ // Step 3

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

5. $\delta(q_1, 1, X) = (q_1, \epsilon)$ // Step 3

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

6. $\delta(q_1, \epsilon, X) = (q_1, \epsilon)$ // Step 3

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

7. $\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$ // Step 3.

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

9. Useless States found:

$[q_1, Z_0, q_0]$ | not in LHS

$[q_0, Z_0, q_0]$ | non terminable after removal of $[q_1, Z_0, q_0]$

$[q_1, X, q_0]$ | not in LHS

$[q_0, X, q_0]$ | non terminable after removal

$[q_0, X, q_0]$ | no longer in RHS.

10. remaining productions are:

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

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

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

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

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

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

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

11. After renaiming

$S \longrightarrow [A$
$A \longrightarrow oBC$
$B \longrightarrow oBD \mid 1$
$D \longrightarrow 1 \mid \epsilon$
$C \longrightarrow \epsilon$

Q. Convert the following PDA to CFG

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

where δ is given by

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$
$\delta[q_0, \epsilon, Z_0) = \{(q_0, \epsilon)\}$
$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$
$\delta(q_0, a, Z) = \{(q_1, Z)\}$
$\delta(q_1, b, Z) = \{(q_1, \epsilon)\}$
$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$

A: 1. $S \longrightarrow [q_0, Z_0, q_0]$
$S \longrightarrow [q_0, Z_0, q_1]$ X | N.T

2. $\delta(q_0, b, Z_0) = (q_0, ZZ_0)$

$[q_0, Z_0, q_0] \Rightarrow b\,[q_0, Z, q_0]\,[q_0, Z_0, q_0]$ X | N.T
$[q_0, Z_0, q_0] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z_0, q_0]$
$[q_0, Z_0, q_1] \rightarrow b\,[q_0, Z, q_0]\,[q_0, Z_0, q_1]$ X | N.T
$[q_0, Z_0, q_1] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z_0, q_1]$ X | N.T

3. $\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$

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

4. $\delta(q_0, b, Z) = (q_0, ZZ)$

$[q_0, Z, q_0] = b\,[q_0, Z, q_0]\,[q_0, Z, q_0]$ X | N.T
$[q_0, Z, q_0] = b\,[q_0, Z, q_1]\,[q_1, Z, q_0]$ X | N.R.H.S
$[q_0, Z, q_1] = b\,[q_0, Z, q_0]\,[q_0, Z, q_1]$ X | N.T
$[q_0, Z, q_1] = b\,[q_0, Z, q_1]\,[q_1, Z, q_1]$

5. $\delta(q_0, a, Z) = (q_1, Z)$

$[q_0, Z, q_0] \Rightarrow a\,[q_1, Z, q_0]$ X | N.R.H.S
$[q_0, Z, q_1] \rightarrow a\,[q_1, Z, q_1]$

6. $\delta(q_1, b, Z) = (q_1, \epsilon)$

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

7. $\delta(q_1, a, Z_0) = (q_0, Z_0)$

$[q_1, Z_0, q_0] \longrightarrow a\,[q_0, Z_0, q_0]$
$[q_1, Z_0, q_1] \longrightarrow a\,[q_0, Z_0, q_1]$ X | N.T

∴ remaining productions

8. $S \longrightarrow [q_0, Z_0, q_0]$

$[q_0, Z_0, q_0] \longrightarrow b\,[q_0, Z, q_1]\,[q_1, Z_0, q_0]$
$[q_0, Z_0, q_0] \longrightarrow \epsilon$
$[q_0, Z, q_1] \longrightarrow b\,[q_0, Z, q_1]\,[q_1, Z, q_1]$
$[q_0, Z, q_1] \longrightarrow a\,[q_1, Z, q_1]$
$[q_1, Z, q_1] \longrightarrow b$
$[q_1, Z_0, q_0] \longrightarrow a\,[q_0, Z_0, q_0]$

After renaming

$S \longrightarrow A$
$A \longrightarrow \in | bBC$
$B \longrightarrow bBD | aD$
$C \longrightarrow aA$
$D \longrightarrow b$

3.8) Pumping Lemma for CFL:

- Pumping Lemma for CFL is used to prove that a language is NOT context Free.
- If A is a context Free Language, then, A has a pumping length 'p' such that any String 's', where $|s| \geq p$ ($|s|$ → size of s) may be divided into 5 pieces $s = uvxyz$ such that the following Conditions must be true:
  (1) uv^ixy^iz is in A for every $i \geq 0$
  (2) $|vy| > 0$
  (3) $|vxy| \leq p$

Steps to prove:

1) For given CFL. A, assume A is Context Free
2) take a pumping length P.
3) take a string S such that $|S| \geq p$
4) divide 'S' into $uvxyz$
5) Show that $uv^ixy^iz \notin A$ for some i.
6) Consider ways that S can be divided into $uvxyz$
7) Show that none can satisfy all 3 conditions.

Q. Show that $L = \{a^N b^N c^N \mid N \geq 0\}$ is Not Context Free

A: Lets assume L is context free.
Lets assume pumping length of L is P
We take a string from L $S = a^P b^P c^P$
for $P = 4$, $S = a^4 b^4 c^4$
$S = aaaa\ bbbb\ cccc$

Case-1: V & Y contain Same type of symbol.

$S = \underbrace{a}_{u}\underbrace{a\ aa}_{v}\underbrace{b\ bbb\ c\ c}_{x}\underbrace{c}_{y}\underbrace{cc}_{z}$

$uv^i xy^i z$ for $i = 2 = aaaaa\ abbbbc\ cc\ cc$

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

$\therefore$ Case-1 failed

Case-2: either V or Y has more than one kind of symbol.

$S = \underbrace{aaaa}_{u}\underbrace{b\ b\ b}_{v}\underbrace{b}_{x}\underbrace{c}_{y}\underbrace{c\ c\ c}_{z}$

$uv^i xy^i z$ for $i = 2$ $aa\ aabbaa\ bb\ bbbcccc$

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

$\therefore$ case-2 failed.

$\therefore$ Both cases fail
Hence, L is not context Free.

Unit-4

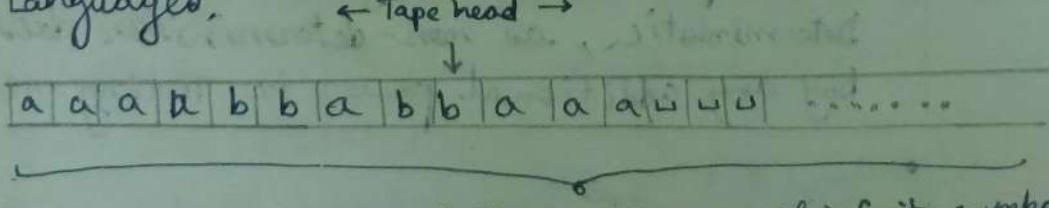
Turing Machine.

Pg-69

★) Topics:

4.1) About Turing Machine:

- In 1936, A.M Turing Proposed the Turing Machine as a model of "any possible computation".
- The Language it accepts is Recursively Enumerable Languages.

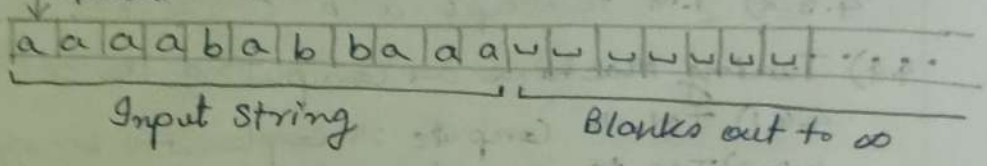


A Tape: Sequence of infinite symbols.

- Tape head can move 1 step to Left or right.
- Tape Alphabet $\Sigma = \{0, 1, a, b, X, Z_0\}$

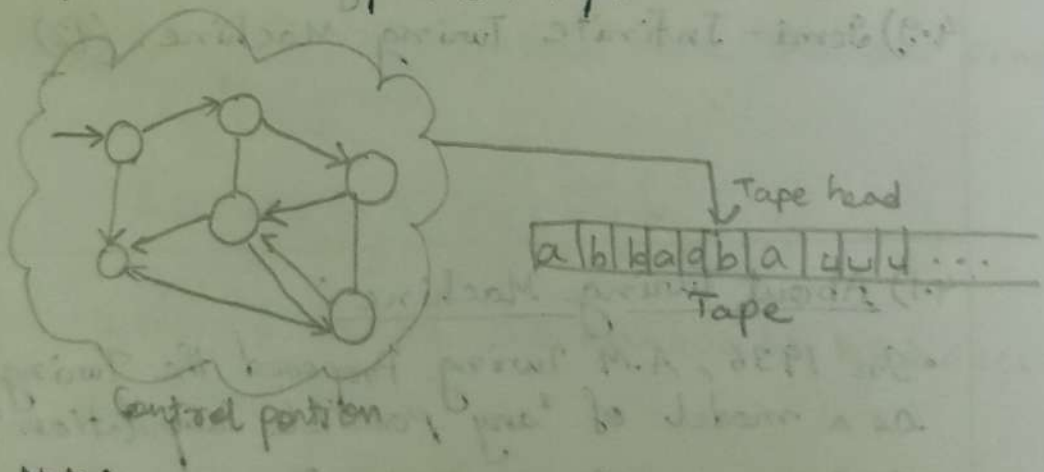
- The Blank ␣ / B is a special symbol, ␣ ∉ Σ
  The blank is a special symbols used to fill empty cells.

- Initial Configuration :



- Operations on the Tape :
  → Read / scan the symbol below the tape head.
  → Update / write a symbol below the tape head.
  → Move the Tape head to one step Left.
  → Move the Tape head to one step right.

- The Control portion of turing machine controlls the operations on Tape, (if tape is Stack, this is the PDA)



Note:
Control portion of Turing Machine is always Deterministic, as non-determinism does not buy any additional power in Turing Machine.

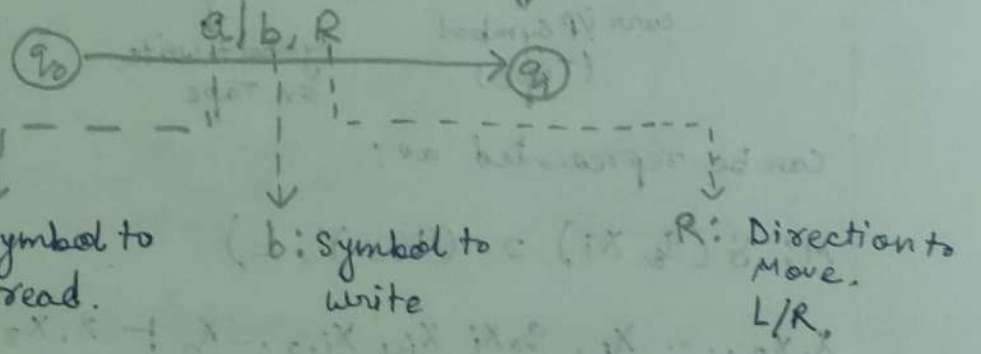
-

- **Rules of each operation:**

At each step of the computation:

→ Read the current symbol
→ Update (i.e. write) the same cell
→ Move exactly one cell to Right or Left.

Note: If we are at the Left end of the tape, and try to move Left, then do not move. Stay on the left end.



Note: If you don't want to update the cell, Just write the same Symbol.

- Computation can either
  1) Halt & Accepted → if stops at accepting state
  2) Halt & Rejected → if stops at non accepting state
  *3) Loop (the machine fails to halt)

4.2) Formal Notation of Turing Machine:

A Turing Machine can be defined as a set of 7 tuples $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$

Q → Set of States (Not Null)
Σ → Set of Symbols (Not Null) (i/p symbols)
Γ → Set of tape Symbols (Not Null) (all Tape symbols)
δ → Transition f^n defined as

$$\delta(Q, \Sigma) \rightarrow \Gamma, (R/L), Q$$

q_0 ⟶ initial state
b ⟶ Blank symbol
F ⟶ Set of Final states (Accept state & Reject state)

The production rules of Turing Machine will be written as:

$$\delta(q_0, a) \longrightarrow (q_1, Y, R)$$

- q_0 → curr state
- a → curr i/p symbol (Tape)
- q_1 → next state
- Y → Symbol to write on Tape
- R → direction to move.

can be represented as:

$$M: \delta(q_0, X_i) = (q_1, Y, L)$$

$$X_1 X_2 \ldots X_{i-1}\ q_0 X_i\ X_{i+1} X_{i+2} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-2}\ q_1 X_{i-1}\ Y\ X_{i+1} X_{i+2} \ldots X_n$$

★ Note:

Turing's Thesis:

Turing's Thesis states that any computation that can be carried out by mechanical machine can be performed by some Turing machine.

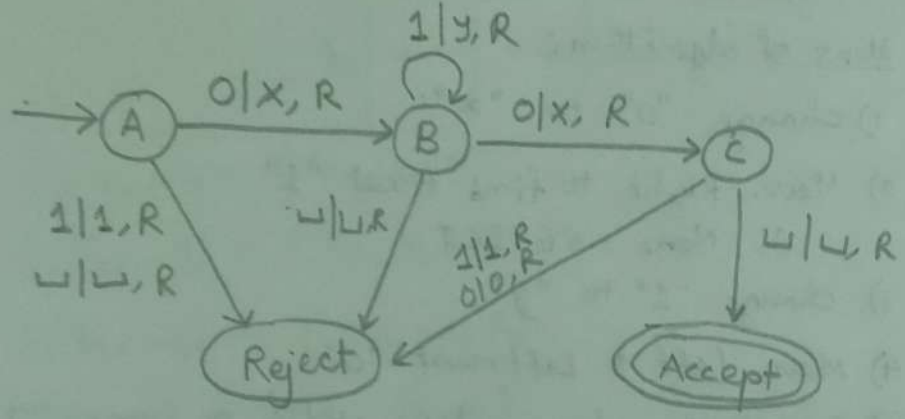
i.e.

- Anything that can be done on an existing digital computer can also be done by turing machine.
- No one has yet been able to suggest a problem solvable by what we consider an algorithm, for which a Turing machine program cannot be written.

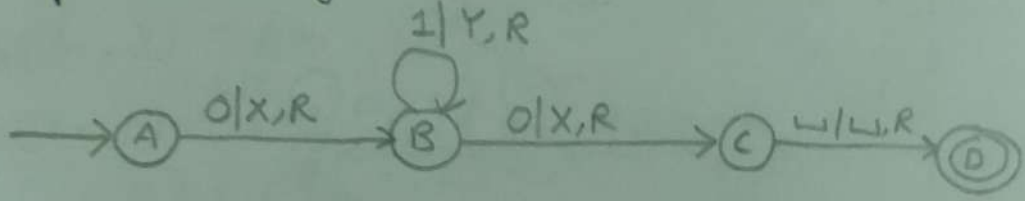
4.3) Turing Machine as Acceptor

Q. Design a turing machine which recognizes the language $L = 01^*0$

A: (This is a Regular Language, so we can solve it using DFA, PDA or T.M)



Simplified design:



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

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

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

$\Gamma = \{0, 1, X, Y\}$

$q_0 = \{A\}$

$b = \{\sqcup\}$

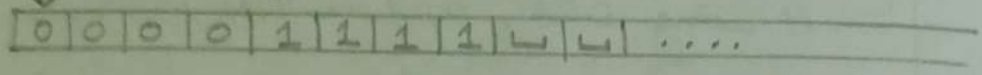
$F = \{D\}$

$\delta \Rightarrow$

	0	1	␣
A	(B, X, R)	–	–
B	(B, X, R)	(B, Y, R)	–
C	–	–	(D, ␣, R)
D	–	–	–

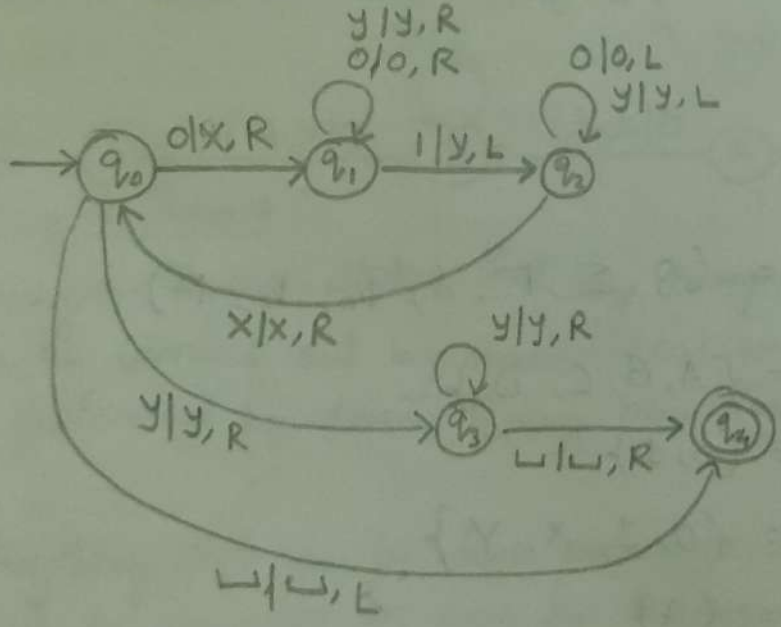
4.3.1Q. Design a Turing Machine which recognizes the language $L = 0^N 1^N$.

A: The following is the algorithm to solve the problem using turing machine:



Steps of algorithm:

1) change "0" to "x"
2) Move Right to find first "1"
   If None : REJECT
3) change "1" to "y"
4) Move Left to Leftmost "0"
5) Repeat the above steps until no more "0"
6) make sure no more "1"s remain.



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

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

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

$\Gamma = \{0, 1, x, y\}$

$q_0 = \{q_0\}$

$b = \{\sqcup\}$

$F = \{q_4\}$

$\delta \Rightarrow$

	0	1	X	Y	$\sqcup$
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	$(q_4, \sqcup, R)$
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)	$(q_4, \sqcup, R)$
q_4	—	—	—	—	—

$\delta(q_0, 0) = (q_1, X, R)$

$\delta(q_0, Y) = (q_3, Y, R)$

$\delta(q_0, \sqcup) = (q_4, \sqcup, R)$

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

$\delta(q_1, 1) = (q_2, Y, L)$

$\delta(q_1, Y) = (q_1, Y, R)$

$\delta(q_2, 0) = (q_2, 0, L)$

$\delta(q_2, X) = (q_0, X, R)$

$\delta(q_2, Y) = (q_2, Y, L)$

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

$\delta(q_3, \sqcup) = (q_4, \sqcup, R)$

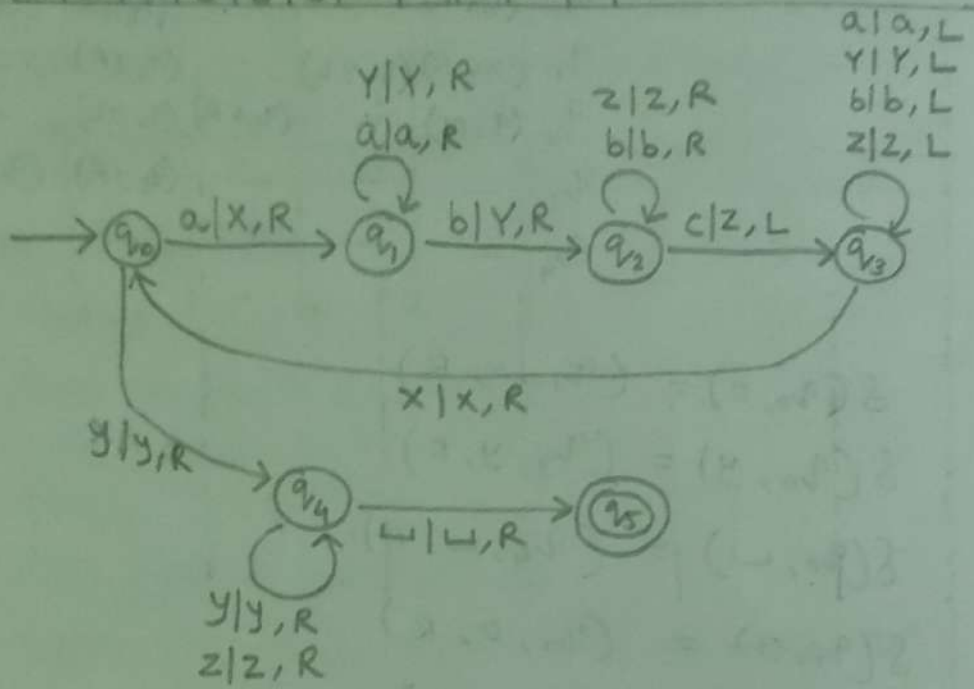
Note: • whenever there is an input in a state for which there is no transition, its a reject due to halt.

• $\xleftarrow[X, R]{X}$ is how we make sure we have the left most "0", we also move the head right to avoid overshooting.

4.3.2 Q. Design a Turing Machine which recognises the language $L = \{a^n b^n c^n \mid n \geq 1\}$

A:

a	a	a	b	b	b	c	c	c	⊔	⊔	...
↓											



Algorithm: Similar as previous one

1) change a to X
2) move right till b is reached
3) change b to Y
4) move right till c is reached
5) change c to Z
6) move left till x is hit, then move right once
7) Repeat from (1) until no more a left
8) make sure no more b & c are left as well

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

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

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

$\Gamma = \{a, b, c, X, Y, Z\}$

$q_0 = \{q_0\}$

$b = \{\sqcup\}$

$F = \{q_5\}$

$\delta \Rightarrow$

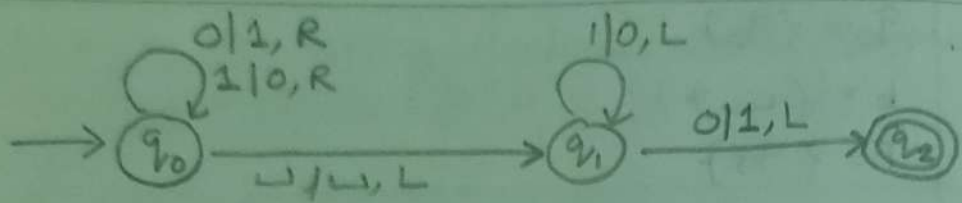
St	a	b	c	X	Y	Z	$\sqcup$
q_0	(q_1, X, R)	–	–	–	(q_4, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, R)	–	–	(q_1, Y, R)	–	–
q_2	–	(q_2, b, R)	(q_3, Z, L)	–	–	(q_2, Z, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	–	(q_0, X, R)	(q_3, Y, L)	(q_3, Z, L)	–
q_4	–	–	–	–	(q_4, Y, R)	(q_4, Z, R)	$(q_5, \sqcup, R)$
q_5	–	–	–	–	–	–	–

4.4) Turing Machine as Computing Device:

4.4.1 Q. Design a Turing machine, which computes the 2's complement of the input binary number.

A: Two's complement can be computed using following algorithm:

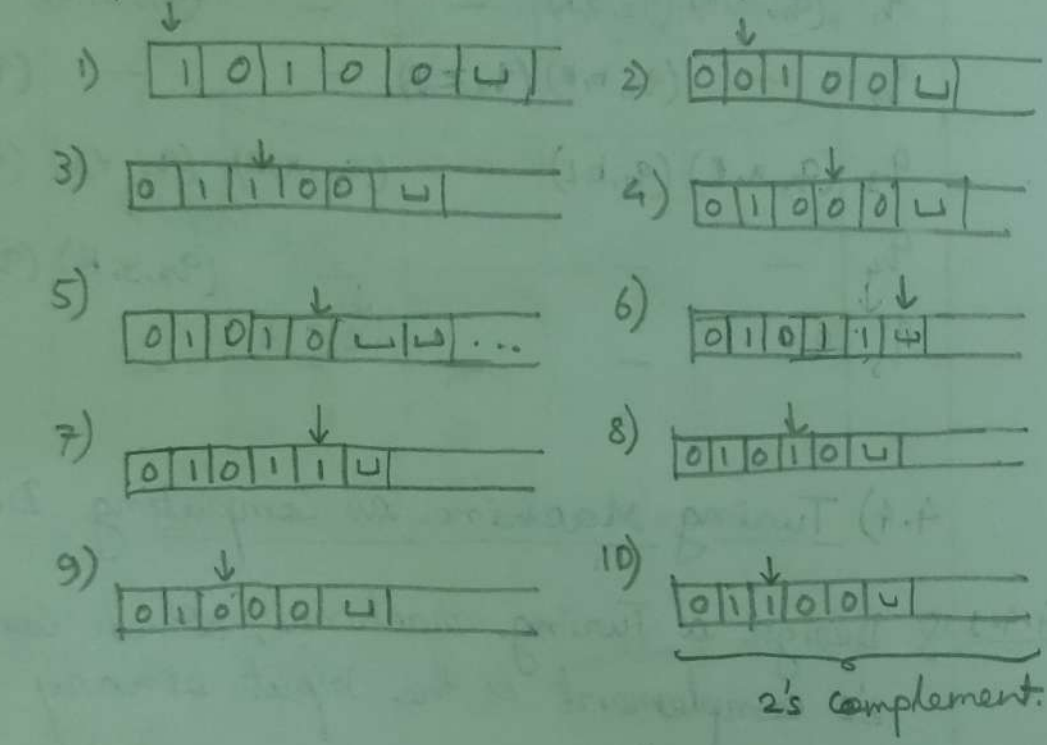
1) first switch all "1"s to "0"s & vice versa till you hit a $\sqcup$
2) Now after hitting $\sqcup$ move left, if its a "1", make it 0 & keep moving left till "0" is found, if it's "0" replace it with "1" & end. This part simulates adding 1 to the binary number.



eg: i/p : 10100

2's comp : 01100

Computation:



2's complement.

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

$Q = \{q_0, q_1, q_2\}$ $\quad b = \{␣\}$

$\Sigma = \{0, 1\}$ $\quad F = \{q_2\}$

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

$q_0 = \{q_0\}$

$\delta \Rightarrow$

State	0	1	␣
q_0	$(q_0, 1, R)$	$(q_0, 0, R)$	$(q_1, ␣, L)$
q_1	$(q_2, 1, L)$	$(q_1, 0, L)$	—

4.4.2 Q. Design a Turing Machine that can perform right shift over $\Sigma = \{0, 1\}$

A: We are to shift all bits towards right one step

Algorithm:

1) ignore all reading & find the end "B"

2) once encountered B move back left

3) if "0" replace with B & go to 4.1
   if "1" replace with B & go to 4.2
   } move right

4.1) replace B with 0 & go Left

4.2) replace B with 1 & go Left

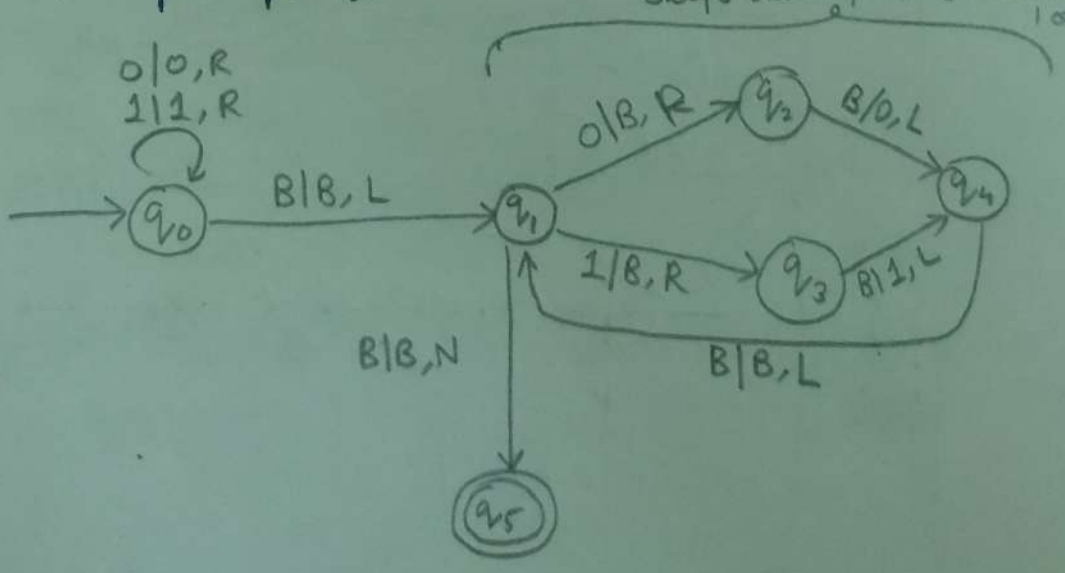
4.3) if B is encountered move Left & repeat from 2, if B is encountered again after moving Left, then Stop.

Expl:

1) go to end.

2) move last bit to end by swapping blank & left of blank.

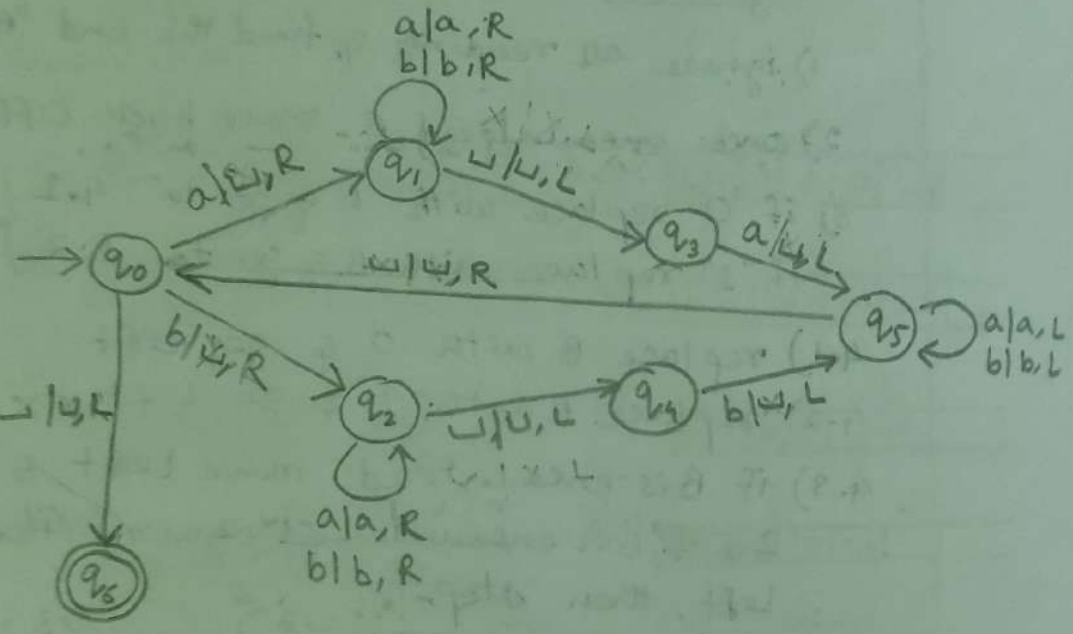
3) keep swapping blank with left of blank.



* Q. Design a Turing Machine that accepts even palindromes.

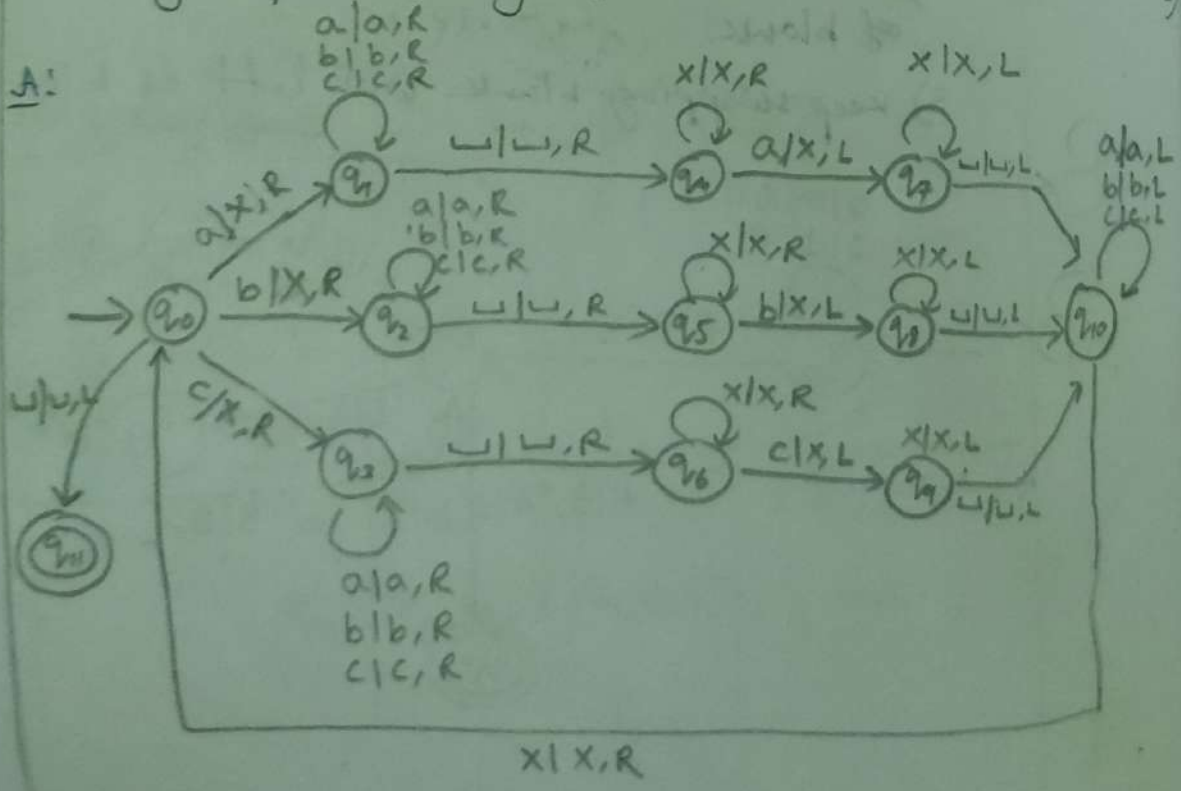
A: In an even palindrome, first & last char are same, 2nd & 2nd last are same & so on.

$L = \{a, b\}$



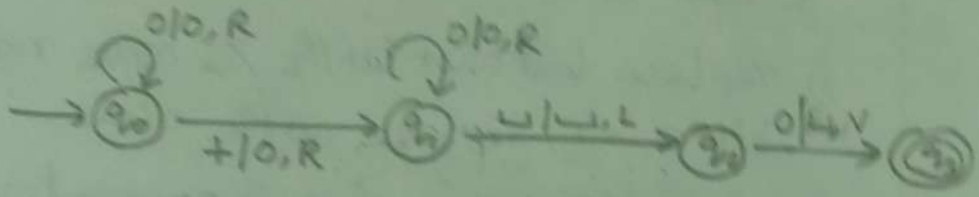
* Q. Design a Turing machine that compares two string separated by symbol ␣ $\{w \text{␣} w \mid w \in \{a,b,c\}\}$

A:

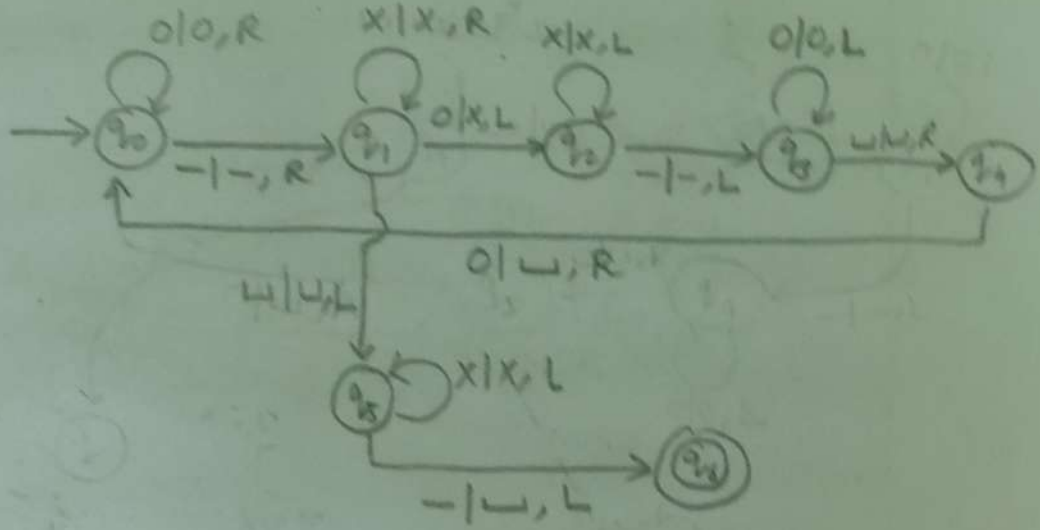


4.4.3 Q. Design a Turing Machine to compute addition of two unary numbers.

A: For unary numbers addition, just replace the operator with a bit then replace the last bit with a blank.



4.4.4 Q. Design a Turing Machine to compute Subtract of two unary numbers.



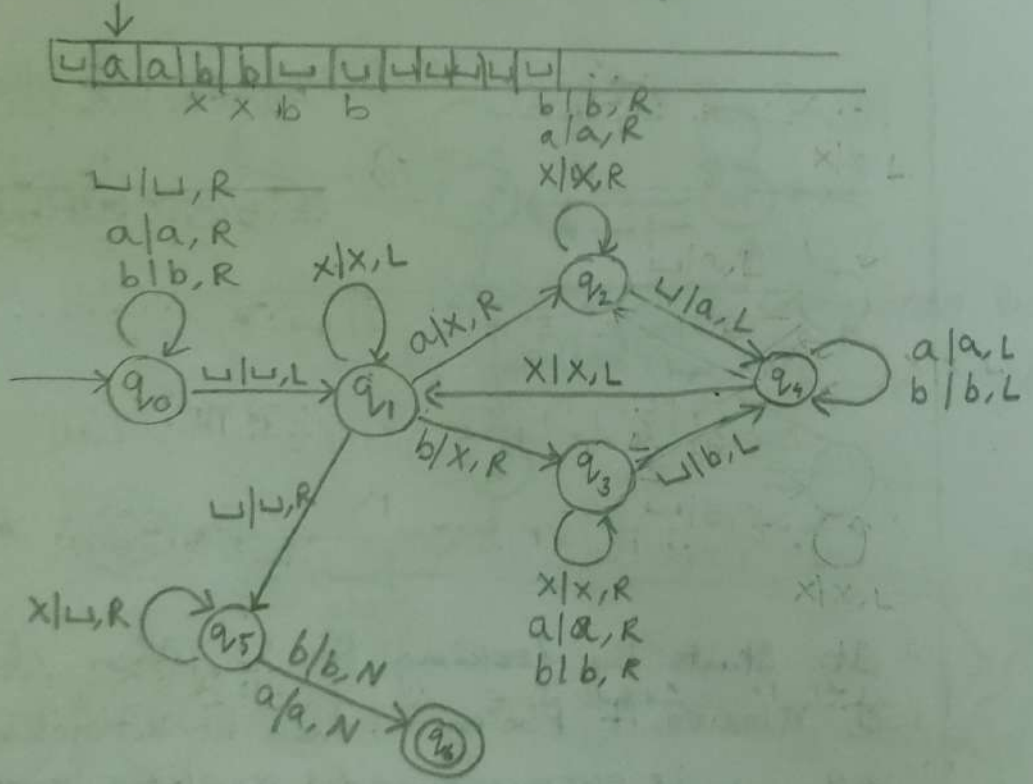
It Starts by looking for '-' then checks for next 0, removes it Placing X, then backtracks to begining till ␣ is encountered & replaces next 0 by X

This is repeated until there are no 0's after '-' in that case '-' is removed & halt is reached.

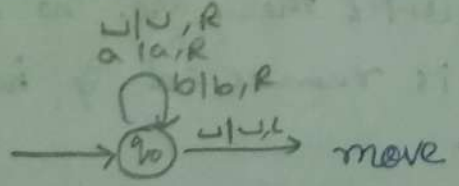
*Q. Design a Turing Machine that can reverse a string consisting of a's & b's.

A: approach:

1) go from left to right till you hit ␣, then go left to reach last char
2) replace last char with X, move right till you hit a ␣, replace with char.
3) go back till a char is encountered & repeat from (2).
4) when reach ␣ on right, halt.

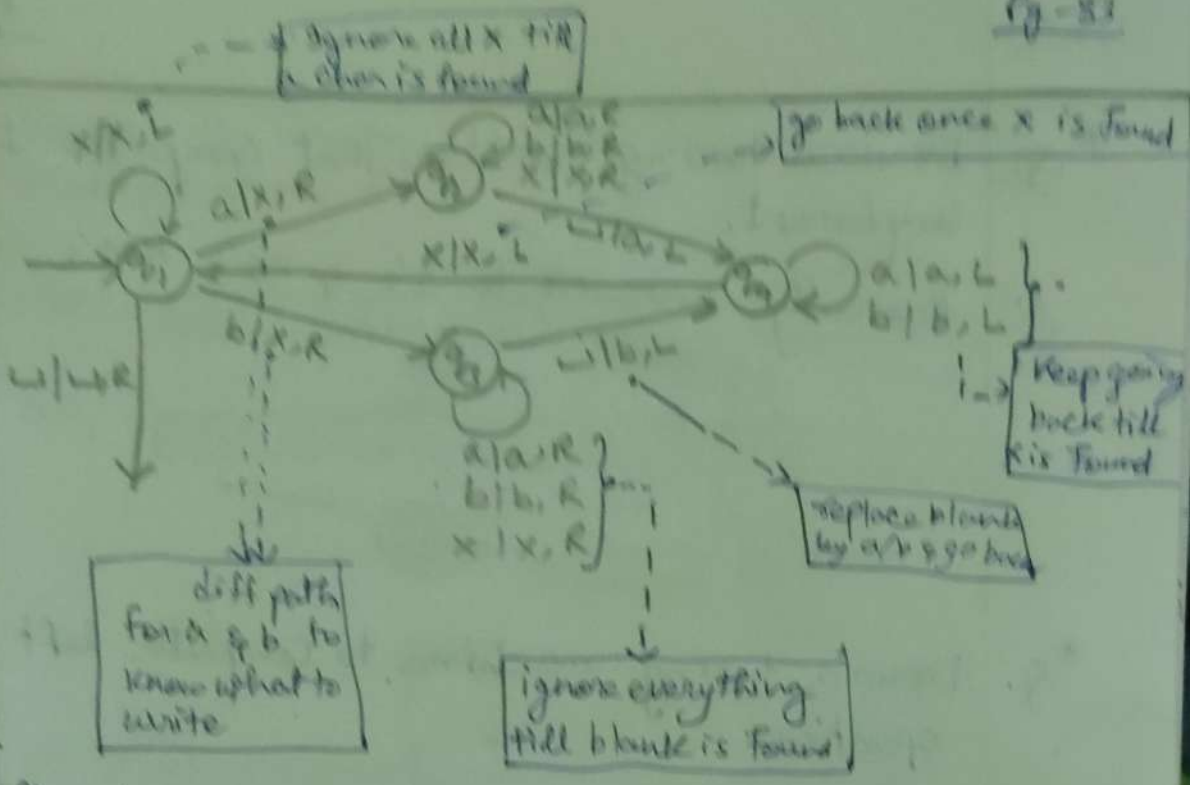


explaination:

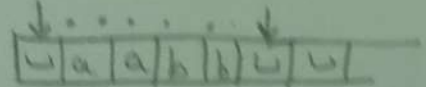


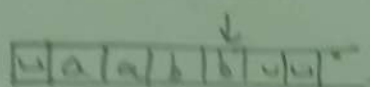
move to right most ele

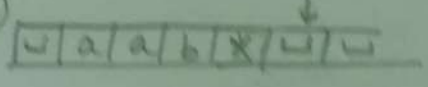
Pg - 83

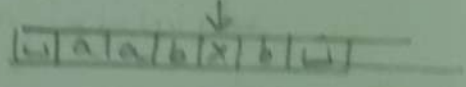


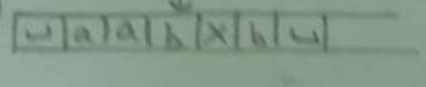
flow:

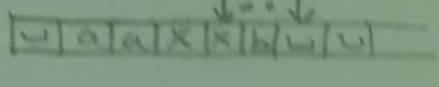
1) 

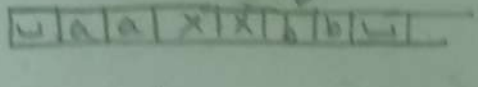
2) 

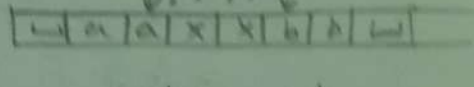
3) 

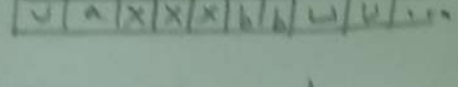
4) 

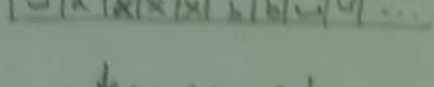
5) 

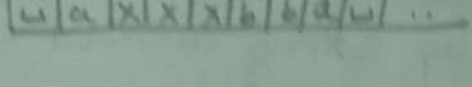
6) 

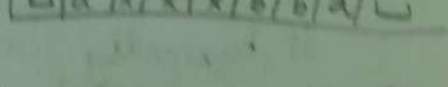
7) 

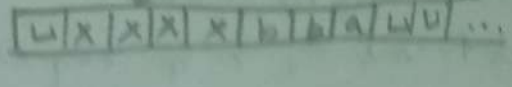
8) 

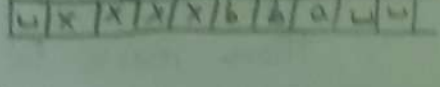
9) 

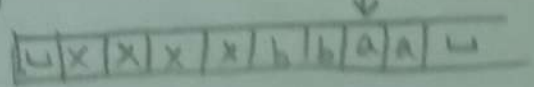
10) 

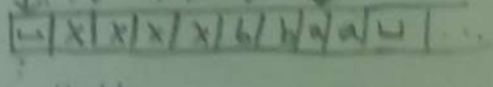
11) 

12) 

13) 

14) 

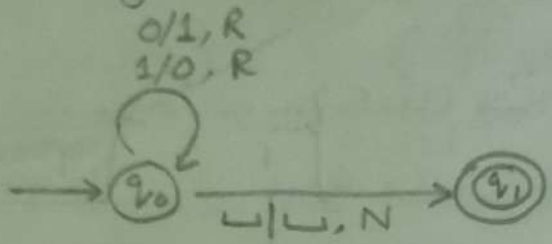
15) 

16) 

Halt

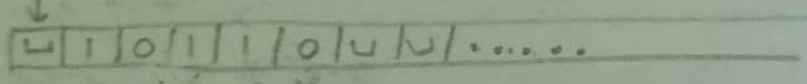
*Q. Draw a turing machine that computes 1's complement.

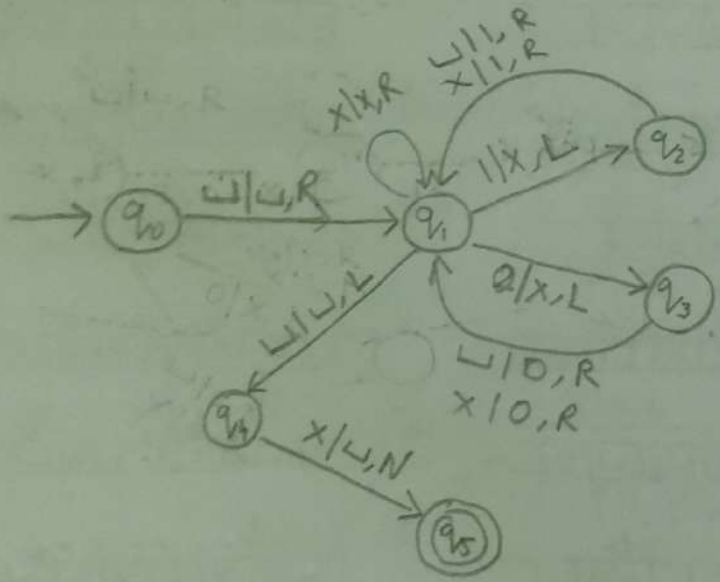
A: Basically turn 0's to 1's & 1's to 0's.



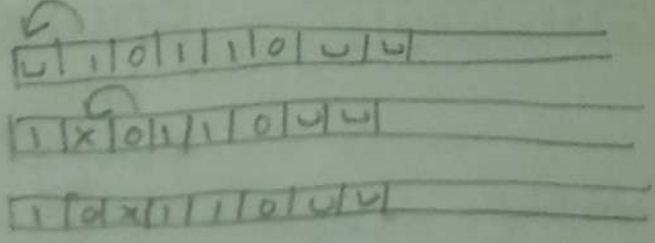
*Q. Draw a turing machine to compute left shift operation.

A:





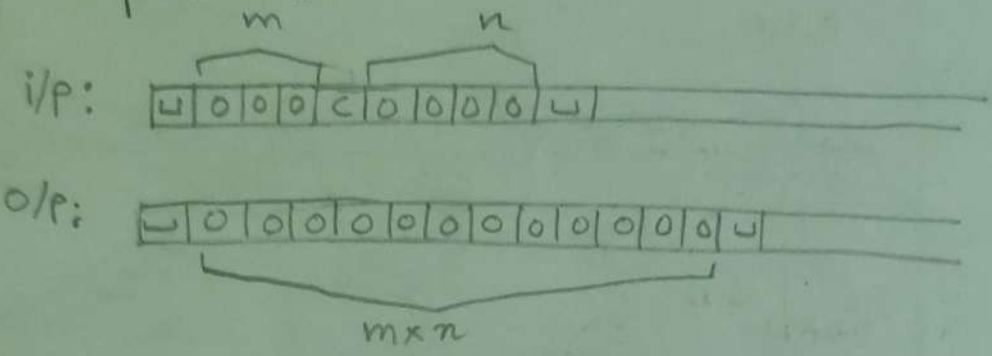
It basically works by setting current val as x then setting previous val as current val then move to next val of x & repeat.

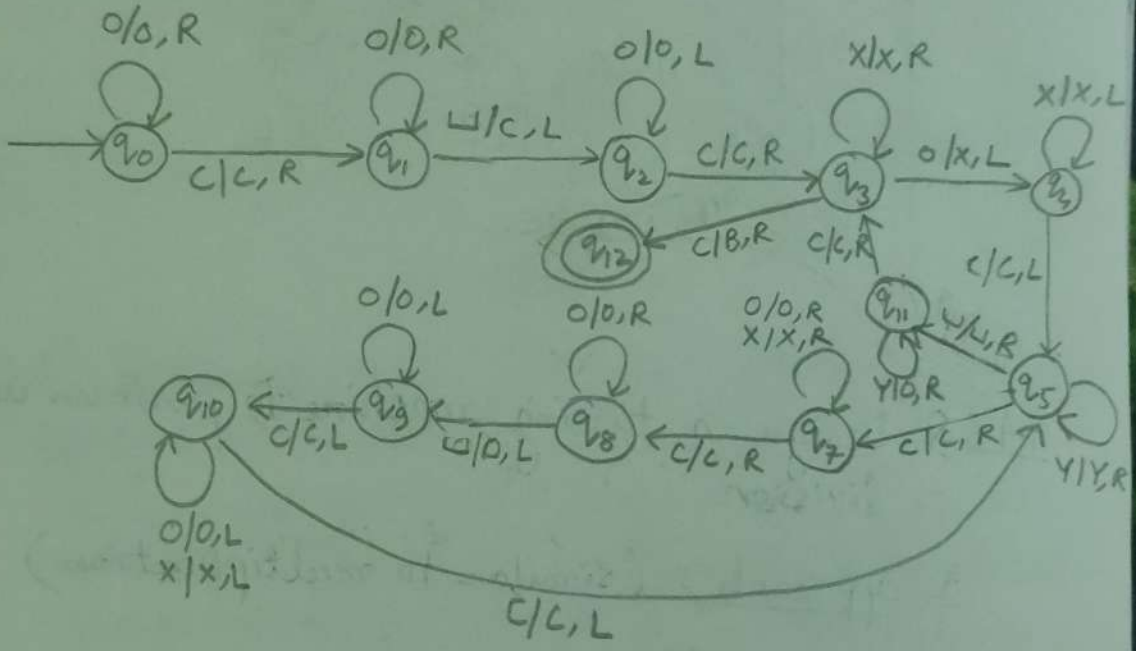


⋮

4.4.5 Q. Design a Turing Machine to perform unary multiplication.

A:





Alt answer:

approach

B 11 C 111 CB B B B

1) for each one on m, do (2)

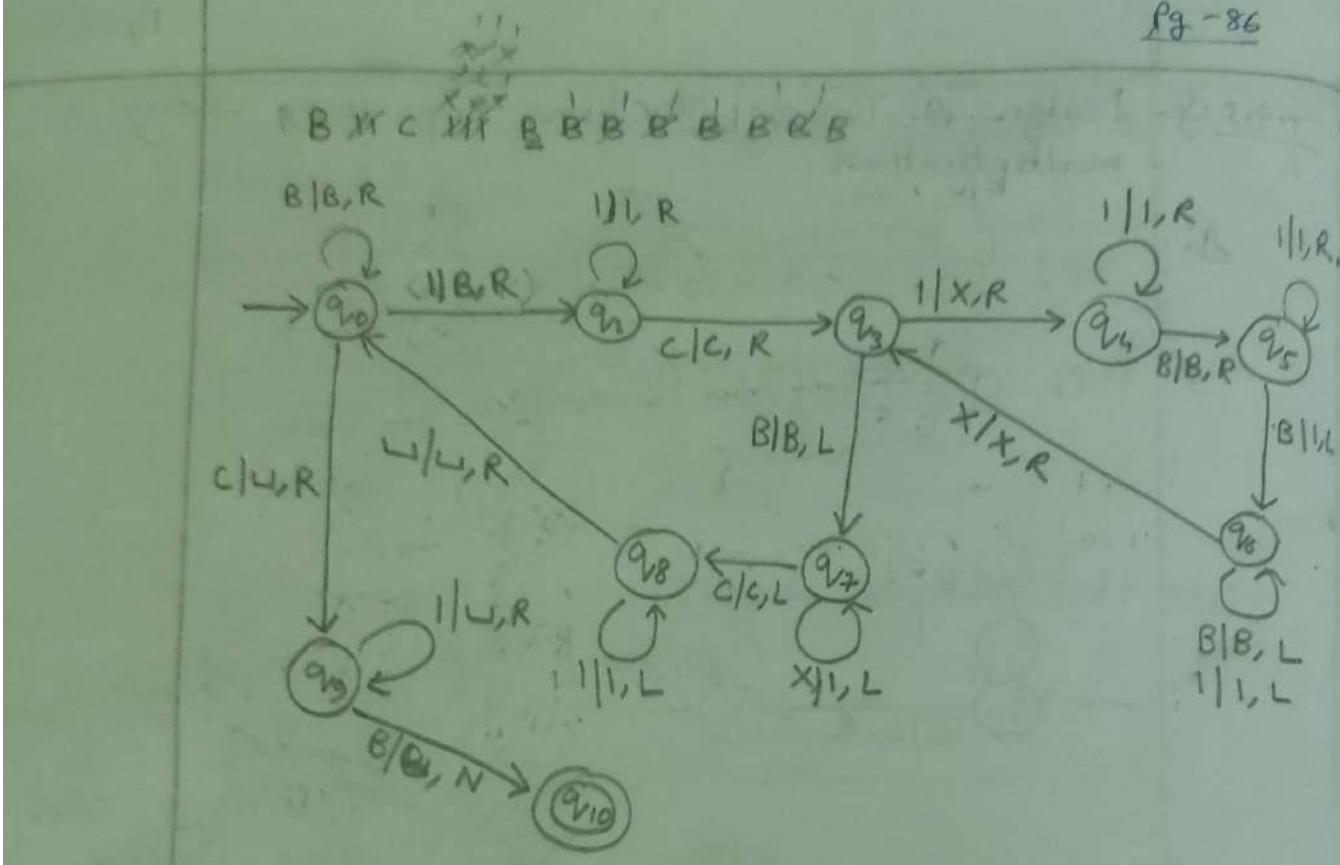
2) for each one on n, add another one after 2nd C

to do this replace 1 with X & B with one till no more 1, then replace all X by 1 & go back to (1).

B X 1 C 11 C 111 B B B B

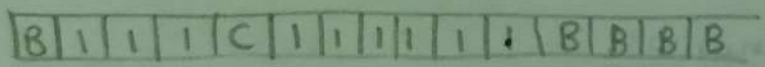
B X X C 11 C 111 11 B B

ans.

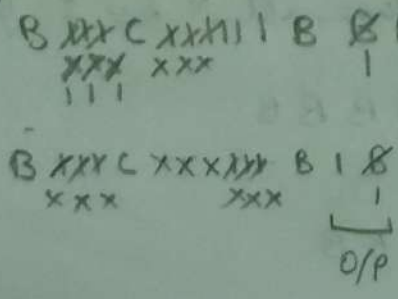


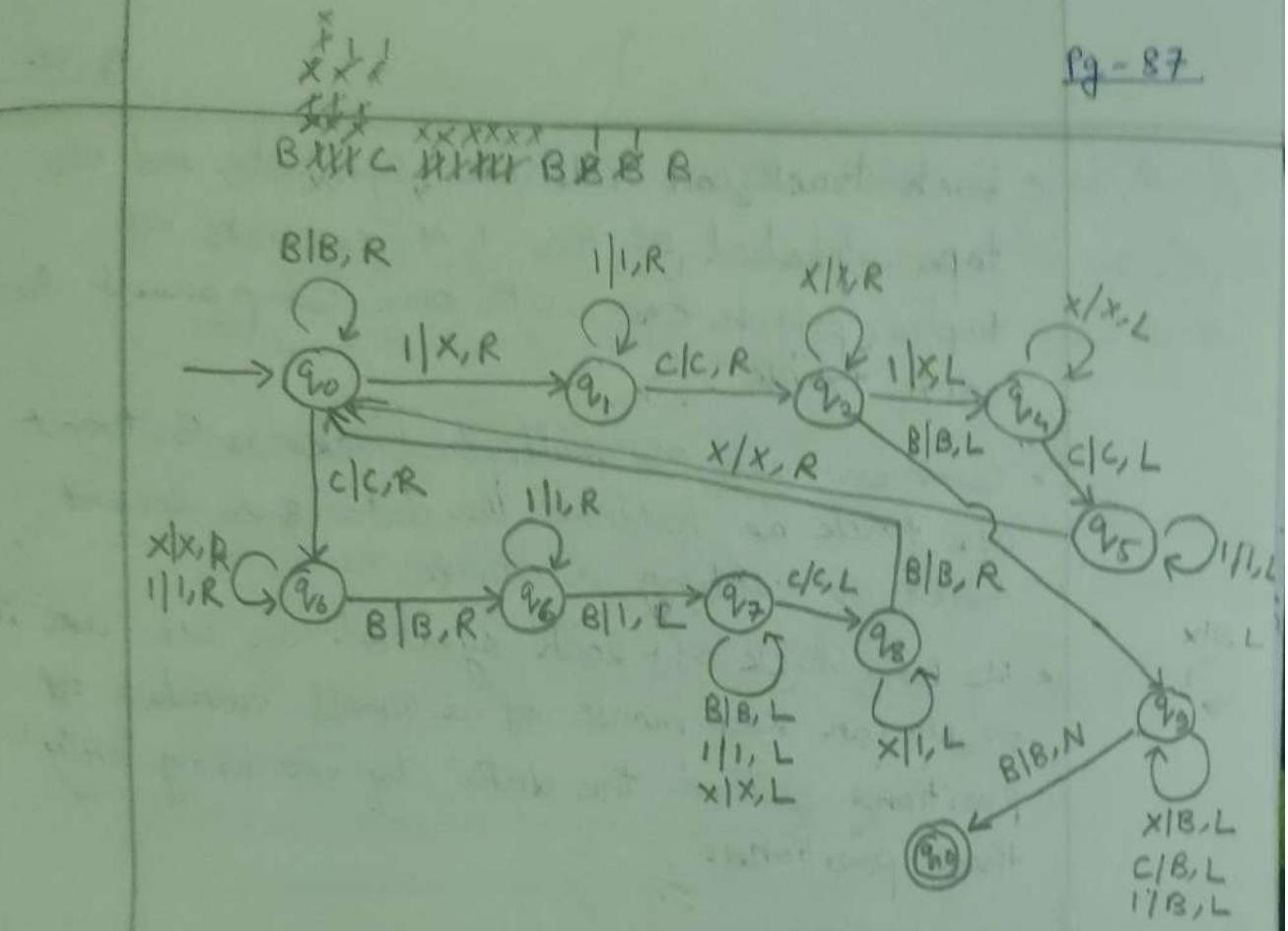
4.4.6 Q. Design a turing machine to perform unary division.

A: approach: (similar to multiplication)



- for each one on LHS of C, remove a 1 from RHS of C, when there is no more 1 in LHS, add a 1 next to the right end blank, then restore LHS & repeat, repeat till there is no more 1's in RHS.

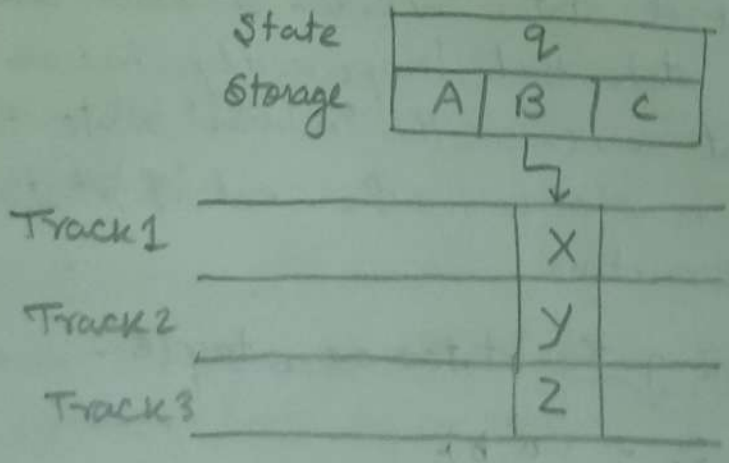




4.5) Techniques for Turing Machine Construction.

(i) Storage in the state :

We can use the finite control not only to represent a position in the "program" of the Turing machine, but to hold a finite amount of data.



(ii) Multiple Tracks :

- To think of the tape of a TM as a Composed of several tracks.

- Each track can hold one symbol, and the tape alphabet of the T.M. consists of tuples, which one with one component for each "track".
- Common use of multiple tracks is to treat one track as holding the data & a second track as holding a mark.
- We can check off each symbol as we use it, or we can keep mark of a small number of positions within the data by marking only those positions.

(iii) Subroutines :

- It helps to think of TM as built from a collection of interacting components or "subroutines."
- A TM subroutine is a set of states that perform some useful process.
- This set of states includes a start state and another state that temporarily has no moves, and that serves as the "return" state to pass control to whatever other set of states called the subroutine.

(iv) Considering the states as a tuple :

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

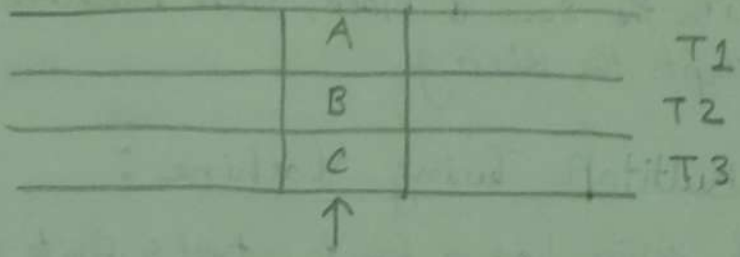
- After reading a 'a' → the machine remembers it by going to 'qa'

- After reading a 'b' → the machine goes to 'q_b'

In general, we can represent the state as $[q, x]$ where $x \in \Sigma$ denoting that it has read a 'x'.

(v) Considering the tape symbol as a tuple:

- Sometimes we may want to mark some symbols without destroying them or do some computation without destroying the input. In such cases, it is advisable to have multiple tracks on the tape.



- There is only one tape head. In the above figure. there are 3 tracks. The head is pointing to a cell which contains A → first track
  B → 2nd track
  C → 3rd track

∴ The tape symbol → 3-tuple [A, B, C]

Some computation can be done in one track by manipulating the respective component of the tape symbol.

(vi) checking off Symbol

- We use one track of tape to mark that some symbols have been read without changing them.
- Consider the eg. T.M for accepting

$$w \# w \# w \quad \forall \; w \in \{a, b\}^*$$

Two tracks tape:

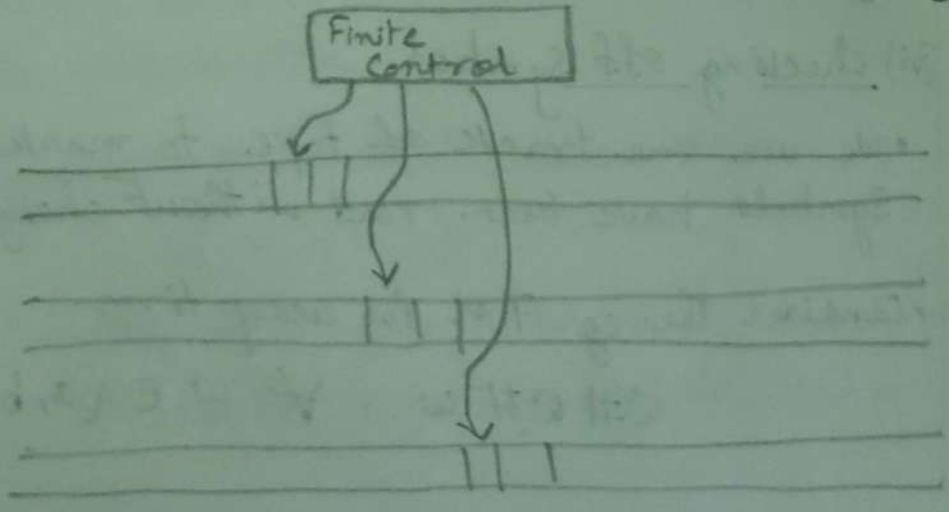
...	a	a	b	#	a	a	b	#	a	a	b	..
...												..

1st track → contains the input.

- When the TM reads the first 'a' in state q_0 it stores it in its memory, checks off 'a' by printing '✓' in the 2nd track & move right
- After the # symbol, do the process again.
- When all the symbols in the first block match with the second block, then machine halts & accepts the string.

4.6) Multitape Turing Machine :

- The device has - finite control & finite no. of tapes.
- Each tape → devided into cells → any symbol of finite tape alphabet.
- The input → placed on the 1st tape.
- All other cells of the tape → blank.
- Finite control → in the initial state
- head of first tape → left end.
- all other tape heads → at some arbitrary cell.



In one move, Multitape TM does the following:

1. The control enters a new state - which could be the same as the previous state.
2. On each tape, a new tape symbol is written on the cell scanned. Any of these symbols may be the same as the symbol previously there.
3. Each of the tape head makes a move, which can be either left, right or stationary.

4.7) Equivalence of One-Tape & Multitape TM's:

Theorem: Every language accepted by a multitape TM is recursively enumerable.

i.e. Anything that can be computed by a multi tape TM. can also be computed by some single tape T.M.

4.8) Non-Deterministic T.M:

A non deterministic TM (NTM) having a transition fⁿ δ such that for some state q and tape symbol x,

$\delta(q, x)$ is a set of triplets,

i.e. $\delta(q, x) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2) \dots\}$

Theorem:

If M_N is a non deterministic TM, then there is a deterministic TM M_D such that

$$L(M_N) = L(M_D)$$

4.9) Semi-Infinite Turing Machine:

- The tape head of the TM to move either left or right from its initial position.
- We can assume, the tape is semi-infinite, i.e. there are no cells to the left of the initial head position.

X_0	X_1	X_2	X_3	X_4	X_5	...
*	X_1	X_{-2}	X_{-3}	X_4	X_{-5}	...

- Restriction → It never writes a blank (B)

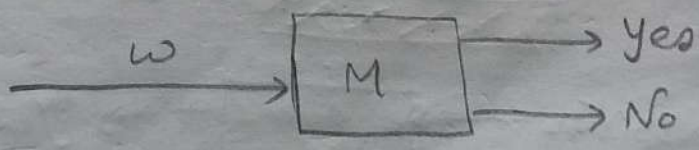
1. The tape is at all times a prefix of non-blank symbols followed by an infinity of blanks.
2. The sequence of non-blanks always begin at the initial tape position.

Unit-V Computation Complexity (T) Pg-93

Topics:

5.1) Undecidability:

- **Recursive Language**: A language is recursive if there exists a Turing machine that accepts every string of the language & rejects the string that is not in the language.



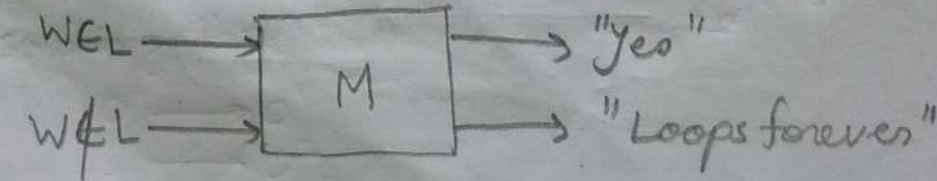
- **Decidable problems:**
  A problem whose language is recursive is said to be decidable otherwise the problem which can be answered as "yes" are called solvable or decidable.
  eg: Sorting, searching etc.
- **Undecidable problems:**
  The problem which can be answered as "No" are called undecidable problems.
  A problem is said to be undecidable if there is no algorithm that takes as input, an instance of the problem & determines weather the output said to that instance is 'yes' or 'no'.
  eg: post correspondance problems, Halting problems etc.

5.2) Recursively Enumerable Language:

A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM that accepts every string $w \in L$ and does not accept strings that are not in L.

If the input string is accepted, M halts with answer "yes".

If the string is not an element of L, then M may not halt & enter into infinite loop.



- The long range goal of proving the undecidability consisting of pairs such that,
  (i) M is a Turing machine encoded in binary
  (ii) W is a string of 0's & 1's
  (iii) M accepts input W.
- If the problem with the binary inputs is undecidable then TM with any other alphabet is undecidable
- The input given to turing machine should be in a well-formed representation using binary values.

* 5.3) Codes for Turing Machine:

To represent a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ as a binary string ($\Sigma = \{0,1\}$)

→ First assign integer to states, type symbols and directions (L & R)

- Assume the states are $q_1, q_2 \ldots q_k$ for some k, using the integers available in the suffix of each state, the string can be represented as

$q_1 \rightarrow 0$, $q_2 \rightarrow 00$, $q_3 \rightarrow 000$ etc.

i.e $q_1 \rightarrow 0^1$, $q_2 \rightarrow 0^2$, $q_3 \rightarrow 0^3$ etc.

- Assume the tape symbols 0, 1, B are represented as $X_1, X_2, X_3 \ldots X_m$ where

$X_1 \rightarrow 0 \rightarrow 0$
$X_2 \rightarrow 1 \rightarrow 00$
$X_3 \rightarrow B \rightarrow 000$
:

- Assume the directions are represented as D_1 & D_2 where

$L \rightarrow D_1 \rightarrow 0$
$R \rightarrow D_2 \rightarrow 00$

After representing each state, symbol & direction using integers, we can encode the transition function.

$\delta(q_i, X_j) = (q_k, X_l, D_m)$ for some integer i, j, k, l, m.

The encoded string is given by $0^i 1 0^j 1 0^k 1 0^l 1 0^m$

The code for entire TM consists of all strings separated by pairs of 1's.

$$C_1 11 C_2 11 C_3 11 \cdots C_{n-1} 11 C_n$$

$C \rightarrow$ Transition Code.

eg: Q. Obtain the code for $\langle M; 1011 \rangle$, where $M = (\{q_1, q_2, q_3, q_4\}, \{0,1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$

has moves: $\delta(q_1, 1) = (q_3, 0, R)$

$\delta(q_3, 0) = (q_1, 1, R)$

$\delta(q_3, 1) = (q_2, 0, R)$

$\delta(q_3, B) = (q_3, 1, R)$

Sol: Assuming the states as encoded

$q_1 \longrightarrow 0^1$

$q_2 \longrightarrow 0^2$

$q_3 \longrightarrow 0^3$

~~$q_4 \longrightarrow 0^4$~~

Assuming tape symbols as encoded

$0 \longrightarrow 0^1$

$1 \longrightarrow 0^2$

$B \longrightarrow 0^3$

Assuming directions as encoded

$L \longrightarrow 0^1$

$R \longrightarrow 0^2$

∴ Transitions are encoded as:

(i) $\delta(q_1, 1) = (q_3, 0, R)$

$C_1 \Rightarrow 0^1 1 0^2 1 0^3 1 0^1 1 0^2$

(ii) $\delta(q_3, 0) = (q_1, 1, R)$

$C_2 \Rightarrow 0^3 1 0^1 1 0^1 1 0^2 1 0^2$

(iii) $\delta(q_3, 1) = (q_2, 0, R)$

$C_3 \Rightarrow 0^3 1 0^2 1 0^2 1 0^1 1 0^2$

(iv) $\delta(q_3, B) = (q_3, 1, L)$

$C_4 \Rightarrow 0^3 1 0^3 1 0^3 1 0^2 1 0^1$

∴ The complete code is given by

$M = C_1\ 11\ C_2\ 11\ C_3\ 11\ C_4$

M = 0 1 0 0 1 0 0 0 1 0 1 0 0 11 0 0 0 1 0 1 0 1 0 0 1
0 0 11 0 0 0 1 0 0 1 0 0 1 0 1 0 0 11 0 0 0
1 0 0 0 1 0 0 0 1 0 0 1 0

∴ Code for <M, 1011> is given by

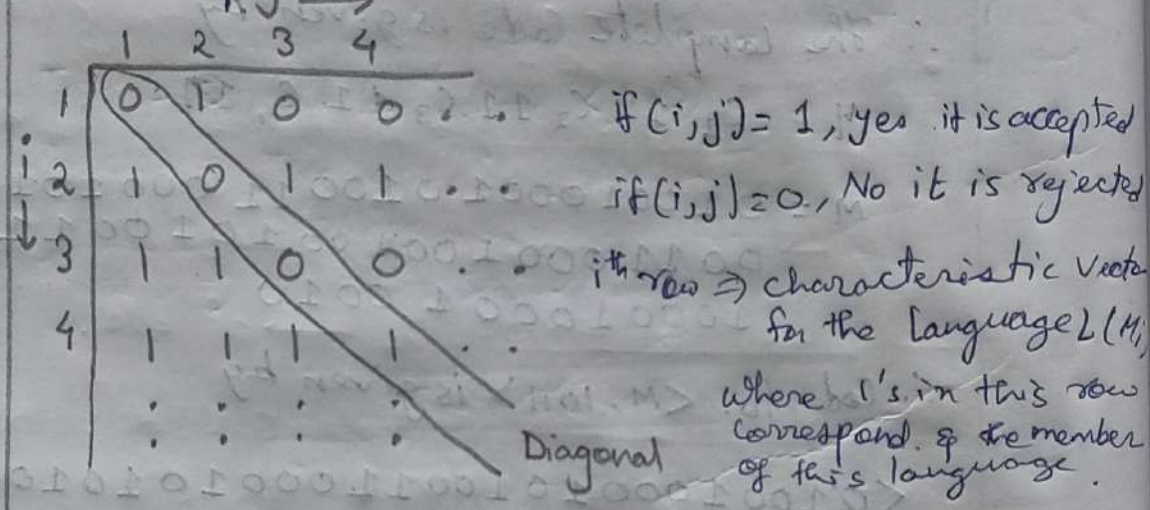
<0 1 0 0 1 0 0 0 1 0 1 0 0 11 0 0 0 1 0 1 0 1 0 0 1 0 11
0 0 0 1 0 0 1 0 0 1 0 1 0 0 11 0 0 0 1 0 0 0 1 0 0 0 1
0 0 1 0, 1 0 1 1>

5.4) Diagnolized Language (L_d):

It consist of all the string w such that the TM represented by w does not accept input w. L_d has no turing machine that accepts it.

- Some integers do not belong to any TM. If w_i is not a valid TM code, then take the TM M_i to be the TM with one state & no transitions.
- Defination: The diagonalization language L_d is the set of string w_i where w_i is not in $L(M_i)$

L_d consists of all strings w such that the TM. M whose code is w does not accept the input w.



In order to find L_d, Complement the diagonal.

Here diagonal value = 0001

After complementing value becomes 1110

- **Diagonalization**: Process of complementing the diagonal to construct the characteristic vector of a language that cannot be language that appears in any row.

- **Property**: L_d is not recursively enumerable

Proof: suppose L_d is accepted by some TM M defined by $L(M)$

Since L_d is a language over alphabet $\{0,1\}$ there may be atleast one code for M, Say i (i.e.) $M = M_i$

check whether w_i is in L_d,

By definition $L_d = \{w_i \mid M_i \text{ does not accept } w_i\}$

Two possibilities,

(i) $w_i \in L_d \Rightarrow (i,j)$ entry is '0' & M_i does not accept w_i But our assumption here is TM M_i which accepts w_i. there is a contradiction.

(ii) $w_i \notin L_d \Rightarrow (i,j)$ entry is '1' and M_i accepts w_i, But by definition of L_d M_i does not accept w_i. So there is a contradiction

Hence proved.

5.5) Universal Language (L_u):

Consider the binary string representation $\langle M, w \rangle$ such that M accepts w, called as universal language L_u

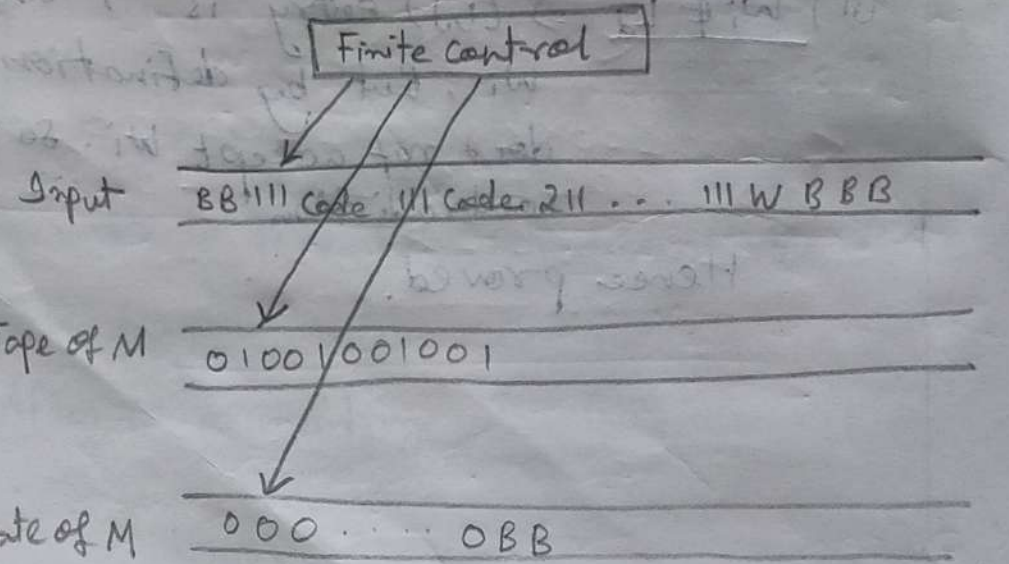
$$L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$$

L_u is the set of strings representing a TM and an input accepted by that TM. So there is a TM U called as Universal Turing Machine.

Theorem: L_u is recursively enumerable

Proof: In order to prove this theorem, it is necessary to construct a TM U that accepts L_u. The turing machine U consists of a Three track input tape.

Track 1 ⇒ Hold the input tape ($\langle M, w \rangle$)
Track 2 ⇒ Tape symbols are written in unary form
Track 3 ⇒ Represents states (unary)



operation:

1. First make sure that the code for M, is a legitimate code for some TM M. otherwise it halts without accepting.
2. Initialize the 2nd tape with input w, if its encoded form keep 0 the start state of M on the third tape & move the head of U's second tape to the first simulated cell.
3. If 0^i is the current state with 0^i then current i/p symbol appeared on the track three & two respectively then U finds the corresponding transition of the form $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ on track 1 & replace 0^i by 0^k & 0^j by 0^l.
4. move the head on the tape two to the position corresponding to the value of M.
5. The universal Turing Machine U accepts $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ if M has the transition of form $\delta(0^i, 0^j) = (0^k, 0^l, 0^m)$ otherwise M halts without accepting.

Thus, U simulated M & accepts w. Thus L_u is recursively enumerable.

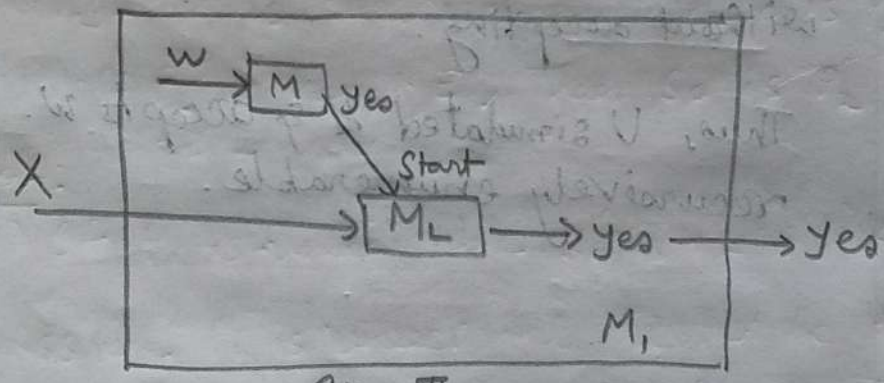
5.6) Rice Theorem :

All natural nontrivial properties of the Recursively Enumerable Languages are undecidable.
i.e. It is not possible to recognize the property by a TM.
A property is trivial if it is either empty which is satisfied by no language or is a all RE language, or else it is nontrivial.

Theorem: Every non-trivial property of the RE language is undecidable.

Proof: Let p be a non-trivial property of RE language.
Assume that ϕ is not in p.
∵ p is nontrivial, There must be some non empty language L that is in p

Here M_1 ⇒ TM accepting L
L_u ⇒ undecidable.



Rice Theorem

Design of M_1 is
(i) if $L(M_1) \neq \phi$, if M does not accept w.
(ii) if $L(M_1) = L$, if M accepts w.

M_1 is a two tape TM. one tape is used to simulate M on w and another tape is used to simulate M_L on input x to M_1

TM, M_1 is constructed as follows:

→ M_1 ignores its input & simulates M on w, If M does not accept w, then M_1 does not accept x. If M accepts w, then M_1 simulates M_L on x, accepting x if and only if M_L accepts x, thus M_L either accepts ϕ or L depending on M

→ It is possible to use the hypothetical M_P to determine if $L(M_1)$ is in P since $L(M_1)$ is in P if and only if $\langle M, w \rangle$ is in L_u.

We have an algorithm for L_u, a contradiction.

Thus P must be undecidable.

5.7) Undecidable problems about Turing Machine:

Reduction:

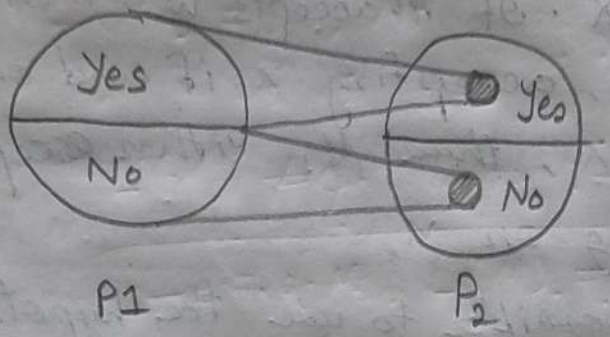
P_1 reduces to P_2 ⇒ If there is an algorithm used to convert instance of a problem P_1 to instance of a problem P_2 with the same answer, then it is said that P_1 reduced P_2

Thus, P_1 is not recursive ⇒ P_2 cannot be recursive

P_2 is non-R.E ⇒ P_2 is not R.E.

(P_1 cannot be R.E)

So any instance of P_1 that has "Yes" answer can be turned into an instance of P_2 with a "Yes" answer, and every instance of P_1 with answer "No" answer can be turned into an instance of P_2 with "No" answer.



Reduction of P_1 to P_2

A reduction can be defined as "A reduction from P_1 to P_2 is TM that accepts/Takes an instance of P_1 written on its tape & halts with an instance of P_2 on its tape."

Theorem: If there is a reduction from P_1 to P_2 then,

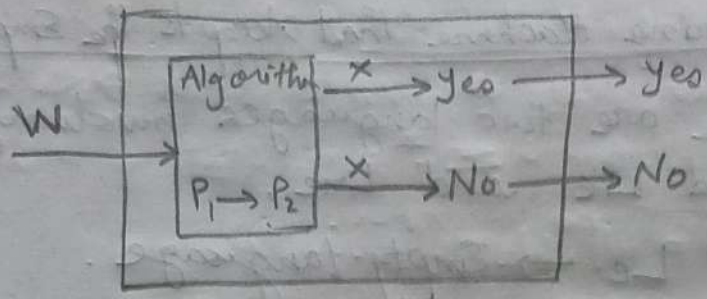
(i) If P_1 is undecidable, Then so is P_2

(ii) If P_1 is non RE, Then So is P_2

Proof:

(i) If P_1 is undecidable, so is P_2

Assume P_1 is undecidable, Let A be the algorithm which converts the instance of P_1 to instance of P_2.



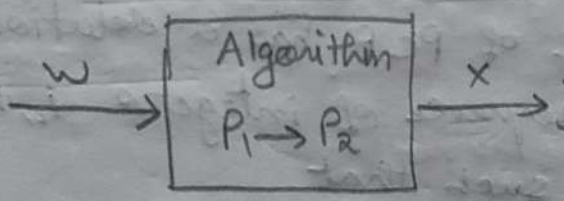
Suppose W be the instance of P_1, given to the algorithm A that converts w into an instance x of P_2

o/p	Inference
Yes	x is in P_2
No	x is not in P_2

∴ If P_1 is undecidable, then P_2 is also undecidable.

(ii) If P_1 is non R.E, then so is P_2

Assume that P_1 is non-RE but P_2 is RE. Since P_2 is RE, have an algorithm to reduce P_1 to P_2 that is there, If a TM gives "yes" output if it is input is in P_2 but may not halt if it is input is not in P_2



If w is in P_1 then x is in P_2, so the TM will accept w

If w is in P_1, then x is not in P_2 so the TM may not halt & will not accept w.

This is a contradiction, thus if P_1 is non RE, then P_2 is non RE.

5.8) Turing Machine that Accepts the Empty language

There are two languages involving TM called L_e and L_{ne},

$L_e \rightarrow$ Empty language.

$L_{ne} \rightarrow$ non-empty language.

If $L(M_i) = \phi$, M_i does not accept any input, then w is in L_e

If $L(M_i) \neq \phi$, language is called L_{ne}.

$$\therefore \boxed{\begin{aligned} L_e &= \{M \mid L(M) = \phi\} \\ L_{ne} &= \{M \mid L(M) \neq \phi\} \end{aligned}}$$

5.9) Post-correspondence problem : (PCP)

An instance of post-correspondance problem (PCP) consists of two lists of string over Σ

$$A = w_1, w_2 \ldots w_k$$
$$B = x_1, x_2 \ldots x_k$$

There instance of PCP has a solution if there is any sequence of integers $i_1, i_2 \ldots i_m$ with $m \geq 1$ such that

$$\boxed{w_{i1}, w_{i2} \ldots w_{im} = x_{i1}, x_{i2}, \ldots x_{ik}}$$

is a solution to this instance of PCP.

eg: Q. Let $\Sigma = \{0, 1\}$, Let A and B be Strings, Find the instance of PCP.

	List A	List B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

Sol: Given $w = (1, 10111, 10) = (w_1, w_2, w_3)$

$x = (111, 10, 0) = (x_1, x_2, x_3)$

Take $M = 4$, Take this combination 2, 1, 1, 3.

By concatinating string in this series.

$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$

$10111\ 1\ 1\ 10 = 10\ 111\ 111\ 0$

∴ Instance of PCP = 2, 1, 1, 3.

Q. Does PCP with two lists $x = (b, bab^3, ba)$ & $y = (b^3, ba, a)$ have a solution?

Sol: Given $x = (b, bab^3, ba) = (x_1, x_2, x_3)$

$y = (b^3, ba, a) = (y_1, y_2, y_3)$

Take $M = 4$, Take the combination 2, 1, 1, 3.

Check the string in this series!

$x_2 x_1 x_1 x_3 = y_2 y_1 y_1 y_3$

$bab^3\ b\ b\ ba = ba\ b^3\ b^3\ a$

$bab^3 b^3 a = bab^3 b^3 a$

∴ PCP has a solution for the given lists.

5.10) Modified PCP :

Given lists A and B of K size strings each from Σ^*

$A = w_1, w_2, w_3 \ldots w_k$

$B = x_1, x_2, x_3 \ldots x_k$

It has a solution $i_1, i_2 \ldots i_r$ such that

$w_1, w_{i_1}, w_{i_2} \ldots w_{i_r} = x_1, x_{i_1}, x_{i_2} \ldots x_{i_r}$

Difference b/w the MPCP & PCP is that in the MPCP, a solution is required to be started with the first string on each list.

eg: Q. Let $\Sigma = \{0, 1\}$ Let A & B the list of string denoted as

i	List A: w_i	List B: x_i
1	1	10
2	110	0
3	0	11

Find the instance of MPCP :

Sol : MPCP is given by

$w_1 w_{i_1} w_{i_2} \ldots w_{i_r} = x_1 x_{i_1} x_{i_2} \ldots x_{i_r}$

instance is 1, 3, 2

$w_1 w_3 w_2 = x_1 x_3 x_2$

w_1 — 1 10 110 = 10 11 0

x_1 — 10 110 = 10 110

$W_1 W_3 W_2 = X_1 X_3 X_2$

$10110 = 10\,1\,10$

both start with the first string.

5.11) Construction of PCP from MPCP:

* Q. Consider the TM M & W = 01, where $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_3\})$ and δ is given by

	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_3	—	—	—

Reduce the above problem to PCP & find the solution.

Sol: An instance of MPCP with two lists can be A & B :

$$\left[\underbrace{\delta(q_i, W_i)}_{A} = \underbrace{\delta(q_j, W_j, D)}_{B}\right]$$

(i) Steps to form list:

→ i.e. we write the LHS for δ transitions in ③ list A where q_i is current state & W_{x+1} is current Tape symbol.

$W_x q_i W_{x+1}$

it shows the relative position of state in tape. $q_i W_x$ if W_x is left most i.e. $q_i W_1$

• for R moves $q_i W_x$ is enough, for L moves $W_{x-1} q_i W_x$ is needed.

→ ③ We write the RHS of δ transition in list B in the form .. $W_x q_j W_{x+1}$ .. where q_j is the next state & W_{x+1} is the next Tape symbol.

eg:

List A	List B
$0q_1 1$	$q_2 00$

for $\delta(q_1, 1) = (q_2, 0, L)$

here tape was 01 initially, q_1 state was there & 1 was on head, the transition made the 1 to 0 & moved the head Left & State changed to q_2.

→ ① Before the transitions, add the initial state of both Lists.

List A : #

Since no transitions has occured yet A is result list which holds results of transition.

List B : # q_1 01 #

It contains the initial state of TM, i.e. Starting state q_1 & Tap head at 0.

→ B.

→ ② After the initial state, add a symbol to symbol mapping.

List A	List B
0	0
1	1
#	#

④ → Now, after the transitions are added, take the final state & put it in every possible position of tape, the result is always the final state itself.

List A	List B
$q_3 0$	q_3
$q_3 1$	q_3
$0 q_3 0$	q_3
$0 q_3 1$	q_3
$1 q_3 0$	q_3
$1 q_3 1$	q_3
$0 q_3$	q_3
$1 q_3$	q_3

⑤ → Lastly add an entry for final state as

List A: $q_f \# \# \mid q_f$ → final state.

List B: #

(ii) Steps to find solution:

→ take the w, i.e. input tape & the transition rules & run the w on the δ

→ write all the states of transition including the initial state in the format

$\# \ldots X_i q_n X_{i+1} \ldots \#$

→ remove all B from the transitions written in previous step.

→ add a # as a delimiter between all the steps.

→ When final state is reached, remove all symbols from tape one by one. eg: q_3 is final state.

eg: $\vdash q_3 101 \rightarrow \vdash q_3 01 \rightarrow \vdash q_3 1 \rightarrow \vdash q_3$

Solution:

Rule	List A	List B	Source
①	#	$\#q_1 0 1 \#$	
②	0	0	
	1	1	
	#	#	
③	$q_1 0$	$1 q_2$	$\delta(q_1, 0) = (q_2, 1, R)$
	$0 q_1 1$	$q_2 00$	$\delta(q_1, 1) = (q_2, 0, L)$
	$1 q_1 1$	$q_2 10$	
	$0 q_1 \#$	$q_2 01\#$	$\delta(q_1, B) = (q_2, 1, L)$
	$1 q_1 \#$	$q_2 11\#$	
	$0 q_2 0$	$q_3 00\#$	$\delta(q_2, 0) = (q_3, 0, L)$
	$1 q_2 0$	$q_3 10\#$	
	$q_2 1$	$0 q_1$	$\delta(q_2, 1) = (q_1, 0, R)$
	$q_2 \#$	$0 q_2 \#$	$\delta(q_2, B) = (q_2, 0, R)$
④	$q_3 0$	q_3	
	$q_3 1$	q_3	
	$0 q_3 0$	q_3	
	$0 q_3 1$	q_3	
	$1 q_3 0$	q_3	
	$1 q_3 1$	q_3	
	$0 q_3$	q_3	
	$1 q_3$	q_3	
⑤	$q_3 \# \#$	#	

Note: for R transition $q_n X_i$ is enough since output is $X_i' q_m$

Note: for L transitions we need $X_{i-1} q_n X_i$ ∴ Output will be $q_m X_{i-1} X_i'$ ↓ needed

now running W=01 through the given δ-transition, we get the following states

$\vdash q_1 01$

$\vdash 1 q_2 1$

$\vdash 10 q_1 B$

$\vdash 1 q_2 0 1$

$\vdash q_3 101$

$\vdash q_3 01$

$\vdash q_3 1$

$\vdash q_3$

∵ q_3 is final state, a solution to the instance of M PCP is written from listA & list B as

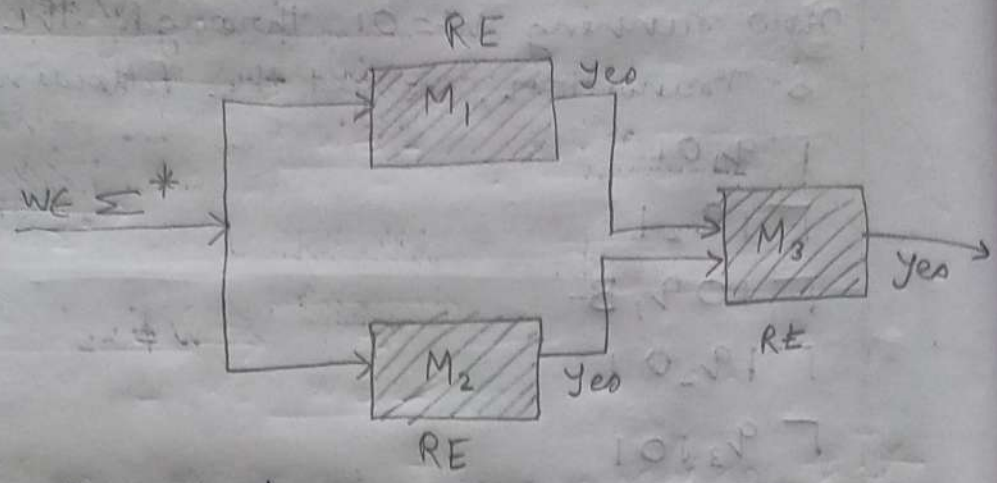
$\# q_1 01 \# 1 q_2 1 \# 10 q_1 \# 1 q_2 01 \# q_3 101 \# q_3 01 \# q_3 1 \# q_3 \# \#$

5.12) Properties of Recursive & Recursively enumerable language:

(i) **Property-1**: The union of two recursively enumerable language is recursively enumerable.

Proof: Let L_1 & L_2 be two recursively enumerable language accepted by the Turing Machine M_1 & M_2.

- If a string $w \in L_1$ then M_1 returns "yes", accepting the input string, else loops forever. Similarly if a string $w \in L_2$ then M_2 returns "yes", else loop forever.



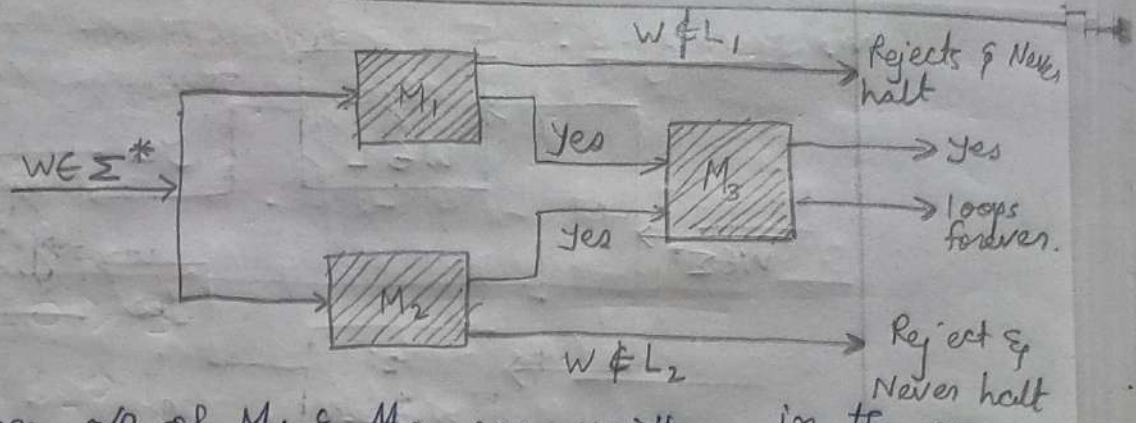
- Here, the output of M_1 & M_2 are written on the input ~~tape~~ tape of M_3
- The TM, M_3 returns "yes" if at least one of the outputs of M_1 & M_2 is "yes".
- M_3 decides on $L_1 \cup L_2$ that halts with the answer "yes" if $w \in L_1$ or $w \in L_2$.
  else M_3 loops forever for both M_1 & M_2 loop forever.

Hence The union of two recursively enumerable language is also recursively enumerable.

(ii) Property-2 : Intersection of two recursively enumerable language is recursively enumerable.

Proof: Let L_1 & L_2 be two recursively enumerable language accepted by the TM M_1 & M_2.

- If a string $w \in L_1$, then M_1 returns yes. else it will not halt if $w \notin L_1$
  Same for $w \in L_2$ & M_2

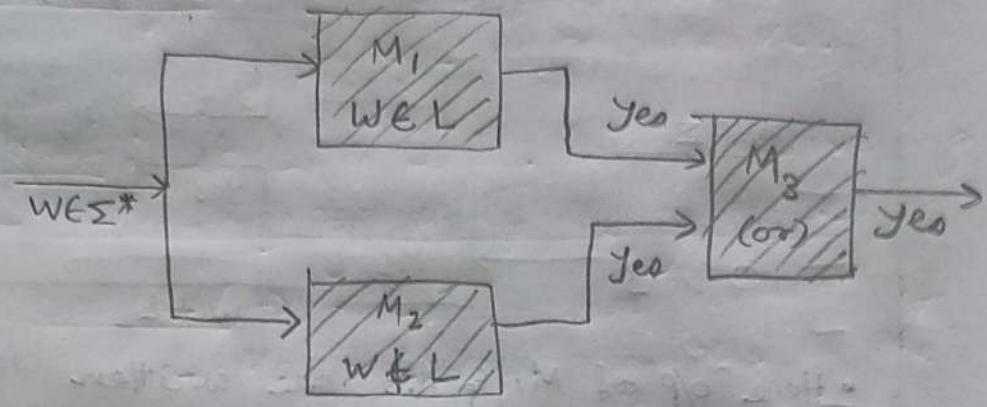


- Here o/p of M_1 & M_2 are written in the i/p tape of M_3.
- M_3 returns "yes" if both the outputs of M_1 and M_2 is "yes"
- If at least one of M_1 or M_2 is "No" it rejects 'w' & never halts.
- Thus M_3 decides on $L_1 \cap L_2$ that halts only if $w \in L_1$ and $w \in L_2$. Else M_3 loops forever along with M_1 or M_2 or both.

(iii) **Property-3**: A language is recursive if and only if both it and its complement are recursively enumerable.

Proof: Let L & $\bar{L}$ be two recursively enumerable Language accepted by TM's M_1 & M_2.

- If a string $w \in L_1$ it is accepted by M_1 & M_2 halts with answer "yes". else M_1 enters infinite loop.
- If $w \notin L$, then M_2 accepts w & halts with yes.
- Since M_1 & M_2 are accepting the complements of each other, one of them is guaranteed to halt for every input $w \in \Sigma^*$.

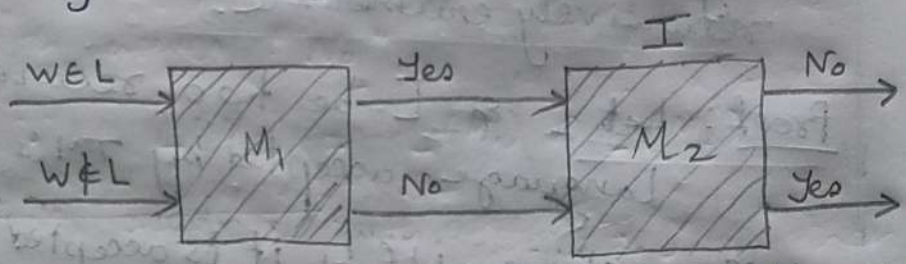


Hence M_3 is a Turing machine that halts for all strings.

(iv) Property-4 : The complement of a recursive language is recursive.

Proof :

- Let L be a recursive language accepted by machine M_1
- Let $\bar{L}$ be a recursive language accepted by TM, M_2.



Let $W \in L$, then M_1 accepts W and halts with "Yes".

M_1 rejects W if $W \notin L$ and halts with "No"

M_2 is activated once M_1 halts.

M_2 works on $\bar{L}$ and hence if M_1 returns "Yes", M_2 halts with "No"

- M_2 is activated once M_1 halts.

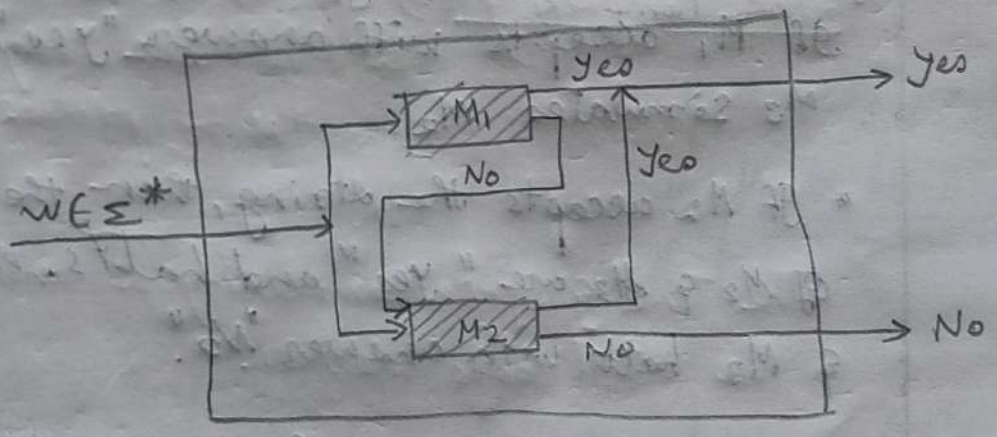
M_2 works on $\overline{L}$ & hence if M_1 returns "yes", M_2 halts with "No". If M_1 returns "No" M_2 halts with "yes"

Thus for all w, where $w \in L$ or $w \notin L$ halts with either "yes" or "No".

(v) Property-5: The union of two recursive language is recursive.

Proof: Let L_1 & L_2 be two recursive language that are accepted by the TM's M_1 & M_2 given by $L(M_1) = L_1$, $L(M_2) = L_2$

- The TM, M_3 first simulates M_1 with input string w.
  If $w \in L_1$ then M_1 accepts & thus M_3 also accepts
  $\therefore L(M_3) = L(M_1) \cup L(M_2)$
- If M_1 rejects, string $[w \notin L_1]$, then M_3 simulates M_2. M_3 halts with "yes" if M_2 accepts w, else returns "No"



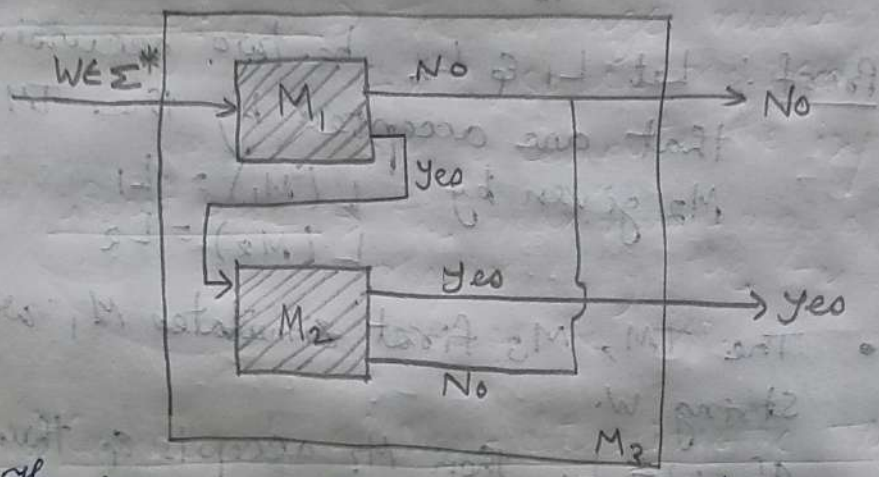
Hence, M_3, M_2, M_1 that halt with either yes or No on all possible inputs.

(vi) Property-6: The intersection of two recursive language is recursive.

Proof: Let L_1 & L_2 be two recursive language accepted by M_1 & M_2 where

$L(M_1) = L_1$

$L(M_2) = L_2$



- The turing machine M_3 simulates M_1 with the input string w.

if $w \notin L_1$, then M_1 halts along with M_3 with answer "No"

$\therefore L(M_3) = L(M_1) \cap L(M_2)$

If M_1 accepts with answer "yes" then M_3 simulates M_2

- If M_2 accepts the string, then the answer of M_2 & M_3 are "yes" and halts. else M_2 & M_3 halt with answer "No".

Thus the intersection of recursive language is also recursive.

5.13) Introduction to Computational Complexity:

Defination:

- Complexity ⇒ It is defined whether the given problem is easy to solve or hard to solve.

Complexity of problems is classified into two types:

(i) Space Complexity: The amount of memory space required by the algorithm/problem to complete its computation.

(ii) Time Complexity: The amount of time required to run the program of the problem.

5.14) Time & Space Complexity of Turing Machine:

(i) Time Complexity (T_T):

- The number of transitions/moves taken by the TM to compute the input is called the time complexity of the TM.
- Let T be a Turing Machine, then the time complexity is a function $T_T(n)$ is the minimum no. of transitions for computing the input, n using T.

T_T depends on the input size of problem.
when 'n' is large, T_T will also be large.
when 'n' is small, T_T will also be small.

(ii) Space Complexity (S_T):

- Space complexity is the number of tape-cells required to process the turing machine, T for computing input 'n'.

- $S_T(n)$ is the minimum number of tape cells used by T to process an input of length 'n'.

5.15) Complexity types / Efficiency types:

(i) Best case ⇒ The minimum number of time / space required by the TM to compute the input.

(ii) Average case ⇒ The average number of time / space required by the TM to compute the input.

(iii) Worst case ⇒ The maximum no. of time/space required by the TM to process the i/p.

5.16) Complexity of Non-deterministic TM:

- Time Complexity ⇒ $T_T(n)$ is function that refers to the max time taken by NTM to compute the input x, where $|x| = n$

- Space Complexity ⇒ $S_T(n)$ is the max amount of tape-cells required by the TM to process x with $|x| = n$.

The Time & Space Complexity is undefined if the NTM loops forever.

5.17) Complexity classes :

(i) class P :

- P refers to the class of problem that can be solved in polinomial time.
- polinomial time algorithms are efficient ones that can be solved more rapidly.
- These are also refered to as tractable problems.
- Eg : Searching, finding min/max etc, Sorting.

(ii) class NP :

- NP refers to the class of problems that are solved by non deterministic polynomial time.
- These types of NP problems are known as intractable problems.
- Eg : Tower of Hanoi, Travelling Salesman, Hamilton circuit etc.

(iii) Np-complete :

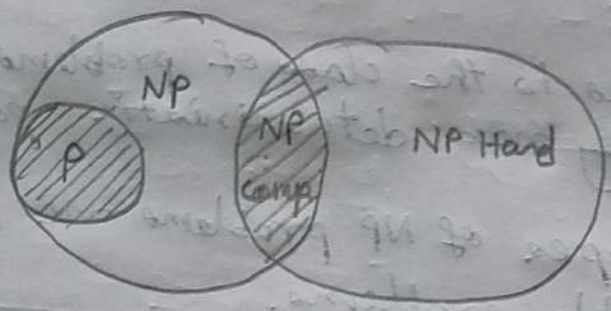
- A problem is said to be Np-complete if it belongs to NP class problem & can be solved in polynomial time.
- They are also called polynomial time reducible problem.
- A Np-complete problem can be transformed into any other in polinomial time.

(iv) NP-Hard problem :

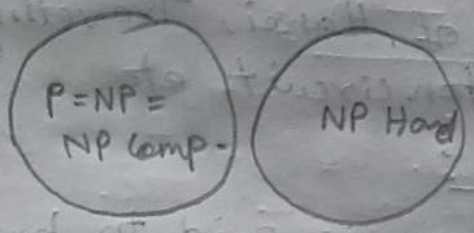
- A problem is said to be Np hard if there exists an algorithm for solving it & it can be translated into one for solving another NP-Problem.

- A problem P_1 is Np-hard if
  - The problem is on Np class problem.
  - For any other problem P_2 in NP, there is a polynomial reduction of L_2 to L_1,
- Every NP complete problem must be NP-hard problem.

Conclusion:



$P \neq NP$



$P = NP$