④ $S \to a \mid YBA \mid AY \mid b$

$A \to YX$

$B \to AY \mid b \mid a$

$X \to BB$

$Y \to a$

It is in CNF.

* Pumping lemma for Context-free language:-

Lemma :-

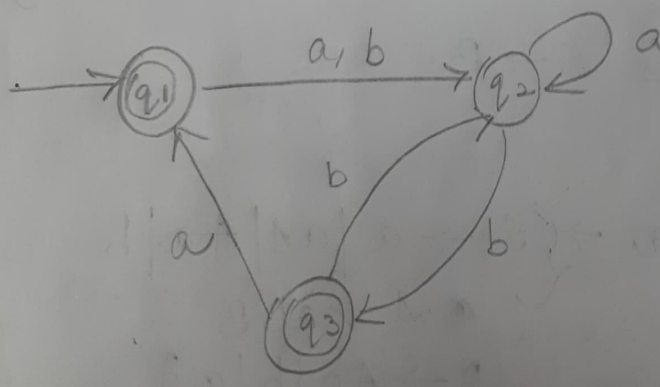
If A is a CFL, then A has a pumping length 'p' such that any string 's', where $|s| \geq p$, may be divided into 5 pieces, $s = uvxyz$, such that the following conditions must be true :

i) $uv^i xy^i z$ is in A for every $i \geq 0$

ii) $|vy| > 0$

iii) $|uxy| \leq p$.

⑤



$q_1 = \varepsilon + q_3 a$ — (1)

$q_2 = q_1(a+b) + q_2 a + q_3 b$ — (2)

$q_3 = q_2 b$ — (3)

$q_2 = q_1(a+b) + q_2a + q_2bb$

$$q_2 = q_1(a+b)(a+bb)^* \quad \text{---(4)}$$

$$q_3 = q_1(a+b)(a+bb)^* b \quad \text{---(5)}$$

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

$$q_1 = ((a+b)(a+bb)^* ba)^* \quad \text{---(6)}$$

$$q_3 = ((a+b)(a+bb)^* ba)^* (a+b)(a+bb)^* b \quad \text{---(7)}$$

$$q_1 + q_3 = ((a+b)(a+bb)^* ba)^* + ((a+b)(a+bb)^* ba)^* (a+b)(a+bb)^* b$$

$$= ((a+b)(a+bb)^* ba)^* \{\varepsilon + (a+b)(a+bb)^* b\}$$

Date - 21.12.23

Example :-

Using pumping lemma, prove that the language $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context free language

Soln - let L is a CFL, then L has a pumping length 'p'.

$$S = a^p b^p c^p = uvxyz \qquad |s| \geq p.$$

(i) $uv^i xy^i z \in L$, for all $i \geq 0$

(ii) $|vy| > 0$

(iii) $|vxy| \leq p$.

let $p = 4$.

$S = a^4b^4c^4 = \underbrace{aaaa}_{u}\underbrace{b}_{v}\underbrace{b}_{x}\underbrace{b}_{y}\underbrace{bccc}_{z}$

$uvxyz = aa\ aa\ bb\ bb\ cccc$

let $i = 2$

(i) $uv^ixy^iz = aa\,aaaa\,bbbbbb\,cccc$

$= a^6b^6c^4 \notin L \rightarrow$ doesn't satisfy

(ii) $|vy| > 0$

$vy = aabb \Rightarrow |vy| = 4 > 0 \rightarrow$ satisfy

(iii) $|vxy| \leq p \Rightarrow |6| \leq 4 \rightarrow$ doesn't satisfy.

∴ Our assumption is wrong and L is not a context-Free language.

*Pushdown Automata :-

Defn- A pushdown automata is a way to implement a context free grammar in a similar way the finite automata is designed for regular grammar.

A pushdown automata is a new type of computational model having an extra component called a stack. The stack provides an additional memory beyond the finite amount

available in the finite state control (finite state machine).

The PDA is more powerful than Finite State Machine. The finite state machine has very limited memory but the PDA has more memory due to the addition of stack.

PDA = finite State Machine + Stack

PDA ≈ NFA + Stack → enhances the memory capability of the machine.

(last In First Out) → principle

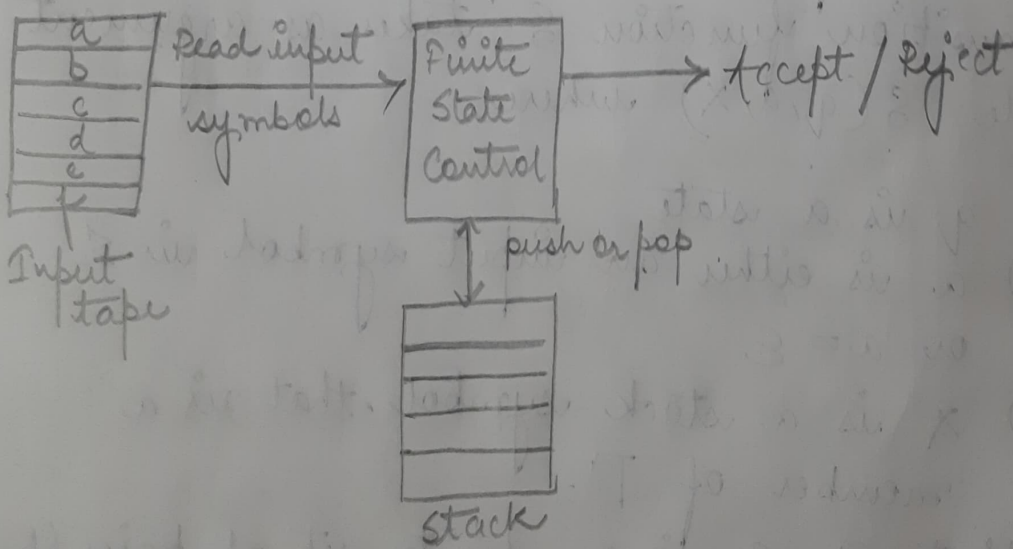
#. Schematic Representation Of PDA :-

→ A PDA has 3 components :-

(1) An input tape (input symbols)

(2) A finite state control unit

(3) A stack



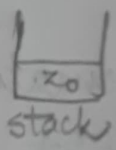
⊙ Formal Definition Of PDA :-

A PDA is formally defined by a 7-tuple as $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, f)$ or a 6 tuple as $P = (Q, \Sigma, \Gamma, \delta, q_0, f)$, where

(i) Q = A finite set of states.
(ii) Σ = A finite set of symbols
(iii) Γ = A finite stack symbols.
(iv) δ = The transition function, defined by.

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow P(Q \times \Gamma) \approx 2^{(Q \times \Gamma)}$$

(v) q_0 = The initial state
(vi) Z_0 = The stack start symbol ($)



(vii) f = The set of final/accept states where $f \subseteq Q$.

* Definition Of Transition function for PDA :-

The transition function δ takes as argument a triple $\delta(q, a, X)$ where,

(i) q is a state
(ii) a is either an input symbol in Σ or $a = \varepsilon$
(iii) X is a stack symbol, that is a member of Γ.

The output of δ is a finite set of pairs (p, γ) where

(i) p is a new state

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

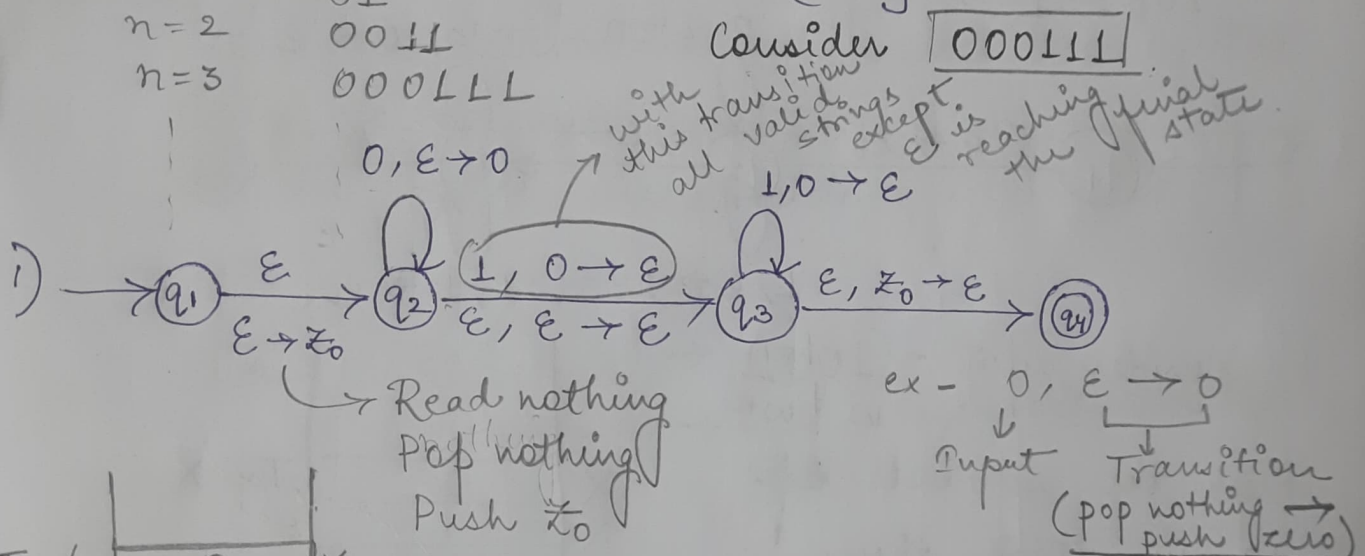
Date - 22.12.23

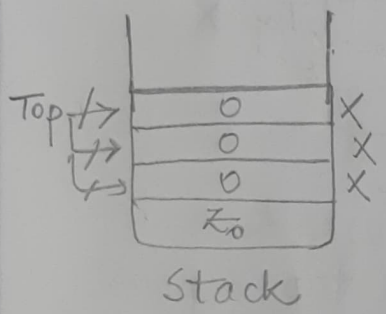
Example :- Construct a PDA that accepts the language $L = \{0^n 1^n / n \geq 0\}$

$n=0$	ε
$n=1$	01
$n=2$	0011
$n=3$	000111

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

Consider 000111





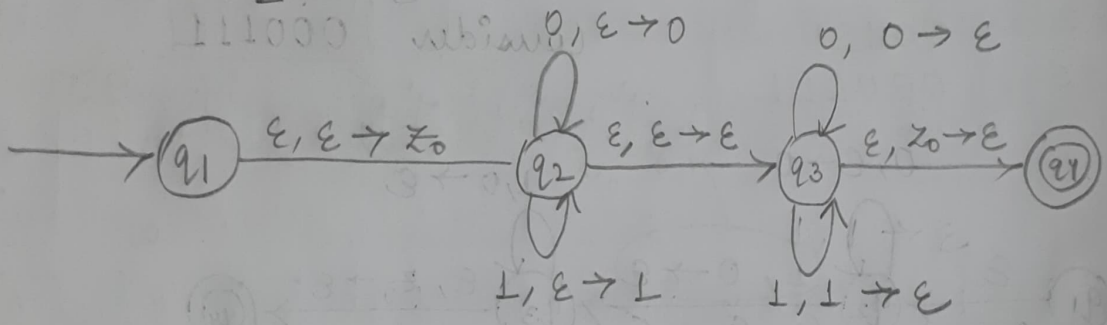
To check whether we have equal no.s of 1s and 0s, we will pop one 'zero' from stack for every 'one' in the string input. If we reach z_0 after popping 0's, then we will have equal no.s of 1s and 0s.

fig 1) State diagram for PDA that accepts $L = \{0^{n_1} n / n \geq 0\}$

Example - Construct a PDA that accepts even palindrome in the form

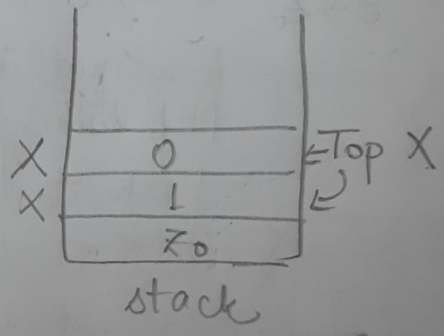
$$L = \{ ww^R \mid w \in \{0,1\}^* \}$$

ε,
00
0110
00111100
0100010
1000001



Consider input - 1001
w | w^R

Push the symbols of w in stack, then transit to q3, after reading all inputs of w and start matching the symbols of w^R with popping the top symbol until we reach Z_0



Date - 04.01.24

* Equivalence with CFG:-

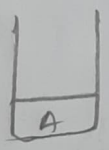
→ Conversion from CFG to PDA:-

Steps -

1) Place the marker symbol Z_0 or $ and the start variable on the stack.

2) Repeat the following steps forever:-

(i) If the top of stack is a variable symbol 'A', non-deterministically select one of the rules for 'A', and substitute 'A' by the string on the right hand side of the rule.

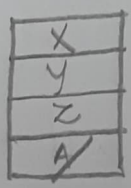


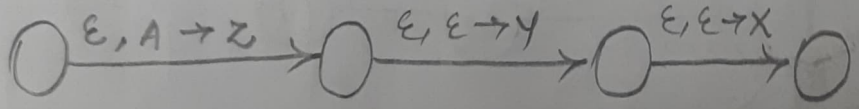
$A \to xx \mid xy \mid xyz$

$\delta(q, \varepsilon, A) \to (q, w)$

where $A \to w$ is a production in the CFG.

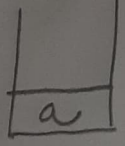
$A \to xyz$



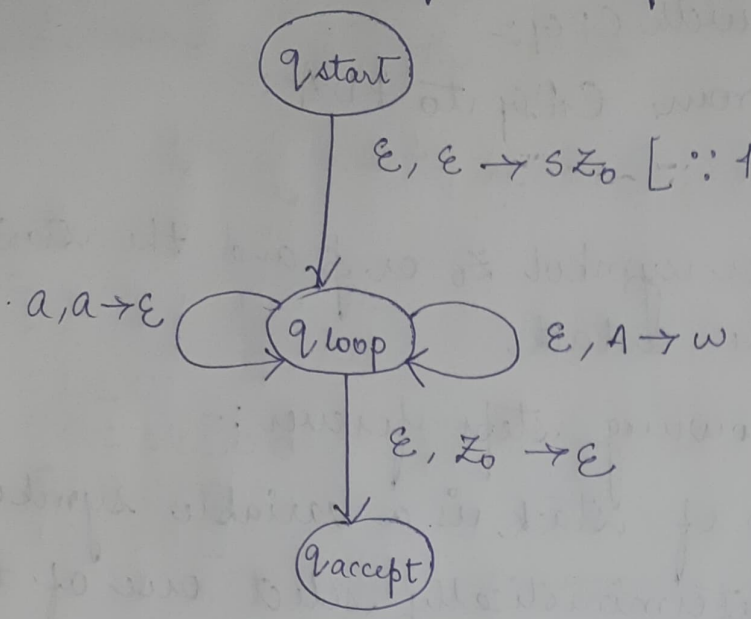


(ii) If the top of the stack is a terminal symbol 'a', pop 'a' from the top of the stack if it matches the input symbol and move forward to the next symbol.

$\delta(q, a, a) \to \delta(q, \varepsilon)$



* General format of PDA from CFG :-



Example: Construct the PDA from the given CFG :-

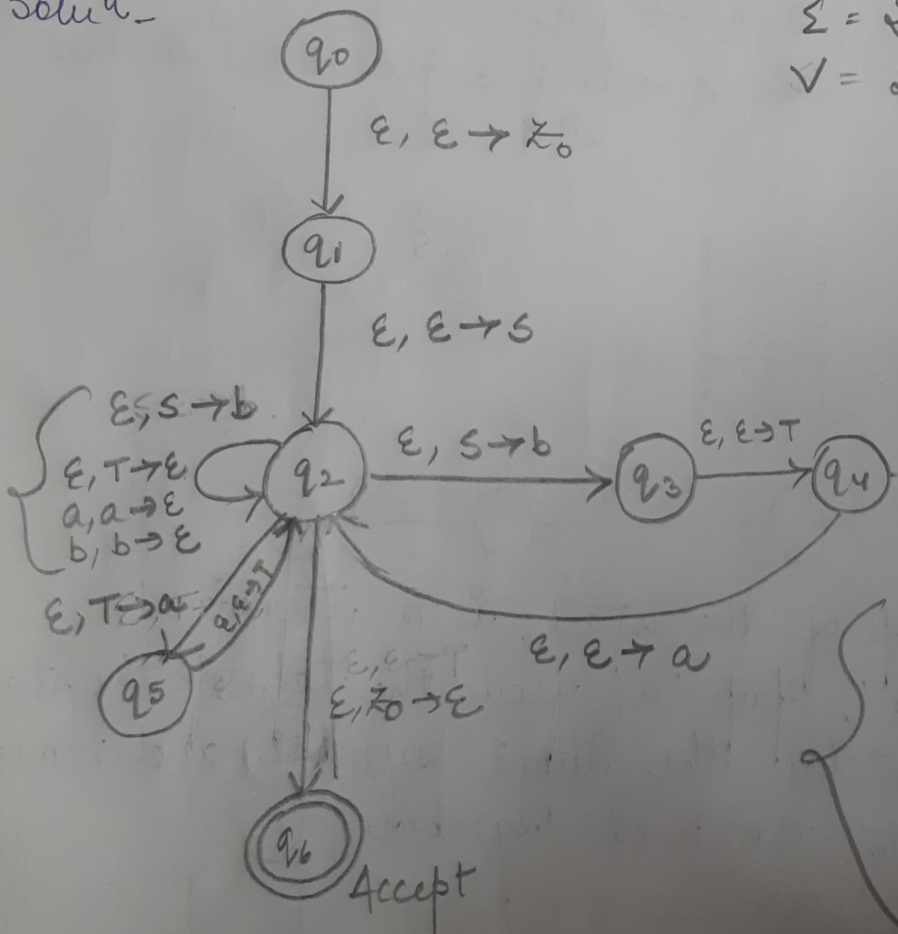
$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \varepsilon$

Soln-

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

$V = \{S, T\}$



Grammar :-

$S \rightarrow b$

$S \rightarrow aTb$ $(T \rightarrow \varepsilon)$

$S \rightarrow ab$

$S \rightarrow aTb$ $(T \rightarrow Ta)$

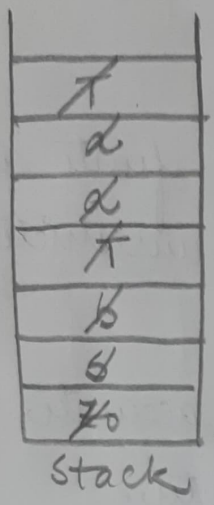
$S \rightarrow aTab$ $(T \rightarrow \varepsilon)$

$S \rightarrow aaab$

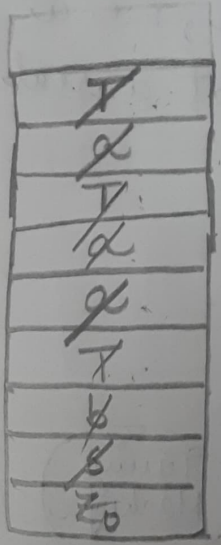
$L = \{b, ab, aab, aaab, \text{——}\}$

$L = \{a^n b \mid n \geq 0\}$

$S \rightarrow aTb$
$\rightarrow aTab \quad [\because T \rightarrow Ta]$
$\rightarrow aab \quad [\because T \rightarrow \varepsilon]$



Now, $S \rightarrow aTb$
$\rightarrow aTab \quad [\because T \rightarrow Ta]$
$\rightarrow aTaab \quad [\because T \rightarrow Ta]$
$\rightarrow aaab \quad [\because T \rightarrow \varepsilon]$



Date - 05.01.24

* Turing Machine :-

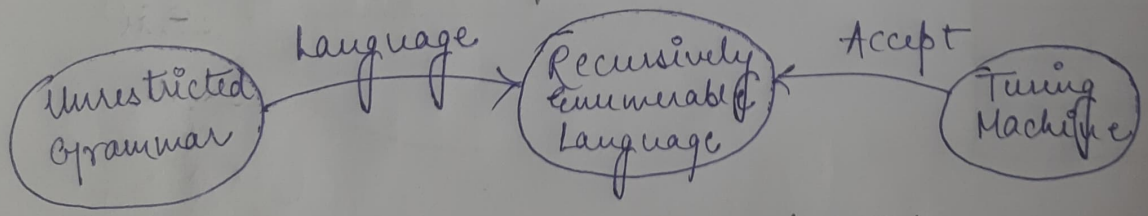
A Alan Turing, first proposed in 1936 a much more powerful model similar to finite automaton is called a Turing Machine.

The Turing Machine is similar to finite automaton with an unlimited and unrestricted memory.

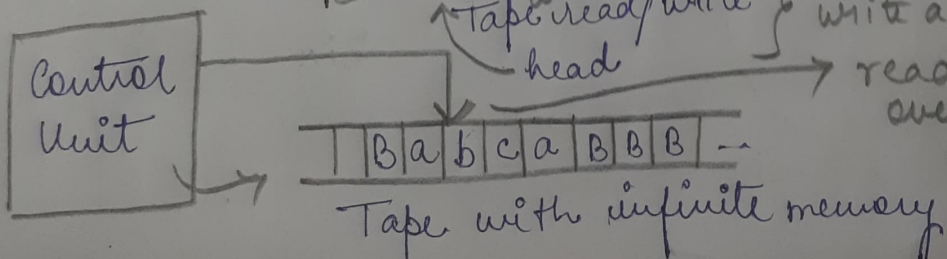
A Turing Machine is a much more accurate model of a general purpose computer.

According to Turing, any computation that can be carried out by mechanical means can be performed by some Turing Machine, i.e a Turing Machine can do everything that a real computer can do.

→ uses Tape (DS (Data Structure)) and infinite memory



* Schematic Representation Of TM :-

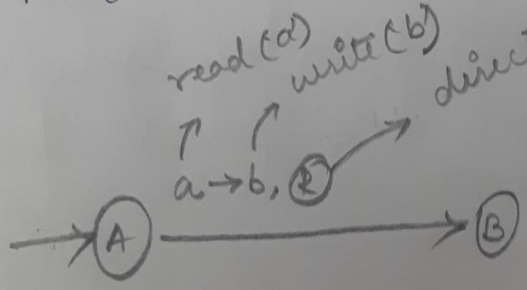


* Formal Definition Of Turing Machine :-

TM is formally defined as a 7 Tuple, $(Q, \Sigma, T, \delta, q_0, B, F)$, where

1) Q = finite set of states.
2) Σ = finite set of input alphabets, other than blank 'B'.
3) T = tape alphabet/symbol
4) $\delta : Q \times T \rightarrow Q \times T \times \{L, R\}$ is the transition function.
   - T (read from tape)
   - T (written on tape)
   - L = left, R = right
5) $q_0 \in Q$ is the initial state.
6) B ($\sqcup$) is the blank character/symbol on tape.
7) $F \subseteq Q$ is the final state (Accept and Reject)

TM → Deterministic
PDA → Non-Deterministic

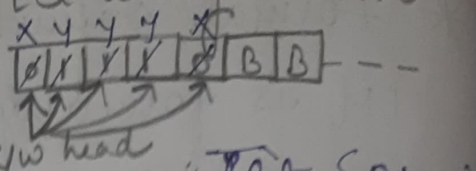


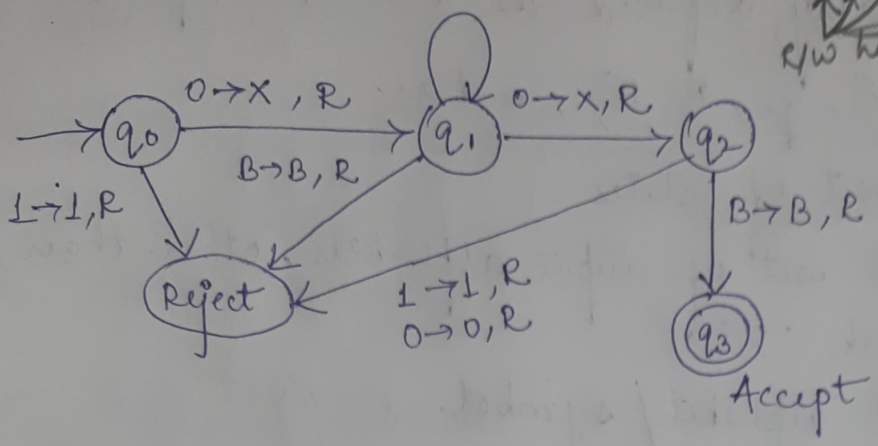
{a → a} implies same symbol will be read & written on the tape.

Example: Construct a TM that recognizes the language $L = 01^*0 = \{01^n0 \mid n \geq 0\}$

00
010
0110
01110

Let string - 01110



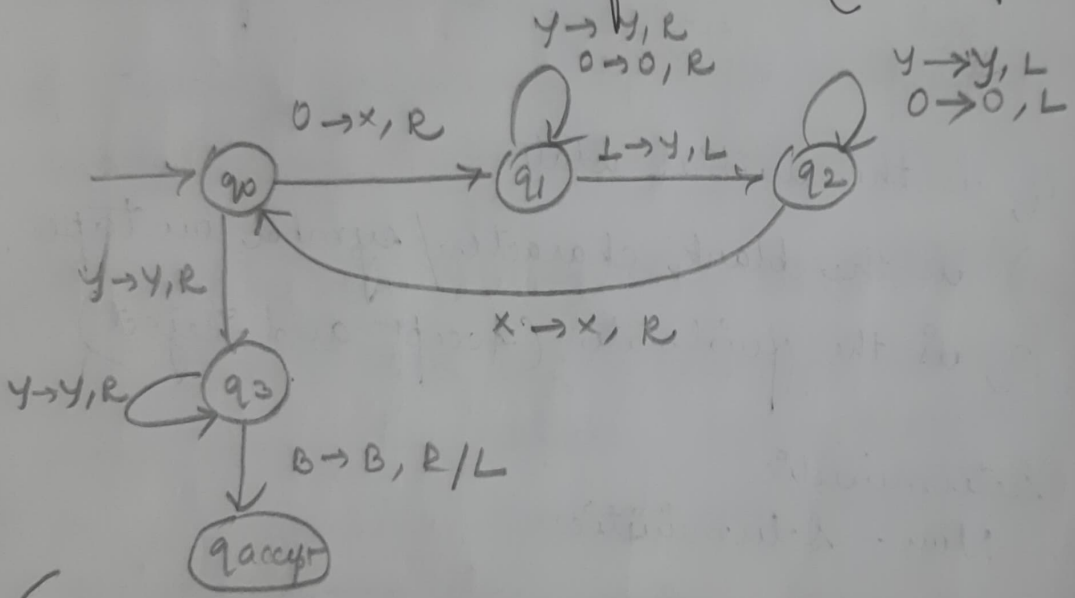


$Q = \{q_0, q_1, q_2, q_3\}$

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

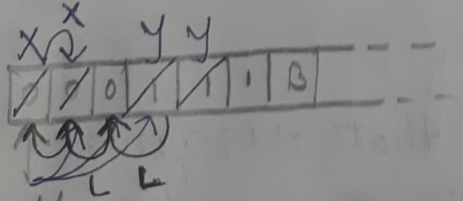
$T = \{0, 1, X, Y, \sqcup\}$

H|W - Construct a TM for $L = \{0^n 1^n \mid n \geq 0\}$



Date - 06.01.24

→ w = 000111



* Variants of Turing Machine :-

1) Multitape Turing Machines
2) Multitape & multihead Turing Machines.
3) Nondeterministic Turing Machines
4) ~~Rough~~ Multihead Turing Machine.
5) Multidimensional Turing Machines.
6) Two-way infinite Turing Machines.

1) A multitape TM has multiple tapes and is controlled by a single head.

2) It is like an ordinary TM with several tapes and each tapes has its own head for reading and writing. (Multitape multihead TM)

3) A nondeterministic TM is a device with a finite control and a single one way infinite tape.

4) A multihead TM contains 2 or more heads to read/write symbols from the same tape. The heads are numbered from 1 through K, and a move of a TM depends on the state and symbol scand by each head.

5) A multidimensional TM has a multidimensional tape where the head can move in any direction, i.e left, right, up or down.

6) ~~A~~ It is a TM with a two-way infinite tape, i.e the the infinite towards

left and right.

* Languages :-

1) Recursive language :-

A language L is said to be recursive if there exists a TM which will accept all the strings in L and reject all the strings not in L.

The Turing Machine will halt every time and give an answer (Accepted or Rejected) for each and every string input.

2) Recursively Enumerable language :-

A language L is said to be enumerable language if there exists a turing machine which will accept and therefore halt for all the input strings in L. But may or may not halt for all input strings which are not in L.

3) Decidable language :-

A language L is decidable if it is a Recursive language.

All decidable languages are recursive languages and vice-versa.

4) Partially Decidable language :-

A language L is partially decidable language if L is a recursively enumerable language.

5) Undecidable language :-

A language is undecidable if it is not decidable.

An undecidable language may sometimes be partially decidable but not decidable.

If a language is not even partially decidable then there exists no TM for that language

* Halting Problem :- (Undecidable problem).

Given a program, will it halt ? → Problem statement

↳ Model H → checks whether P will halt or not

↓

Program P is given as input.

H(P, I) → input to P.

{ Read complexity classes P, NP, NPC, NPH.

* Interpretation :-

This means that, can we design a machine in such a way that if we give a program to that machine as input, then that machine should be able to tell whether that program will halt or not halt on a particular input to that program