PUSHDOWN AUTOMATA

Just as the regular expression have an equivalent automaton - the finite automaton, the CFGs have their machine counterpart - the pushdown automaton.

The pushdown automaton is essentially a F.A with control of both an input tape and a stack [FILO].

A pushdown automata M is a system $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ — 7-tuple

where

i) Q is a finite set of states

ii) Σ is an alphabet called the input alphabet.

iii) Γ is an alphabet, called the stack alphabet.

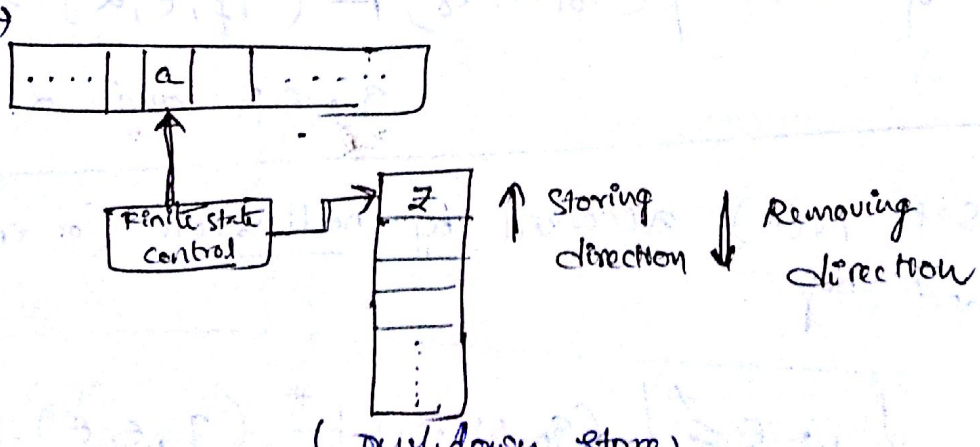
(iv) q_0 in Q is the initial state

(v) Z_0 in Γ is a particular stack symbol called the start symbol.

(vi) $F \subseteq Q$ is the set of final states.

(vii) δ is a mapping from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$

Model (Input tape) →



Instantaneous descriptions:

To formally describe the configuration of a PDA at a given instant we define an instantaneous description (ID). The ID must record the state and stack contents. we define an ID to be a triple (q, w, γ), where q is a state, w a string of input symbols, and γ a string of stack symbols.

Def: Let A be a pda. A move relation (denoted by $\vdash$) between IDs is defined as

$$\begin{cases} (q, a_1 a_2 \cdots a_n, Z_1 Z_2 \cdots Z_n) \vdash (q', a_2 \cdots a_n, \beta Z_2 \cdots Z_m) \\ \text{if } \delta(q, a_1, Z_1) \text{ contains } (q', \beta). \end{cases}$$

Acceptance by PDA

we can define acceptance of input strings by pda in terms of final states (or) in terms of PDS [empty stack].

Def: Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pda. The set accepted by pda by final state is defined by

$$T(A) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q_f, \epsilon, \alpha) \text{ for some } q_f \in F \text{ and } \alpha \in \Gamma^* \}$$

Def: The set $N(A)$ accepted by null store (or empty store) is defined by

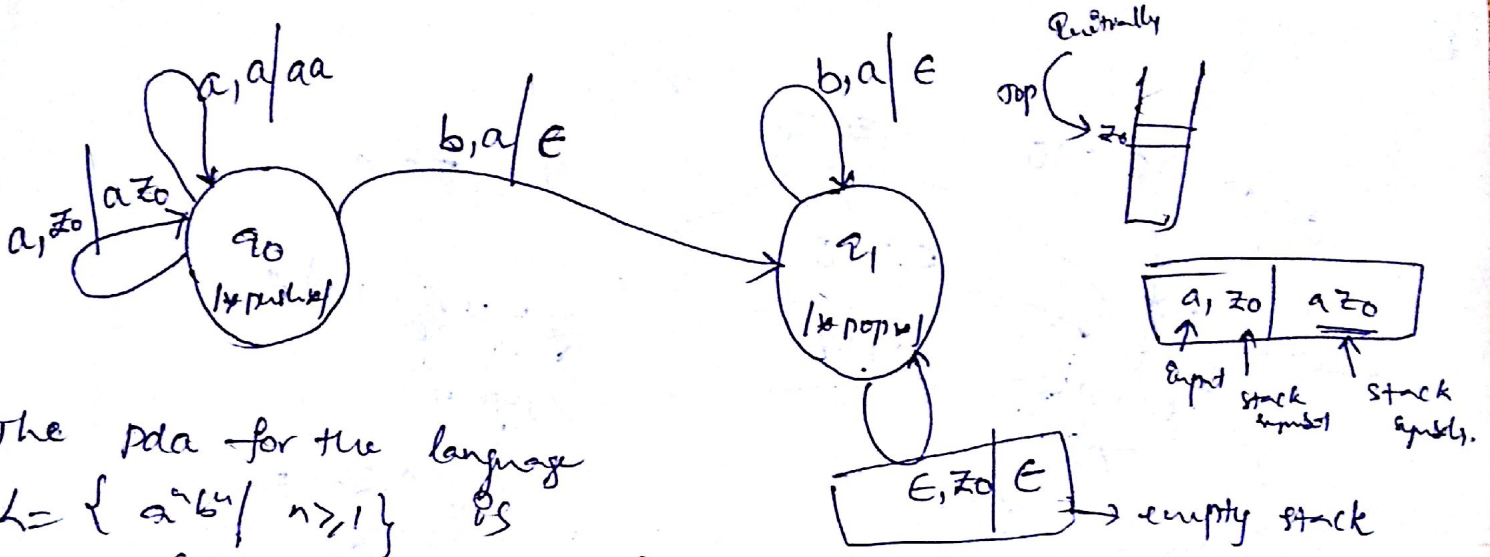
$$N(A) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon) \text{ for some } q \in Q \}$$

Q) Construct pushdown automata which accepts the language (2)

$L = \{ a^n b^n / n \geq 1 \}$ by empty stack [or empty store]

Sol: The language is $L = \{ ab, aabb, aaabbb, \ldots \}$

Logic: we have to push a's into stack. For every 'b', we have to pop 'a' from the stack. If stack is empty, then it is said to be accepted by the pda.



The pda for the language $L = \{ a^n b^n / n \geq 1 \}$ is

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

$Q = \{q_0, q_1\}$ $\Sigma = \{a, b\}$, $\Gamma = \{a, Z_0\}$,

δ is given by

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

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

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

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

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

Only change $\delta(q_1, \epsilon, Z_0) = \{(q_f, Z_0)\}$ where q_f is final state. Then it will give you a PDA by final state.

MOVES:

example: input: aabb

$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0)$

$\vdash (q_0, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

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

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

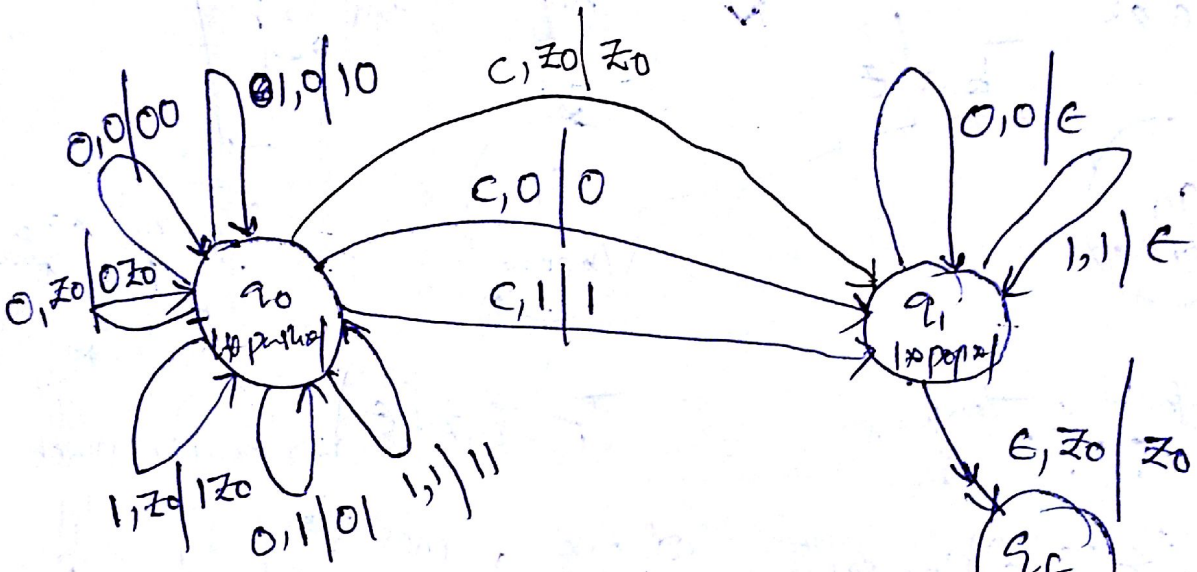
empty stack.

So, $L = \{ a^n b^n / n \geq 1 \}$ is accepted by the above pda

(Q2) Construct PDA which accepts the language

$L = \{ wcw^R / w \in (0+1)^* \}$ by final state.

Sol: Logic: Here we have to push 0's and 1's into the stack until we get 'c'. On input 'c', change the state and pop the element '0' from stack if top of the stack is '0' and input symbol is also '0', pop '1' if top of the stack is '1' and input symbol is also '1'



The PDA is $Q = \{q_0, q_1, q_f\}$, $\Sigma = \{0,1\}$, $\Gamma = \{0,1,Z_0\}$, $F = \{q_f\}$

q_0 & Z_0 initial. where δ is defined as

$\delta(q_0,0,Z_0) = \{(q_0,0Z_0)\}$, $\delta(q_0,1,Z_0) = \{(q_0,1Z_0)\}$

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

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

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

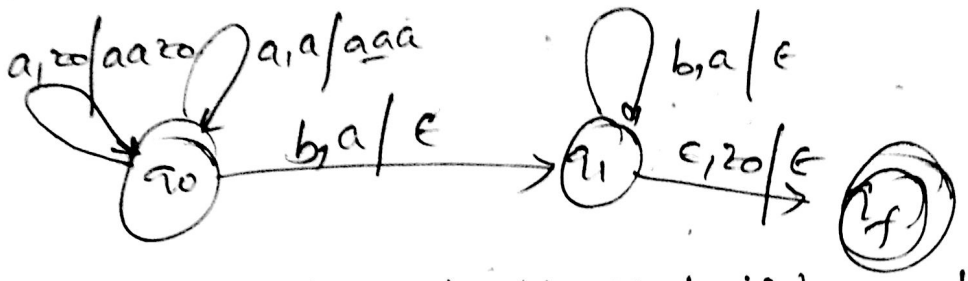
$\delta(q_0,c,0) = \{(q_1,0)\}$, $\delta(q_0,c,1) = \{(q_1,1)\}$

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

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

(6) Construct PDA for the language $L = \{a^n b^{2n} / n \geq 1\}$ by final state.

Sol: Logic: For each i/p symbol 'a', we have to push two a's into the stack.
For each 'b', pop one 'a' from the stack. $L = \{abb, aabbbb \ldots\}$



- $Q = \{q_0, q_1, q_f\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, z_0\}$, $F = \{q_f\}$

$\delta(q_0, a, z_0) = \{(q_0, a a z_0)\}$

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

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

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

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

Verification: Input : aabbbb

$(q_0, aabbbb, z_0)$ ↑

$\vdash (q_0, a\,bbbb, aa z_0)$ // push two a's

$\vdash (q_0, bbbb, aaaa z_0)$ // push two a's

$\vdash (q_0, bbb, aaa z_0)$ // pop one 'a'

$\vdash (q_0, bb, aa z_0)$ // pop one 'a'

$\vdash (q_0, b, a z_0)$ // pop one 'a'

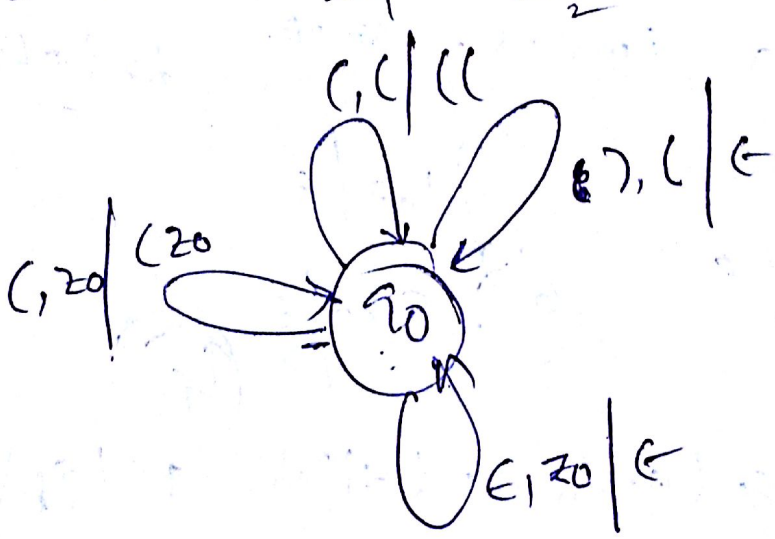
$\vdash (q_0, \epsilon, z_0)$ // pop one 'a'

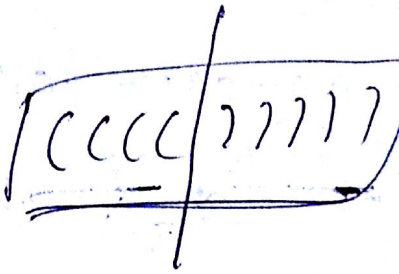
$\vdash (q_1, \epsilon, \epsilon)$ // (stack is empty. so it is accepted)

Accepted

(9) Obtain a PDA to accept the strings of balanced p

$L = \{ (), (())(), ((())), (()()) \ldots \}$





Sol:

① $\delta(q_0, (, Z_0) = \{(q_0, (Z_0)\}$

② $\delta(q_0, (, () = \{(q_0, (()\}$

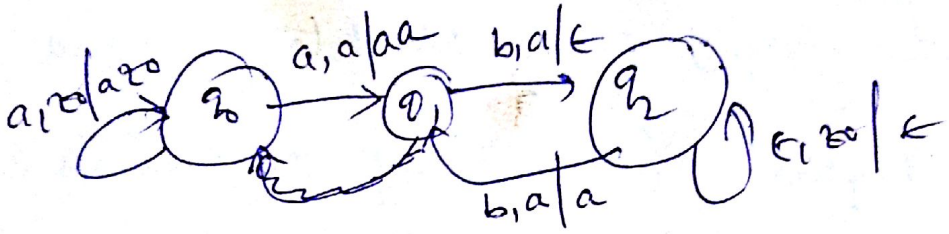
③ $\delta(q_0,), () = \{(q_0, \epsilon)\}$

④ $\delta(q_0, \epsilon, Z_0) = \{(q_0, \epsilon)\}$

Design PDA for the languages

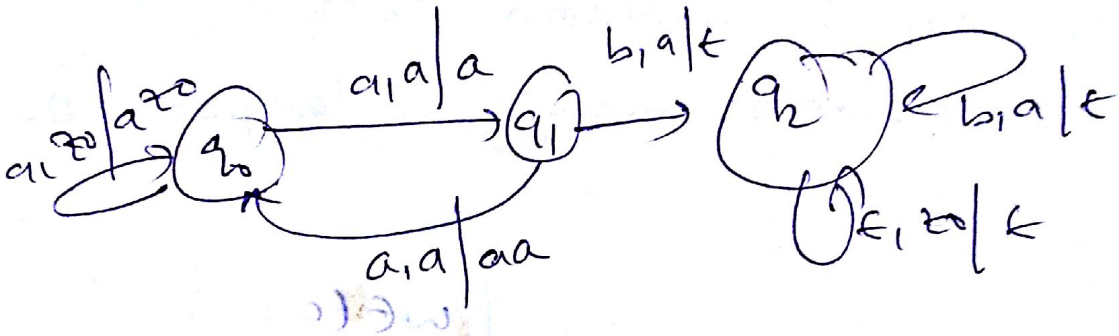
aabbbb ←
1 1 1 1 1

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



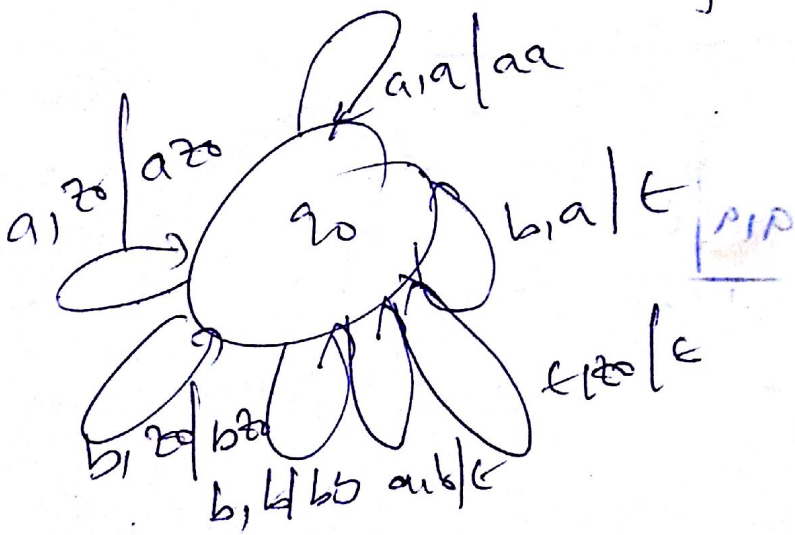
② $L = \{ a^{2n} b^n \mid n \geq 1 \}$

aaaabb ←



③ $L = \{ w \in (a+b)^* \mid n_a(w) = n_b(w) \}$

no of a's = no of b's in 'w'



PDA

Deterministic PDA (DPDA)

∴ A pda $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if

ⓐ $\delta(q, a, Z)$ is either empty or a singleton. and

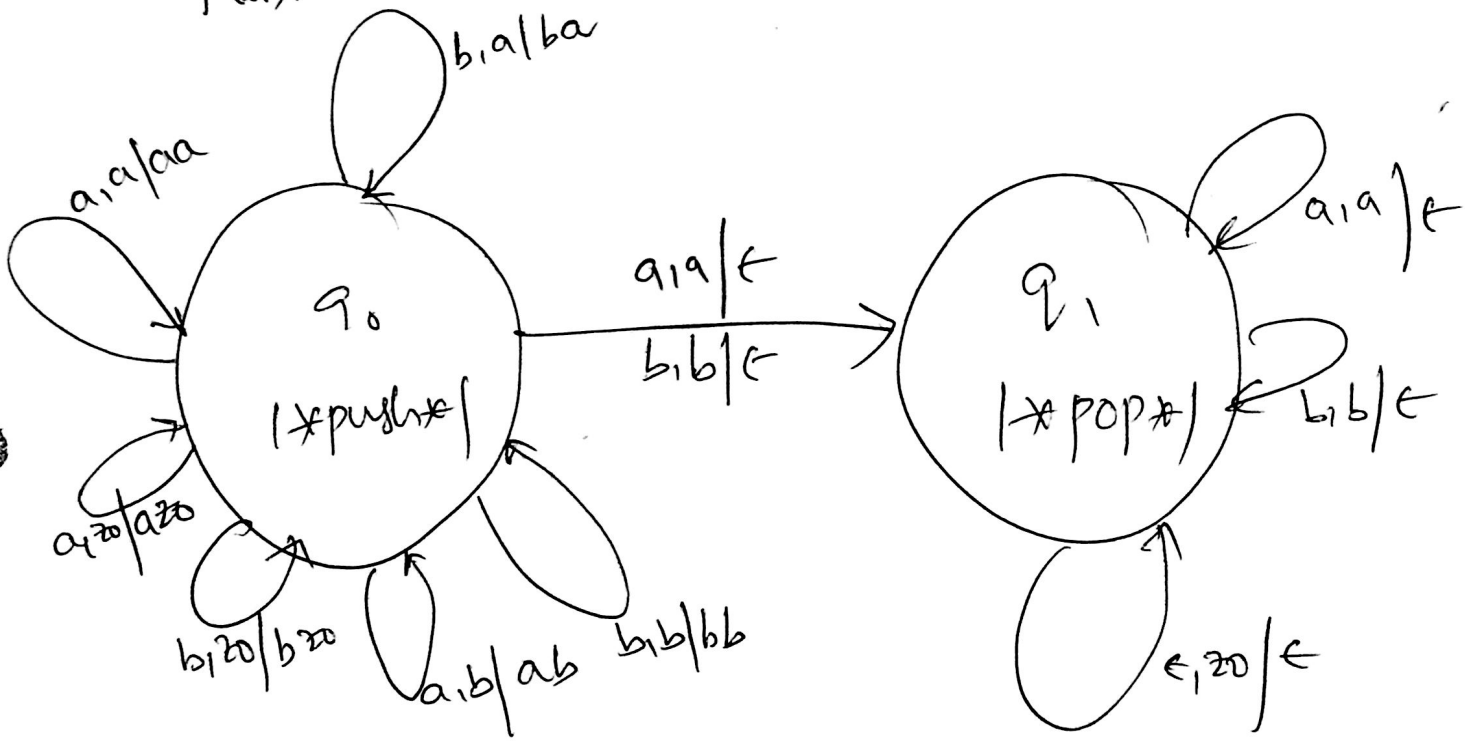
ⓑ $\delta(q, \in, Z) \neq \phi$ implies $\delta(q, a, Z) = \phi$ for each 'a' $\in \Sigma$.

⇒ PDA for the language $\{wcw^R \mid w \in (a+b)^*\}$ is DPDA

Here deterministic in the sense that atmost one move is possible from any instantaneous description.

→ DPDA accepts Deterministic context free languages (DCFL) as inputs.

⇒ PDA for the language $\{ww^R \mid w \in (a+b)^*\}$ is Non-deterministic PDA.



Getting PDA from the given CFG. (CFG → PDA)

In order to design a PDA from a CFG, CFG must be in Greibach Normal Form. (GNF).

Procedure:

Step1: Convert the given CFG into GNF.

Step2: Let q_0 be the start state and Z_0 is the initial symbol on the stack. without consuming any input, push the start symbol 'S' onto the stack and change the state to q_1.

$$\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$$

Step3: For each production of the form $A \to a\alpha$ introduce the transition

$$\delta(q_1, a, A) = (q_1, \alpha)$$

$A \to a\epsilon$

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

Step4: Finally, in state q_1, without consuming any input, change the state to q_f which is final state.

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

examples 2009, 10

Example:

Q) construct a PDA to the following CFG.

$$\text{CFG} \begin{cases} S \to aABC \\ A \to aB \mid a \\ B \to bA \mid b \\ C \to a \end{cases}$$

Sol: Step1: CFG is already in GNF

Step2: $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0) \to (1)$

① $S \to aSbb \mid aab$

② $S \to aAA$

$A \to aS \mid bS \mid a$

CFG → PDA

Step 3: For each production $A \to aα$ introduce the transition,

$\delta(q_1, a, A) = (q_1, α)$

Production	Transition
$S \to aABC$	$\delta(q_1, a, S) = (q_1, ABC)$
$A \to aB$	$\delta(q_1, a, A) = (q_1, B)$
$A \to a$	$\delta(q_1, a, A) = (q_1, \epsilon)$
$B \to bA$	$\delta(q_1, b, B) = (q_1, A)$
$B \to b$	$\delta(q_1, b, B) = (q_1, \epsilon)$
$C \to a$	$\delta(q_1, a, C) = (q_1, \epsilon)$

Step 4:

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

∴ PDA for the given CFG is

$M = (\{q_0, q_1, q_f\}, \{a, b\}, \{S, A, B, C, Z_0\}, \delta, Z_0, \{q_f\})$

where δ is defined as

1. $\delta(q_0, \epsilon, Z_0) = \{(q_1, SZ_0)\}$
2. $\delta(q_0, a, S) = \{(q_1, ABC)\}$
3. $\delta(q_1, a, A) = \{(q_1, B), (q_1, \epsilon)\}$
4. $\delta(q_1, b, B) = \{(q_1, A), (q_1, \epsilon)\}$
5. $\delta(q_1, a, C) = \{(q_1, \epsilon)\}$
6. $\delta(q_1, \epsilon, Z_0) = \{(q_f, Z_0)\}$

$S \Rightarrow a\underline{A}BC$
$\Rightarrow aa\underline{B}BC$
$\Rightarrow aab\underline{B}C$
$\Rightarrow aabbC$
$\Rightarrow$ aabba

derived from the given CFG

verification: Input string: aabba

$(q_0, aabba, Z_0) \vdash (q_1, aabba, SZ_0)$
$\vdash (q_1, abba, ABCZ_0)$
$\vdash (q_1, bba, BBCZ_0)$
$\vdash (q_1, ba, BCZ_0)$
$\vdash (q_1, a, CZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_f, \epsilon, Z_0)$

accepted

(Q2) Obtain a PDA from the given CFG

$S \rightarrow aAA$

$A \rightarrow aS \mid bS \mid a$

Sol: ① $\delta(q_0, \epsilon, Z_0) = \{(q_1, SZ_0)\}$

Production	Transition
$S \rightarrow aAA$	$\delta(q_1, a, S) = \{(q_1, AA)\}$
$A \rightarrow aS$	$\delta(q_1, a, A) = \{(q_1, S)\}$
$A \rightarrow bS$	$\delta(q_1, b, A) = \{(q_1, S)\}$
$A \rightarrow a$	$\delta(q_1, a, A) = \{(q_1, \epsilon)\}$

③ $\delta(q_1, \epsilon, Z_0) = \{(q_f, Z_0)\}$

Verification:

$S \Rightarrow aAA \Rightarrow aaA$

$\Rightarrow aaSA \Rightarrow aabSA$

$\Rightarrow aabbSA \Rightarrow aabaAA$

$\Rightarrow aabaaA$

$\Rightarrow$ aabaaa string derived from the given CFG

I/P string: aabaaa

$(q_0, aabaaa, Z_0) \vdash (q_1, aabaaa, Z_0)$

$\vdash (q_1, abaaa, AZ_0)$

$\vdash (q_1, baaa, AAZ_0)$

$\vdash (q_1, aaa, SAZ_0)$

$\vdash (q_1, aa, AAAZ_0)$

$\vdash (q_1, a, AAZ_0)$

$\vdash (q_1, \epsilon, AZ_0)$

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

Accepted by PDA

Q) obtain a PDA from the given CFG

$S \to c \mid aSa \mid bsb$ $\quad L = \{wcw^R \mid w \in (a+b)^*\}$

Sol: Step1: Convert the given CFG into GNF

$S \to c$

$S \to aSA, \quad A \to a$

$S \to bSB, \quad B \to b$

Step2:

① $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$

Step3:

production	Transition
$S \to c$	$\delta(q_1, c, S) = \{(q_1, \epsilon)\}$
$S \to aSA$	$\delta(q_1, a, S) = \{(q_1, SA)\}$
$S \to bSB$	$\delta(q_1, b, S) = \{(q_1, SB)\}$
$A \to a$	$\delta(q_1, a, A) = \{(q_1, \epsilon)\}$
$B \to b$	$\delta(q_1, b, B) = \{(q_1, \epsilon)\}$

Step4: $\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

$S \Rightarrow bsb$
$\Rightarrow basab$
$\Rightarrow bacab$

Verification: Take bacab as input string

$(q_0, bacab, Z_0) \vdash (q_1, bacab, SZ_0)$
$\vdash (q_1, acab, SBZ_0)$
$\vdash (q_1, cab, SABZ_0)$
$\vdash (q_1, ab, ABZ_0)$
$\vdash (q_1, b, BZ_0)$
$\vdash (q_1, \epsilon, Z_0)$
$\vdash (q_f, Z_0)$

Accepted by the PDA.

Obtaining CFG from a given PDA: [PDA → CFG]

If $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is a PDA, then there exists a CFG G such that $L(G) = N(M)$, where $N(M)$ is the language accepted by empty stack.

Procedure:

Step 1: we define $G = (V, T, P, S)$ where

$$V = \{S\} \cup \{[q, z, q'] \mid q, q' \in Q, z \in \Gamma\}$$

[i.e any element of V is either the new symbol 'S' acting as the start symbol for G (or) an ordered triple whose first and third elements are states and the second element is a pushdown symbol.]

Step 2: The productions in 'P' are induced by moves of PDA as follows:

Rule 1: S-productions are given by $S \to [q_0, z_0, q]$ for every $q \in Q$.

Rule 2: Each move erasing a pushdown symbol given by

$(q', \epsilon) \in \delta(q, a, z)$ induces the production $[q, z, q'] \to a$

$\delta(q, a, z) = (q', \epsilon)$
$[q, z, q'] \to a$

Rule 3: Each move not erasing a pushdown symbol given by

$(q_1, z_1 z_2 \cdots z_m) \in \delta(q, a, z)$ induces many productions of the form

$$[q, z, q'] \to a [q_1, z_1, q_2] [q_2, z_2, q_3] \cdots [q_m, z_m, q']$$

where each of states $q, q_2, \cdots q_m$ can be any state in Q.

Each move yields many productions because of rule 3.

$\delta(q, a, z) = (q_1, z_1 z_2)$
$[q, z, q'] \to a [q_1, z_1, q_2] [q_2, z_2, q']$

Q: construct a CFG which accepts $N(A)$ where

$A = (\{q_0, q_1\}, \{a, b\}, \{z_0, z\}, \delta, q_0, z_0, \phi)$

δ is given by

$\delta(q_0, b, z_0) = \{(q_0, z z_0)\}$

$\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$

$\delta(q_0, b, z) = \{(q_0, zz)\}$

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

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

$\delta(q_1, a, z_0) = \{(q_0, z_0)\}$

Sol:

Let $G = (V, \{a, b\}, P, S)$ where V consists of

S, $[q_0, z_0, q_0]$, $[q_0, z_0, q_1]$, $[q_0, z, q_0]$, $[q_0, z, q_1]$, $[q_1, z_0, q_0]$, $[q_1, z, q_0]$, $[q_1, z_0, q_1]$, $[q_1, z, q_1]$

The productions are

P_1: $S \to [q_0, z_0, q_0]$

P_2: $S \to [q_0, z_0, q_1]$

} rule ①

$\delta(q_0, b, z_0) = (q_0, z z_0)$ gives

P_3: $[q_0, z_0, q_0] \to b\,[q_0, z, q_0]\,[q_0, z_0, q_0]$

P_4: $[q_0, z_0, q_0] \to b\,[q_0, z, q_1]\,[q_1, z_0, q_0]$

P_5: $[q_0, z_0, q_1] \to b\,[q_0, z, q_0]\,[q_0, z, q_1]$

P_6: $[q_0, z_0, q_1] \to b\,[q_0, z, q_1]\,[q_1, z_0, q_1]$

$\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$ gives

P_7: $[q_0, z_0, q_0] \to \epsilon$

$\delta(q_0, b, z) = \{(q_0, zz)\}$ gives

P_8: $[q_0, z, q_0] \to b\,[q_0, z, q_0]\,[q_0, z, q_0]$

P_9: $[q_0, z, q_0] \to b\,[q_0, z, q_1]\,[q_1, z, q_0]$

P_{10}: $[q_0, z, q_1] \to b\,[q_0, z, q_0]\,[q_0, z, q_1]$

P_{11}: $[q_0, z, q_1] \to b\,[q_0, z, q_1]\,[q_1, z, q_1]$

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

$\therefore$ P_{24}: $[q_0, z, q_0] \to a[q_1, z, q_0]$

P_{25}: $[q_0, z, q_1] \to a[q_1, z, q_1]$

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

P_{26}: $[q_1, z, q_1] \to b$

$\delta(q_1, a, z_0) = \{(q_0, z_0)\}$ gives

P_{27}: $[q_1, z_0, q_0] \to a[q_0, z_0, q_0]$

P_{28}: $[q_1, z_0, q_1] \to a[q_0, z_0, q_1]$

$P_1 - P_{16}$ give the productions in P'

(9) construct a CFG which accepts $N(A)$ where A is $(\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, z_0\}, \delta, q_0, z_0, q_2)$ where δ is given by

$L = \{ wcw^R \mid w \in \{a,b\}^* \}$

$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, c, z_0) = (q_1, z_0)$
$\delta(q_1, c, a) = (q_1, a)$
$\delta(q_1, c, b) = (q_1, b)$
$\delta(q_1, a, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$

PDA

(9) Construct a CFG from the following PDA

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

where δ is given by

① $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$

② $\delta(q_0, 0, X) = \{(q_0, XX)\}$

③ $\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$

④ $\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$

⑤ $\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$

⑥ $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

$[q, Z, q']$ $Z \in \Sigma$, $q, q' \in Q$

Sol: Let $G = \{V, \{0,1\}, P, S\}$ where V consists of

$V = \{S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1], [q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1]\}$

Productions are:

P_1: $S \to [q_0, Z_0, q_0]$

P_2: $S \to [q_0, Z_0, q_1]$

using rule ①

By using rule ②, we can write the productions for the last four transitions

Take ③ $\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$

P_3: $[q_0, X, q_1] \to 1$

④ $\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$

P_4: $[q_1, X, q_1] \to 1$

⑤ $\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$

P_5: $[q_1, X, q_1] \to \epsilon$

⑥ $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$

P_6: $[q_1, Z_0, q_1] \to \epsilon$

By using rule ②

Diagram rule ②

rule ① $\delta(q_0, 0, z_0) = \{(q_0, x z_0)\}$

rule ② $\delta(q, a, z) = (q_1, z_1 z_0 \ldots z_n)$ pop ②

$[q, z, q'] \rightarrow \delta(q, a, z)\,[z_1 z_0 \ldots z_n \cdot q_n]$

p_0: $[q_0, z_0, q_0] \rightarrow 0\,[q_0, x, q_0]\,[q_0, z_0, q_0]$ $\Rightarrow$ $q' = q_0$, $q_1 = q_0$, $q_2 = q_0$

p_1: $[q_0, z_0, q_0] \rightarrow 0\,[q_0, x, q_1]\,[q_1, z_0, q_0]$ $\Rightarrow$ $q' = q_0$, $q_1 = q_0$, $q_2 = q_1$

p_{10}: $[q_0, z_0, q_1] \rightarrow 0\,[q_0, x, q_0]\,[q_0, z_0, q_1]$ $\Rightarrow$ $q' = q_1$, $q_1 = z_0$, $q_2 = q_0$

p_{11}: $[q_0, z_0, q_1] \rightarrow 0\,[q_0, x, q_1]\,[q_1, z_0, q_1]$ $\Rightarrow$ $q' = q_1$, $q_1 = z_0$, $q_2 = q_1$

rule ② $\delta(q_0, 1, x) = \{(q_0, xx)\}$

p_{12}: $[q_0, z_0, q_0] \rightarrow 1\,[q_0, x, q_0]\,[q_0, x, q_0]$

p_{13}: $\rightarrow 1\,[q_0, x, q_1]\,[q_1, x, q_0]$

p_{14}: $[q_0, z_0, q_1] \rightarrow 1\,[q_0, x, q_0]\,[q_0, x, q_1]$

p_{15}: $[q_0, z_0, q_1] \rightarrow 1\,[q_0, x, q_1]\,[q_1, x, q_1]$

(b) For the given PDA, the CFG containing the productions $p_1 - p_{15}$.

(4) construct PDA for the CFG

① $S \to aA, A \to aABC \mid bB \mid a, B \to b, C \to c$.

② $S \to aBC, A \to abc, B \to ab, C \to AB, C \to c$

③ $S \to aB \mid bA, A \to a \mid aS \mid bAA, B \to b \mid bS \mid aBB$

④ $S \to aABB \mid aAA$

$A \to aBB \mid a$

$B \to bBB \mid A$

$B \to A$ is a unit production.

$B \to A \to aBB \Rightarrow B \to aBB$

$B \to A \to a \Rightarrow B \to a$

finally CFG in GNF is $S \to aABB \mid aAA$

$A \to aBB \mid a$

$B \to bBB \mid aBB \mid a$

$S \to aABB$, $S \to aAA$, $A \to aBB$, $A \to a$, $B \to bBB$, $B \to aBB$, $B \to a$

① $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$

② $\delta(q_1, a, S) = (q_1, ABB)$

③ $\delta(q_1, a, S) = (q_1, AA)$

④ $\delta(q_1, a, A) = (q_1, BB)$

⑤ $\delta(q_1, a, A) = (q_1, \epsilon)$

⑥ $\delta(q_1, b, B) = (q_1, BB)$

⑦ $\delta(q_1, a, B) = (q_1, BB)$

⑧ $\delta(q_1, a, B) = (q_1, \epsilon)$

⑨ $\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$ By Final State

$(q_0, aaa, Z_0) \vdash (q_1, aaa, SZ_0)$

$\vdash (q_1, aa, AAZ_0)$

$\vdash (q_1, a, AZ_0)$

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

$\vdash (q_f, Z_0)$

Accepted

$S \Rightarrow aAA$

$\Rightarrow aaA$

$\Rightarrow aaa$

(5) $S \to aB \mid bA$, $A \to a \mid aS \mid bAA$, $B \to b \mid bS \mid aBB$

$S \to aB$

$A \to aB \mid a$

$B \to b$

(6) $S \to aSbb \mid aab$

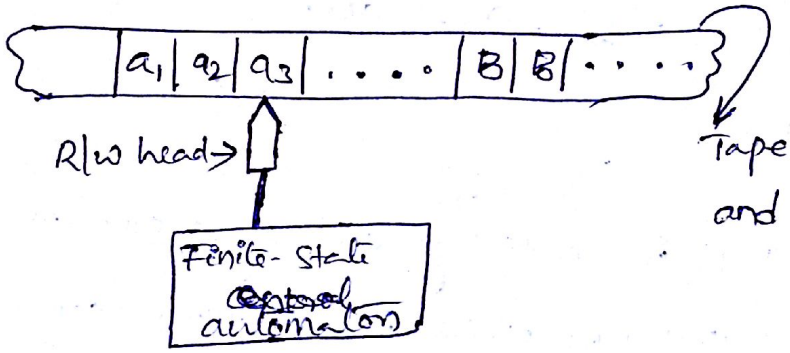
TURING MACHINE

The Turing machine (TM) is a simple mathematical model of a general purpose computer. In other words, Turing machines models the computing power of a computer. i.e. the Turing machine is capable of performing any calculation which can be performed by any computing machine.

Turing machine model :-

The Turing machine can be thought of as a finite-state automata connected to a R/W (read/write) head. It has one tape which is divided into a no of cells.

Block Diagram : [Turing machine model]



Each cell can store only one symbol. The input to and output from the finite-state automaton are effected by the R/W head which can examine one cell at a time. In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automaton to determine :-

(i) a new symbol to be written on the tape in the cell under the R/W head,

(ii) a motion of the R/W head along the tape: either the head moves one cell left (L) or one cell right (R).

(iii) the next state of the automaton

(iv) whether to halt (or) not.

Def: A Turing machine M is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where

① Q is a finite nonempty set of states.

② Γ is a finite nonempty set of tape symbols.

③ $B \in \Gamma$ is the blank.

④ Σ is a nonempty set of input symbols and is a subset of Γ and $B \notin \Sigma$.

⑤ δ is the transition function mapping the states of F.A and tape symbols to states, tape symbols and movement of the head, i.e. $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

⑥ $q_0 \in Q$ is the initial state

⑦ $F \subseteq Q$ is the set of final states.

→ A Turing machine (TM) is nothing but a FSM with an infinite auxiliary memory in which information can be stored (or) recalled in any manner by moving the tape left (or) right of any no of cells.

→ The TM can write intermediate (or) final results on that tape.

TM is more powerful than FA, PDA with no limitations and has external memory capable of remembering long no of sequence I/ps.

TM finds: a) well-formedness of parentheses

b) Addition (or) Multiplication of unary numbers.

c) GCD of two +ve numbers

d) Recognition of palindromes.

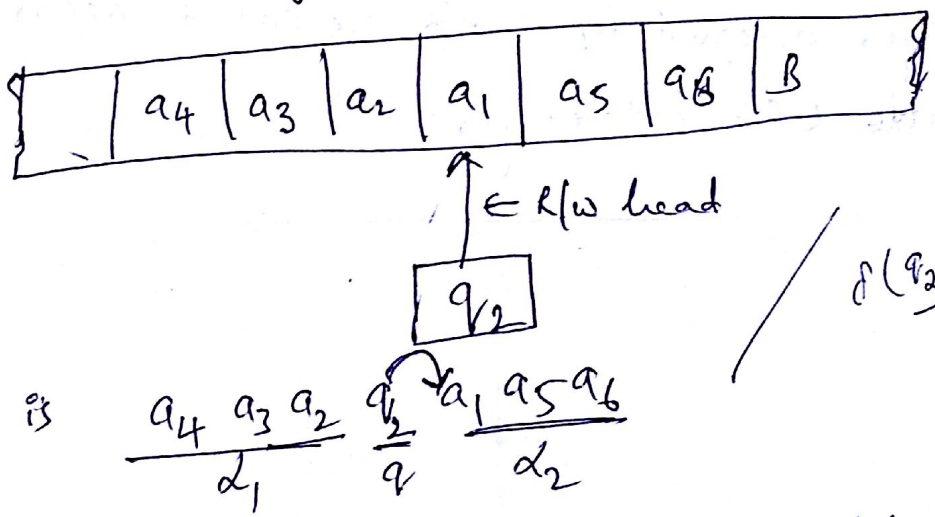
Instantaneous Description (ID)

The current configuration of a TM can be described by a finite-length string that includes all the tapes cells from left-most to right-most non-blank. The state and the position of the head are shown by placing the state within the sequence of tape symbols, just to the left of the cell to be scanned, we denote an ID of the TM as follows:

$\alpha_1 \, q \, \alpha_2$ where α_1 is the substring of i/p string is scanned by TM.

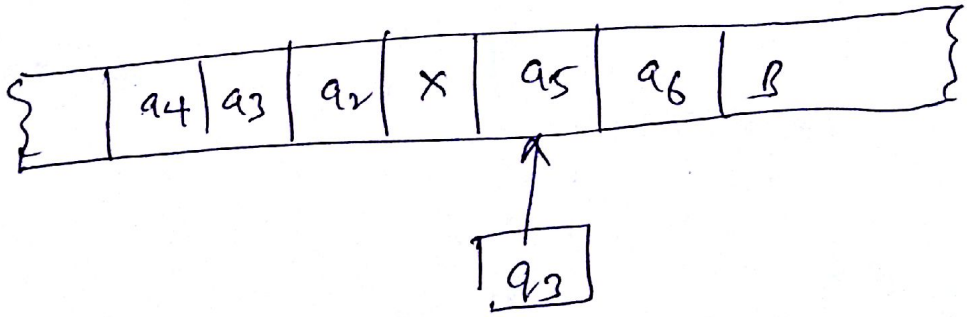
The first symbol of α_2 is the current symbol under R/W head.

Example



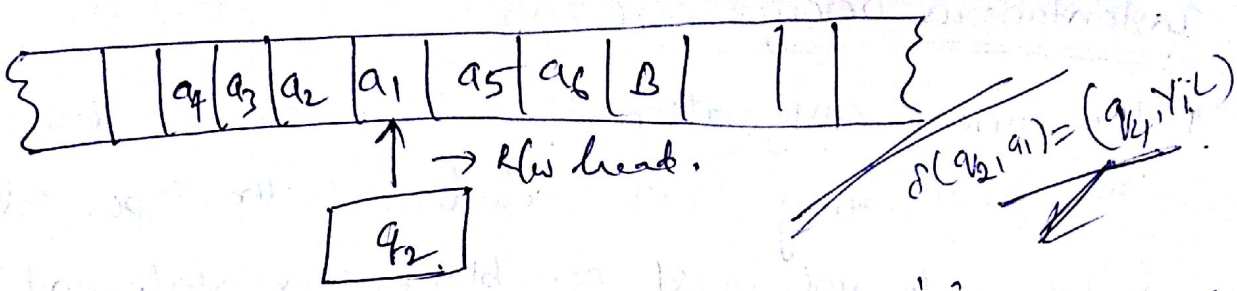
Right move

for example, a_1 is replaced 'X' and move right and change state to 'q_3'. i.e



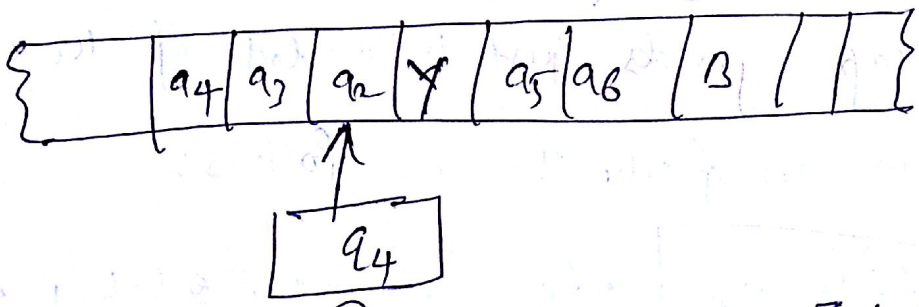
ID is $a_4 a_3 a_2 X q_3 a_5 a_6$ [Right-move]

Left move:



for example, a_1 is replaced by Y, change state to q_4 and move left

i.e



ID is $a_4 a_3 q_4 a_2 Y a_5 a_6$ [left-move]

→ The languages accepted by TM's are called Recursively Enumerable (RE) languages.

Design of Turing Machines :-

Guidelines:-

(i) The fundamental objective in scanning a symbol by R/W-head is to know what to do in the future. The machine must remember the past symbols scanned. The TM can remember this by going to the next unique state.

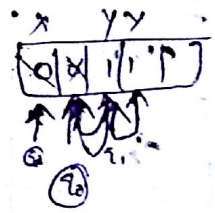
(ii) The no of states must be minimised. This can be achieved by changing the states only when there is a change in the written symbol (or) when there is a change in the movement of R/W head.

(5) Design a TM to accept the language $L=\{0^n 1^n / n \geq 1\}$

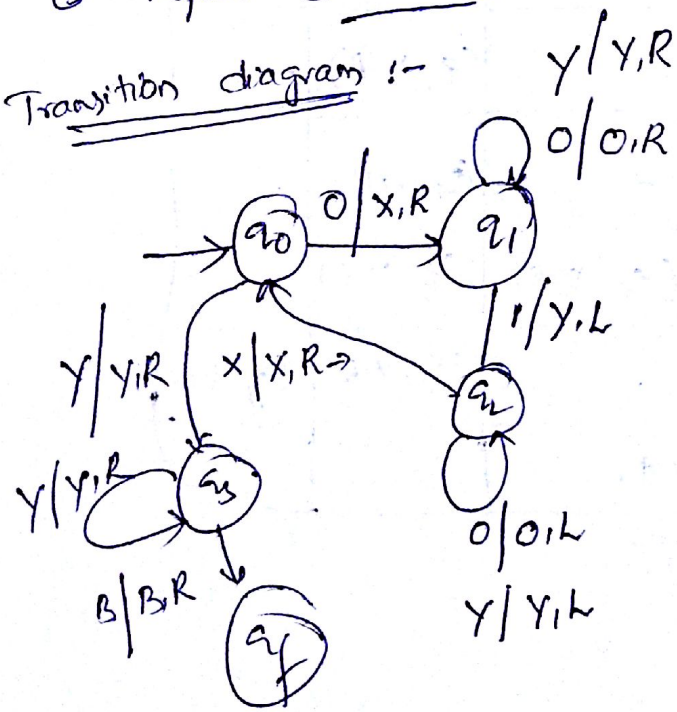
Sol: ① Replace the leftmost 0 by X and change the state to q_1 and then move the R/W head towards right. This is because, after a zero is replaced, we have to replace the corresponding 1, so that no of 0's matches with no of 1's.

② Search for the leftmost 1 and replace it by the symbol Y and move towards left (so as to obtain the leftmost '0' again)

③ Repeat ① and ②.



Transition diagram :-



So, the TM to accept the language $L = \{0^n 1^n / n \geq 1\}$ is given by

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

$Q = \{q_0, q_1, q_2, q_3, q_f\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, X, Y, B\}$

$q_0 \in Q$, $B \in \Gamma$, $F = \{q_f\}$, where δ is defined as

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

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

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

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

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

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

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

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

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

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

I/p string : 0011

(I.D) (Instant)

$q_0 0011B \vdash X q_1 011B$

$\vdash X 0 q_1 11B$

$\vdash X q_2 0 Y 1 B$

$\vdash q_2 X 0 Y 1 B$

$\vdash X q_0 0 Y 1 B$

$\vdash X X q_1 Y 1 B$

$\vdash X X Y q_1 1 B$

$\vdash X X q_2 Y Y B$

$\vdash X q_2 X Y Y B$

$\vdash X X q_0 Y Y B$

$\vdash X X Y q_3 Y B$

$\vdash X X Y Y q_3 B$ (Final I.D)

$\vdash X X Y Y B q_f B$ (Accepted)

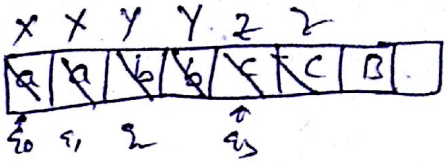
Transition table

δ

States	0	1	X	Y	B
→ q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, X, 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_f, B, R)
q_4	Final state, accepted.				

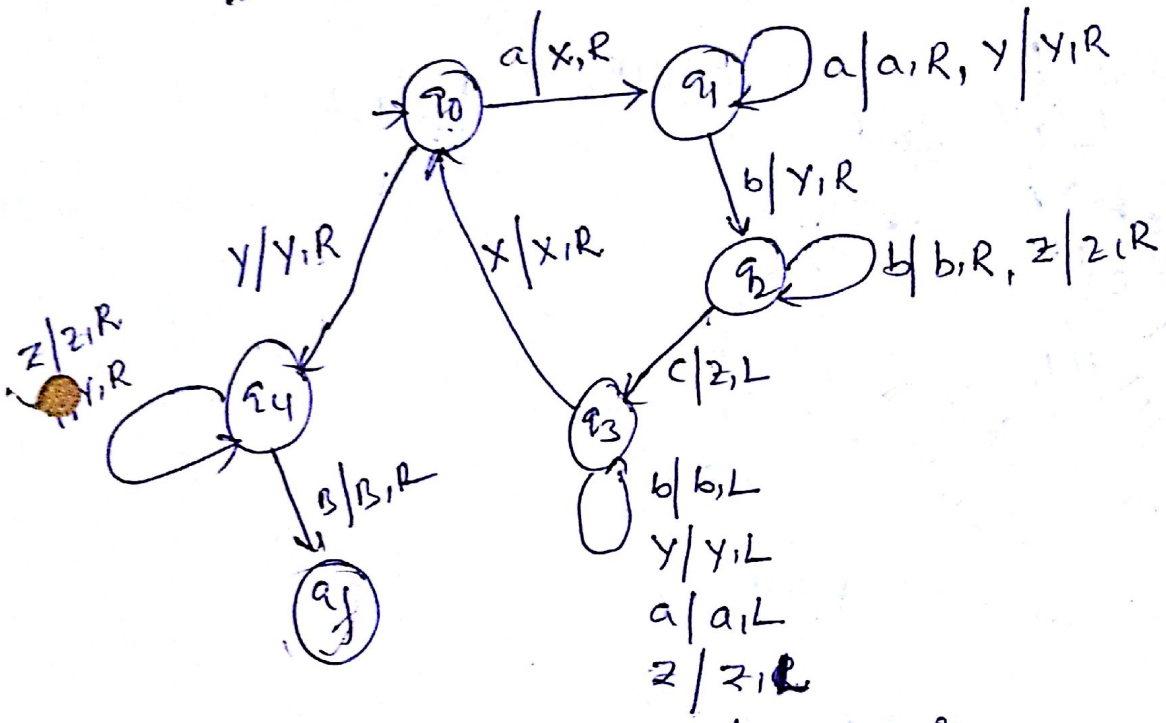
(9) Design a turing machine to accept the language

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



Sol:

Transition diagram:



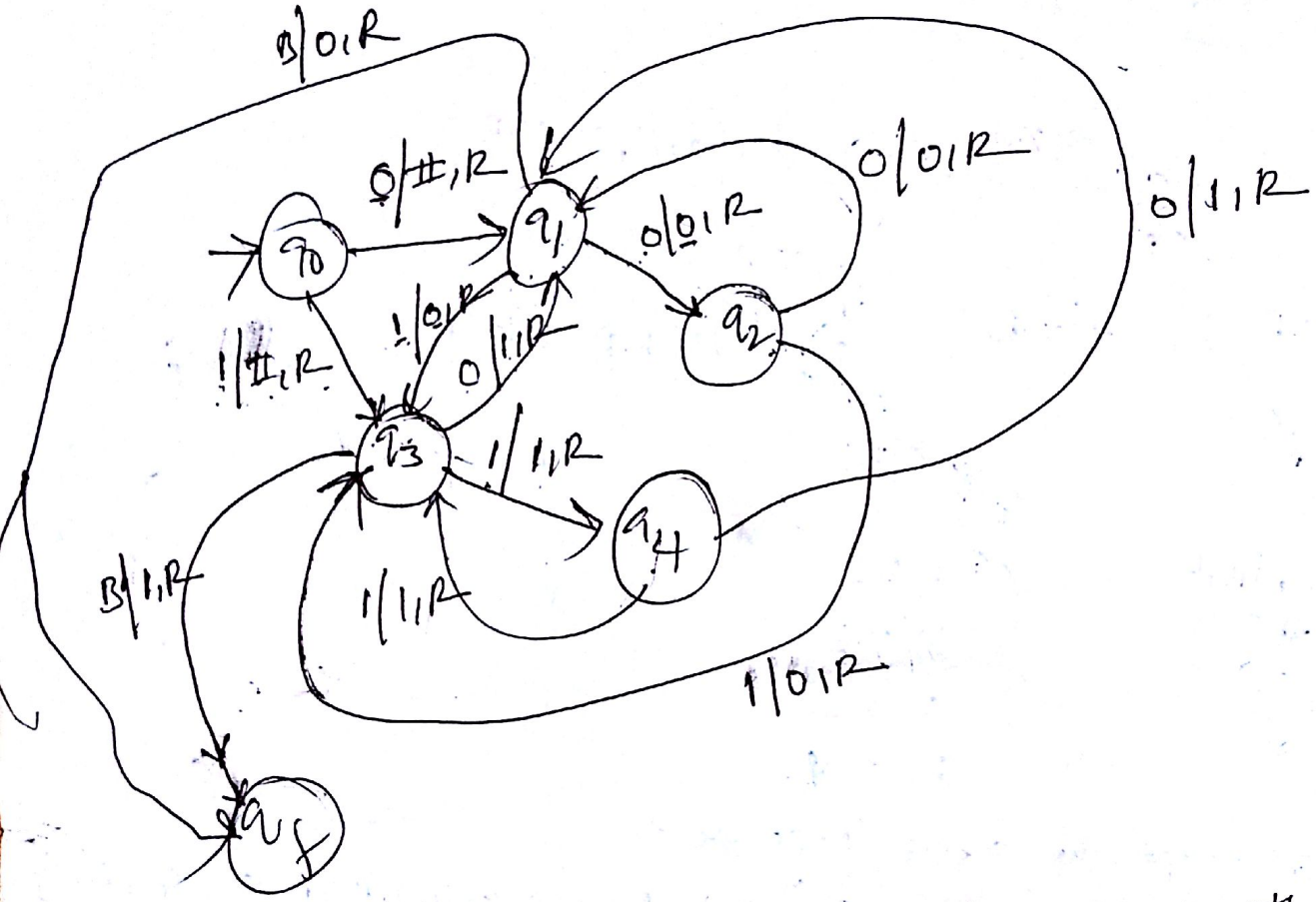
The TM to accept the above language is

$M = (\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{a, b, c, X, Y, Z, B\}, \delta, q_0, B, \{q_f\})$

where δ is defined as

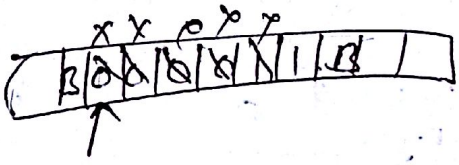
Inputs → / States ↓	a	b	c	X	Y	Z	B
q_0	(q_1, X, R)	-	-	-	(q_4, Y, R)	-	-
q_1	(q_1, a, R)	(q_2, Y, R)	-	-	(q_1, Y, R)	-	-
q_2	~~(q_2, X, R)~~ -	(q_2, b, R)	(q_3, Z, L)	-	-	(q_2, Z, R)	-
q_3	(q_3, a, L)	(q_3, b, L)		(q_0, X, R)	(q_3, Y, L)	(q_3, Z, L)	-
q_4					(q_4, Y, R)	(q_4, Z, R)	(q_f, B, R)
q_f	Final State; Accepted						

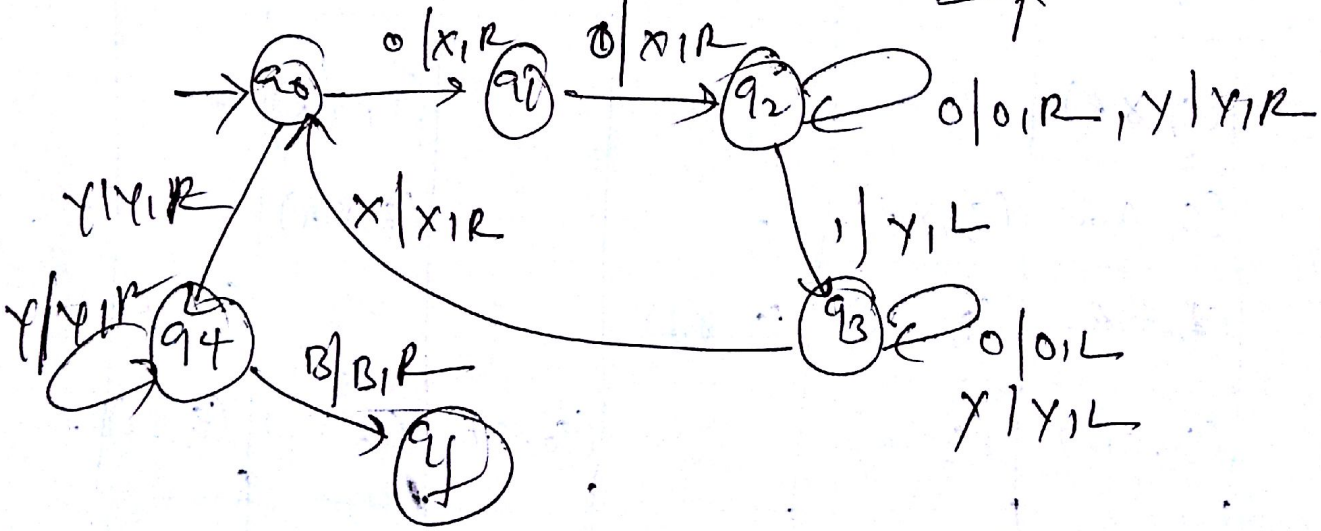
Q) Design a turing machine that shifts the input string by one position right by inserting # as the first character.



Q) Design a turing machine to accept the language

$L = \{ 0^{2^n} | n | \; n \geq 1 \}$

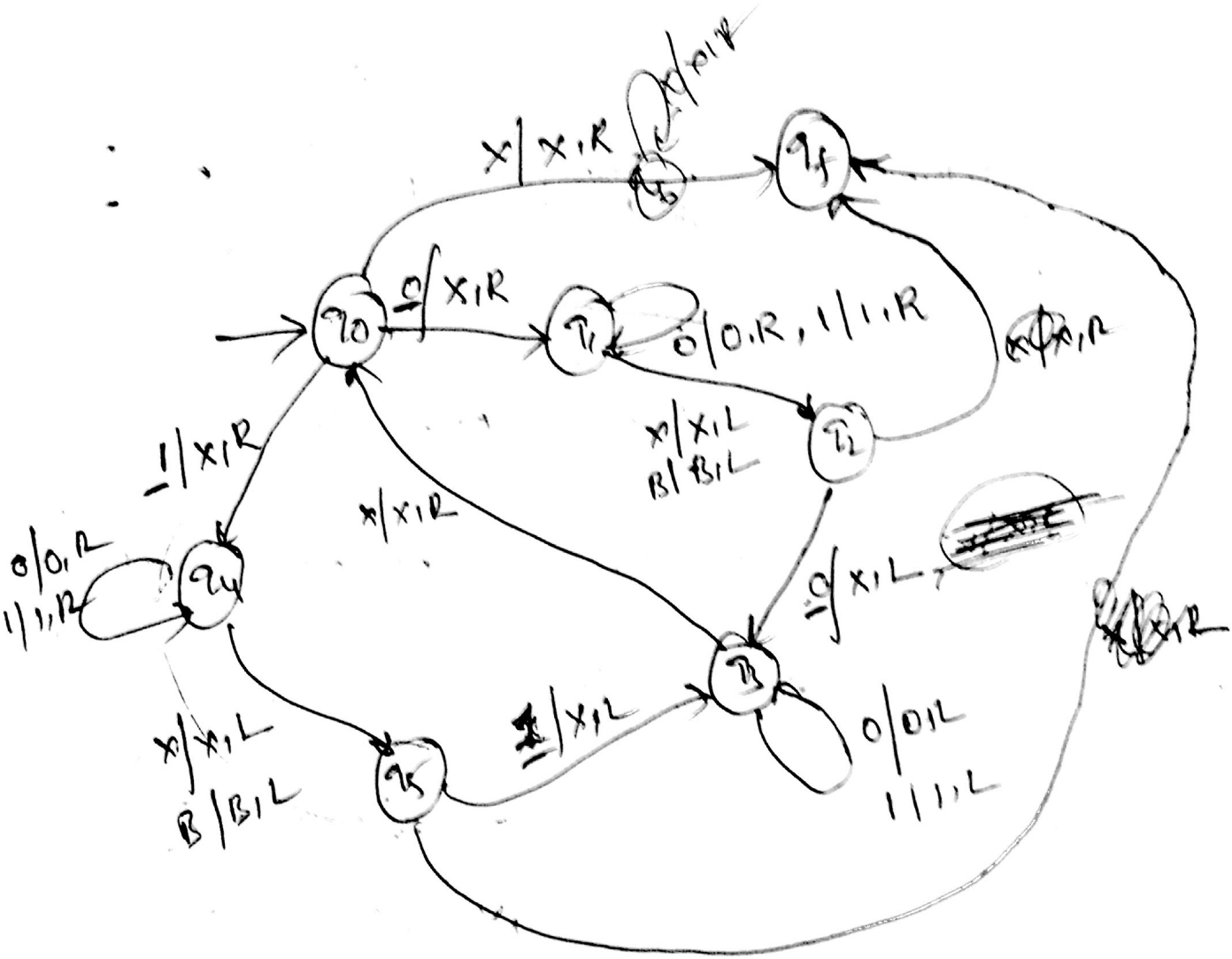




(5) Design a Turing machine to check palindromes.

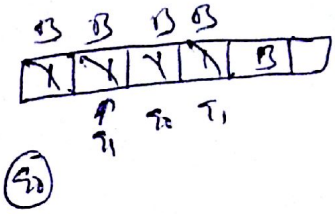
(or)

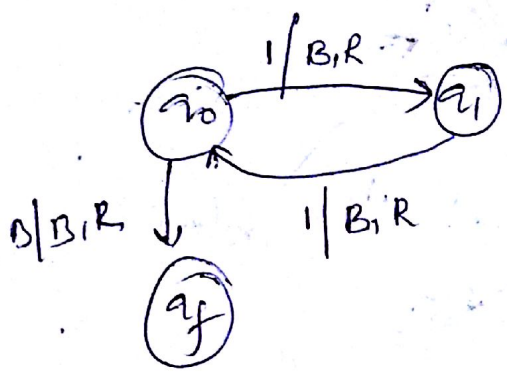
$L = \{ wcw^R \mid w \in (0+1)^* \}$



(Q). Design a Turing machine to accept set of strings consisting of even no of 1's over $\Sigma = \{1\}$.

Sol:





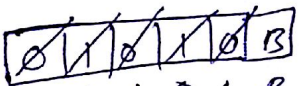
Input / States	1	B
q_0	(q_1, B, R)	(q_f, B, R)
q_1	(q_0, B, R)	—
q_f	final state.	

verification:

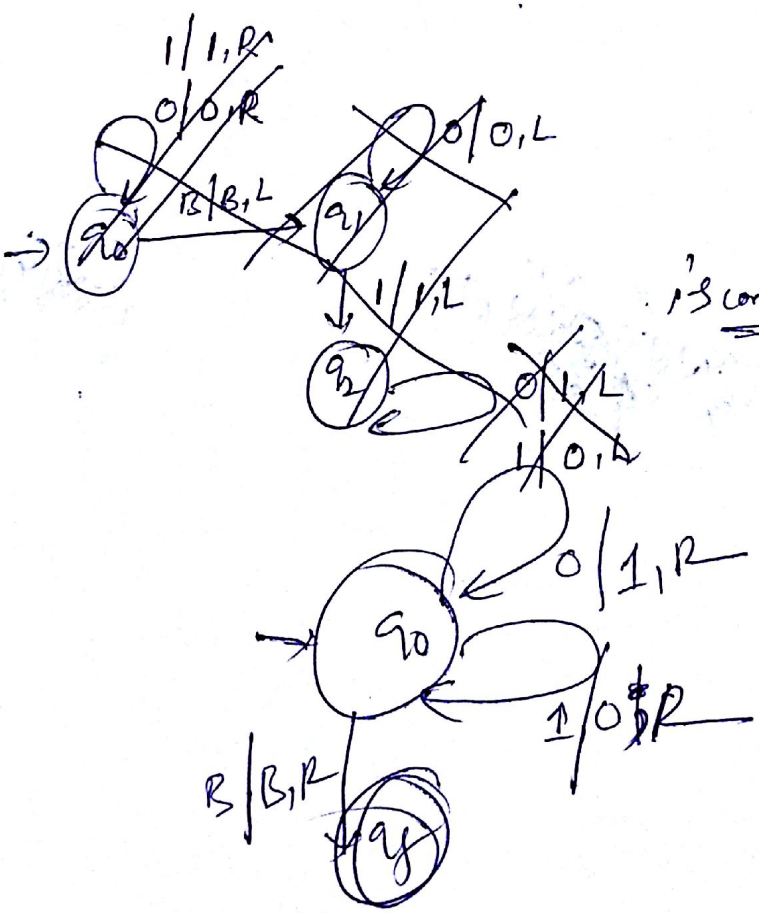
$q_0 1111B \vdash Bq_1 111B \vdash BBq_0 11B \vdash BBBq_1 1B \vdash BBBBq_0 B \vdash BBBBBq_f$

Accepts.

(Q) Design a TM to compute 1's complement of a given binary number.



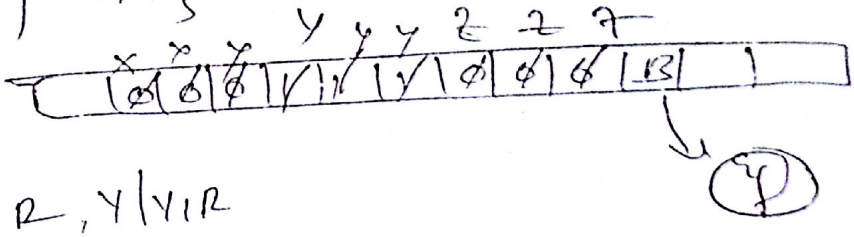
1's complement: 1 0 1 0 1 B

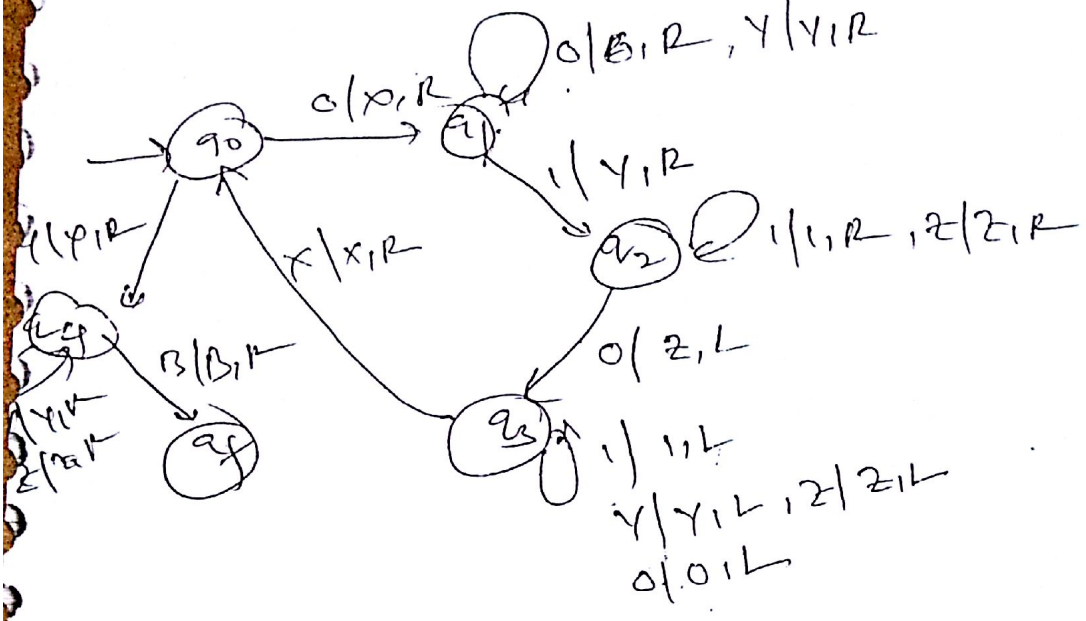


Input / States	0	1	B
q_0	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_f, B, R)
q_f	final state		Accept

(a) Design a Turing machine

$L = \{ 0^n 1^n 0^n \mid n \geq 1 \}$





(7) CFG to PDA

$S \rightarrow aAA$

$A \rightarrow aS \mid bS \mid a$

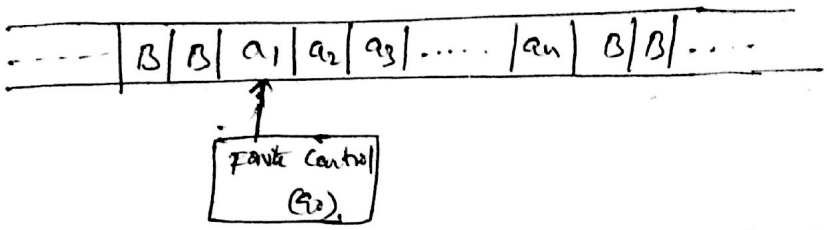
(1) $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$

PRODUCTION	TRANSITION
(2) $S \rightarrow aAA$	$\delta(q_1, a, S) = (q_1, AA)$
(3) $A \rightarrow aS$	$\delta(q_1, a, A) = (q_1, S)$
(4) $A \rightarrow bS$	$\delta(q_1, b, A) = (q_1, A)$
(5) $A \rightarrow a$	$\delta(q_1, a, A) = (q_1, \epsilon)$

(6) $\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

Types of Turing machines:

① Turing machine with two-way infinite tape:-

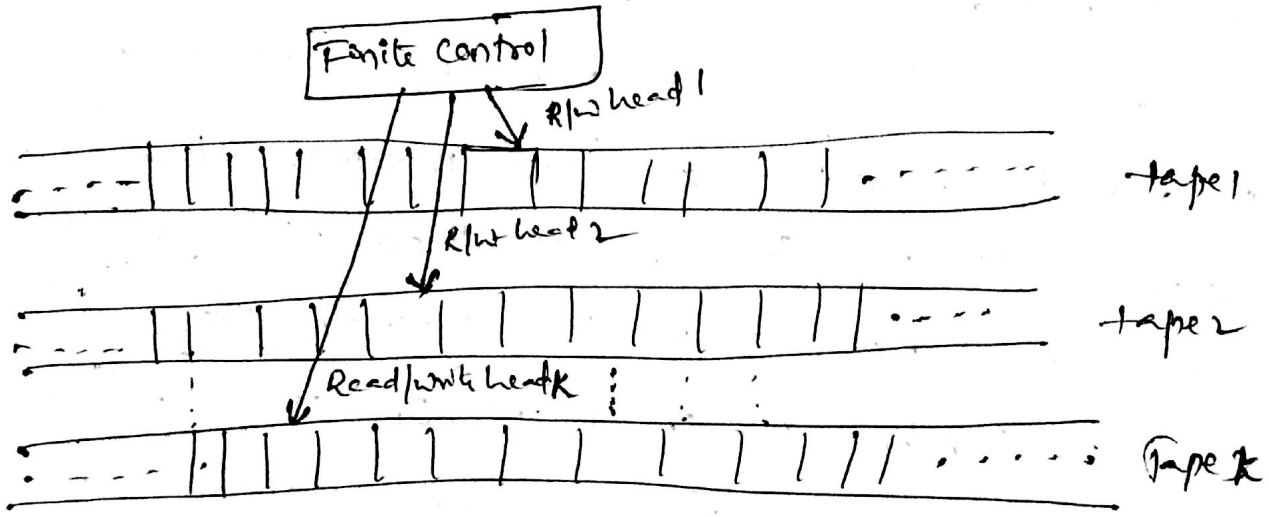


A TM with a two-way infinite tape is denoted by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ as in the original model. As its name implies, the tape is infinite to the left as well as to the right. we denote an ID of such a device as for the one-way infinite TM.

The ID of two-way infinite TM is same as that of one way infinite TM. Here the I/P string is $a_1 a_2 \ldots a_n$. Note that the tape is loaded with all blanks (B) to the left of a_1 and to the right of a_n. So if q_0 is the initial state of the TM and the ID corresponding to the initial state will be

$$q_0 a_1 a_2 \ldots a_n B$$

② ~~Multiple~~ Multitape TM:-



A multitape TM consists of a finite control with k no of tapes and k no of heads; each tape is infinite ~~to~~ in both direction

In a single move, depending on the state of the finite control and the symbols scanned by each of the tape heads, the machine can

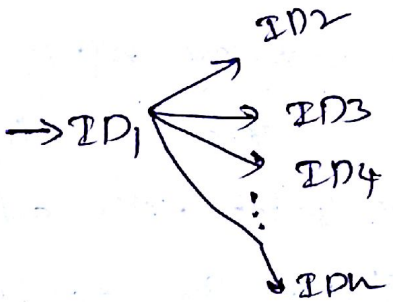
(a) change the state

(b) print a new symbol on each of the cells scanned by its tape heads.

(c) move each of its tape heads, independently, one cell to the left (or) right (or) keep it stationary.

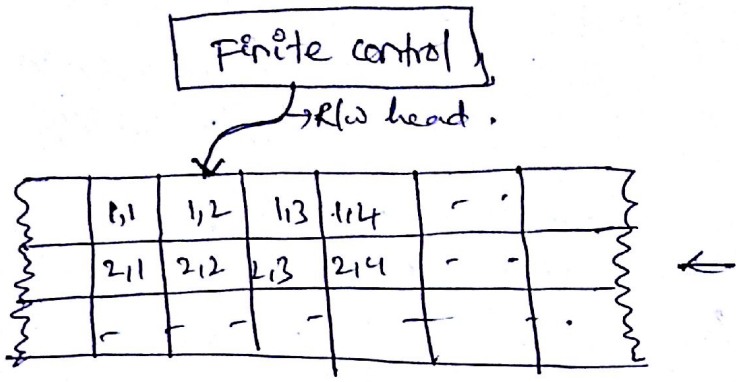
Initially, the i/p appears on the first tape, and the other tapes are blank.

(3) Nondeterministic TM :-



A nondeterministic TM is a device with a finite control and a single, one way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite no of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. Note that nondeterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and/or direction of head motion are selected from other choices. The non-deterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

(4) Multidimensional TM



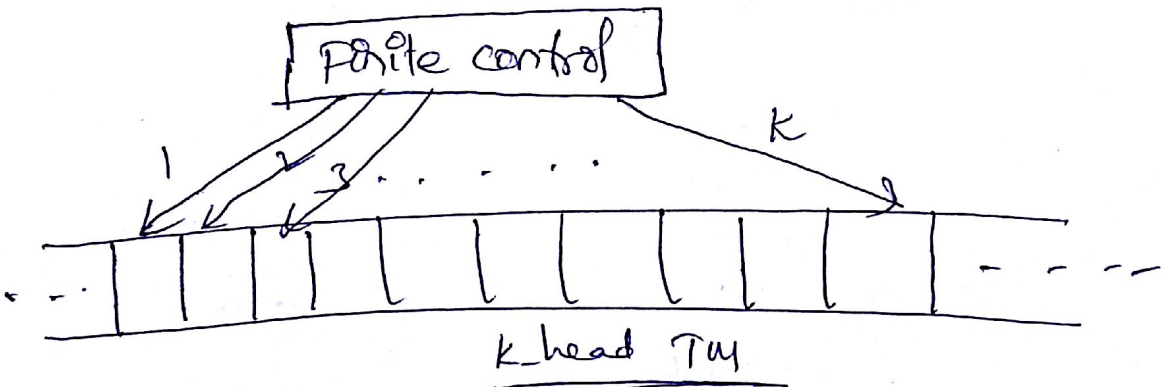
← Two dimensional address scheme.

A multidimensional TM is one in which the tape can be viewed as extending infinitely in more than one dimension. A Two dimensional TM is shown in the above figure.

A multidimensional TM mapping function is defined as

$Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$ where 'U' and 'D' specify movement of read/write head up and down respectively.

(5) Multihead TM

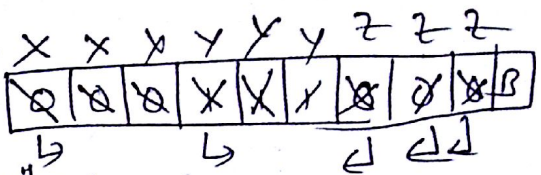


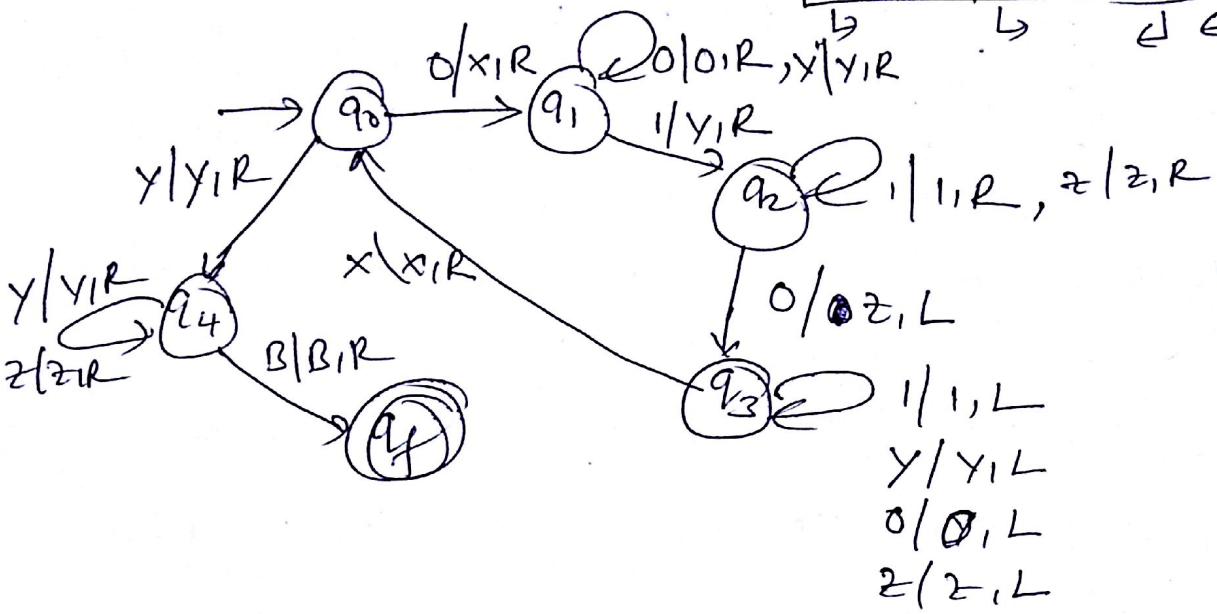
K-head TM

A K-head TM has some fixed number 'K' of heads. The heads are numbered 1 through K, and a move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may move independently left, right, or remain stationary.

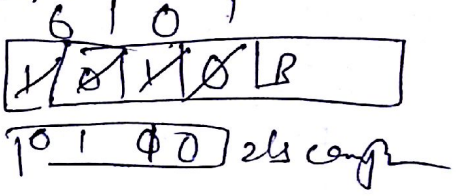
⑨ Design Turing Machine for the language

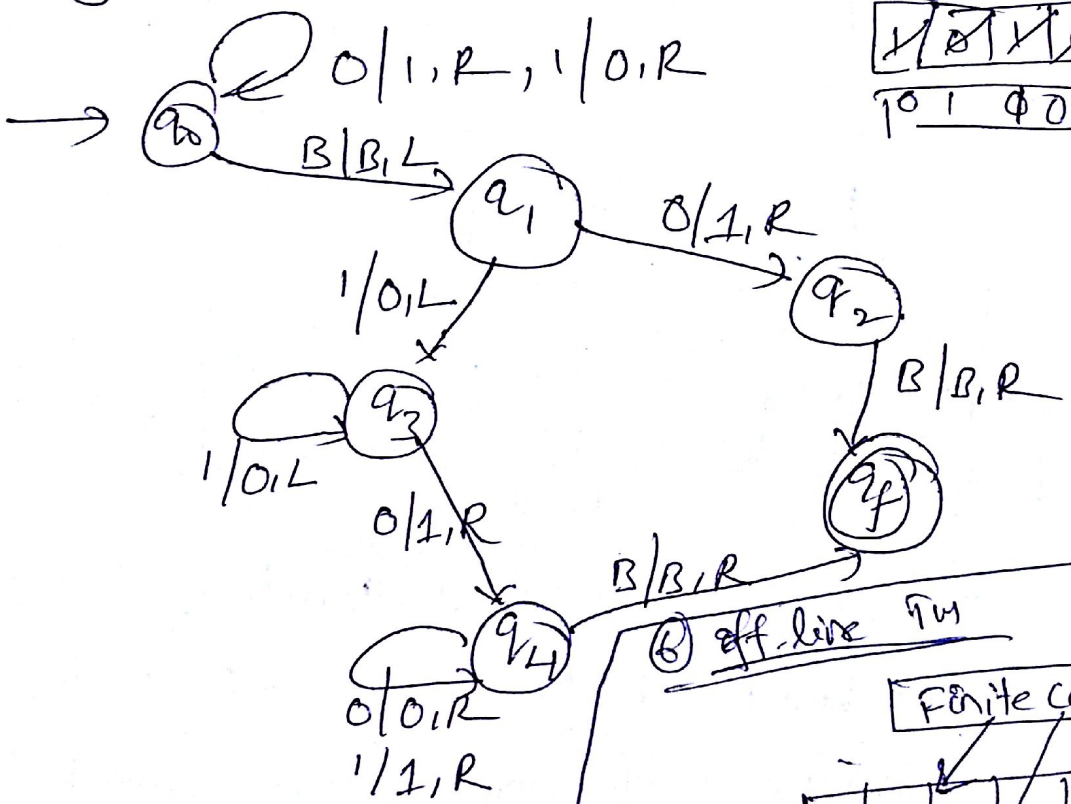
$L = \{ 0^n 1^n 0^n \mid n \geq 1 \}$



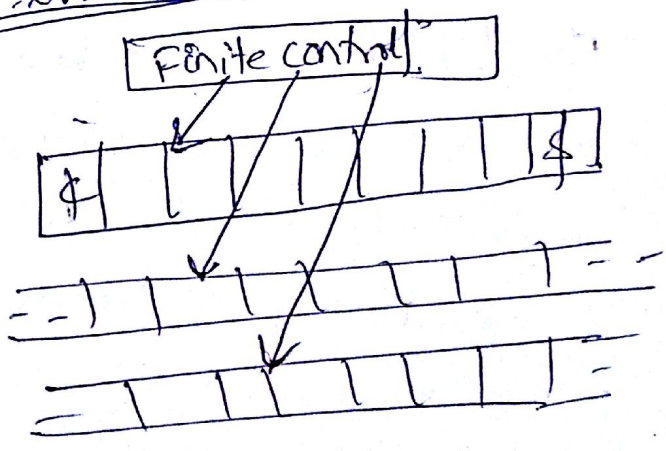


⑩ Design TM for 2's complement.





⑪ Off-line TM



An off-line TM is a multitape TM whose i/p is read only. [i/p surrounded by end markers ¢ and $). The first thing it copies its own i/p onto the extra tape and simulates M on extra tape.

Recursive and Recursively enumerable languages:

For every TM 'T', which runs on strings from the alphabet Σ, we can break the set of all finite strings over Σ into three disjoint sets.

$$\Sigma^* = \text{accept}(T) + \text{Loop}(T) + \text{reject}(T)$$

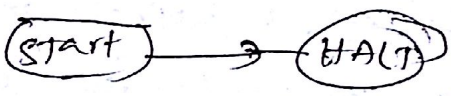
Based on this, we have two definitions for the concept of what languages are recognized by TM's. These are

① Recursive languages and ② recursively enumerable languages.

Recursive languages :-

The language 'L' over the alphabet Σ is called recursive if there is a TM 'T' that accepts every word in L and rejects every word in L'. i.e

$$\text{accept}(T) = L$$
$$\text{reject}(T) = L'$$
$$\text{Loop}(T) = \phi$$



Example: If we design a TM over $\Sigma = \{a, b\}$, if the string starts with 'a', it is accepted, otherwise rejected. So this language is recursive.

Recursively Enumerable languages:

The language L over Σ is called recursively enumerable if there is a TM 'T' that accepts every word in L, and either rejects or loops for every word in the language L',

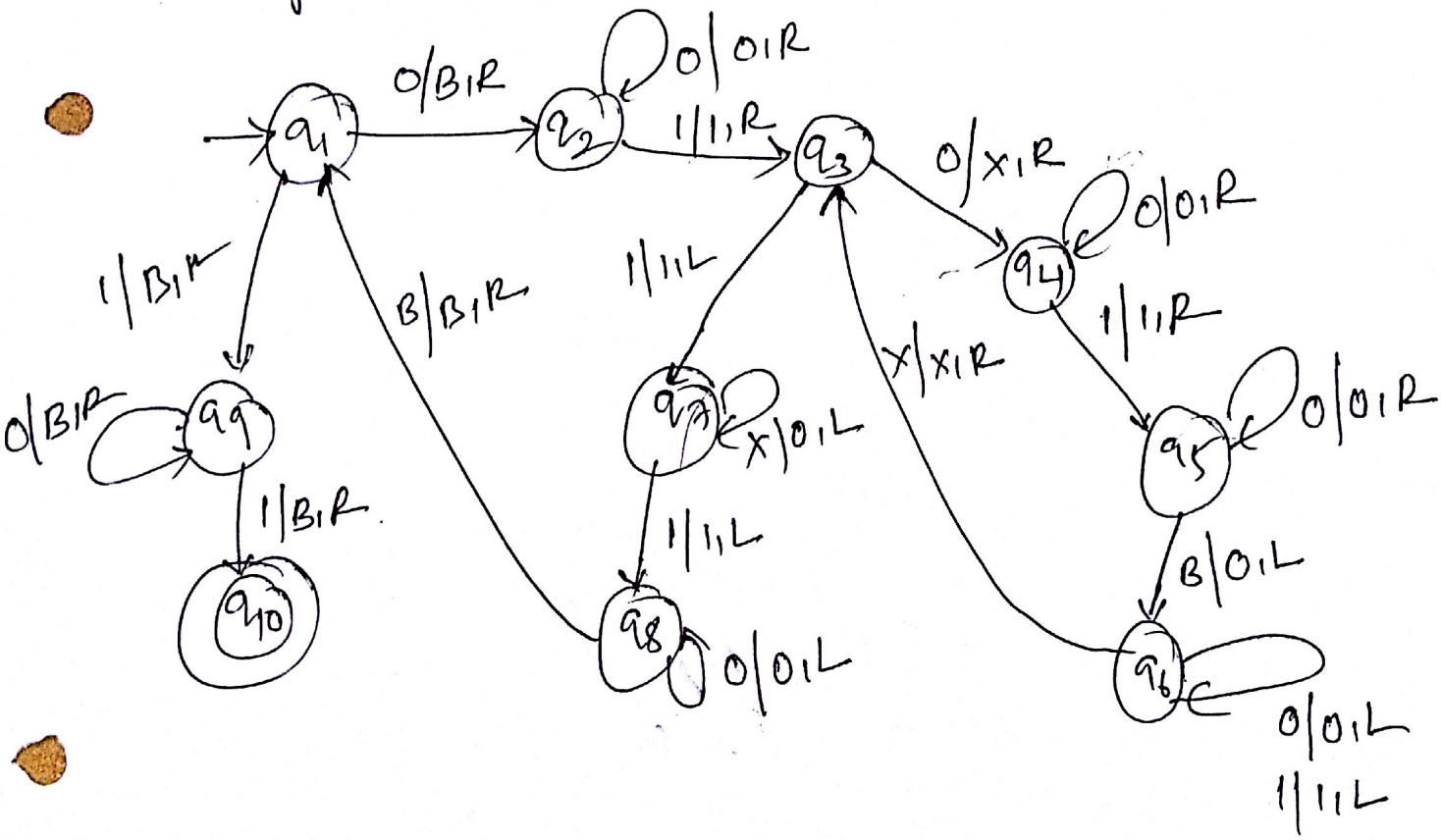
i.e $\text{accept}(T) = L$

$$\text{reject}(T) + \text{Loop}(T) = L'$$

Ex: A TM accepting a language $L = \{a^n b^n a^n\}$ and loops on rejects all words not in L. So $L = \{a^n b^n a^n\}$ is r.e.

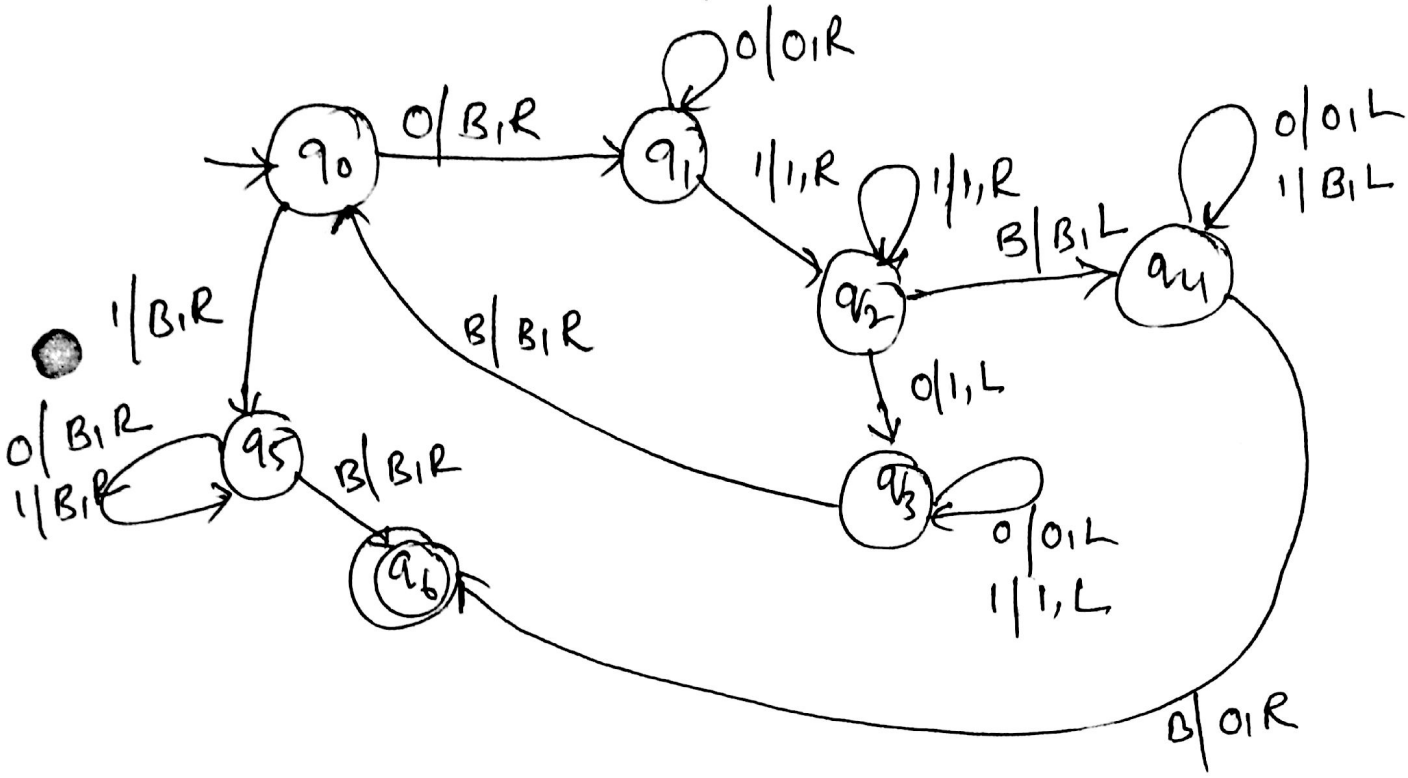
Q) Design a TM to implement 'multiplication'

Sol: M starts with $0^m 1 0^n$ on its tape and end with 0^{mn} surrounded by blanks. The general idea is to place a '1' after $0^m 1 0^n$ and then copy the block of n 0's onto the right end 'm' times, each time erasing one of the m 0's. The result is $1 0^n 1 0^{mn}$. Finally, the prefix $1 0^n 1$ is erased, leaving 0^{mn}.



Q): Design a TM for subtraction $m-n$ is defined as $m-n$ for $m \geq n$ and zero for $m < n$.

Sol: The TM $M = (\{q_0, q_1, \ldots q_6\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \phi)$ defined below, started with $0^m 1 0^n$ on its tape, halts with 0^{m-n} on its tape.



Universal Turing Machine :

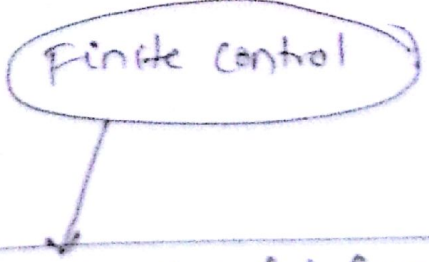
So far we have seen that a turing machine is designed to execute a specific computation. If we have a turing machine for computing one function then computing different function or doing some other calculation requires a different machine. Hence it represents a special purpose computer. On the other hand digital computers are general purpose machines which do different jobs by executing instructions stored in memory. Hence to obtain the functionality of general purpose computer a reprogrammable turing machine called Universal Turing Machine (UTM) is designed.

A UTM is given as input a string composed of two parts. Encoded program of any turing Machine 'M' followed by a marker and second part consists of a string called data. The UTM operates on data as if it were M. If M would accept the input, so the UTM.

Depending on the encoding scheme used and algorithm used for simulating the TM different UTMs exist.

Encoding scheme : It must be simple and must provide all the information regarding M. The information includes no of states used in M, no of tape symbols, input alphabet, starting state, final state, end marker, blank and transition function.

Model



Input M # w in encoded form — Tape 1

Tape of 'M' 10001000 1 . . . — Tape 2

State of 'M' 000 . . . — Tape 3

Universal Turing Machine

The tape of UTM is partitioned into three track: one track holds input M#, another tape holds data and third tape holds the state in which the UTM is in.

The UTM simulates TM 'M' on data 'w' as follows:

1. First it checks M# w to check whether 'M' is valid encoding data 'w' on turing machine 'M' and 'w' is a valid encoding of string over M's I/p alphabet. If M is not correct encoding UTM halts.
2. If it is correct, UTM copies encoded data 'w' from tape 1 to tape 2.
3. The starting state is copied onto the tape 3.
4. UTM then first looks at the content of tape 3 to determine the configuration of M. It then consults tape 1 to check what 'M' would do in this configuration
5. If 'M' has no transition that matches with simulated state and tape symbol then no transitions found and it halts.
6. If 'M' enters final state then it accepts 'w'.

Turing Reducibility:

Turing reducibility is a more general technique to reduce a language to another language i.e., L_1 to L_2 it consists of simply of showing that L_1 is recursive in L_2.

If we have an algorithm to convert instances of a problem P_1 to instances of a problem P_2 that have the same answer, then it is said that P_1 cannot be recursively enumerable. Care to be taken to reduce a known hard problem to prove to be atleast as hard, never the opposite.

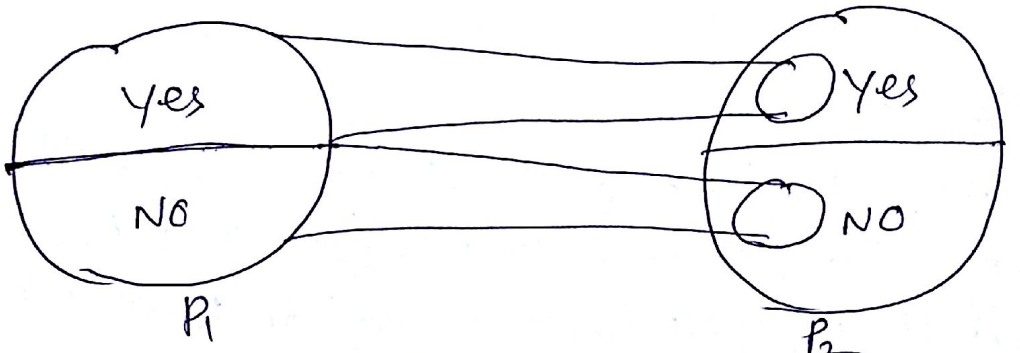


Fig: Reduction turn positive instances to positive and negative instances to Negative.

A reduction must turn any instance of P_1 that has a 'yes' answer into an instance of P_2 with a 'yes' answer and every instance of P_1 with a 'no' answer must be turned into an instance of P_2 with a 'no' answer.

Note that it is not essential that every instance of P_2 be the target of one or more instances of P_1 and it is quite common that only a small fraction of P_2 is a target of the reduction.

Formally a reduction from P_1 to P_2 is a TM that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape. If there is a reduction from P_1 to P_2 then ① If P_1 is undecidable then so is P_2,

② If P_1 is non-Recursively Enumerable then so is P_2.

Properties of Recursive and Recursively Enumerable Languages:

① If the language 'L' and its complement '$\bar{L}$' are both recursively enumerable, then L (& $\bar{L}$) is recursive.

② The union of two recursive languages is recursive.

③ The union of two recursively enumerable languages is recursively enumerable

④ The complement of recursive language is recursive.

→ A problem whose language is recursive is said to decidable. otherwise, the problem is undecidable. That is, a problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is 'yes' or 'no'.

MPCP (Modified post's correspondance problem)

Given lists A and B, of K strings each from Σ^+, say

$A = w_1 w_2 \dots w_k$ and $B = x_1, x_2 \dots x_k$ does there exist a sequence of integers $i_1, i_2 \dots i_r$ such that

$$w_1 w_{i_1} w_{i_2} \dots w_{ir} = x_1 x_{i_1} x_{i_2} \dots x_{ir} ?$$

The difference b/w PCP and MPCP is a solution is required to start with the first string on each list.

⇒ If PCP is decidable, then MPCP would be decidable i.e MPCP reduces to PCP

Decidability and undecidability:

A decision problem is a restricted type of an algorithmic problem where for each input there are only two possible outputs (yes or no).

A decision problem is a function that associates with each input instance of the problem a truth value true or false (yes or no).

A decision algorithm is an algorithm that computes the correct truth value for each input instance of a decision problem. The algorithm has to terminate on all inputs!

A decision problem is decidable if there exists a decision algorithm for it. otherwise it is undecidable.

Examply:

① DFA membership

Instance: A DFA $M = (Q, \Sigma, \delta, q_0, F)$ and a string $w \in \Sigma^*$

Question: Is $w \in L(M)$?

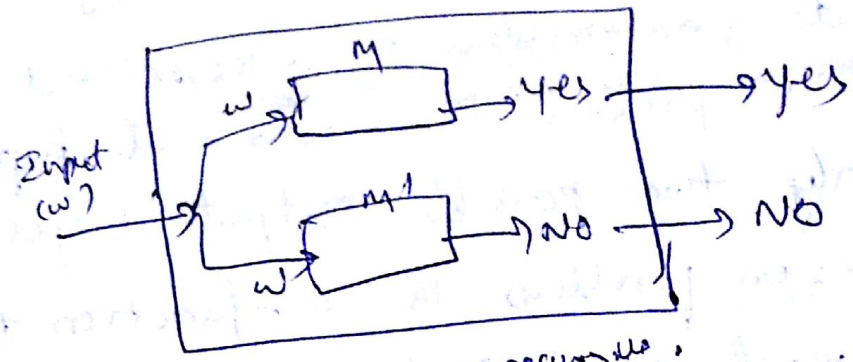
proposition: It is decidable.

② DFA Equivalence

Instance: Two DFAs M_1, M_2

Question: Is $L(M_1) = L(M_2)$?

Theorem: DFA Equivalence is decidable.



→ A language is decidable (or recursive) if some TM decides it. All computations of a decider TM must halt. Decidable languages are often called also recursive languages.

→ A language is turing-recognizable (recursively enumerable) if it is recognized by a TM. i.e, all words in the languages are accepted by the TM. all words not belonging to the language, the computation of the TM either rejects or goes on forever (loop).

→ $L_{TM_{accept}}$ is undecidable.

→ halting problem of TM ✓

→ * PCP

→ equivalence btw two TMs.