SEM - V^{th}

SOLUTION OF FLAT (2013)

(SECTION - A)

Ques 1 Fill in the blanks :→

(1) FSM can recognize only REGULAR language.

(2) For input null the output produced by a Mealy machine is NULL.

(3) The output of mealy machine depends on PRESENT STATE & PRESENT INPUT.

(4) The number of states of the FSM required to simulate the behaviour of a company with a memory capable of sorting 'm' words each of length 'm' bits is 2^{mm}.

(5) In Moore machine the output is associated with PRESENT STATE.

(6) Pumping lemma is generally used for proving a grammar is NOT REGULAR.

(7) A formal notation for context free grammar $G = (V, T, P, S)$.

(8) Every regular grammar is context free. TRUE (T/F)

(9) A string of symbols obtained by reading the leaves of the tree from left to right, omitting any ε's encountered is called Yielding.

(10) If in a derivation tree every leaf has a label from $V \cup T \cup \{\epsilon\}$ then it is called PARTIAL DERIVATION TREE.

SEM - V^{th}

SOLUTION OF FLAT (2013)

(SECTION - A)

Ques 1 Fill in the blanks :->

(1) FSM can recognize only REGULAR language.

(2) For input null the output produced by a Mealy machine is NULL.

(3) The output of mealy machine depends on PRESENT STATE & PRESENT INPUT.

(4) The number of states of the FSM required to simulate the behaviour of a company with a memory capable of sorting 'm' words each of length 'n' bits is 2^{mn}.

(5) In Moore machine the output is associated with PRESENT STATE.

(6) Pumping lemma is generally used for proving a grammar is NOT REGULAR.

(7) A formal notation for context free grammar $G = (V, T, P, S)$.

(8) Every regular grammar is context free. TRUE (T/F)

(9) A string of symbols obtained by reading the leaves of the tree from left to right, omitting any ε's encountered is called Yielding.

(10) If in a derivation tree every leaf has a label from $V \cup T \cup \{\epsilon\}$ then it is called PARTIAL DERIVATION TREE.

UNIT- 1 SECTION - B

Ques 2 (a) Construct an equivalent DFA for NFA $M = (\{p, q, r, s\}, \{a, b\}, \delta, p, \{q, s\})$, where δ is given below:→

(NFA)

PRESENT STATE	a	b
p	{q, s}	{q}
q	{r}	{q, r}
r	{s}	{q, r}
s	-	{p}

Sol^n Let equivalent DFA is M_1, & $M_1 = (Q, \Sigma, \delta, \{p\}, F)$

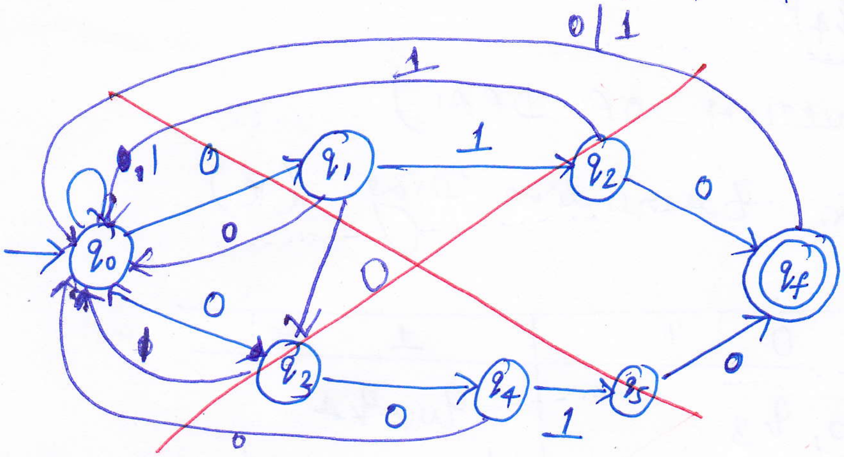
S/Σ	a	b
→[p]	[q, s]	[q]
[q]	[r]	[q, r]
[q, s]	[r]	[p, q, r]
[r]	[s]	[q, r]
[q, r]	[r, s]	[q, r]
[p, q, r]	[q, r, s]	[q, r]
[s]	ϕ	[p]
[r, s]	[s]	[p, q, r]
[q, r, s]	[r, s]	[p, q, r]

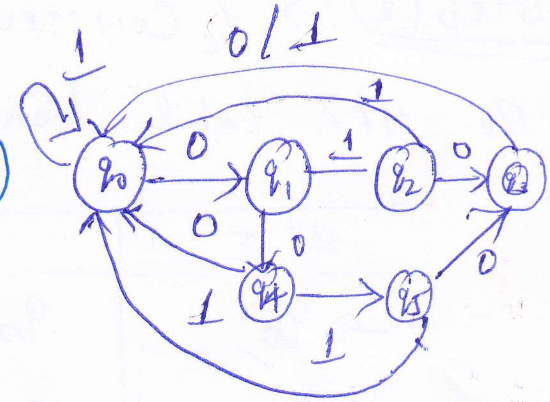
$Q = \{\{p\}, \{q\}, \{r\}, \{s\}, \{q, r\}, \{r, s\}, \{q, s\}, \{p, q, r\}, \{q, r, s\}\}$

$\Sigma = \{a, b\}$, $\{p\}$ → is the starting state

Ques (b) Construct a finite automata accepting all strings over $\{0,1\}$ ending in 010 or 0010.

Sol^n Let FA $M = (Q, \Sigma, \delta, q_0, F)$





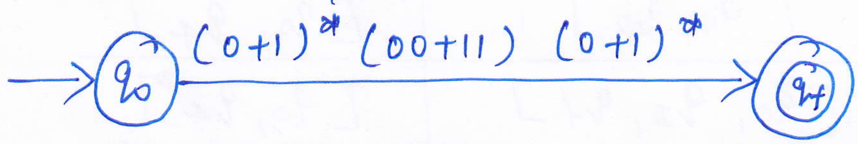
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$

$\Sigma = \{0,1\}$, $q_0 \rightarrow$ Initial state, $q_3 \rightarrow$ is the final state.

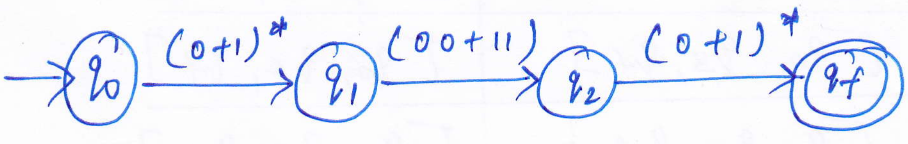
Ques (c) Construct an FA equivalent to the regular expression $(0+1)^*(00+11)(0+1)^*$.

Sol^n → **STEP-1** :→ (Construction of transition graph). First of all we construct the transition graph with Λ-moves using the construction of transition graph. Then we eliminate Λ-moves.

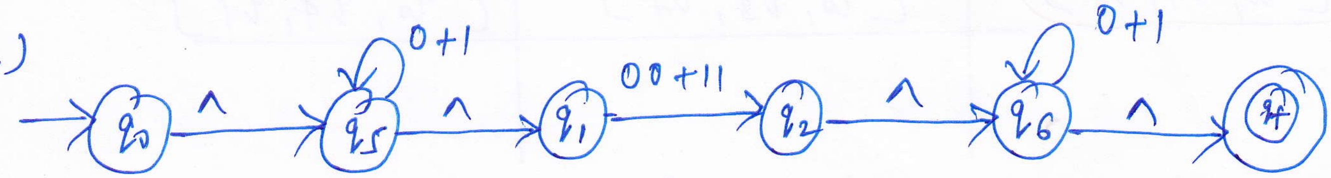
(a)



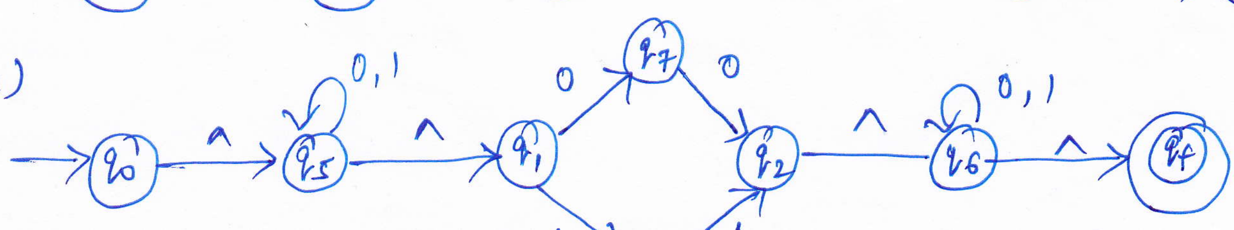
(b)



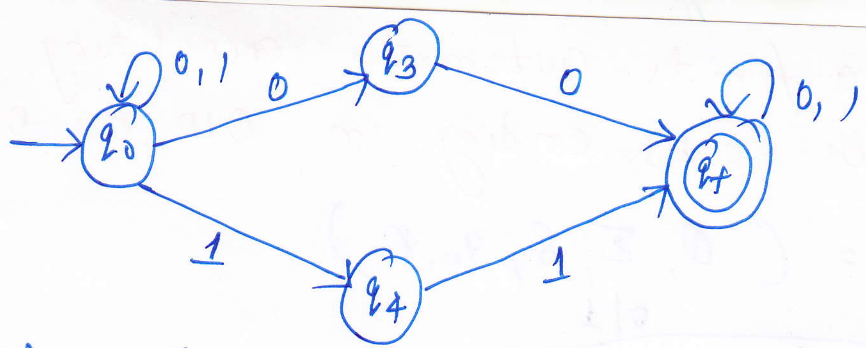
(c)



(d)



(e)



Step (2) :-> (CONSTRUCTION OF DFA)

So, NFA table from transition Diag. (e)

S / Σ	0	1
→ q_0	q_0, q_3	q_0, q_4
q_3	q_f	–
q_4	–	q_f
q_f	q_f	q_f

=> Now, Transition table of equivalent DFA

Q	Q_0	Q_1
→ [q_0]	[q_0, q_3]	[q_0, q_4]
[q_0, q_3]	[q_0, q_3, q_f]	[q_0, q_4]
[q_0, q_4]	[q_0, q_3]	[q_0, q_4, q_f]
[q_0, q_3, q_f]	[q_0, q_3, q_f]	[q_0, q_4, q_f]
[q_0, q_4, q_f]	[q_0, q_3, q_f]	[q_0, q_4, q_f]

UNIT - II

Ques 3 (a) Define CFG. Generate CFG for the language $L = \{a^n b^m, n \neq m\}$. [1+3]

Sol^n (a) Definition of CFG :⇒ A Grammar $G = (V_n, \Sigma, P, S)$ is said to be Context-free grammar (CFG) if the production rules of G are of the form :-

$$A \to \alpha, \text{ where } \alpha \in (V_n \cup \Sigma)^*$$

The right hand side of a context-free grammar is not restricted and it may be null or a combination of variable and terminals, $(V_n + \Sigma)^*$. As we know that a context-free grammar has no context, neither left nor right. i.e it is called as context free.

⇒ The CFG for the language

$$L = \{a^n b^m, n \neq m\}$$

is

$$\boxed{\begin{array}{l} S \to aAb \mid a \mid b \\ A \to aAb \mid a \mid b \end{array}}$$

Ques (b) Define GNF? Consider the production rule of CFG: $S \rightarrow S+S / S*S / a / b$ and find an equivalent grammar in GNF. [1+3]

Sol^n A Context-free grammar is $G = (V_n, \Sigma, P, S)$ is said to be in Griebach normal form (GNF) if its all production rules are of type $A \rightarrow a\alpha$, where $\alpha \in V_n^*$ (string of variables including null string) and $a \in \Sigma$. A grammar in GNF is the natural generalization of a regular grammar.

=> Let G_1 is the equivalent grammar in GNF. Renaming the variable, we have following production rules :-

P_1 : $S_1 \rightarrow S_1 + S_1$ (Not in GNF)

P_2 : $S_1 \rightarrow S_1 * S_1$ (Not in GNF)

P_3 : $S_1 \rightarrow a$ (IN GNF)

P_4 : $S_1 \rightarrow b$ (IN GNF)

P_1 & P_2 are left recursive production rule, so removing the left recursion, we get following production rules :-

$S_1 \rightarrow aS_2 \mid bS_2 \mid a \mid b$, where

$S_2 \rightarrow +S_1S_2 \mid *S_1S_2 \mid +S_1 \mid *S_1$

All the prod^{n}s are in GNF, which are :-

$S_1 \rightarrow aS_2 / bS_2 / a / b$

$S_2 \rightarrow +S_1S_2 / *S_1S_2 / +S_1 / *S_1$

OR

$P_1 : S_1 \rightarrow S_1 S_2 S_1$ [Not in GNF]

$P_2 : S_1 \rightarrow S_1 S_2 S_1$ [Not in GNF]

$P_3 : S_2 \rightarrow + \mid *$ [In GNF]

$P_4 : S_1 \rightarrow a S_2 S_1$ [In GNF]

$P_5 : S_1 \rightarrow b S_2 S_1$ [In GNF]

So, all the prodns. are :-

$S_1 \rightarrow a S_2 S_1 \mid b S_2 S_1 \mid a \mid b \mid + \mid *$

Ques (c) Describe the decision algorithms for Context free language with one example.

Soln In a given CFG $G = (V, T, S, P)$ there exists an algorithm for deciding whether or not $L(G)$ is empty, finite or infinite.

⇒ Assume the given language does not contain ϵ. Find the reduced grammar (i.e, eliminate useless prodns., ϵ-prodns. & unit productions) of the given grammar.

① If the reduced grammar vanish then the given language is empty.

② Draw a graph with the productions of the reduced grammar. If the graph contains cycle the given grammar generates infinite language, otherwise it generates finite language.

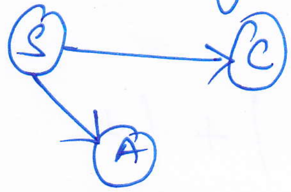
Ex:- $S \rightarrow AB \mid CA$
$B \rightarrow BC \mid AB$
$A \rightarrow a$
$C \rightarrow aB \mid b$

On eliminating useless symbol we get following productions :-
$S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$

As there are no ϵ-productions or no unit productions this is the reduced grammar.

(1) As grammar doesn't vanish the given grammar doesn't generate empty language.

(2) Now draw the graph as shown below



As this graph does not contain cycle, the given grammar generates finite language.

UNIT - III

Ques (a) Obtain PDA to accept strings of balanced parenthesis & verify by a suitable example.

Soln :→ The PDA for the balanced paranthesis is :→ $[Q, \Sigma, \Gamma, \delta, q_0, z_0, F]$, $Q = q_0$, $\Sigma = \{(,)\}$

$\Gamma = \{z_0, x\}$,

$z_0 \rightarrow$ stack symbol

$F = \{q_f\}$

$$\delta(q_0, (, z_0) = (q_0, x z_0)$$

$$\delta(q_0, (, x) = (q_0, xx)$$

$$\delta(q_0,), x) = (q_0, \epsilon)$$

$$\delta(q_0, \not b, z_0) = (q_f, \epsilon)$$

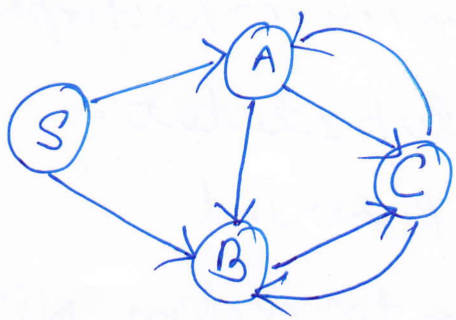
for example, $(q_0, (()), z_0) \vdash (q_0, ()), x z_0)$

$\vdash (q_0,)), x x z_0) \vdash (q_0, \not b, z_0) \vdash$

(q_f, ϵ) ...

Ques 4 (b) Verify the grammar is finite, infinite or empty $S \to AB$, $A \to BC/a$, $B \to CC/b$, $C \to AB/a$

Soln (b) :-> As there is no null production, useless production, unit production, so the grammar will be as it is and the graph is shown below:->



As this graph contains cycle the given grammar generate infinite language.

Ques 4 (c) Define DPDA? Describe the closure properties of CFL's.

Soln (c) Deterministic PDA (DPDA) is just like ~~PDA~~ DFA, which has at most one choice to move for certain input. A PDA $M = (Q, \Sigma, T, \delta, s, Z_0, F)$ is deterministic if it satisfies both the conditions given following :->

① for any $q \in Q$, $a \in (\Sigma \cup \{\epsilon\})$, & $Z \in T$, $\delta(q, a, Z)$ has at most one choice of move.

② For any $q \in Q$ and $Z \in T$, if $\delta(q, \epsilon, Z)$ is defined, i.e. $\delta(q, \epsilon, Z) \neq \phi$, then $\delta(q, a, Z) = \phi$ for all $a \in \Sigma$.

CLOSURE PROPERTIES OF CFL :-

Closure properties of CFL are

1. CFL are closed under Union operation.
2. CFL are closed under Concatenation operation
3. CFL are closed under Kleene closure ".
4. CFL are closed under Homomorphism ".
5. " " " " Inverse homomorphism "
6. " " " " Substitution "
7. " " " " Reversal "
8. " " " " Intersection with regular sets.

UNIT - IV

Ques 5

(a) Define DTM? Describe the different types of TM?

⇒ A Turing machine can be described by 7-Tupples $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$ where

$Q \rightarrow$ is the finite set of states, not including the halt state (h).

$\Sigma \rightarrow$ is the non empty set of input symbols, $(b \notin \Sigma)$

$\Gamma \rightarrow$ is a finite nonempty set of tape symbols.

$b \in \Gamma$ is the blank.

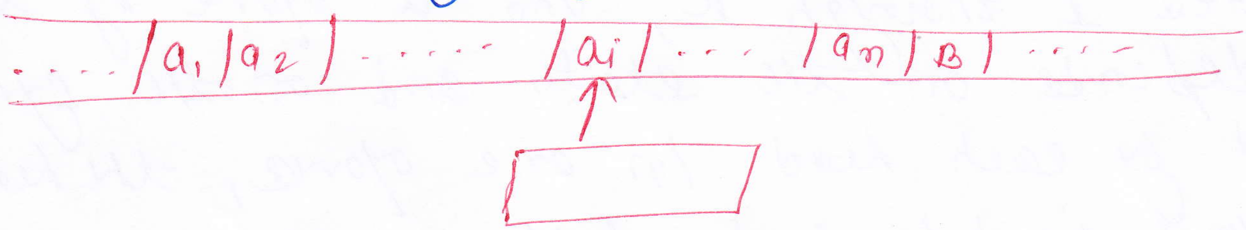
δ is the transition function.

q_0 is the initial state

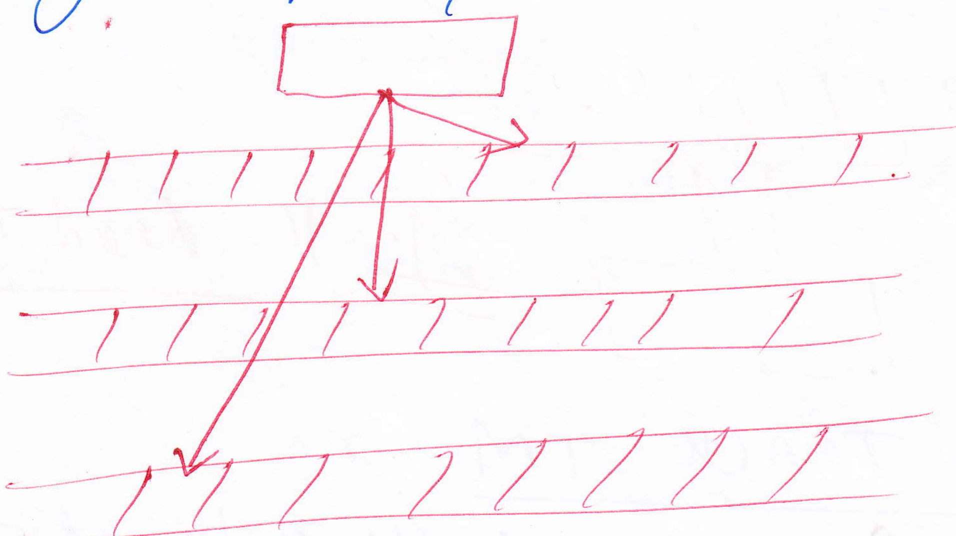
⇒ If Turing machine has at most one move in transition for all input symbols then it is called deterministic Turing machine.

TYPES OF TM :->

① **Two-way infinite TM :->** L is recognized by a turing machine with a two-way infinite tape if and only if it is recognized by a TM with a one-way infinite tape.



② **MULTITAPE-TM :->** If a language L is accepted by a multi tape turing machine, it is accepted by a single-tape machine.



③ **Non-Deterministic TM :->** In this machine, the state and input symbol a NTM has at least one choice to move (finite number of choices for the next state, a new state, a
... ... consisting of a new state, a

④ **Multi Dimensional TM :->** In K- dimensional TM the tape consists of K- dimensional array of cells infinite in all 2K Direction, for some fixed K. If L is accepted by a K- dimensional turing machine M1, then L is accepted by some single tape turing machine M.

⑤ **MULTI 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. If L is accepted by some K-head TM M_1, it is accepted by a one head TM M_2

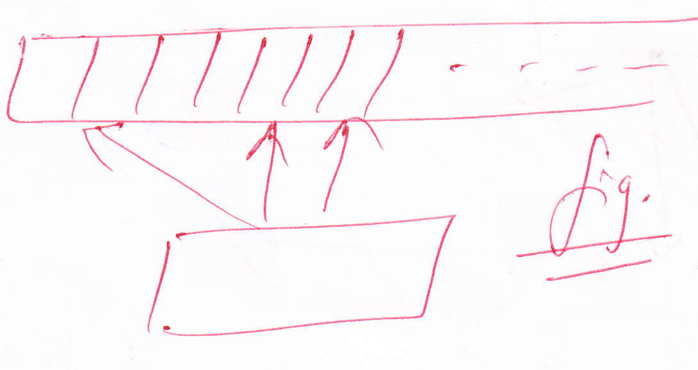
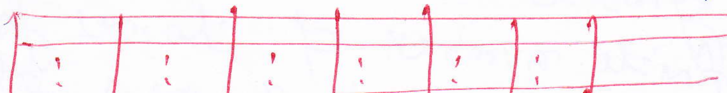


fig. K- head TM

⑥ **MULTI - TRACK TM :->**

We can imagine that the tape of the TM is divided into K tracks, for any finite K.

K-tracks.



Ques 5 (b) Design TM that replaces all occurrences of '111' by '101' from sequence of 0's & 1's.

Soln

STEP-1

δ / Σ	0	1	b
$\rightarrow q_0$	$0 R q_0$	$1 R q_1$	
q_1	—	$1 R q_2$	
q_2	$0 R q_2$	$1 R q_3$	
q_3	$0 L q_4$	$1 L q_4$	
q_4	—	$1 L q_5$	
q_5	$0 R q_0$	$0 L q_6$	
q_6	$0 L q_6$	$1 L q_6$	$b R q_7$
q_7	—	—	—

STEP-2

Ex :-> $\boxed{q_0\ 0111010}$ $\rightarrow$ $\boxed{b\ q_7\ 0101010}$

I/p — O/p

STEP-3

Tuples $(Q, \Sigma, \Gamma, \delta, q_0, z_0, b, F)$

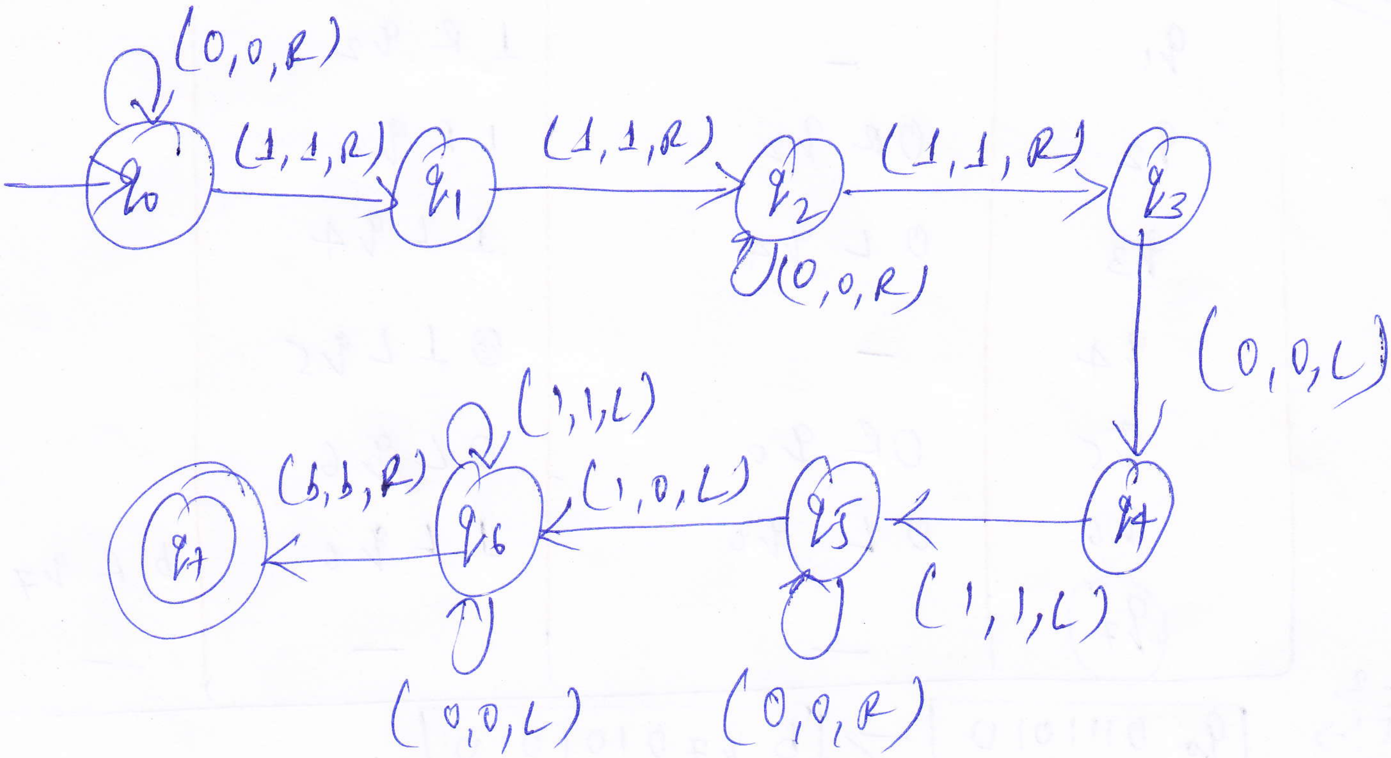
$(\{q_0, q_1 \cdots q_7\}, \{0,1\}, \{0,1\}, \delta, q_0, b, \{q_7\})$

Step 4

EXPLANATION :->

As soon as we come across 3 consecutive 1's we will take a left shift and make the middle 1 as 0, q_0 is the initial state and q_7 is the final state here.

Step 5 TRANSITION DIAGRAM



Ans 5(c) Write a short note on Church's Hypothesis.

Soln "The principle that Turing machines are formal version of algorithms and no computational procedure will be considered an algorithm unless it can be implemented as a TM is known as Church's Thesis.

Some arguments for accepting the Church's Thesis are following :->

(1) No one has been able to suggest a counter example to disprove till now.

(2) Some Turing machines can also perform any thing that can be performed by a digital computer.

(3) There are several alternative model of computation like random access machine (RAM), Post machine (PM) etc, but no one is more powerful than Turing machine.

=> Church actually said that any machine that can do certain list of operations will be able to perform all conceivable algorithms. He tied together what logicians had called recursive functions & computab-

UNIT- V

Ques. 6(a) Write a short note on Universal TM.

Soln The Turing machine is an "un programmable" piece of hardware, specialized at solving one particular problem, with instructions that are "hard-wired at the factory."

UNIVERSAL TM 'U' takes two arguments, a description of a machine Tm, 'Tm', and a description of an input string w, "w", We want U to have the following property :→ U halts on input "Tm" "w" if and only if Tm halts on input w.

$$U(\text{"m" "w"}) = \text{"m(w)"}$$

It is the functional notation of universal turing machine. So, a Turing machine which can simulate the behavior of any turing machine, such general turing machine is called as a Universal turing machine.

The operations of UTM involves the initial contents of tape and the initial description of turing machine (program).

(b) Write a short note on PCP.

Write a short note on Post correspondence problem.

Soln PCP stands for Post correspondence problem.

The PCP of 'n' strings on some alphabet 'Σ'

say, $A = w_1, w_2, w_3 \cdots w_n$ &

$B = v_1, v_2, v_3 \cdots v_n$

We say that there exist a PCP soln (PC-soln)

$W_i, W_j \cdots W_k = V_i, V_j, \cdots V_k$

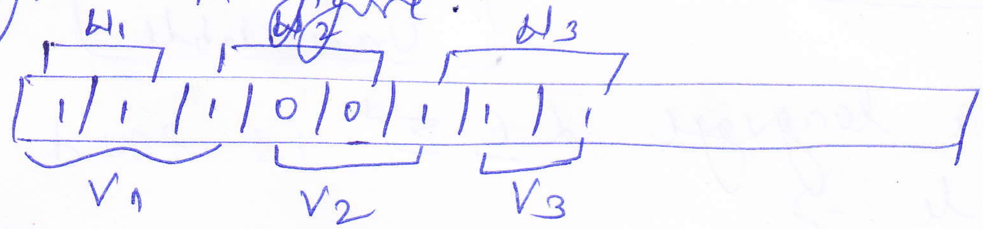
The post correspondence problem is to derive an algorithm that tell us for any (A, B) whether or not there exist a PC-soln.

Ex:-) Let $\Sigma = \{0, 1\}$ & take A, B as

$W_1 = 11$, $W_2 = 100$, $W_3 = 111$

$V_1 = 111$, $V_2 = 001$, $V_3 = 11$

for this case, there exist a PC soln which are given in figure.



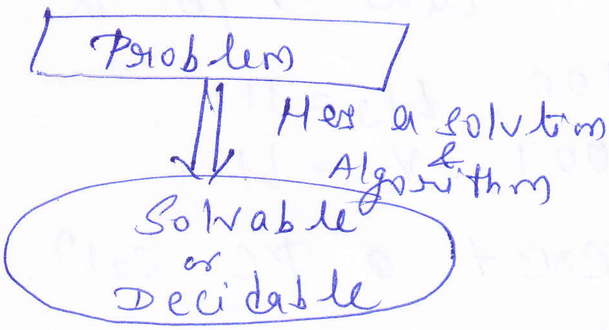
Qus. 6 (c) Write a short note on Decidability of problem.

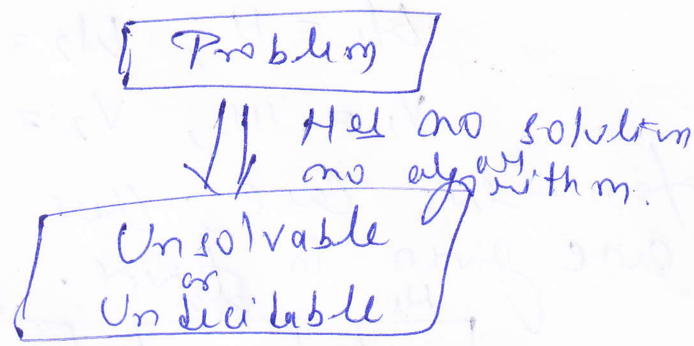
Soln:-> Every problem either have a solution or not. Such problems are restricted to a certain set {yes, no} that is if they will have a solution that is "yes" otherwise they don't have any solution that is "no."

So thus if a problem is having a solution then that is solvable otherwise the problem is called unsolvable.

If a problem has a solution either "yes" or "no" but not both on the basis of some algorithm then that problem is called a "Decidable Problem" and this decision leads to the decidability of the problem.

In some cases if a problem has a solution "yes" and in some case "no" then such problems are called as undecidable problem which has both the solution, sometimes "yes" and sometimes "no".





formally, a language $L \subseteq \Sigma^*$ is said to be decidable :->

1. If the L is recursive
2. It has an algorithm (or answer)

<u>Example of Decidable problems</u> :->

1. Union of two regular language is a regular language $\Rightarrow$ [Yes]
2. Intersection of CFL and Regular Language is a CFL $\Rightarrow$ [Yes]

While,

PCP is an undecidable problem.