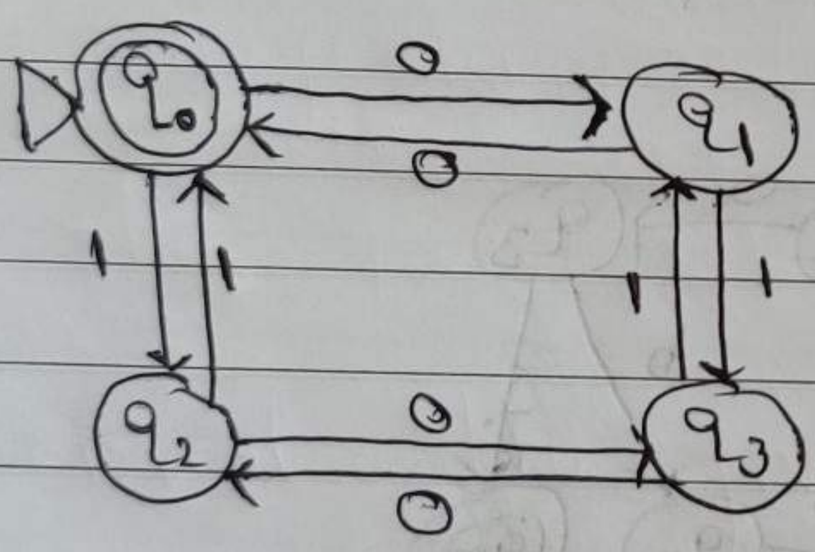
Tutorial No. 1

Q.1) Design F.A with $\Sigma = \{0, 1\}$ accepts even number of 0's & even number of 1's

Soln:- * q_0 = initial state with no 0's & no 1 i.e (even no. of 0 & 1)

q_1 = odd 0 & Even 1
q_2 = Even 0 & odd 1
q_3 = odd 0 & odd 1
* q_4 = Even 0 & 1 i.e * $q_0 \Leftrightarrow$ * q_4

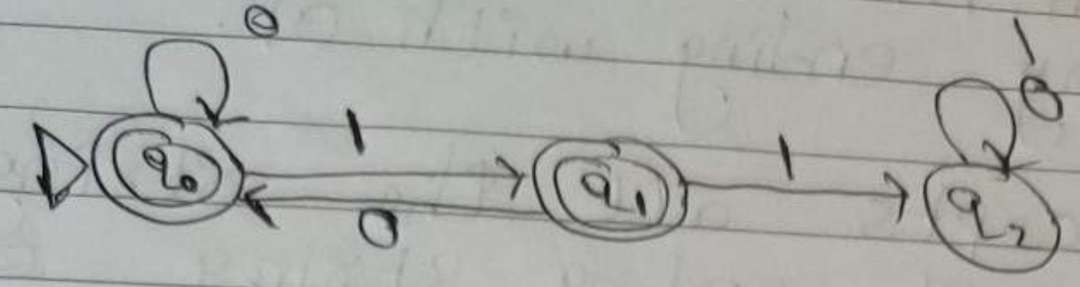
D.F.A =>



Q.2) Design a DFA $L(M) = \{w | w \in \{0,1\}^*\}$ and W is a string that doesn't Contain Consecutive 1's.

Soln) * q_0 = Initial State with no Consecutive 1's
* q_1 = To remember one 1
q_2 = halting State Consisting of Consecutive 1's.

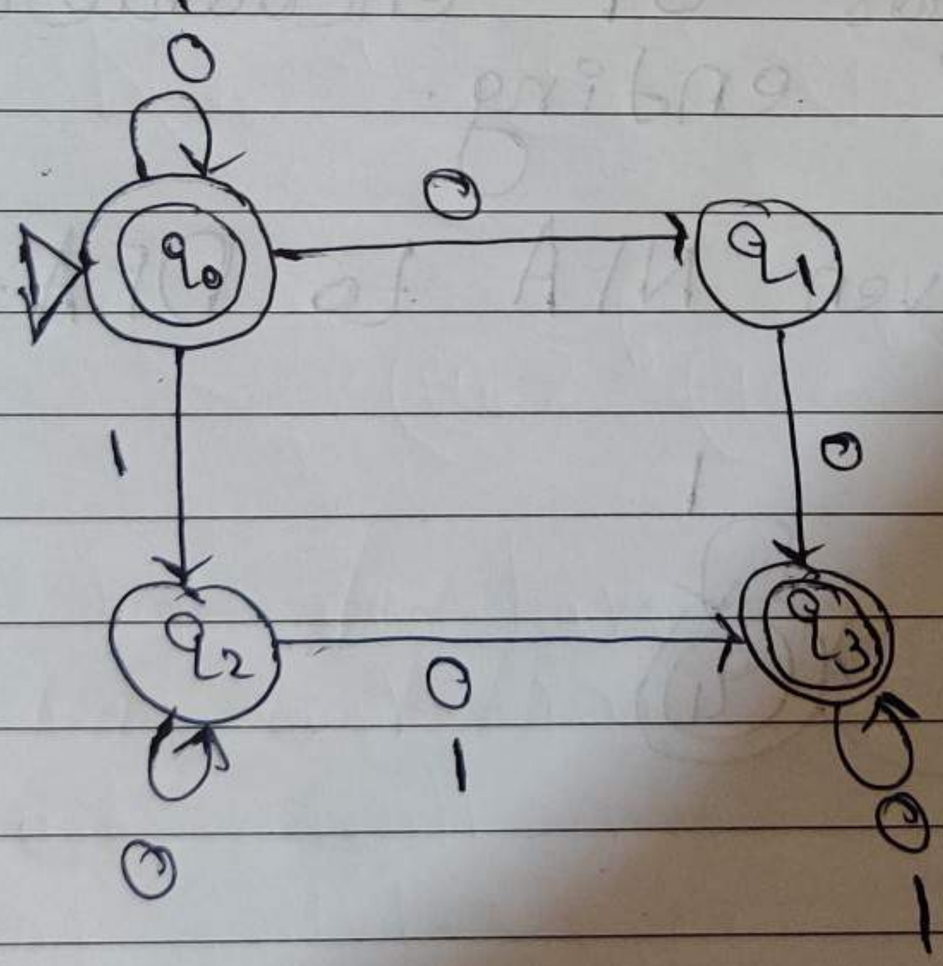
Q. F. A =>



Q.3) Design a NFA for the transition table a given below.

Present State	0	1
$*q_0$	q_0, q_1	q_0, q_2
q_1	q_3	ϵ
q_2	q_2, q_3	q_3
$*q_3$	q_3	q_3

Ans) NFA =>

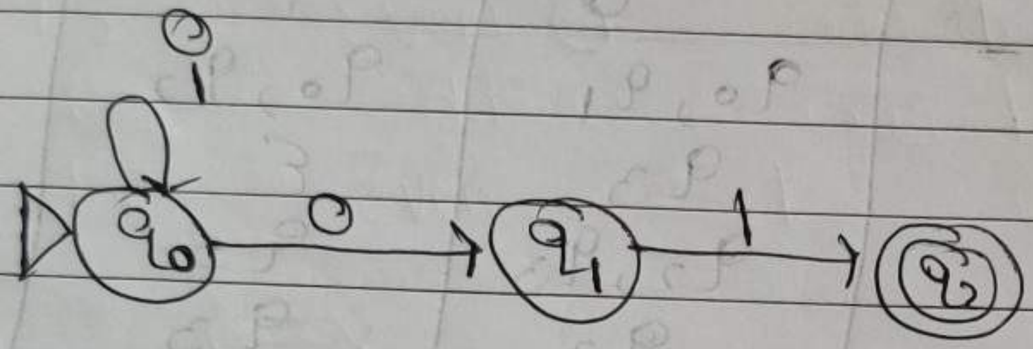


Q.4) Design an NFA with $\Sigma = \{0, 1\}$ accepts all string ending with 01.

Ans) q_0 = Accepts all the strings except ending string 01.

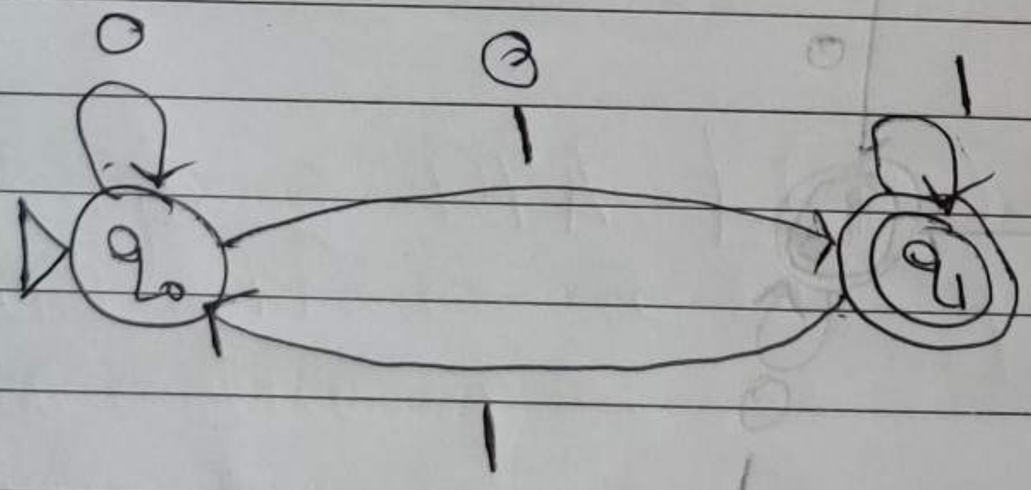
q_1 = accept 0 from 01

*q_2 = accept 1 from 01

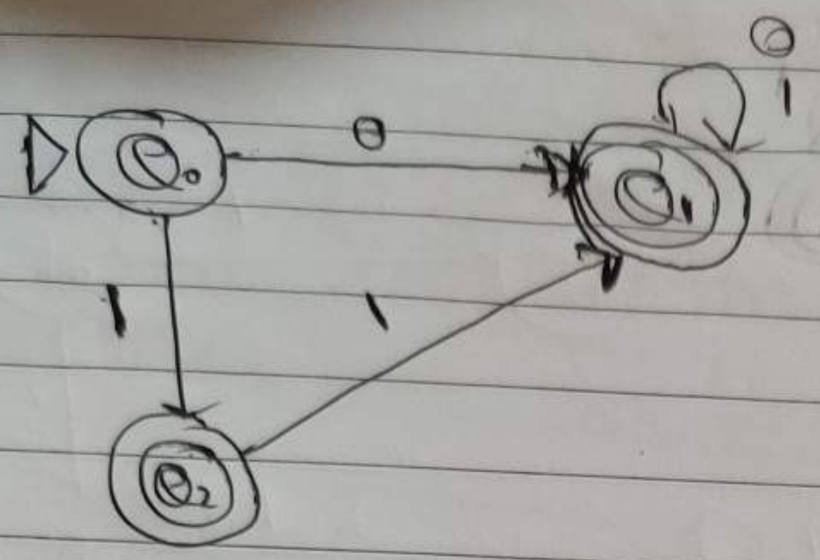


q_0 is non-deterministic because it may happen (most of the time) the 0 ~~encount~~ or 01 encountered isn't the part of ending.

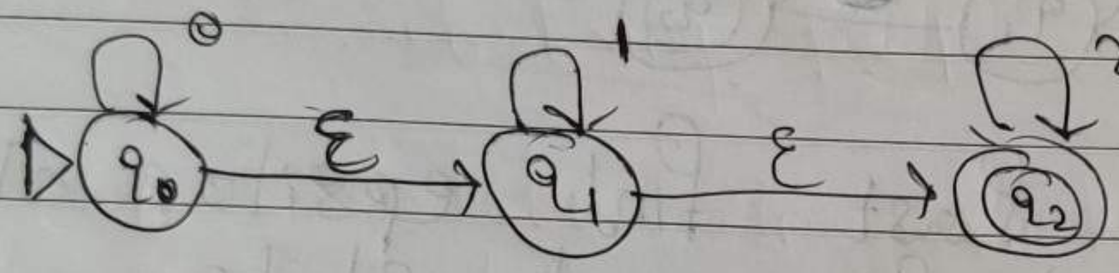
Q.5) Convert the given NFA to DFA.



Present-State	transition on 0	transition on 1
Q_0 $\{q_0\}$	Q_1 $\{q_0, q_1\}$	Q_2 $\{q_1\}$
*Q_1 $\{q_0, q_1\}$	Q_1 $\{q_0, q_1\}$	Q_2 $\{q_0, q_1\}$
*Q_2 $\{q_1\}$	$\{\}$	Q_1 $\{q_0, q_1\}$



Q.) Convert given epsilon NFA into its equivalent DFA.



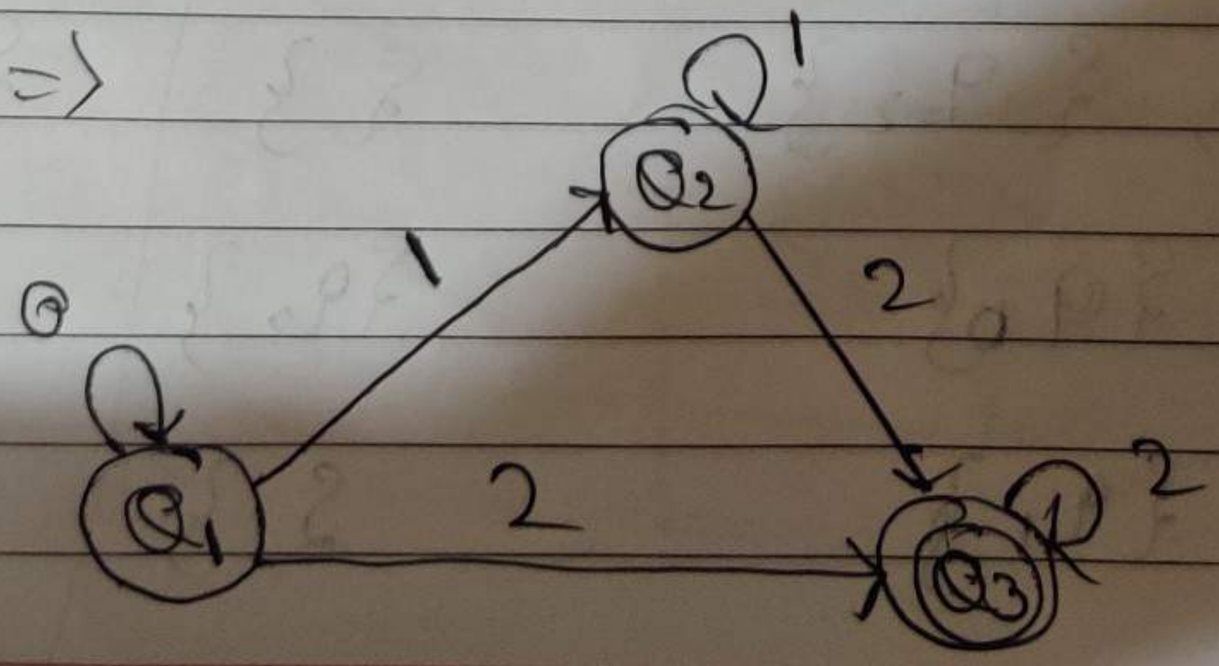
Ans) ε-closure $(q_0) = \{q_0, q_1, q_2\}$ Q_1

ε-closure $(q_1) = \{q_1, q_2\}$ Q_2

ε-closure $(q_2) = \{q_2\}$ Q_3

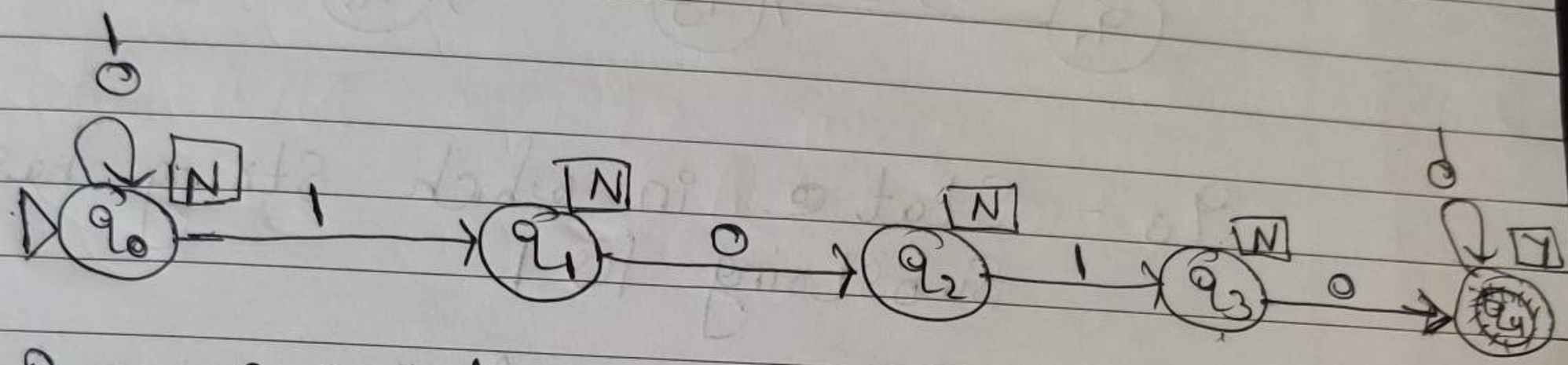
State	0	1	2
Q_1	Q_1	Q_2	Q_3
Q_2	$\{\}$	Q_2	Q_3
*Q_3	$\{\}$	$\{\}$	Q_3

DFA =>



Q.7) Design a Mooxe machine with the input alphabet {0, 1} & output alphabet {Y, N} which produces Y as output if input sequence contains 1010 as a substring otherwise, it produces N as output.

Ans)

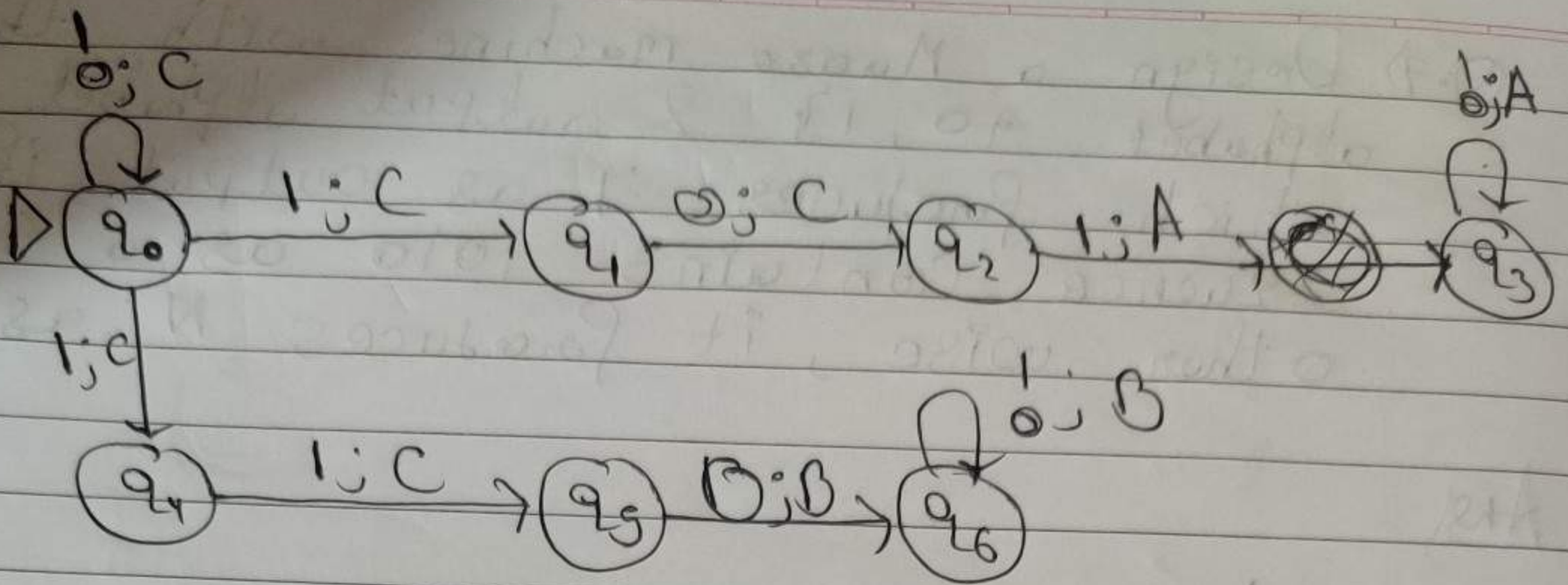


q_0 - Initial state Consumes all the 0 & 1 which doesn't lead to substring 1010

q_1 - Remember 1, q_2 - remember 10
q_3 - remember 101, *q_4 - final State output Y

Q8) Design a Mealy machine for a binary input sequence such that if it has substring 101, the machine output A, if the input has substring 110, it outputs B otherwise it outputs C.

Ans)



q_3 - State in which string has a Substring 101

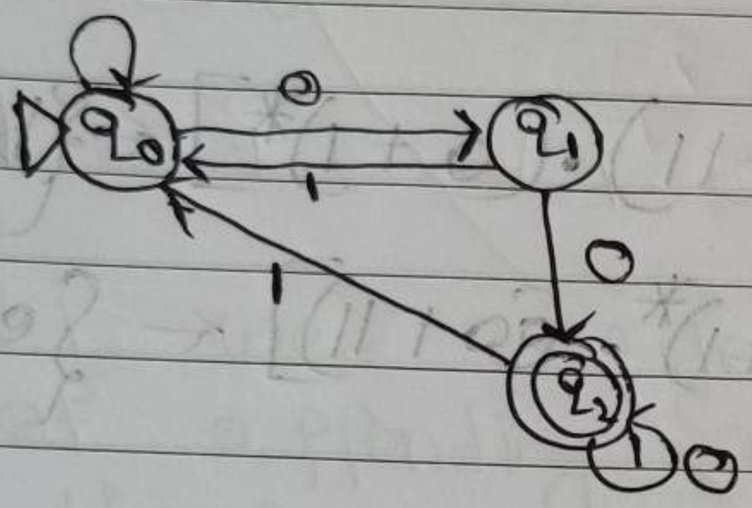
q_6 - State in which string has a Substring 110.

q_0, q_1, q_2, q_4 & q_5 - String without Substrings 101 & 110.

Tutorial # 2

Q.1) Write RE for the language accepting all the string which are ending with 00 over alphabet {0,1}

Ans)



R.E = (0+1)*.0.0

Q.2) Write RE for the language accepting all the strings which are starting with 1 & ending with 0 over the alphabet {0,1}

Ans) RE = 1.(0+1)*.0

Q.3) Write RE for the language accepting all the Strings which are Starting and ending with 1 and any Combination of 0's in between over the alphabet {0,1}.

Ans) R.E = 1.(0)*.1

Q.5) Write RE to denote language L which accept all the string begin or end with either 00 or 11

Ans) RE ⇒ L = ξ

$L_1 = [(00 + 11) \cdot (0 + 1)^*]$ ← {begins with 00 or 11}

$L_2 = [(0+1)^* \cdot (00 + 11)]$ ← {ends with 00 or 11}

$\therefore L = L_1 + L_2 = [(00+11) \cdot (0+1)^*] + [(0+1)^* \cdot (00+11)]$

$\therefore L = \{U \cdot V + V \cdot W$

$\therefore L = \{U \cdot V + V \cdot U \mid V \in \Sigma^* \text{ and } U \in \{00, 11\}\}$

Q.6) Write R.E for the language accepting even length of string over the alphabet {0}

Ans) R.E = $(0 \cdot 0)^*$

Q.7) Write RE for the language accepting odd length of string over the alphabet {1}.

Ans) R.E = $1 \cdot (1 \cdot 1)^*$

Q4) Write RE to denote Language L which is over the alphabet {a, b, c} in which every string such that any number of a's followed by any number of c's

Ans) R.E = (a)* . (C)*

L = { v . v | v ∈ a* and v ∈ c* }

If we consider there is no effect of appending {a, b, c} at the statxt then:

R.E = (a+b+C)* . (a)* . (c)*

L = { v . v . w | v ∈ (a+b+C)*, v ∈ a* and w ∈ c* }

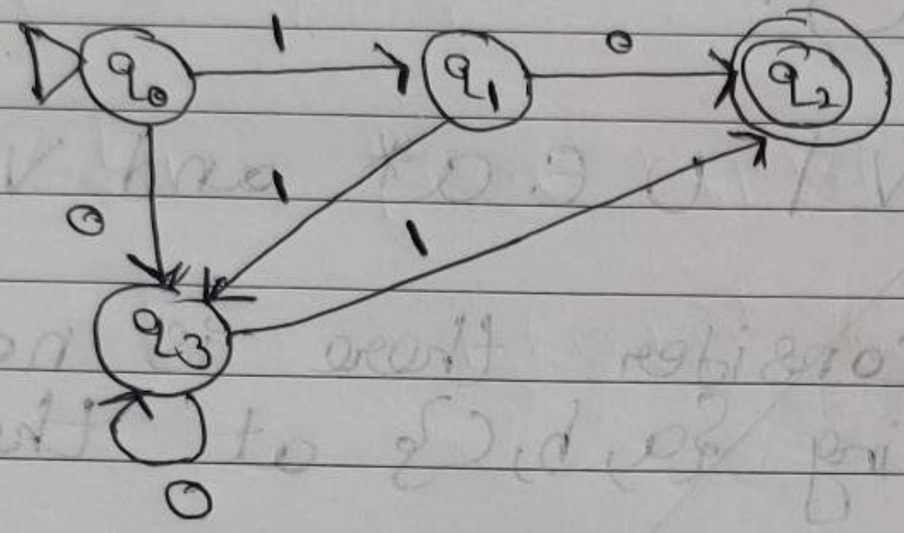
Q4) Write RE to denote the Language L over the alphabet {a, b, c} in which every string such that any number of a's followed by any number of b's & any number of c's

Ans) R.E = a* . b* . c*

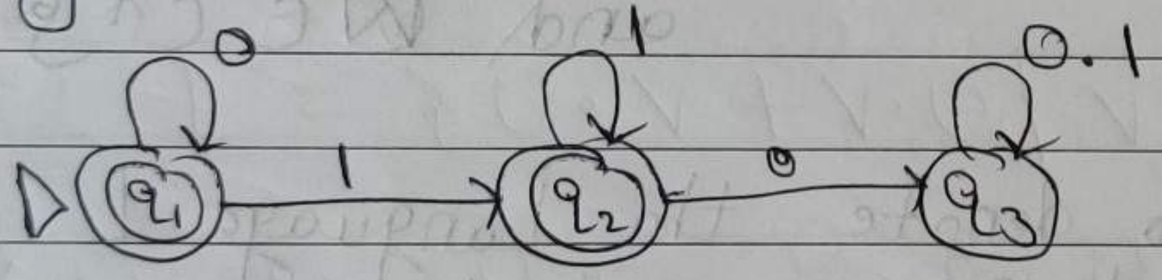
∴ L = { $a^m . b^n . c^i$ | where {m, n, i} ∈ z }

Q8) Design a F.A from given regular expression
$10 + (0 + 11) \cdot 0^* 1$

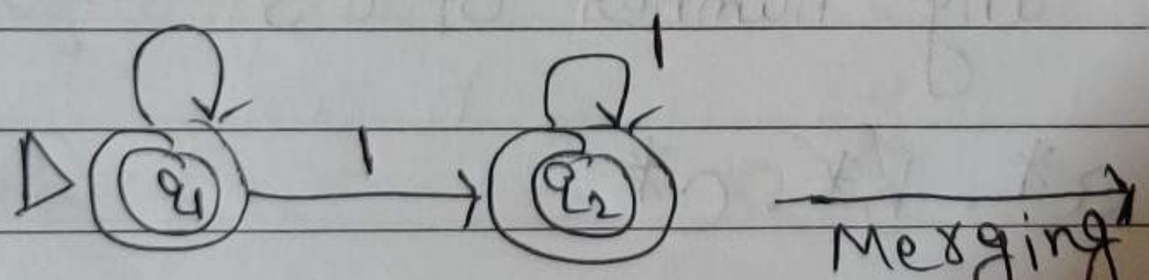
Soln)-



Q.9) Construct the regular expression for the given DFA



removing q_3 not needed for result



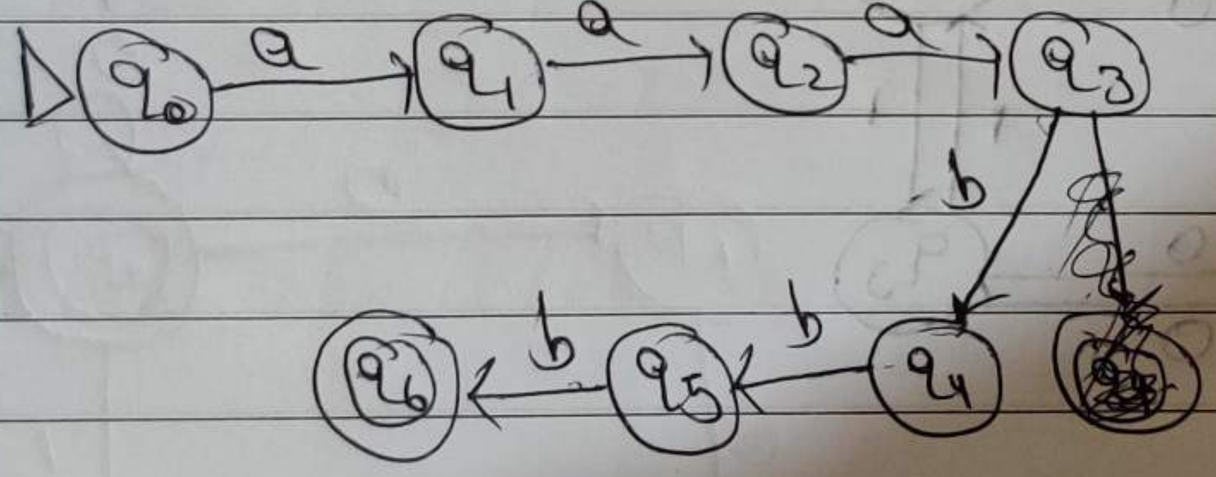
$0^* + 0^* \cdot 1^* \Leftrightarrow \boxed{0^* \cdot 1^*}$

$\therefore \boxed{R \cdot E = 0^* \cdot 1^*}$

Q.10) Prove that $L = \{a^i . b^i \mid i \geq 0\}$ is not regular.

Ans) $L = \{a^i . b^i \mid i \geq 0\}$ we need to generate equivalent number of a's followed by b's in that case we need to remember number of a's generated in an expression to generate equivalent number of b's and there is no such mechanism for regular Language (memorizing).

Let, us show it by example :-
Consider, a.a.a.b.b.b which satisfied L
If we generate NFA or DFA for it :-



If we have infinite number of a&b i.e a,b $\to \infty$ or $a^i . b^i$ where $i \to \infty$ we will need to generate infinite number of states for a&b. Hence, this prove our Question.

Q.10) Proof by contradiction using pumping Lemma.

Considering $L=\{a^i.b^i \mid i>,0\}$ is regular

Let, m be some arbitrary integer for which $a^m b^m \in L$

By Pumping Lemma

$|X.Y| <= m$ & $|Y| >, 1$

for String $X.Y.Z$

$\therefore$ Let, ~~$X=a^1$~~ $X.Y = a^K.a^l$, $\Rightarrow \therefore X.Y = a^{K+l}$

$X=a^K, Y=a^l$ ↗

By above conditions $K+l <= m$

& $K+l+\gamma = m$

$\therefore Z = a^\gamma.b^m$

$\therefore X.Y^i.Z = a^K.(a^l)^i.a^\gamma.b^m$

By Pushing down i.e $i=0$

$\therefore X.Y^0.Z = a^K.a^\gamma.b^m = a^{K+\gamma}.b^m$

but, $|Y| >= 1 \Rightarrow l >= 1 \Rightarrow K+\gamma < m$

which is a contradiction

$\therefore L=\{a^i.b^i \mid i >= 0\}$ is not regular.

Tutorial No. 3

Q.1) Construct a CFG for a language $L = \{wcw^R \mid \text{where } w \in (a,b)^*\}$

Soln:- By observation we can say that this is a Str language for set of odd length palindrome Strings

$\therefore S \Rightarrow aSa \mid bSb \mid c$

aSa insert a at the start & at the end at starting state same goes for bSb, as string grows in size C.F.G ensures that character at i^{th} position is same as character at (n-i+1 position i.e string is of the form $w \cdot w^R$ & at end it insert c which makes our strings of odd length i.e $w \cdot c \cdot w^R$.

Q.2) Construct a C.F.G for the language $L = a^n \cdot b^{2n}$; $n >= 1$.

Soln:- C.F.G:-

$S \Rightarrow aSbb \mid abb$

aSbb recursively generate a' & bb on both side of the string & abb ensures that the length of string is atleast 3 i.e $n = 1$.

Q.3) Check whether the given grammar G_1 is ambiguous or not.

$E \to E+E$, $E \to E-E$, $E \to id$

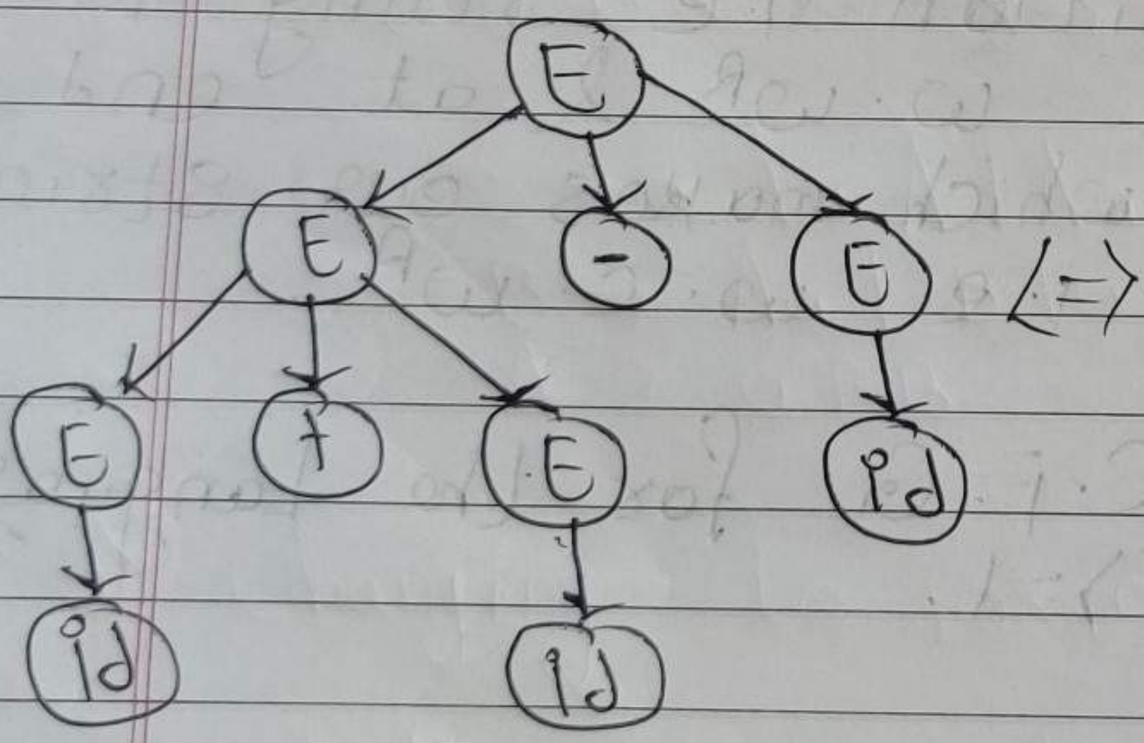
Given a String: $id + id - id$

Soln:- By observation we can say that given grammar is ambiguous. Let us show that with a Given String.

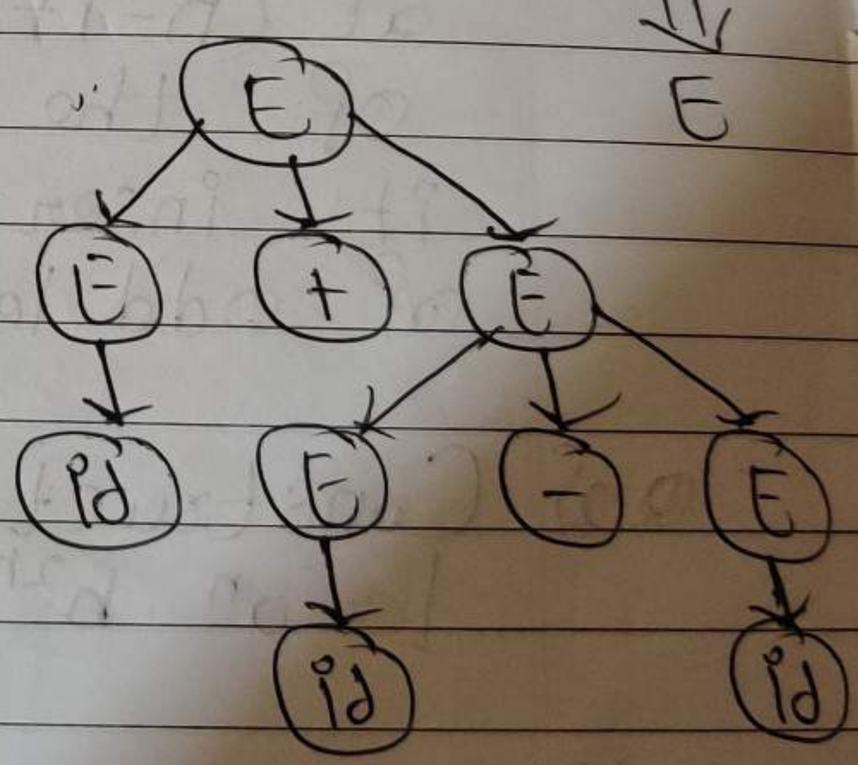
$id + id - id \Rightarrow E + E - id \Rightarrow E - id$

$E \Leftarrow E - E$

$id + id - id \Rightarrow id + E - E \Rightarrow id + E \Rightarrow E + E$



$\Leftrightarrow$



But if we follow only Left most Parsing or Right most Parsing the ambiguity can be removed.

2- Different Parse trees can be generated for same String $\therefore$ Grammar is ambiguous.

Q.4) Check whether the given grammar G is ambiguous or not.

$S \to aSb \mid SS$, $S \to \varepsilon$

Soln:- As we can see there is ε transition which always led to ambiguous grammar.

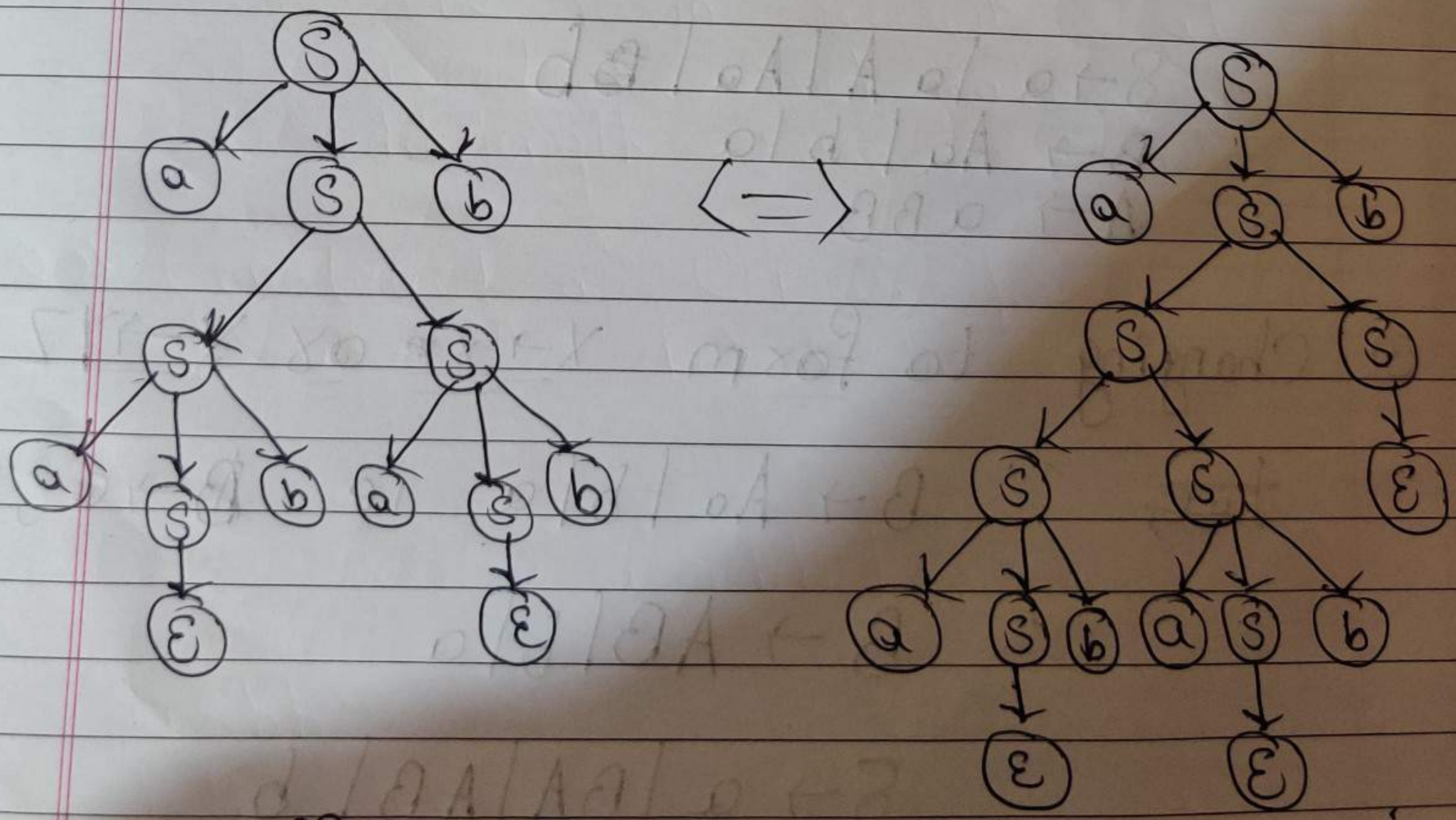
Example:- Let us Parse : aababb

aababb → aaεbaεbb → aasbasbb

S ← aSb ← aSSb

aababb → aaεbaεbεb → aasbasbSb

S ← aSb ← aSSb ← aSSSb



2-Different parse trees can be generated for same ~~gramma~~ String ∴ Grammar is ambiguous.

Q.5) Convert the given CFG to CNF. Consider the given grammar G1:

C.F.G:- $S \to a \mid aA \mid B$, $A \to aBB \mid \varepsilon$, $B \to Aa \mid b$.

Converting to C.N.F:-

1) Removing ε transition:-

$A \to aBB \Rightarrow B \to Aa \mid ba$

$S \to a \mid aA \mid B$

2) Removing unit production:-

$S \to B \Leftrightarrow S \to Aa \mid ba$

$\therefore S \to a \mid aA \mid Aa \mid ba$

$B \to Aa \mid ba$

$A \to aBB$

3) Changing to form $X \to xx$ or $X \to YZ$:-

~~Let~~ $\therefore B \to Aa \mid ba$ ie $B \to a$; $B \to ba$

$\therefore B \to AB \mid ba$

$S \to a \mid BA \mid AB \mid ba$

$A \to BBB$; let $X \to B \cdot B$

$\therefore A \to X \cdot B$

Consider

a|b.

∴ C.N.F :-

S → AB|BA|a|b
A → X.B
X → B.B
B → A.B|b|a. //

Tutorial No. 4

Q.1) Design a PDA for accepting a language $\{0^n.1^m.0^n \mid m, n >= 1\}$

Soln: $S \rightarrow 0S0 \mid 0T0$

$T \rightarrow 1T \mid 1$

Converting to G.N.F:

$A \rightarrow 0 \Rightarrow: S \rightarrow 0SA \mid 0TA$

$T \rightarrow 1T \mid 1$

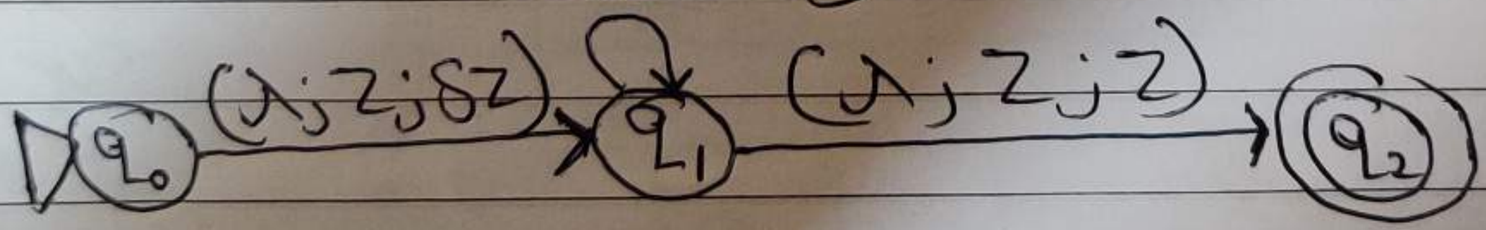
G.N.F to P.D.A: Assuming z last element of Stack is always present

(1, T, T)

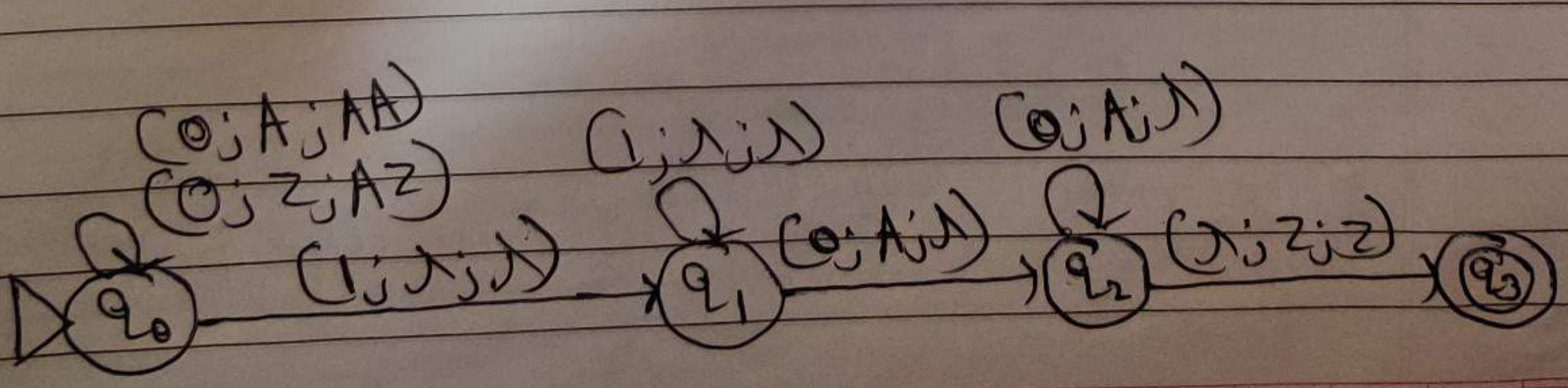
(1, T, T) ; (0, A, λ)

(0, S, TA)

(0, S, SA)



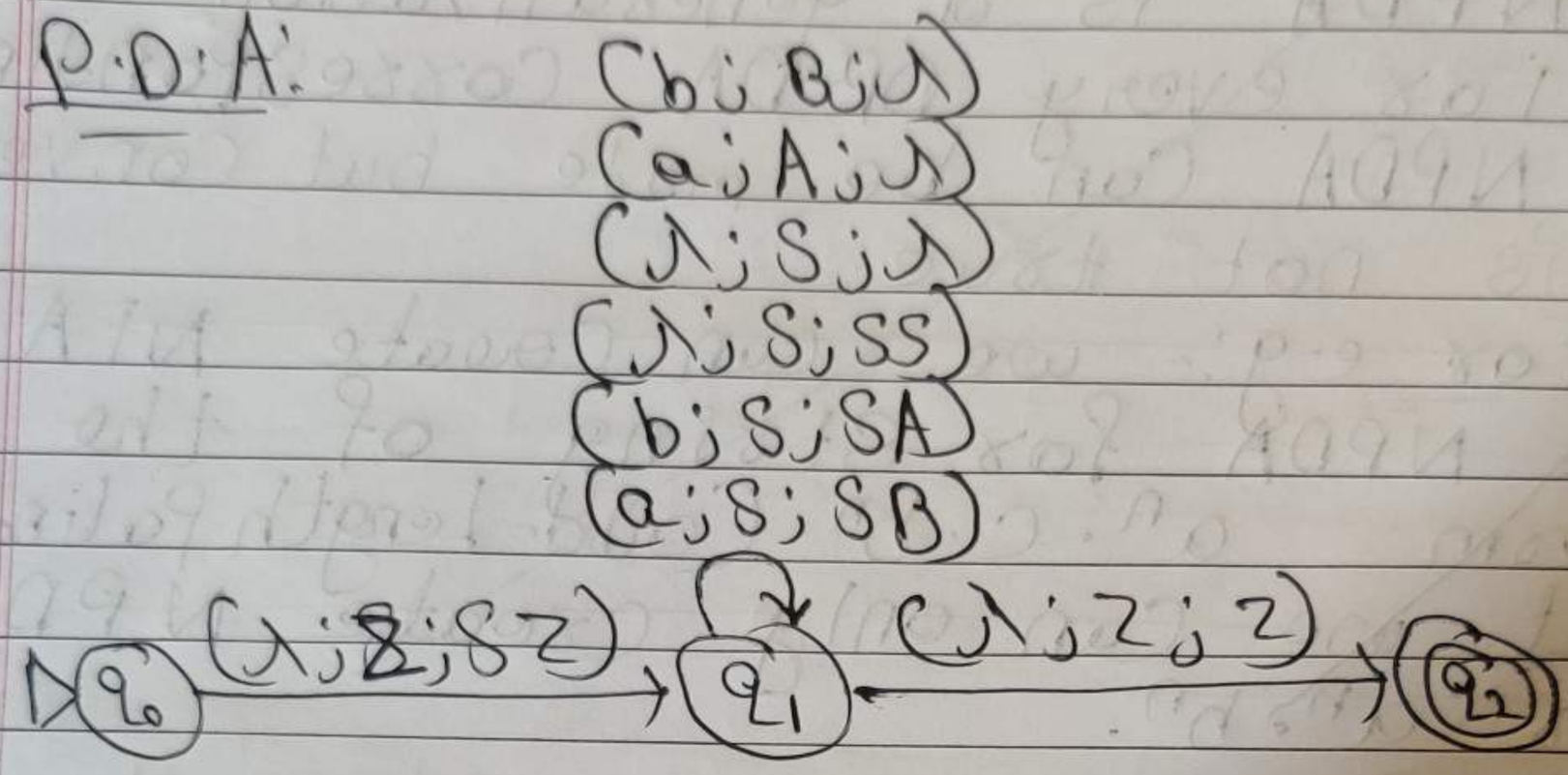
Deterministic P.D.A: For each 0 triggered Push A in q_0 & Pop A in q_2.



Q.2) Design a PDA for equal no. of a's and b's

C.F.G: $S \to aSB \mid bSA \mid SS \mid \lambda$
$A \to a$; $B \to b$

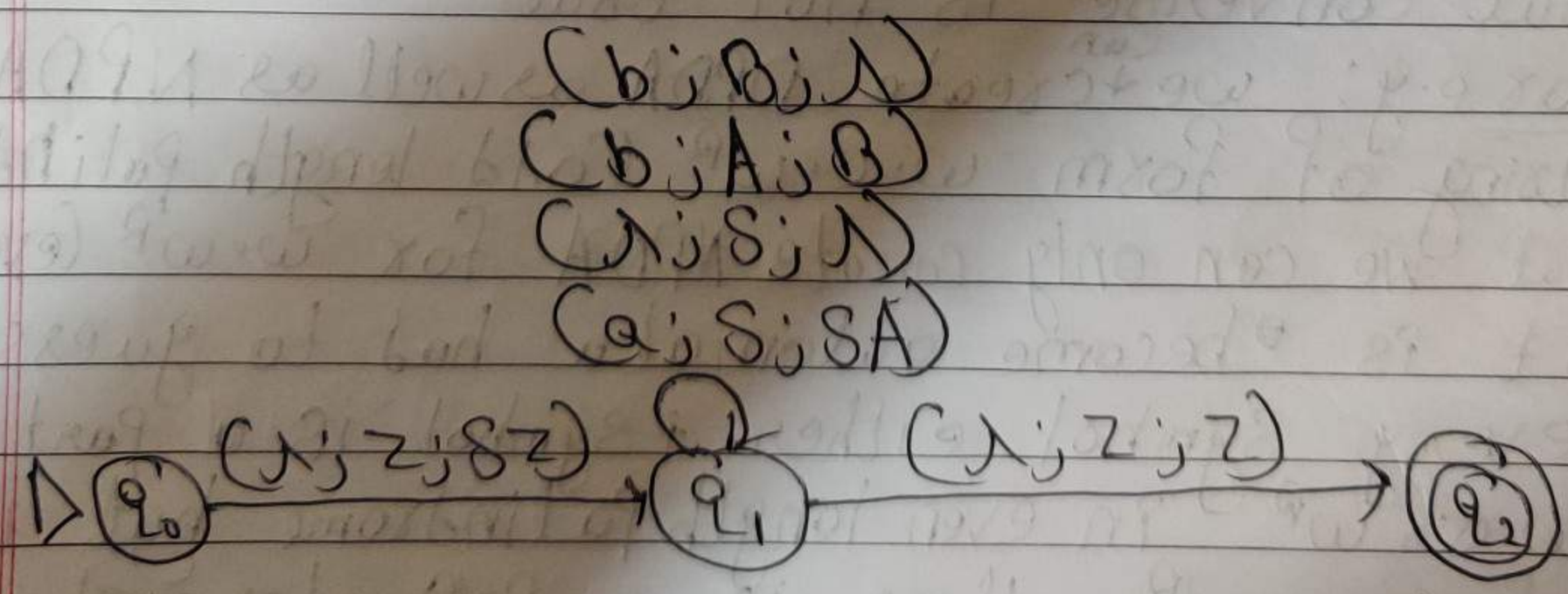
P.D.A:



Q.3) Design a PDA corresponding to the grammar

$S \to aSA \mid \lambda$
$A \to bB$
$B \to b$

P.D.A:



Q.4) Differentiate between PDA & NPDA

Ans) (i) Like, NFA & DFA there also exist Deterministic & Non-deterministic PDA.
(ii) NPDA is a generalization of DPDA.
(iii) For every DPDA corresponding NPDA can be made but converse is not true.
(iv) For e.g:- we can create DPDA & NPDA for string of the form $a^n.c.b^n$ (odd-length palindrome) but we can only create NPDA for $a^n.b^n$.
(v) It is because we can have sta

Q.4) Differentiate between PDA & NPDA

Ans) (i) Like, NFA & DFA there also exist Deterministic and Non-Deterministic PDA.
(ii) NPDA is a generalization of DPDA.
(iii) For every DPDA corresponding NPDA can be created but converse is not true.
(iv) For e.g:- we can create DPDA as well as NPDA for string of form $w.c.w^R$ (odd-length palindrome) but we can only create NPDA for $w.w^R$ (even length)
(v) It is because automata had to guess for every symbol either i symbol is a part of w or w^R in even length palindrome but in case of $w.c.w^R$ there is a seperator 'c' which leads to DPDA
(vi) DPDA $\subseteq$ NPDA
(vii) NPDA can be in a multiple states at any instance but DPDA will always be in a single state.

Q.4) Differentiate between PDA & NPDA

Ans) (I) Like, NFA & DFA there also exist Deterministic and Non-Deterministic PDA.

(II) NPDA is a generalization of DPDA.

(III) For every DPDA corresponding NPDA can be created but converse is not true.

(IV) For e.g:- we can create DPDA as well as NPDA for String of form $w \cdot c \cdot w^R$ (odd-length palindrome) but we can only create NPDA for $w \cdot w^R$ (even length)

(V) It is become automata had to guess for every Symbol either i Symbol is a part of w or w^R in even length palindrome but in case of $w \cdot c \cdot w^R$ there is a seperator 'c' which leads to DPDA

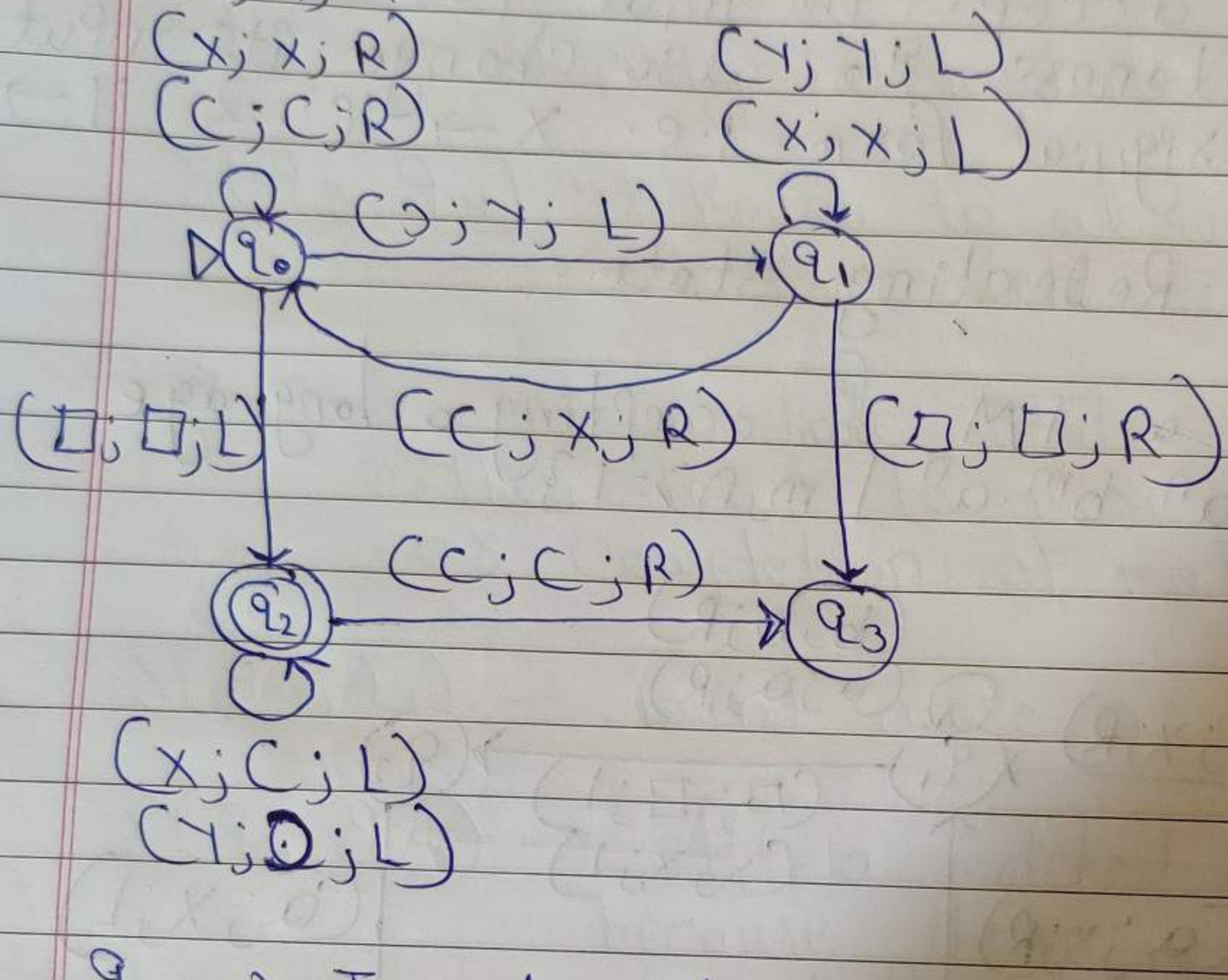
(VI) DPDA $\subseteq$ NPDA

(VII) NPDA can be in a multiple states at any instance but DPDA will always be in a single state.

Tutorial no. 5

Q.1) Design a TM to check well formedness Parathesis

Ans)



q_0 → Initial State & Skip already matched Paranthesis & opening paranthesis (Jolly ride)

q_1 → on triggering closing Paranthesis ')' → Y from q_0 transition is made to q_1
q_1 tries to find if there is any matching opening paranthesis '(' for ')' if there is, it will make transition to q_0 marking '(' → X else, if input is exhausted and no matching paranthesis found automata Halt in q_3 (Rejecting State).

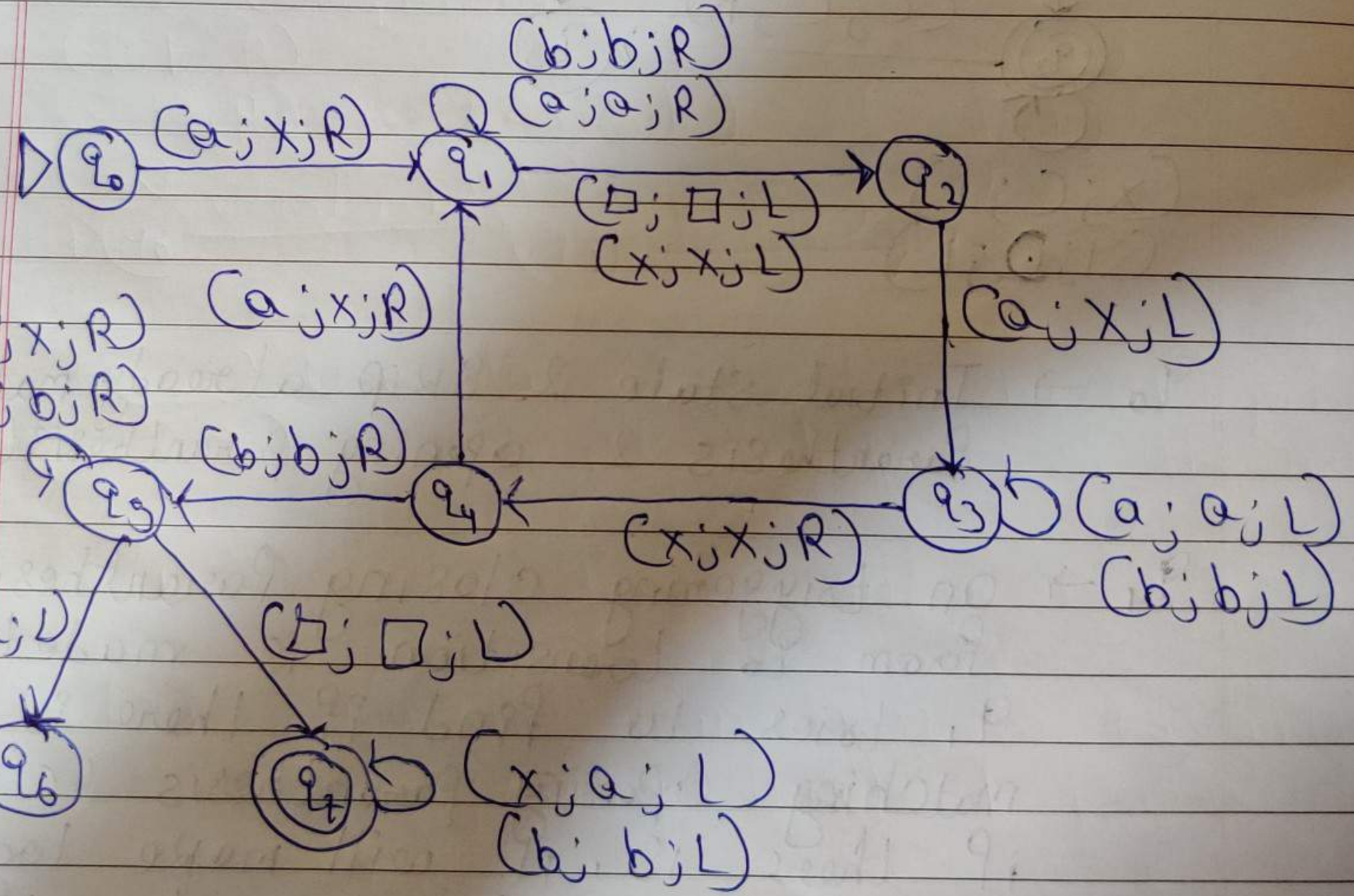
*q_2 → when there is no more ')' ~~to~~ in input

automata make transition to q_2, Here it will check if there is any extra '(' if there is automata make transition to q_3 & reject input else it will stay in q_2 & accept the input. For sake of Completeness q_2 also change its input to original form. i.e. X → '(' & Y → ')'

q_3 → Rejecting State.

Q.2) Design a TM for accepting a language $\{a^n . b^m . a^n \mid m, n >= 1\}$?

Soln.



q_0 → Initial State

$\delta(q_0, q_1)$ → To ensure atleast two a present i.e $n=1$ and start matching of a

q_1 → Skip intermediate a & b

$\delta(q_1, q_2)$ → on triggering end of the input or already matched 'a'.

$q_2 \rightarrow$ Matching 'a' with 'a' ~~triggered~~ triggered at $\delta(q_0, q_1)$ or $\delta(q_4, q_1)$

$\delta(q_2, q_3) \rightarrow$ 'a' Matched Successfully

$q_3 \rightarrow$ Skipping intermediate a & b

$\delta(q_3, q_4) \rightarrow$ Reach to already matched 'a' from Left side of the string.

$q_4 \rightarrow$ Intermediate State which will either lead to more matching or Completion of matching

$\delta(q_4, q_1) \rightarrow$ found more a^+ to be matched

$\delta(q_4, q_5) \rightarrow$ All 'a' matched successfully. because there is no ~~'a'~~ more 'a' before b.

$q_5 \rightarrow$ checking if any extra a's after b is present, if it is then $\delta(q_5, q_6)$ transition is made to reject the input else (q_6, q_7) transition which leads to acceptance of input

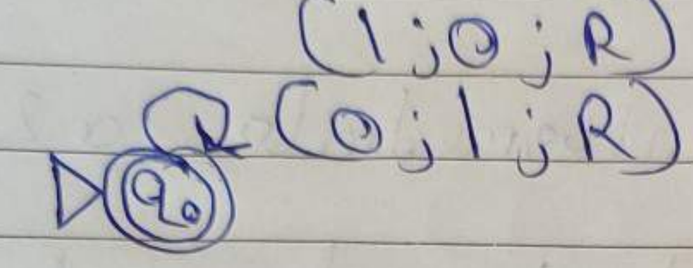
$q_6 \rightarrow$ Rejecting state.

$*q_7 \rightarrow$ Final state where $a^n . b^m . a^n$ is satisfies and for completness each X is resubstitud from 'a'

Epsilon, $m=0$ machine halt at q_4
, $n=0$ machine halt at q_0

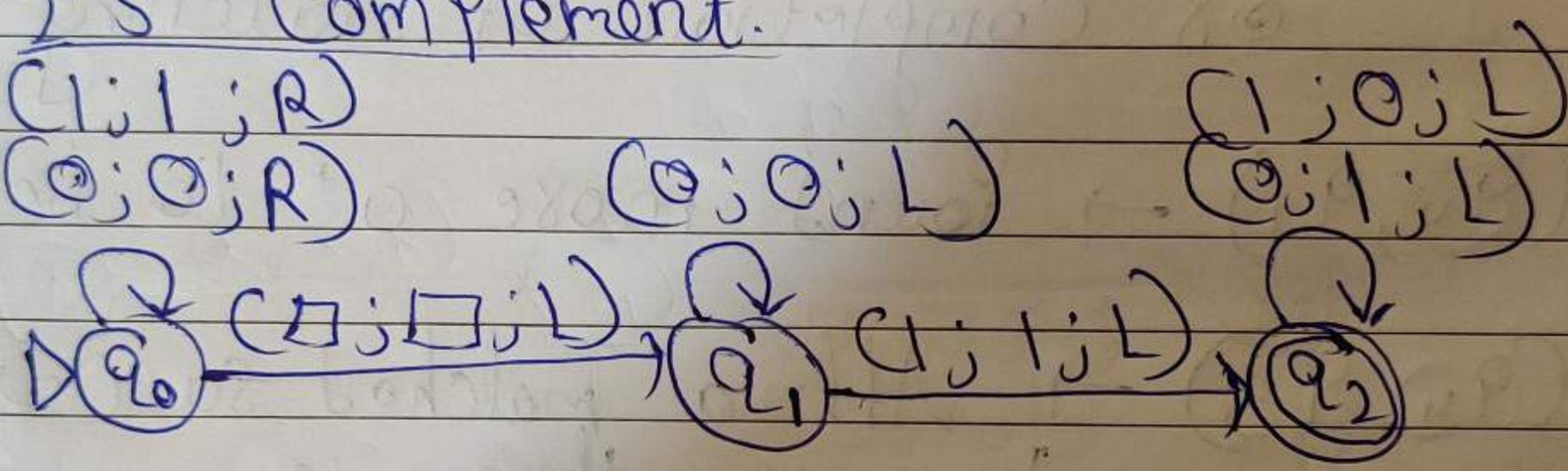
Q.) Design a T.M for 1's and 2's ~~or~~ complement

Sol^n 1's complement:



q_0 → changing every 0 with 1's and every 1 with 0.

2's Complement:



q_0 → use to get to the end of input

$\delta(q_0, q_1)$ → ~~to~~ Right most part of the input

q_1 → skiping all initial zero's

$\delta(q_1, q_2)$ → Making transition to q_2 after triggering first 1 from Right Side.

q_2 → Changing every 0 with 1 & every 1 with 0 after triggering 1.

Q.4) Explain Universal Turing Machine.

Ans) A Universal Turing Machine is a "programmable" machine that can compute anything that is computable as opposed to separate Turing machines computing individual functions that they were hardcoded to compute. A universal machine M_U should be able to simulate the computation carried out by any hardcoded Turing Machine M. Turing proposed such a machine by providing the standard Turing machine with more than one tape (and with one independent reading head for each tape). The universal Turing M_U has three tapes:

1. The input-output tape is used to simulate the tape of the particular machine M (or computable function) which it is simulating. The input as well as output is written on this tape.
2. The compiled program tape stores "the program", that is, the table of instructions or an encoding of the transition functions of the Turing machine M which it is simulating. The tape is read-only tape.
3. The working memory tape, on which the current state of the machine M being simulated is maintained, along with any working memory (i.e., scratchpad) it needs.

The three tapes and their independent reading heads are controlled by a finite control unit with its own states and transition functions. This unit can very well be considered the operating system of the universal Turing machine. It works as follows:

1. Look up the current state information maintained in working memory and the current symbol on the input-output tape.
2. With these two, scan the compiled program tape to find a transition for the given state & current symbol.
3. Execute the transition, that is, write the new symbol on the input-output tape & move its reading head to right or left as specified in the transition; note the new state mentioned in the transition in the working memory.
4. Repeat these ~~stp~~ steps until there are no more transitions in the compiled program.
5. When it halts, if the current state is a final state in the machine M being simulated, then the input is accepted; otherwise, the input is rejected.

Tutorial no. 6

Q.1 Write a short note on: Rice's Theorem.

Ans) Rice theorem states that any non-trivial semantic property of a language which is recognized by a Turing machine is undecidable. A property, P, is the language of all Turing machines that satisfy that property.

Formal Definition: If P is a non-trivial property, and the language holding the property, L_P, is recognized by Turing machine M, then

$L_P = \{\langle M \rangle \mid L(M) \in P\}$ is undecidable.

Trivial property: There exists Turing machines, M_1 and M_2 that recognize the same language, i.e $\langle M_1 \rangle, \langle M_2 \rangle \in L$ or $\langle M_1 \rangle, \langle M_2 \rangle \notin L$

Non-Trivial property: There exists Turing machines, M_1 & M_2 such that $\langle M_1 \rangle \in L$ & $\langle M_2 \rangle \notin L$.

Proof: Suppose, a property P is non-trivial & $\phi \in P$
∴ P is non-trivial, atleast one language satisfies P, i.e., $L(M_0) \in P$, where, M_0 is a Turing Machine.

let, w be an input in a particular instant & N is a Turing Machine which follows:
On input x
1) Run M on w

2) If M doesn't accept then don't accept x.
3) If M accepts w then run M_0 on x. If M_0 accepts x, then accept x.

A function that maps an instance A_{TM} = $\{\langle M, w\rangle \mid M$ accepts $w\}$ to N such that
i) If M accepts w & N accepts the same language as M_0, then $L(M) = L(M_0) \in P$
ii) If M doesn't accept w & N accepts ϕ, Then $L(N) = \phi \notin P$

∴ A_{TM} is undecidable & it can be reduced to L_P, L_P is also undecidable.

Q.2) Write a short note on: Post Correspondence Problem

Ans) Post correspondance problem (PCP) is non-trivial problem of comparing just two strings.
Two sets of string W & V over the same alphabet each having n strings:

$W = \{w_1, w_2, w_3, \ldots w_n\}$
$V = \{v_1, v_2, v_3, \ldots, v_n\}$

The problem is to find concatenation of one or more strings from set W, in any order, where resulting concatenated string is identical to the corresponding concatenation in the same order, of strings from the set V.

Suppose we concatenate all the strings in W in the order in which the elements of W are given: $w_1, w_2, w_3, \ldots$ Similarly, $v_1 \cdot v_2 \cdot v_3 \ldots$ will the two concatenations give us identical strings? i.e. $w_1 \cdot w_2 \cdot w_3 \ldots w_n = v_1 \cdot v_2 \cdot v_3 \ldots v_n$?
Answer is not always.

It turns out that only some instances of the PCP have solutions. For some other pairs of sets W & V, there is no solution to the P.C.P

e.g: $W = \{bb, aba, bbb\}$
$V = \{bbb, bab, b'b\}$

It can observe that solutions to P.C.P for above sets are $w_1 \cdot w_3 = v_1 \cdot v_3$; $v_3 \cdot v_1 = w_3 \cdot w_1$
i.e $bb \cdot bbb = bbb \cdot bb$

what if we consider, $W = \{bb, aba\}$ & $V = \{bbb, bab\}$
There is no solution to P.C.P for above sets.

Q.3) Write a short note on Recursive and Recursive enumerable languages.

Ans) Recursive Language:
A language L is recursive (decidable) if L is the set of strings accepted by some Turing Machine (TM) that halts on every input. ~~i.e Machine (M) reaches final State~~. To put it simply there exist Turing Machine (M) halts when M reaches a State q & a current Symbol 'a' to be scanned so that $\delta(q, a)$ is undefined.

~~Recursive~~
Recursive Enumerable Language:
A language L is recursively enumerable if L is the set of strings accepted by some TM.

If L is a recursive enumerable language then:
If $w \in L$ then a TM halts in a final State,
If $w \notin L$ then a TM halts in a non-final State or loops forever.

If L is a recursive language then:
If $w \in L$ then a TM halts in a final State.
If $w \notin L$ then TM halts in a non-final State.

Recursive languages are also recursive enumerable.