SR 22

St. Peter's Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. QUESTION BANK						Dept.	:	CSE(AIML)
						Academic Year 2023-24		
Subject Code	:	AS22-66PC02	Subject	:	AUTOMATA THEORY & COMPILER DESIGN			
Class/Section	:	B. Tech.	Year	:	II	Semester	:	II

BLOOMS LEVEL					
Remember	L1	Understand	L2	Apply	L3
Analyze	L4	Evaluate	L5	Create	L6

*****

Q. No	Question (s)	Marks	BL	CO
	UNIT – III			
1	Define Push Down Automata (PDA).	1M	L1	C222.2
	Define Turing Machine.	1M	L1	C222.2
	Define Instantaneous Description of PDA.	1M	L1	C222.3
	Define Γ in PDA.	1M	L1	C222.3
	Define Undecidability.	1M	L1	C222.3
2	Discuss about Push Down Automata.	3M	L2	C222.2
	Discuss about Turing Machine.	3M	L2	C222.3
	Discuss about Language of PDA.	3M	L2	C222.2
	Discuss about Decidable Problem with examples.	3M	L2	C222.3
	Discuss about Undecidable Problem with examples.	3M	L2	C222.3
3	Design a PDA for Language L = $\{WCW^r \mid W \in (a + b)^*\}$.	5M	L4	C222.2
	Explain about Instantaneous Description of PDA.	5M	L4	C222.2
	Design a PDA for the Language L = $\{a^n b^{2n} \mid n>0\}$.	5M	L4	C222.2
	Explain in detail about Universal Turing Machine.	5M	L4	C222.3
	Design a PDA for the Language L = $\{WW^r \mid W \in (a,b)^*\}$.	5M	L4	C222.2
4	Explain about Turing Machine as an Adder and Language of a Turing Machine.	10M	L4	C222.3
	Explain about Recursive Language and Recursively Enumerable Languages with examples.	10M	L4	C222.3
	Explain about Turing Machine as a Comparator and as a subtractors.	10M	L4	C222.3

ANSWERS

1.

Define Push Down Automata (PDA).

Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. A Pushdown Automata (PDA) can be defined as: Q is the set of states. $\sum$ is the set of input symbols. Γ is the set of pushdown symbols (which can be pushed and popped from stack).

Define Turing Machine.

A Turing machine is a finite automaton that can read, write, and erase symbols on an infinitely long tape. The tape is divided into squares, and each square contains a symbol.

Define Instantaneous Description of PDA.

ID is an informal notation of how a PDA computes an input string and make a decision that string is accepted or rejected. An instantaneous description is a triple (q, w, α) where: q describes the current state. w describes the remaining input.

Define Γ in PDA.

Γ in PDA is defined as the set of stack symbols.

Define Undecidability.

A decision problem that admits no algorithmic solution is said to be undecidable. No undecidable problem can ever be solved by a computer or computer program of any kind. In particular, there is no Turing machine to solve an undecidable problem.

2.

Discuss about Push Down Automata.

Pushdown Automata is a finite automata with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. This article describes pushdown automata in detail.

Pushdown Automata

A Pushdown Automata (PDA) can be defined as :

- Q is the set of states
- ∑is the set of input symbols
- Γ is the set of pushdown symbols (which can be pushed and popped from the stack)
- q0 is the initial state
- Z is the initial pushdown symbol (which is initially present in the stack)
- F is the set of final states
- δ is a transition function that maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, the PDA will read the input symbol and stack symbol (top of the stack) move to a new state, and change the symbol of the stack.

Discuss about Turing Machine.

A Turing machine consists of a tape of infinite length on which read and writes operation can be performed. The tape consists of infinite cells on which each cell either contains input symbol or a special symbol called blank. It also consists of a head pointer which points to cell currently being read and it can move in both directions.

Figure: Turing Machine

A TM is expressed as a 7-tuple (Q, T, B, ∑, δ, q0, F) where:

- Q is a finite set of states
- T is the tape alphabet (symbols which can be written on Tape)
- B is blank symbol (every cell is filled with B except input alphabet initially)
- ∑ is the input alphabet (symbols which are part of input alphabet)
- δ is a transition function which maps Q × T → Q × T × {L,R}. Depending on its present state and present tape alphabet (pointed by head pointer), it will move to new state, change the tape symbol (may or may not) and move head pointer to either left or right.
- q0 is the initial state
- F is the set of final states. If any state of F is reached, input string is accepted.

Discuss about Language of PDA.

A language can be accepted by Pushdown automata using two approaches:
1. Acceptance by Final State: The PDA is said to accept its input by the final state if it enters any final state in zero or more moves after reading the entire input.
Let P =(Q, ∑, Γ, δ, q0, Z, F) be a PDA. The language acceptable by the final state can be defined as:

1. L(PDA) = {w | (q0, w, Z) ⊢* (p, ε, ε), q ∈ F}

2. Acceptance by Empty Stack: On reading the input string from the initial configuration for some PDA, the stack of PDA gets empty.
Let P =(Q, ∑, Γ, δ, q0, Z, F) be a PDA. The language acceptable by empty stack can be defined as:

1. N(PDA) = {w | (q0, w, Z) ⊢* (p, ε, ε), q ∈ Q}

Equivalence of Acceptance by Final State and Empty Stack

- If L = N(P1) for some PDA P1, then there is a PDA P2 such that L = L(P2). That means the language accepted by empty stack PDA will also be accepted by final state PDA.
- If there is a language L = L (P1) for some PDA P1 then there is a PDA P2 such that L = N(P2). That means language accepted by final state PDA is also acceptable by empty stack PDA.

Example:
Construct a PDA that accepts the language L over {0, 1} by empty stack which accepts all the string of 0's and 1's in which a number of 0's are twice of number of 1's.
Solution:
There are two parts for designing this PDA:

- If 1 comes before any 0's
- If 0 comes before any 1's.

We are going to design the first part i.e. 1 comes before 0's. The logic is that read single 1 and push two 1's onto the stack. Thereafter on reading two 0's, POP two 1's from the stack. The δ can be

1. δ(q0, 1, Z) = (q0, 11, Z) Here Z represents that stack is empty
2. δ(q0, 0, 1) = (q0, ε)

Now, consider the second part i.e. if 0 comes before 1's. The logic is that read first 0, push it onto the stack and change state from q0 to q1. [Note that state q1 indicates that first 0 is read and still second 0 has yet to read].
Being in q1, if 1 is encountered then POP 0. Being in q1, if 0 is read then simply read that second 0 and move ahead. The δ will be:

1. δ(q0, 0, Z) = (q1, 0Z)
2. δ(q1, 0, 0) = (q1, 0)
3. δ(q1, 0, Z) = (q0, ε) (indicate that one 0 and one 1 is already read, so simply read the second 0)
4. δ(q1, 1, 0) = (q1, ε)

Now, summarize the complete PDA for given L is:

1. δ(q0, 1, Z) = (q0, 11Z)
2. δ(q0, 0, 1) = (q1, ε)
3. δ(q0, 0, Z) = (q1, 0Z)
4. δ(q1, 0, 0) = (q1, 0)

5. $\delta(q1, 0, Z) = (q0, \varepsilon)$
6. $\delta(q0, \varepsilon, Z) = (q0, \varepsilon)$ ACCEPT state

Discuss about Decidable Problem with examples.

A problem is said to be Decidable if we can always construct a corresponding algorithm that can answer the problem correctly. We can intuitively understand Decidable issues by considering a simple example. Suppose we are asked to compute all the prime numbers in the range of 1000 to 2000. To find the solution to this problem, we can easily construct an algorithm that can enumerate all the prime numbers in this range.
Now talking about Decidability in terms of a Turing machine, a problem is said to be a Decidable problem if there exists a corresponding Turing machine that halts on every input with an answer- yes or no. It is also important to know that these problems are termed Turing Decidable since a Turing machine always halts on every input, accepting or rejecting it.
Semi Decidable Problems.

Semi-decidable problems are those for which a Turing machine halts on the input accepted by it but it can either halt or loop forever on the input which the Turing Machine rejects. Such problems are termed as Turing Recognisable problems.
Example
We will now consider some few important Decidable problems:

- Are two regular languages L and M equivalent: We can easily check this by using Set Difference operation. L-M =Null and M-L =Null. Hence (L-M) U (M-L) = Null, then L,M are equivalent.
- Membership of a CFL: We can always find whether a string exists in a given CFL by using an algorithm based on dynamic programming.
- Emptiness of a CFL By checking the production rules of the CFL we can easily state whether the language generates any strings or not.

Discuss about Undecidable Problem with examples.

The problems for which we can't construct an algorithm that can answer the problem correctly in finite time are termed as Undecidable Problems. These problems may be partially decidable but they will never be decidable. That is there will always be a condition that will lead the Turing Machine into an infinite loop without providing an answer at all.
We can understand Undecidable Problems intuitively by considering Fermat's Theorem, a popular Undecidable Problem which states that no three positive integers a, b and c for any n>2 can ever satisfy the equation: a^n + b^n = c^n. If we feed this problem to a Turing machine to find such a solution which gives a contradiction then a Turing Machine might run forever, to find the suitable values of n, a, b and c. But we are always unsure whether a contradiction exists or not and hence we term this problem as an Undecidable Problem.
Example
These are few important Undecidable Problems:

- Whether a CFG generates all the strings or not: As a Context Free Grammar (CFG) generates infinite strings, we can't ever reach up to the last string and hence it is Undecidable.

- Whether two CFG L and M equal: Since we cannot determine all the strings of any CFG, we can predict that two CFG are equal or not.
- Ambiguity of CFG: There exist no algorithm which can check whether for the ambiguity of a Context Free Language (CFL). We can only check if any particular string of the CFL generates two different parse trees then the CFL is ambiguous.
- Is it possible to convert a given ambiguous CFG into corresponding non-ambiguous CFL: It is also an Undecidable Problem as there doesn't exist any algorithm for the conversion of an ambiguous CFL to non-ambiguous CFL.
- Is a language Learning which is a CFL, regular: This is an Undecidable Problem as we cannot find from the production rules of the CFL whether it is regular or not.

3.

Design a PDA for Language L = {WCW^r | W ∈ (a + b)*}.

Read the string w and push it onto the stack until it encounters c. After that, read each symbol, if it matches with the top of the stack, pop off the symbol. When input is read completely, if stack becomes empty, then string is accepted.

The PDA can be given as follows:

Let q0 be initial state, qf be final state and Z0 be the bottom of the stack as shown in Figure.

δ(q0, a, Z0) = (q0, aZ0)
δ(q0, a, a) = (q0, aa)
δ(q0, a, b) = (q0, ab)
δ(q0, b, Z0) = (q0, bZ0)
δ(q0, b, a) = (q0, ba)
δ(q0, b, b) = (q0, bb)
δ(q0, c, Z0) = (q1, Z0)
δ(q0, c, a) = (q1, a)
δ(q0, c, b) = (q1, b)
δ(q1, a, a) = (q1, ε)
δ(q1, b, b) = (q1, ε)
δ(q1, ε, Z0) = (qf, Z0)

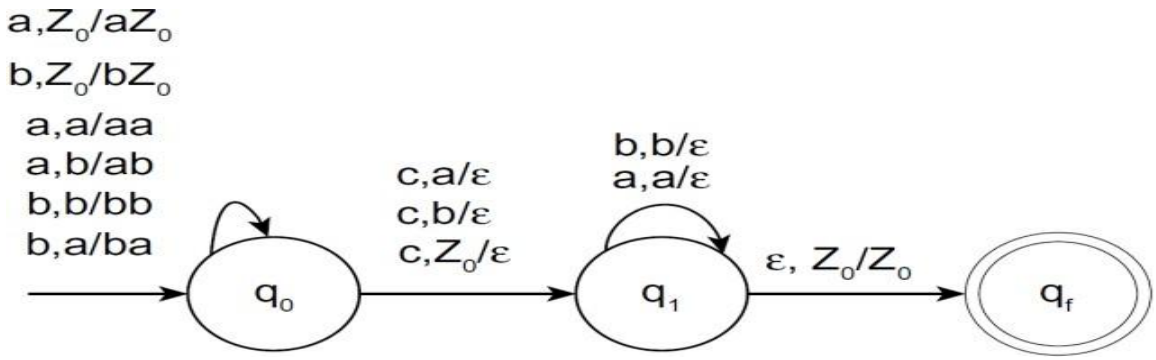


Fig. *PDA*

The final PDA is given by M = {(q_0, q_1, q_f), (a, b, c), (a, b, Z_0), δ,q_0, Z_0, {q_f}}.

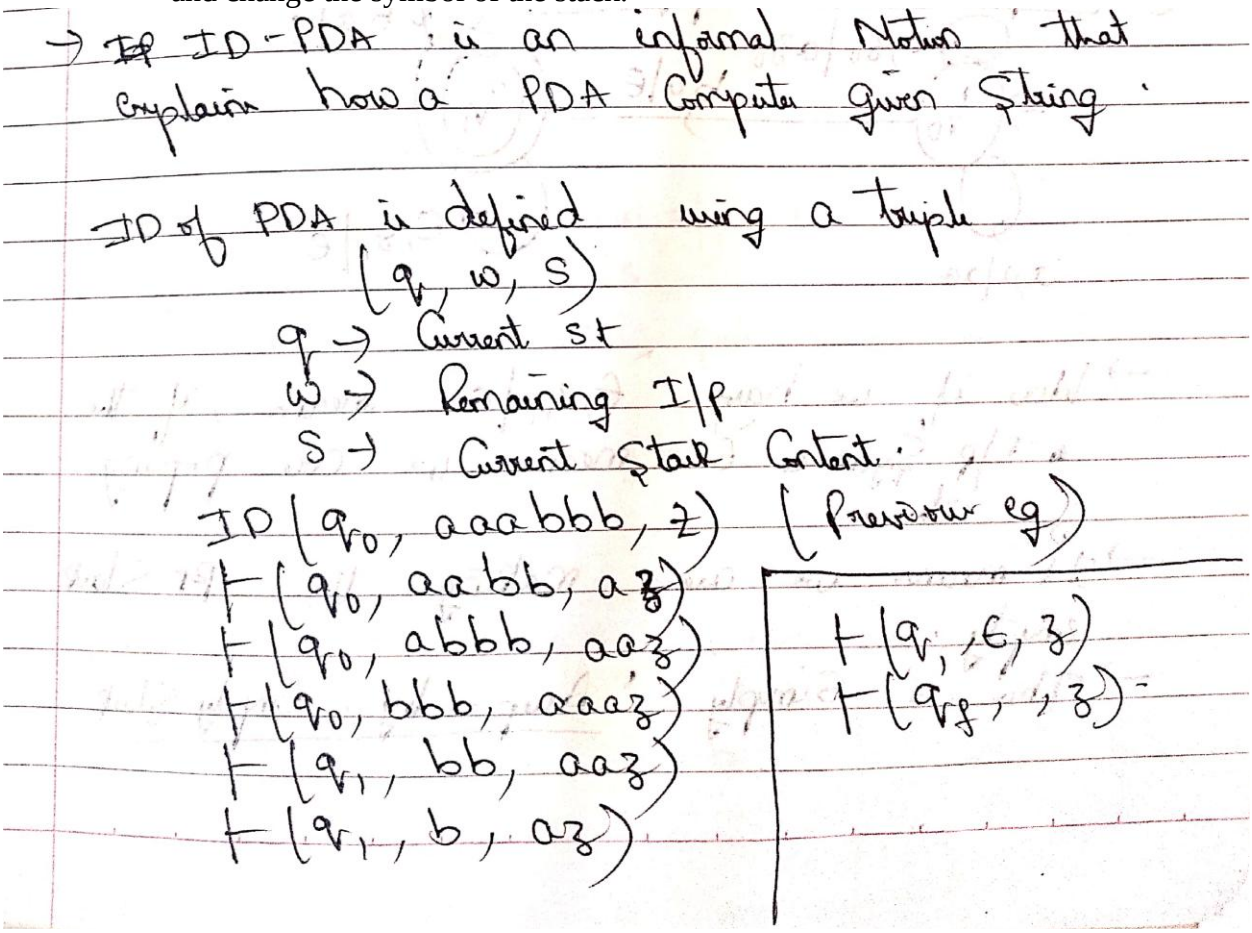
Explain about Instantaneous Description of PDA.

Pushdown Automata is a finite automata with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. This article describes pushdown automata in detail.

Pushdown Automata

A Pushdown Automata (PDA) can be defined as :

- Q is the set of states
- ∑is the set of input symbols
- Γ is the set of pushdown symbols (which can be pushed and popped from the stack)
- q0 is the initial state
- Z is the initial pushdown symbol (which is initially present in the stack)
- F is the set of final states
- δ is a transition function that maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, the PDA will read the input symbol and stack symbol (top of the stack) move to a new state, and change the symbol of the stack.



Design a PDA for the Language L = {$a^n b^{2n}$| n>0}.

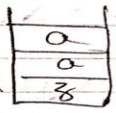
Eg's:

$\{L = a^n b^{2n}, n>0\}$

$L = \{abb, aabbbb, \ldots\}$

aabbbb

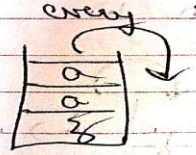
→ 'L' is a Lang with no of a's followed by double the no of b's



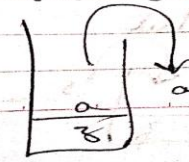
→ So, whenever we get 'a', push down to the stack.

aabbbb

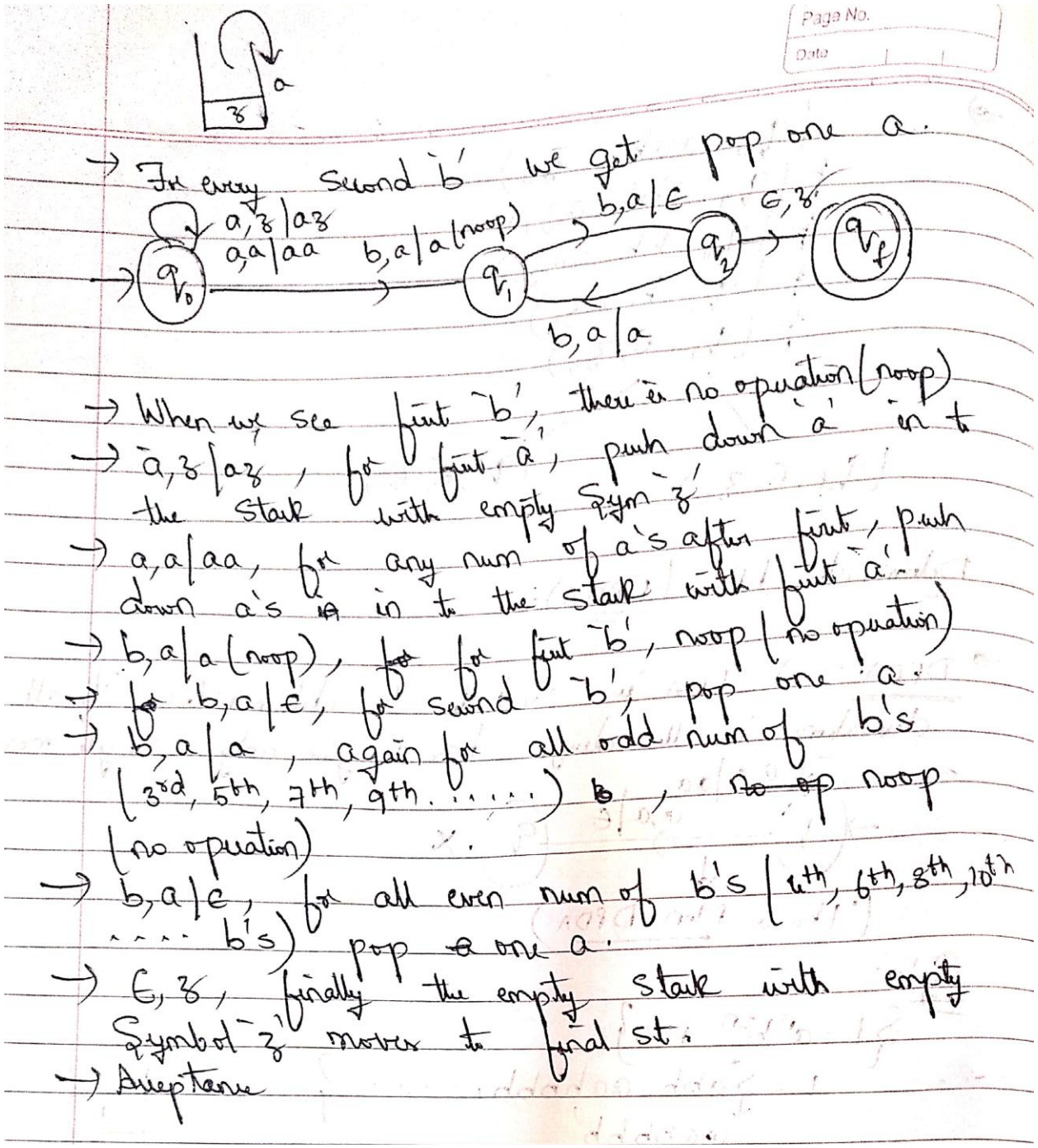
→ And for every 2 b's, pop one a.



bbbb →



bb

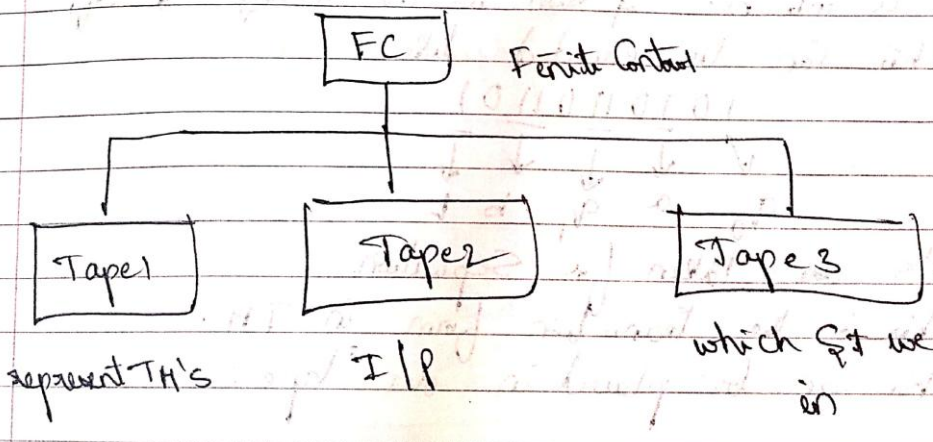


Explain in detail about Universal Turing Machine.

B → Blank Sym
F → Set of Final State's

Universal TM (UTM)

→ TM work like today's Comp. So, whatever Computations that can be done by today's Comp's & done by TM
→ TM works as an Adder, Substractor, Multiplier & even as a Comparison Operator.
→ Today's Comp's works according to the option we select. For eg: Apps, according to our requirement we select a particular & Comp will perform that action.
→ But whereas in TM, TM's don't work according to it. We have a Separate TM for addition, Substration etc.
→ So, TM Can work like today's Comp by UTM
→ UTM works like today's Comp's.



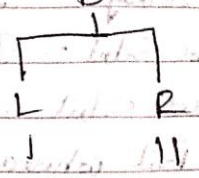
→ UTM Consists of FC, this is going Control all the TM's Tape.
→ 3 Tape - Tape1, Tape2 & Tape3.
→ Tape1 is used to represent the TM.

- Eg: we need a TM for addition, or Substitution
  Ec it will be placed on Tape1 according to our requirement.
→ Tape2 - looks after the I/P.
→ Tape3, ~~repeat~~ looks after 'which st we are in'.
→ By using FC, we can execute any TM.
→ This works exactly like our Today's Comp.

Representation of TM

q_0, q_1, q_2 — a, b

1 11 111 — 1 11



$$\delta(q_0, a) = [q_1, b, L]$$

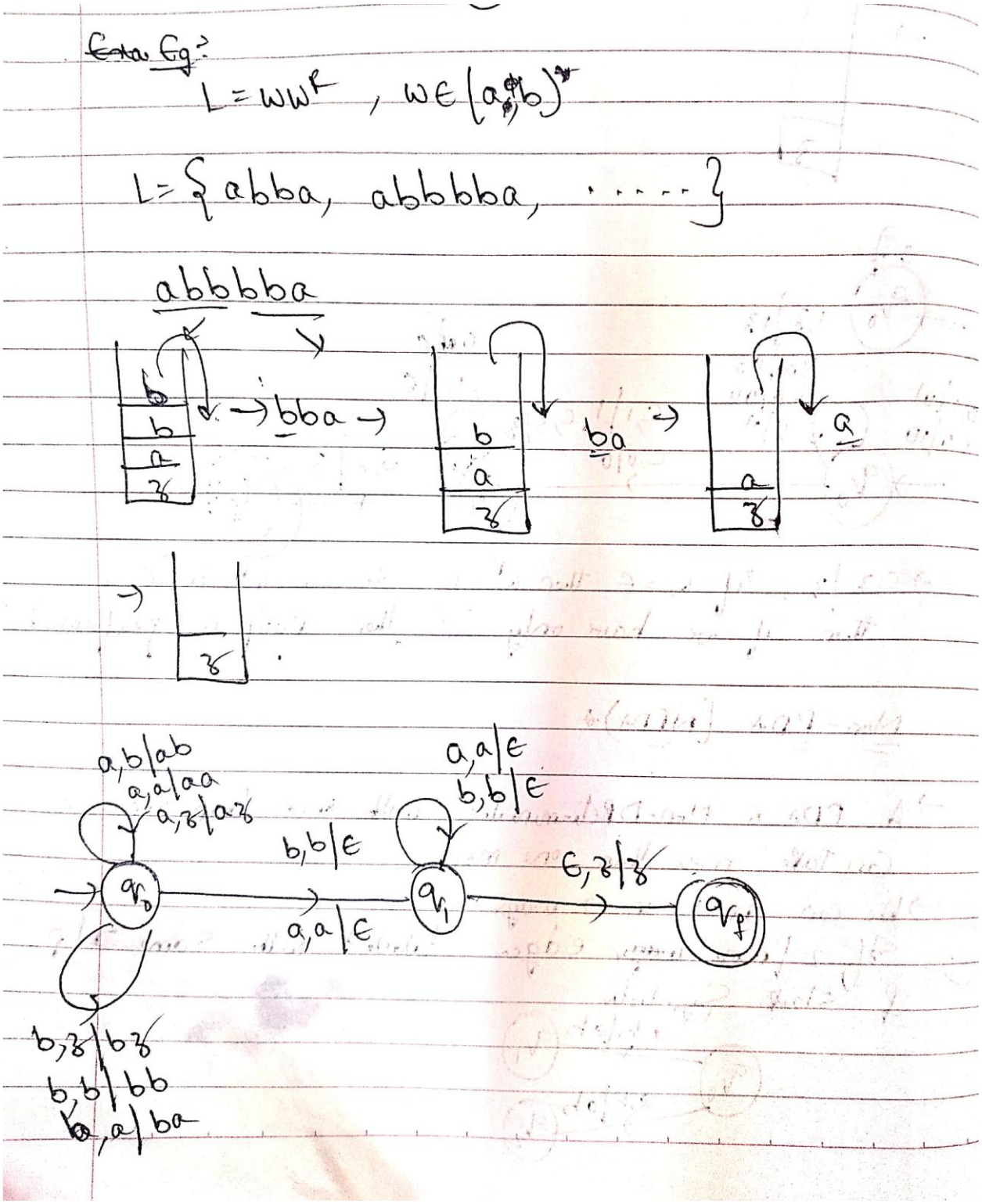
1 0 1 0 11 0 11 0 1

→ With 'a' I/P Sym q_0 moves to q_1 & Converting a into b. & then we are moving to Left.
→ This can be encoded like

1 0 1 0 11 0 11 0 1
↓ ↓ ↓ ↓ ↓
q_0 a q_1 b L

→ 0's are taken for Separation
→ This is the transition form in TM.
→ This can be placed in our Tape.

Design a PDA for the Language L = {WW^r | W ∈ (a,b)*}.



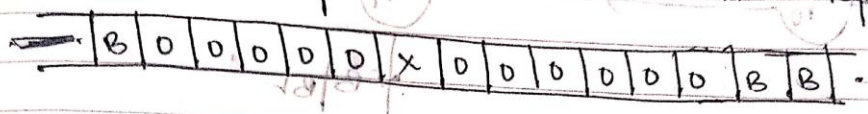
10.

Explain about Turing Machine as an Adder and Language of a Turing Machine.
Explain about Recursive Language and Recursively Enumerable Languages with examples.
Explain about Turing Machine as a Comparator and as a Substractor.

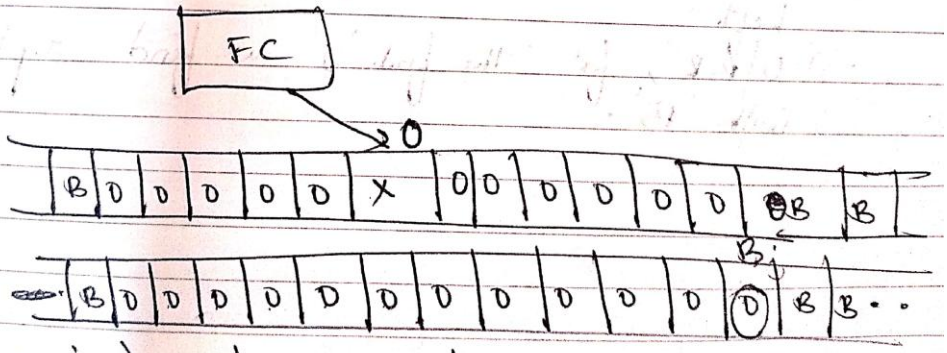
Explain about Turing Machine as an Adder and Language of a Turing Machine.

TM as an Adder

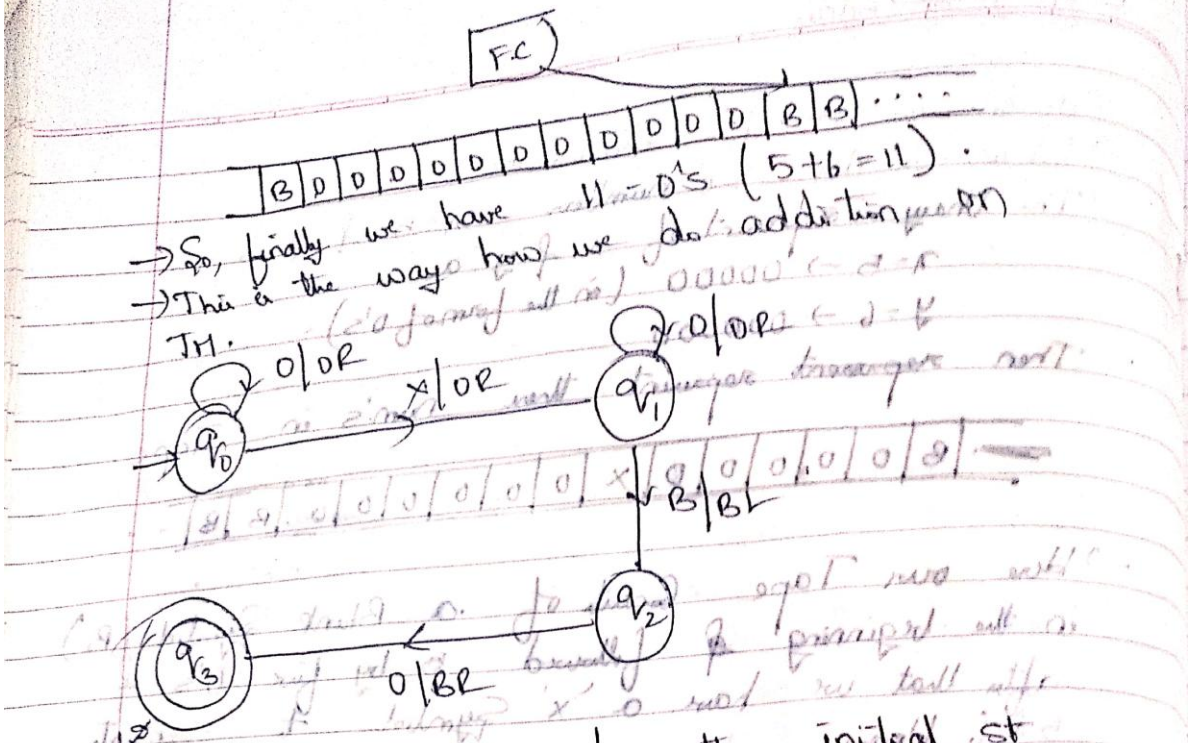
→ We can use TM for addition.
→ Let us Consider two numbers $x=5$ & $y=6$
→ The Unary represent of x & y are
$x=5 \rightarrow 00000$ (in the form of 0's).
$y=6 \rightarrow 000000$
→ Then ~~represent~~ represent them num's in Tape.



→ Here our Tape Consits of a Blank Symbol (B) in the beginning & followed by five 0's & after that we have a 'x' Symbol to seperate 5 & 6 Unary num's
→ Then we have '6' B's (Unary form)
→ This is the way how we represent our I/P on the Tape. Starting from left
→ When we add 5 & 6 we need to get 11 (in Unary representation, we will get ~~11~~ eleven 0's)
→ So, we start from Blank Sym 'B' & we go upto 'x' & replace that 'x' as '0'
→ So, finally we will have ~~11~~ 12 0's.



→ But we need only 11-0's, So, we will move to the end & we will Convert the last 0 as B

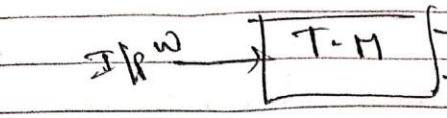


→ So, finally we have 11 0's (5+6=11).
→ This is the way how we do addition on TM.
→ First we start from the initial st `q0`, if we get `0`, move right (0/0R) for all 0's
→ X/0R, whenever we find `x` replace it with `0` & move right; Again `x` come on the Tape, the m/c moves from `q0` to `q1`.
→ 0/0R, for any num of 0's, move right &
→ B/BL, whenever we find Blank `B`, move Left.
→ 0/BR, for the first `0` we find replace it with `B`.

Explain about Recursive Language and Recursively Enumerable Languages with examples.

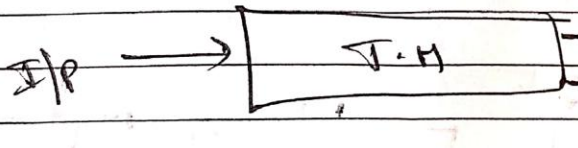
Recursive & Recursively enumerable Lang

Recursive: A lang 'L' over alphabets 'Σ' is called recursive Lang if their is a TM that accepts every word 'L' or reject every word 'L'.



→ This has only given O/p as either Accept or Reject.

Recursively Enumerable: A lang 'L' over the alphabet 'Σ' is called Recursively Enumerable if their is a TM that accepts every word in 'L' & either reject or loop for every other words.



→ Here the TM may Accept ✓, or Reject ✗ enter into a Infinite loop.

Recursive Lang: TM will halt (accept / Reject)

Recursive Enumerable Lang:
TM may halt sometimes accept & may not halt.

→ TM may halt (accept / reject) or may not halt (Infinite loop).

Page No.
Date / /

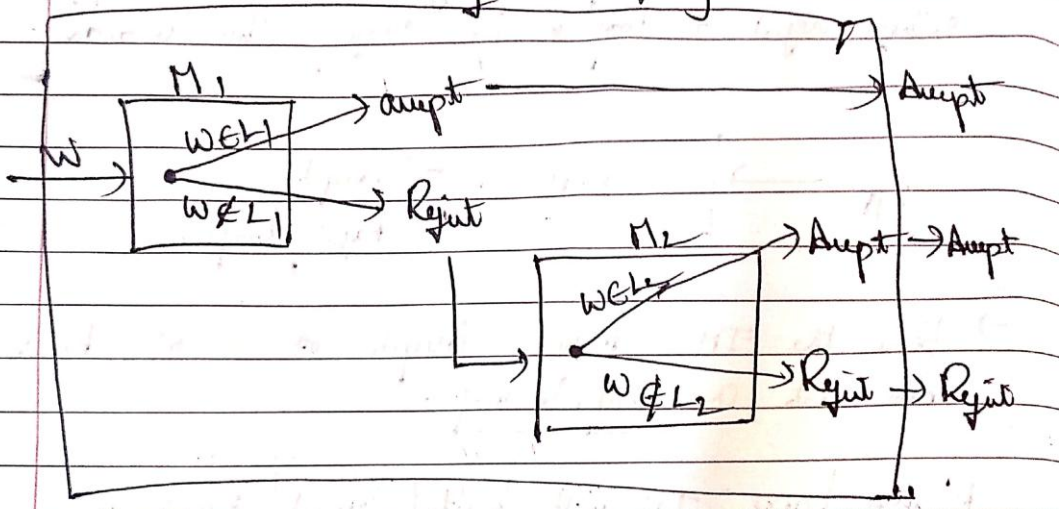
Decidable Lang : Recursive Lang
Partially D.L : Recursively Enumerable Lang
(Halt / may not Halt)
Undecidable : No TM exists for these Lang's

~~Key Properties of Enumerable~~
Properties of Recursive & Recursively Enumerable Lang's :

(I) The Union of 2 Recursive Lang's is Recursive :

→ Let L_1 be a recursive Lang accepted by TM M_1 & L_2 be a recursive Lang accepted by TM M_2.

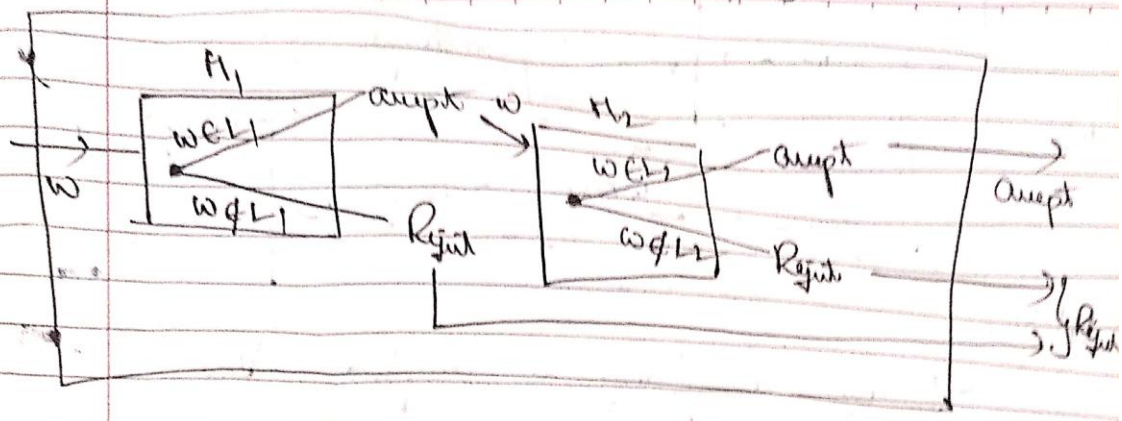
$L_1 \cup L_2$ { it should be accepted by any 1 M/c }



(II) The intersection of 2 Recursive Lang's is Recursive

→ Let L_1 be a recursive Lang accepted by TM M_1 & L_2 be a Recursive Lang accepted by TM M_2

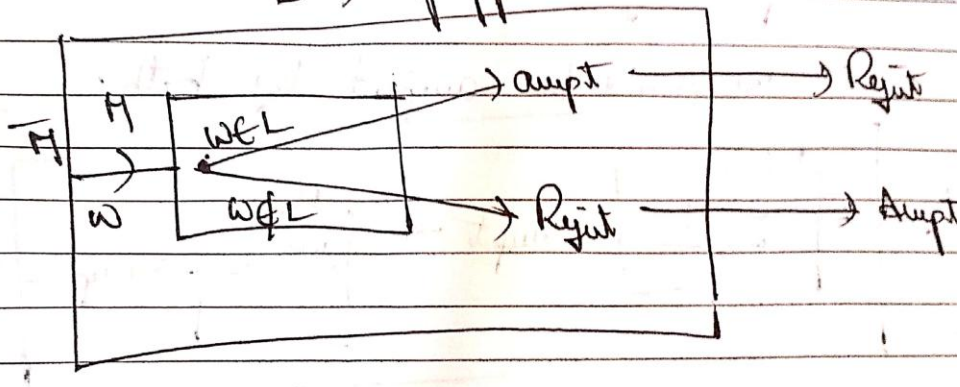
$L_1 \cap L_2$ it must be accepted by both the Lang's



(III.) The Complement of 2 Recursive Lang's also Recursive:

→ Let 'L' be a recursive Lang accepted by M_1 then L' be Complement which is also recursive

L' ⇒ opposite.

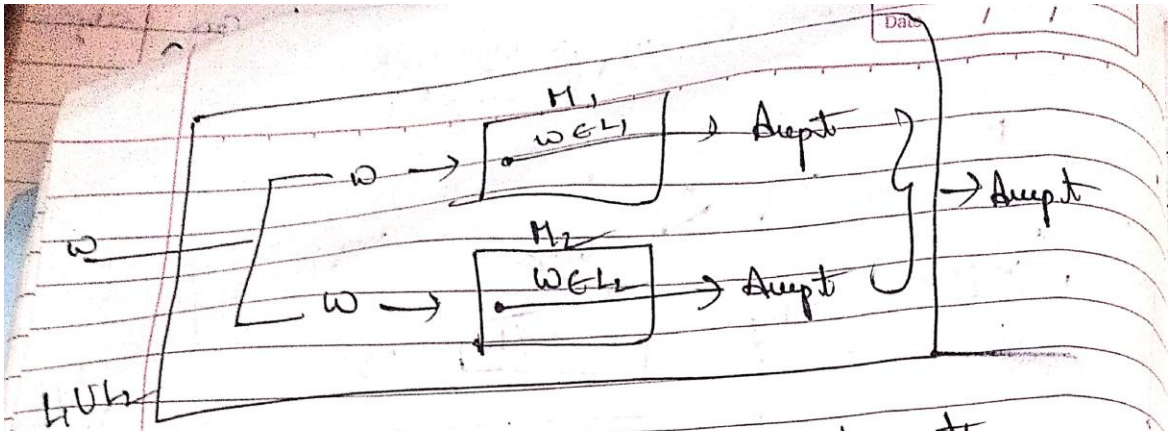


(IV.) The Union of 2 Recursively enumerable Lang's is also Recursively Enumerable

→ Let L_1 & L_2 be recursively enumerable Lan accepted by M_1 & M_2

→ If 'w' is accepted by any TM Then it is accepted.

→ If

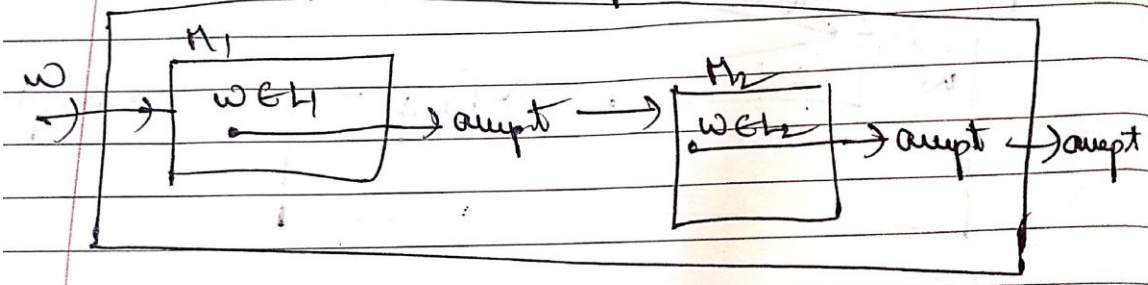


→ If any one Lang is accepted then it is accepted.

(V) The intersection of 2 Recursively enumerable Lang is also recursively enumerable.

→ Let L_1 & L_2 be recursively enumerable Lang accepted by TM, M_1 & M_2.

w should be accepted by both.



→ If both M/c's accepted then it is accepted, if one M/c Rejects also it is ~~rejected~~ rejected.

Explain about Turing Machine as a Comparator and as a Subtractor.

TM as Comparator and Subtractor

Proper Subtraction

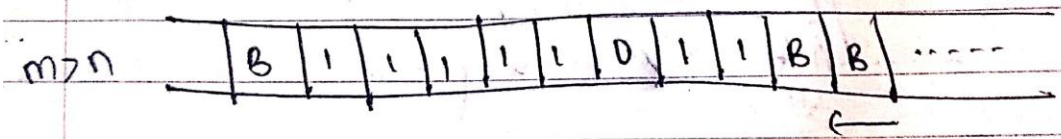
→ Fun $f(m,n) = \begin{cases} m-n, & \text{if } m>n \\ 0, & \text{if } m \le n \end{cases}$

Unary num's m=5, n=2

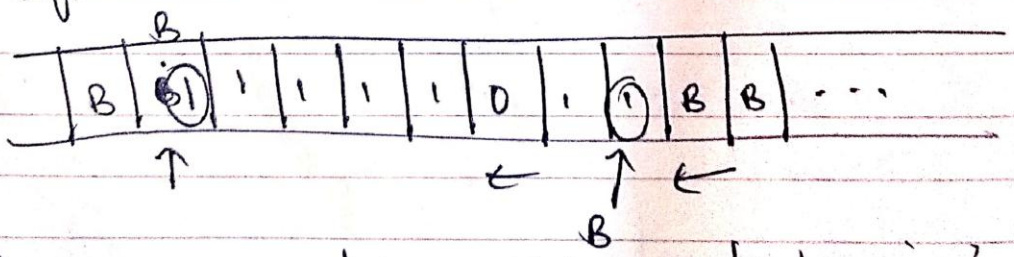
m = 5 = 1 1 1 1 1

n = 2 = 1 1

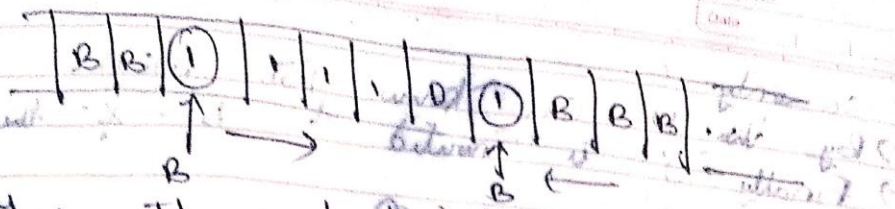
FC



→ First Convert the first '1' as B, & move forward until we get B, whenever we get B move left.

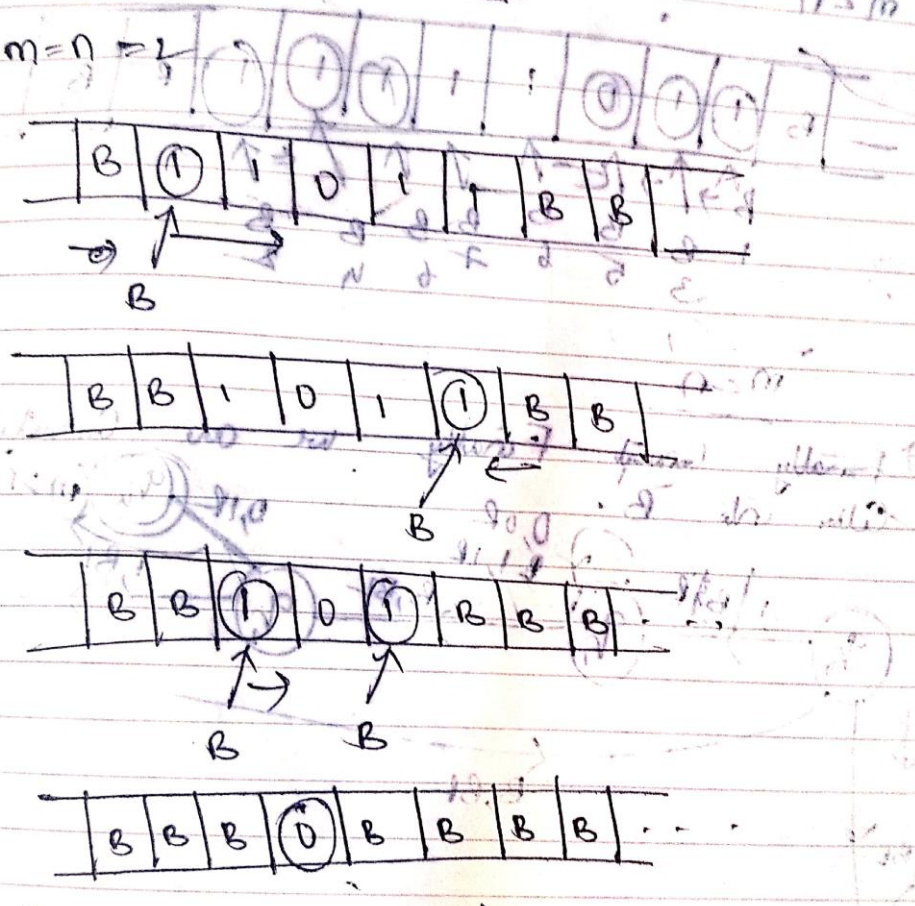


→ Again move left until we find 'B', then move right & Convert this into 'B' & move right



→ Move until we find 'B' & move left & Convert that into 'B'.

→ Again move right. until we find 'B' & move right & after '0', we don't find any '1', by this TM can know that m is Greater than n. (m>n).



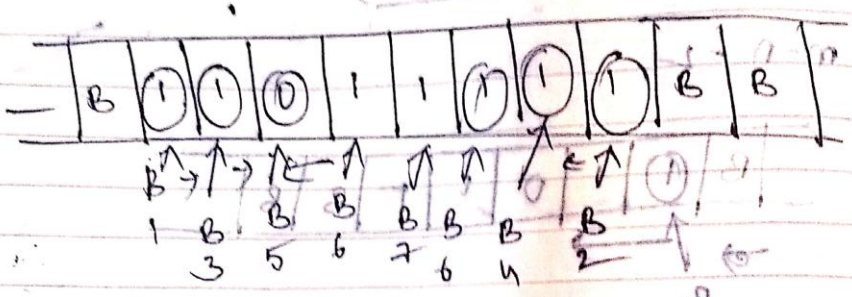
→ So here on right side we Converted 2 1's into blank & on left side we Converted two 1's in to blank.

→ By this TM will Come to Know that after 0 & if it move on left it Can

So only 0.
→ By this TM can know that m=n?
→ Finally 0 is converted to B. If the result is 0

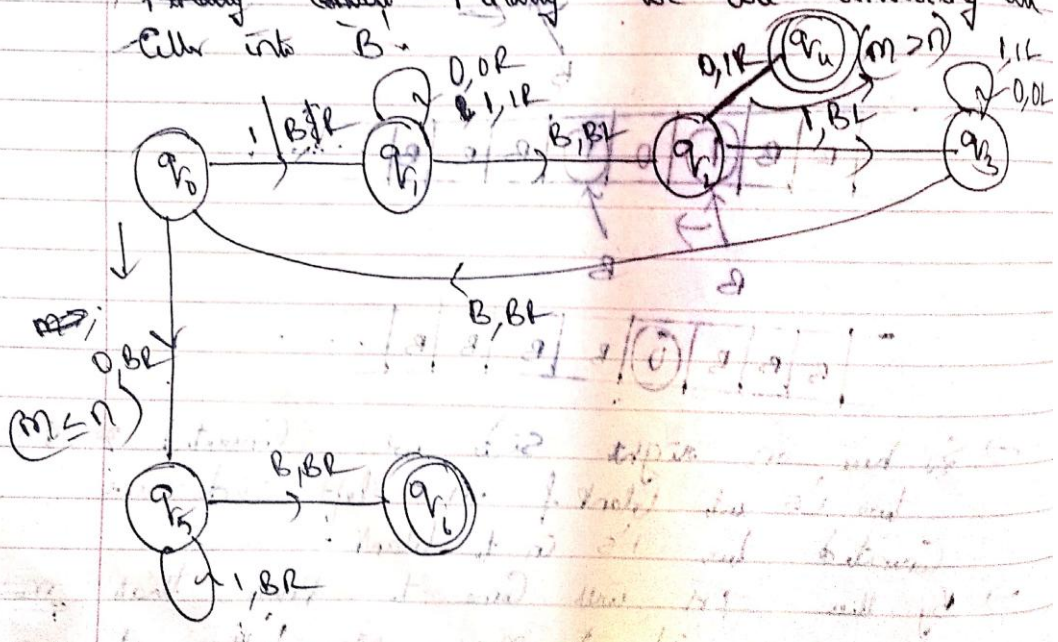
→ Also B's in equal no of 1's (Condition).

m<n



m<n

→ Finally we are converting all 1's into B.



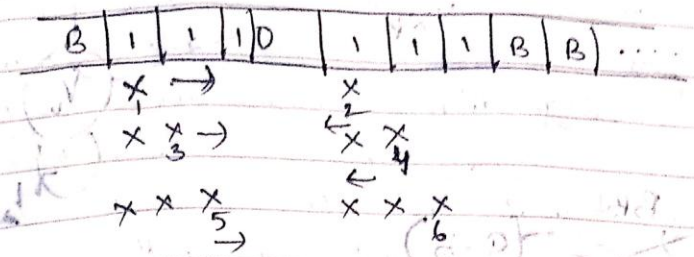
TM as a Comparator

$a = b$

$a < b$

$a > b$

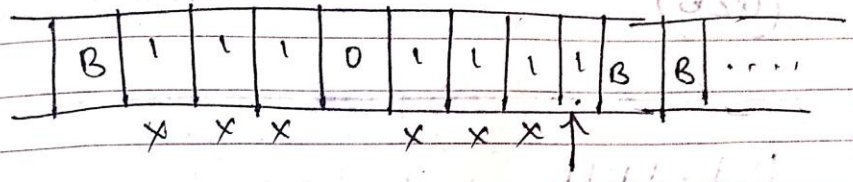
$a = b$



⇒ Whenever we found '0' left side num is one more right if found '0' so right side num is one.

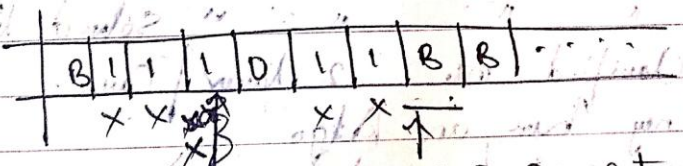
→ So, both num's are equal.

$a < b$



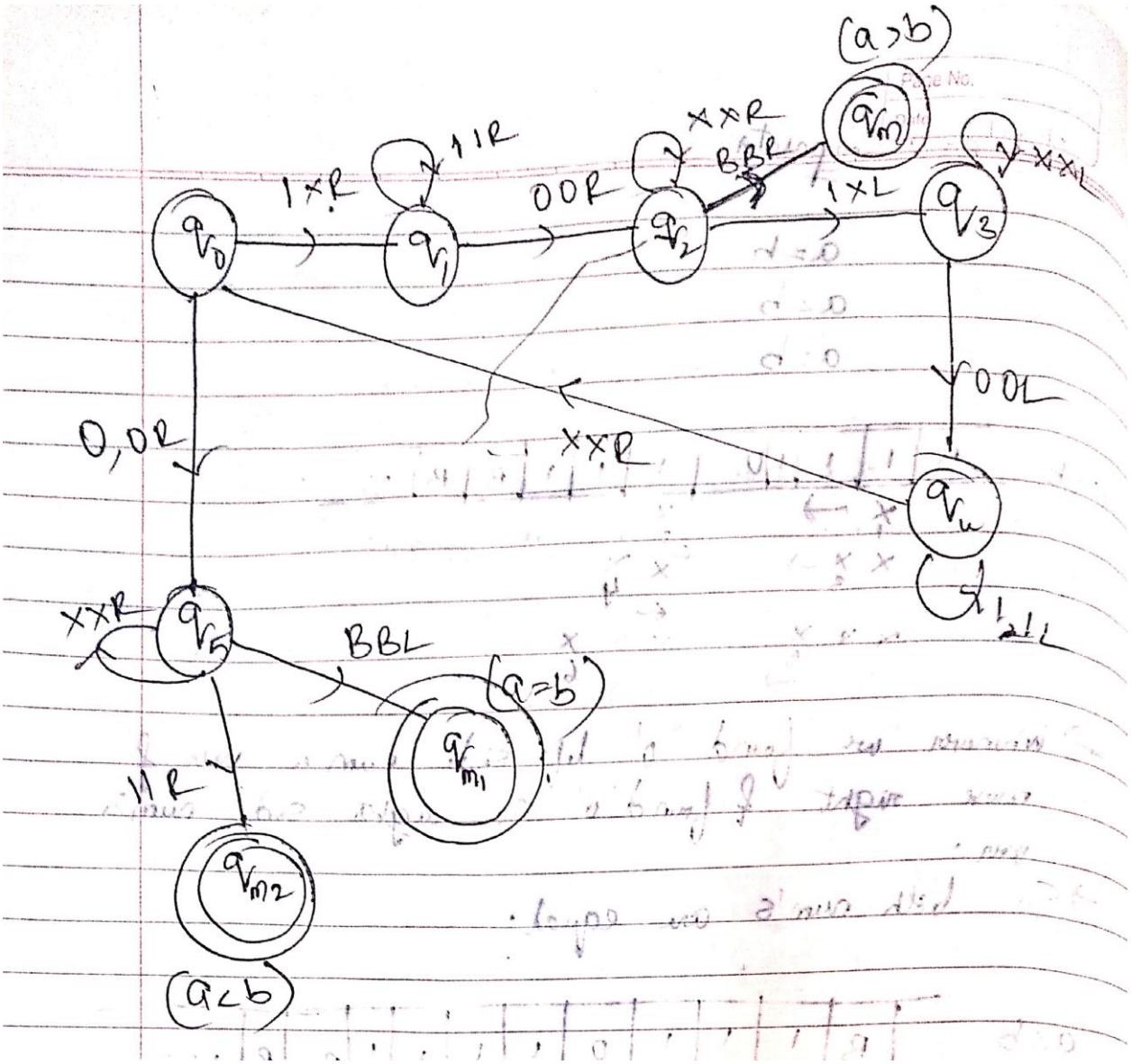
→ Apply the same logic as $a = b$, if on right hand side we will have one extra '1' by this TM will come to know that $(a < b)$.

$a > b$



$(a > b)$

There is no extra '1' on right side, by this 'TM' will Count the the no of 1's Converted into X on both sides. So, the no of 1's Converted on left side is ⇒ than right side


(a>b)
1XR
1 1R
0 0R
X X R
B B R
1XL
X X L
0 0 L
0,0R
X X R
X X R
B B L
(a=b)
1 1 R
(a<b)