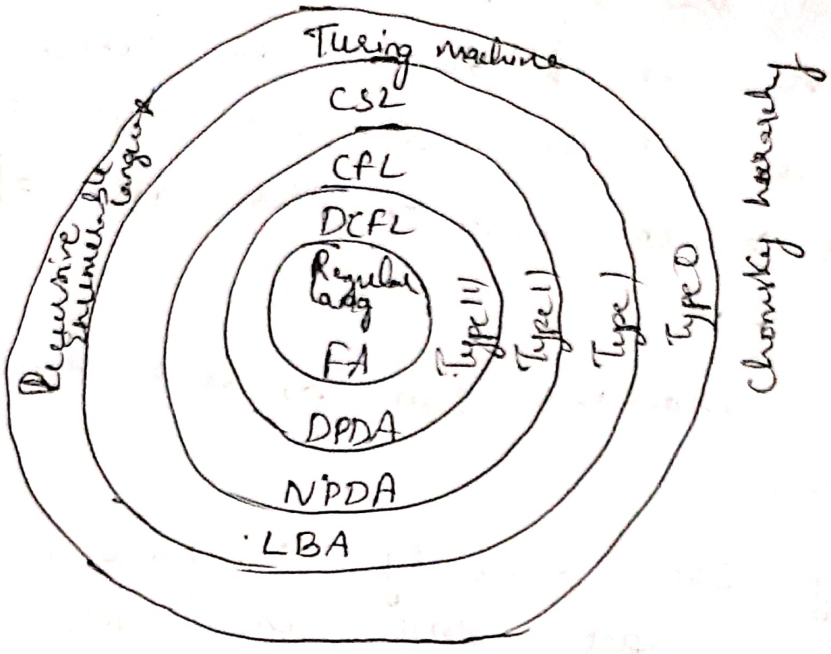
Turing Machine

Turing machine can solve every problem that is solvable, compute everything that is computable.



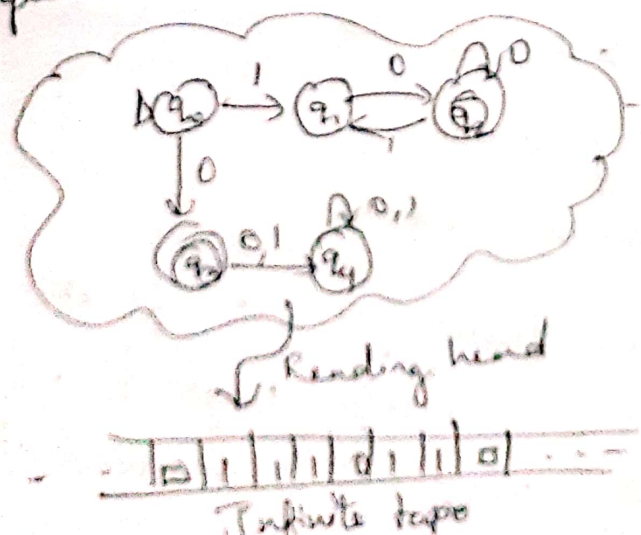
Turing machine

→ halts and accepts

→ halts and rejects

→ never halts (infinite loop) i.e Halting problem

Turing machine can give you a yes / no as output or even compute something.



At the start of a computation, the control unit is in the start state q_0, the reading head is pointing to cell S_1 on the tape. By convention, the input string S is already written on the tape. Sometimes, a special symbol such as \$ or $\lhd$ is used to indicate the starting cell and special marker such as $\rhd$ to indicate the end of the input string.

Turing machine can produce two kinds of outputs.

1. They can actually compute a result from the input string and write the result on the tape.
2. If a possible result of a computation is given as a part of the inputs, they can give us a either Yes or No.

Definition of Turing machine.

A Turing machine M has five elements

$M = (Q, \Sigma, q_0, f, \delta)$

Q - set of all states

Σ - set of symbols

q_0 - is the start state

δ : It is a function δ from $Q \times \Sigma$ to $Q \times \Sigma \times \{Left, Right\}$ i.e a mapping from the current state and the current

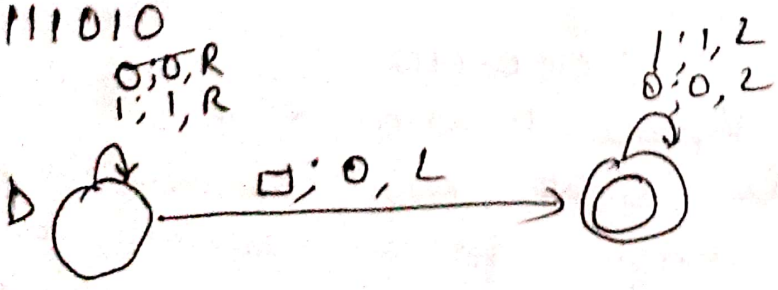
symbol under the reading head on the tape to a new state, a new symbol in the cell and a movement of the reading head by one cell to the left or to the right. In addition, a special symbol □ denotes a blank cell.

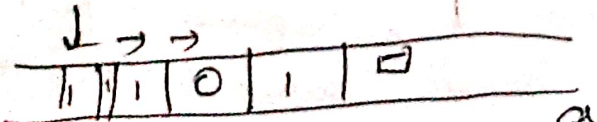
Construct a simple turing machine for doubling a binary number.

Consider the problem of doubling number i.e multiplying by 2. The input number is written on the tape in binary with the reading head at the beginning of the input. Each transition is defined by the current state of the finite control and the current cell at which the reading head is pointing.

Algorithm

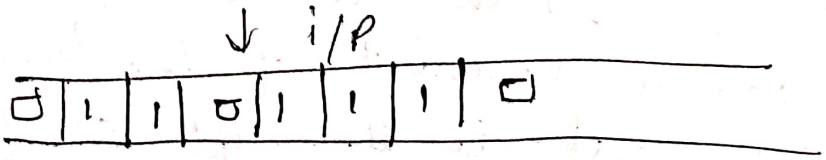
The machine starts in state q_0 and move to the end of the number and append 0. For example if the input is 11101 then output is 111010



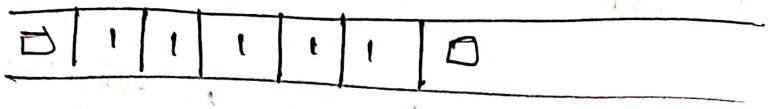


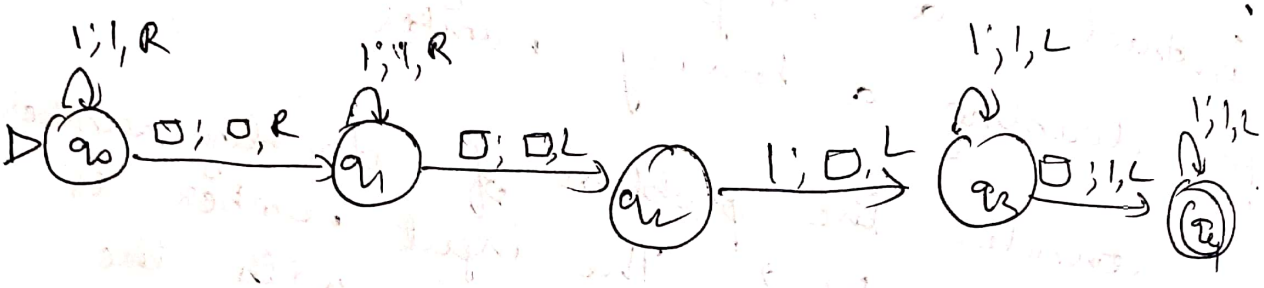
Keep moving towards right once if you find □ replace □ with 0

Turing machine for adding unary number

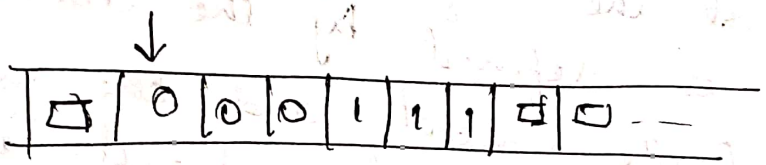


o/p

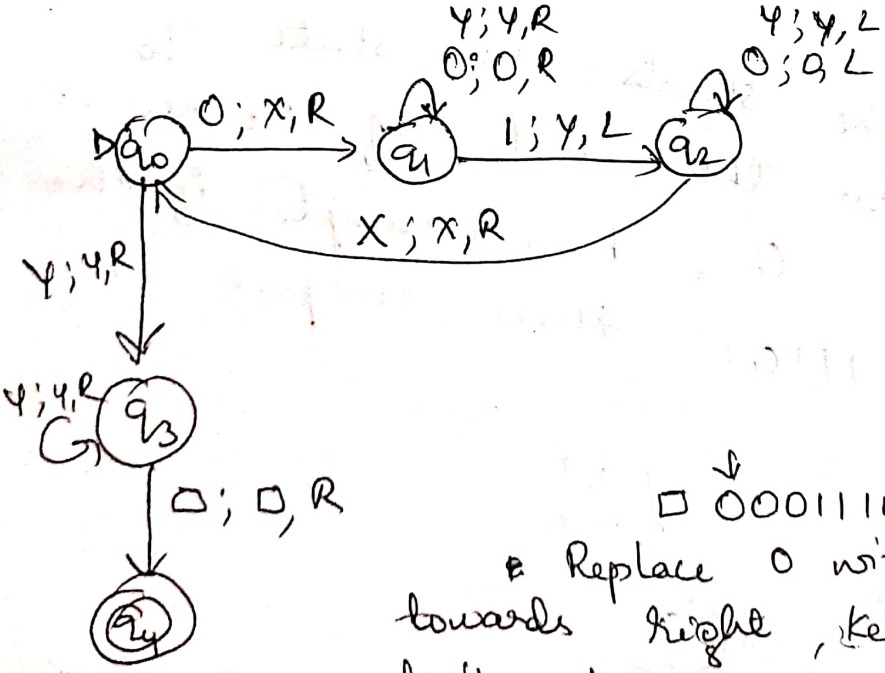




Turing machine for 0^n1^n



For every 0, we have a 1. We replace a 0 with X & 1's with Y.



* Replace 0 with X & move towards right, keep reading 0 & don't change for 0, retain 0 as it is.

If you find a first 1, replace first 1 with Y & the reading head towards with left

for second 0, while moving towards left, you will get 0 keep 0 as it is & if you find X

i.e ↓

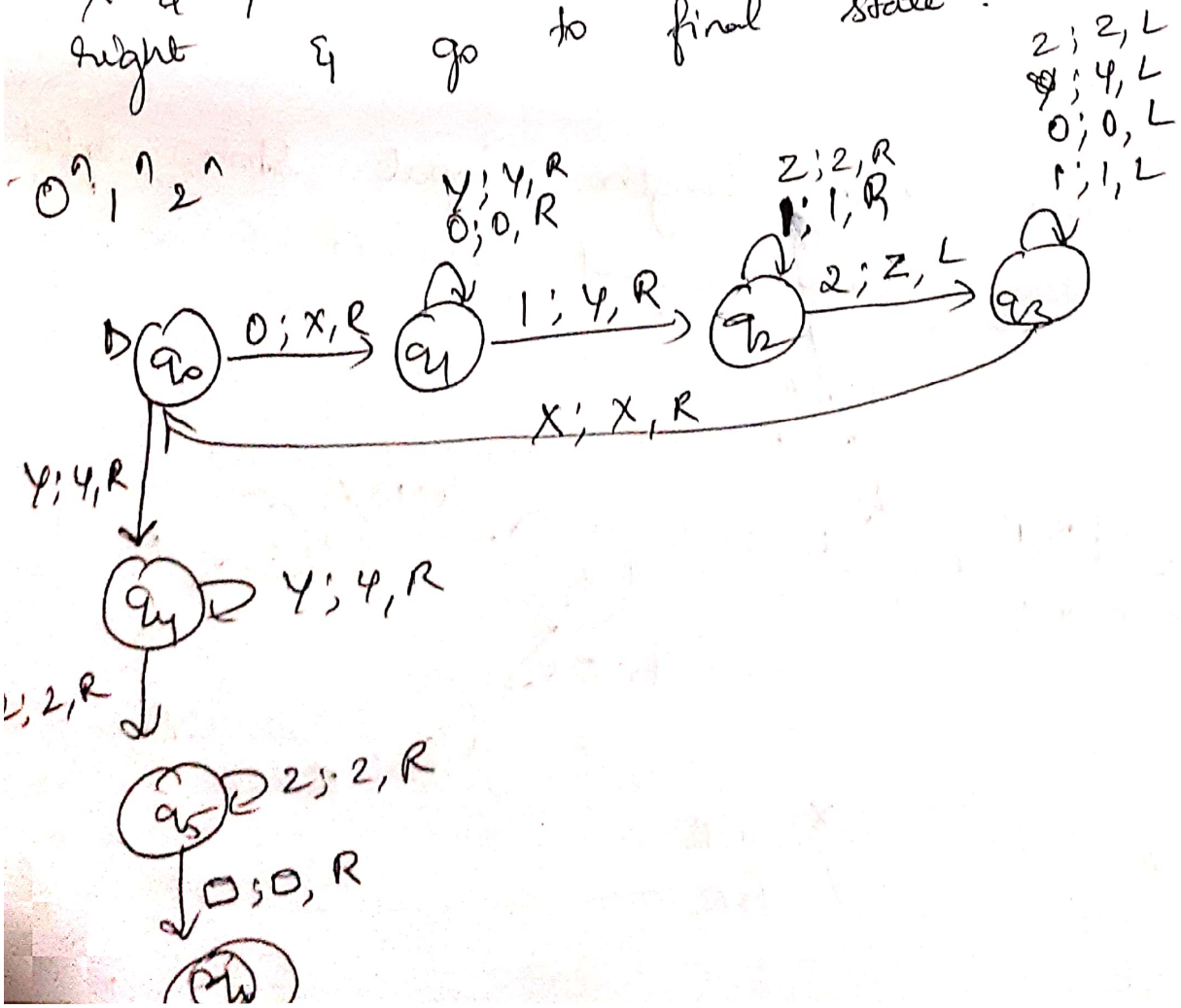
1. | □ | X | 0 | 0 | Y | 1 | 1 | □ |

2. | □ | X | X | 0 | Y | 1 | 1 | □ |

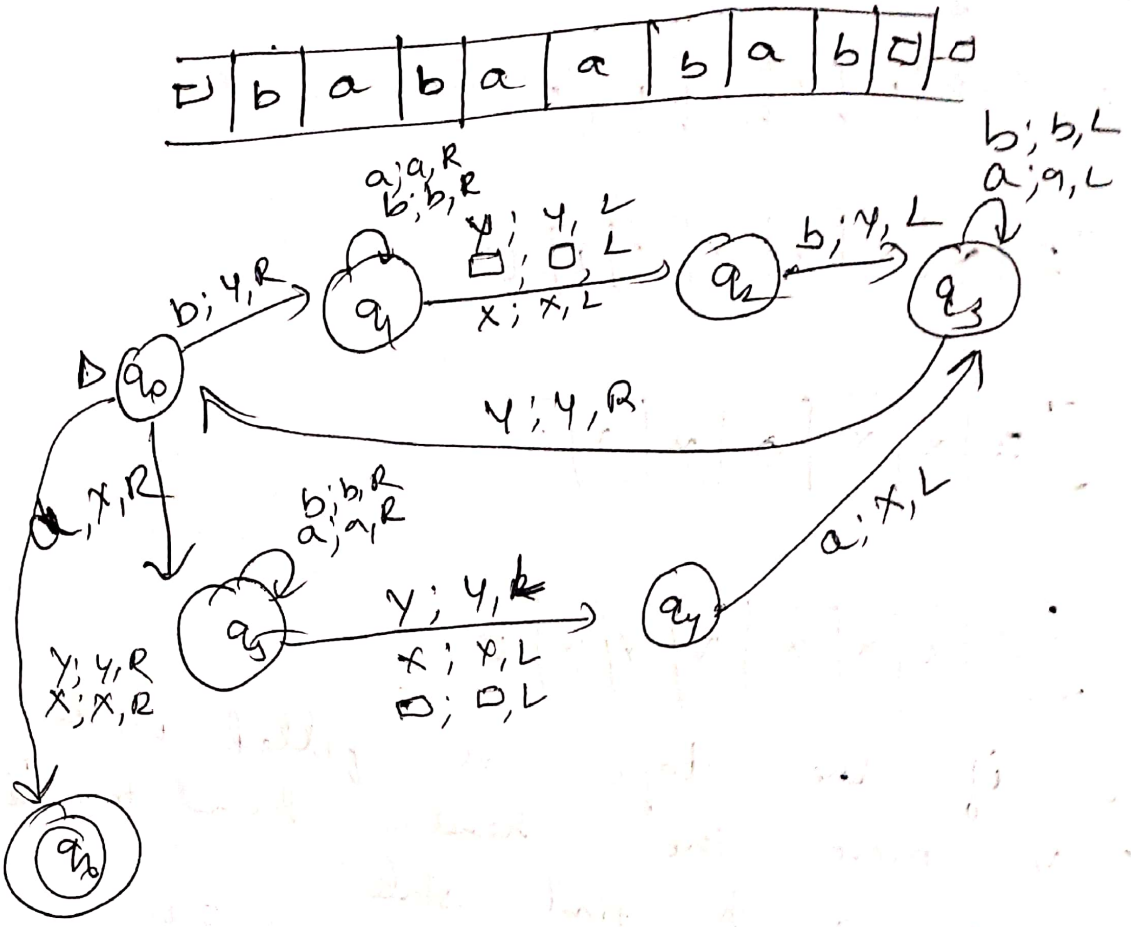
| □ | X | X | 0 | Y | Y | 1 |

| □ | X | X | X | Y | Y | Y | □ |

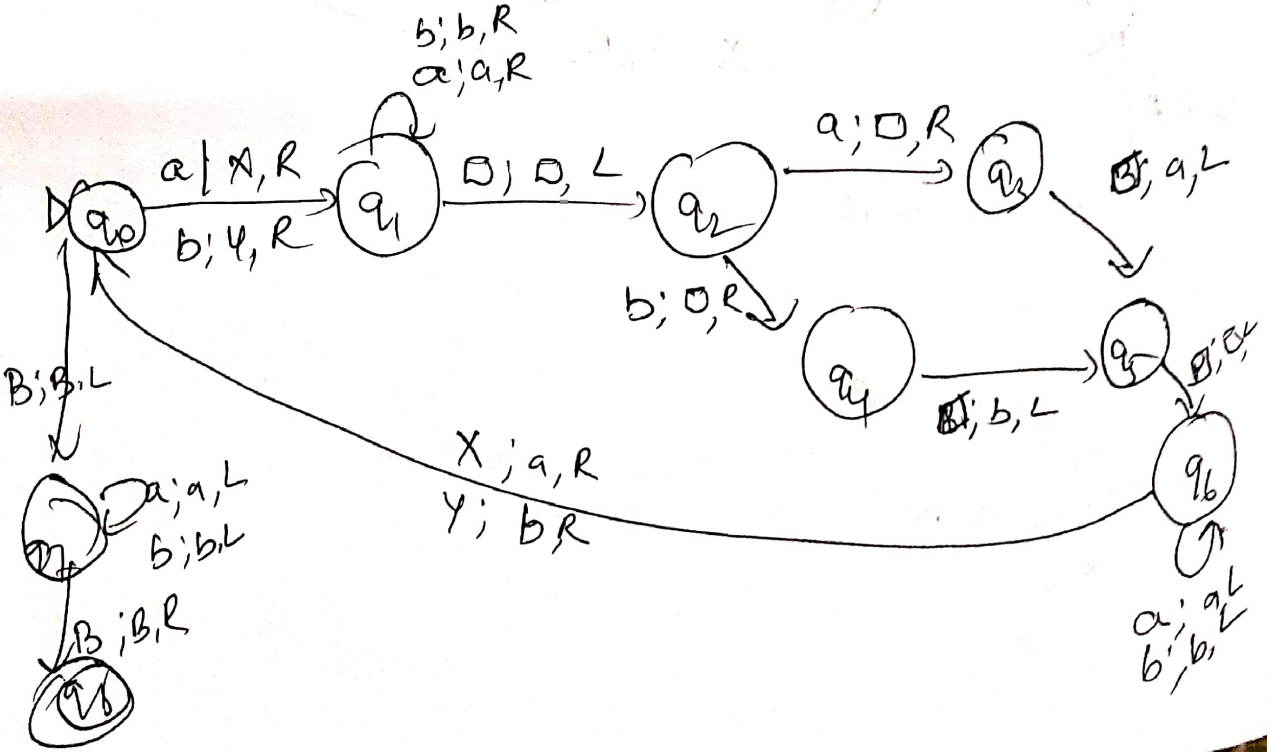
Once if the tape is filled with X & Y move the reading head towards right & go to final state.



Turing machine for WW^R over $\Sigma = \{a, b\}$

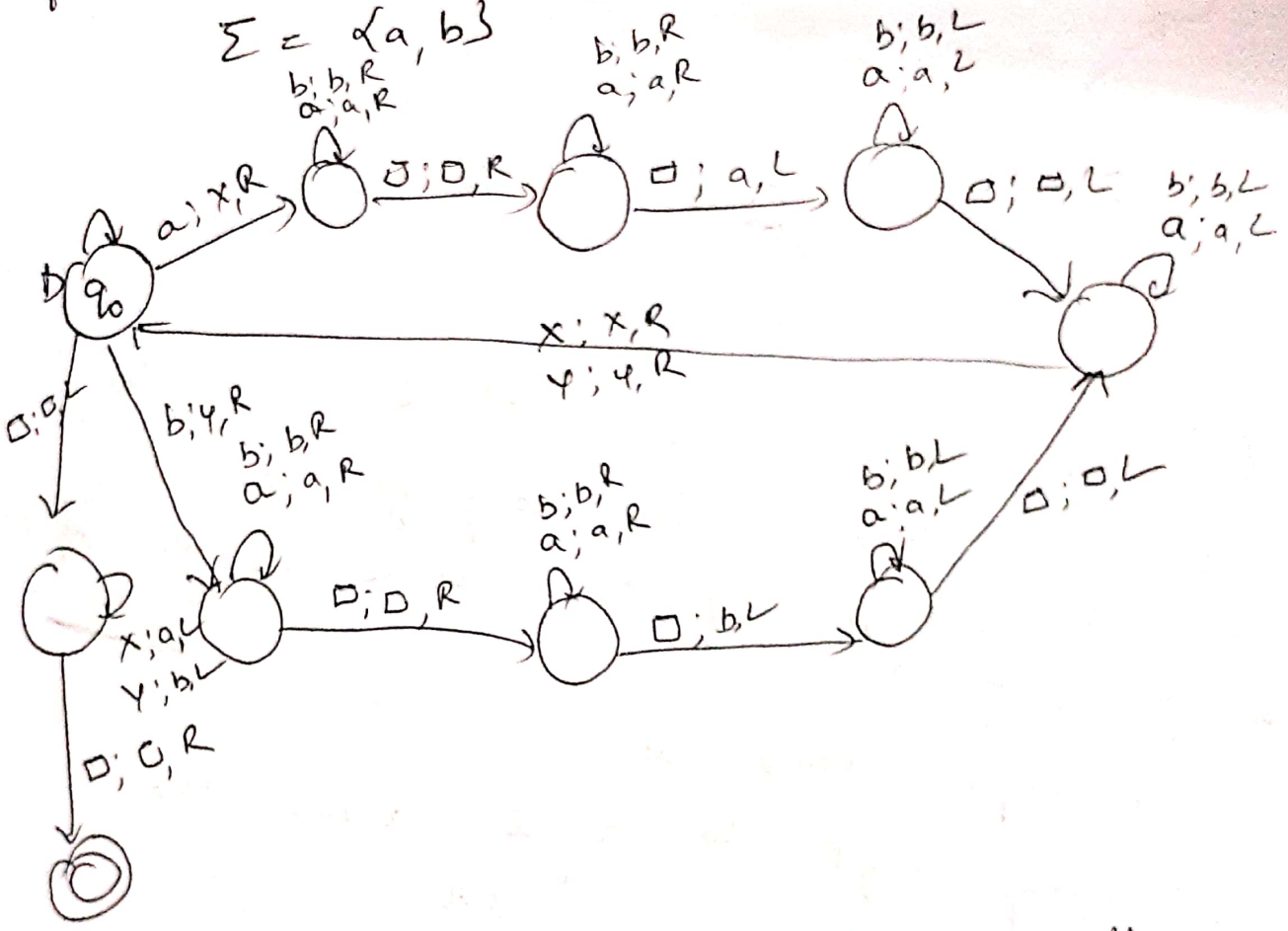


Given an even length input string, split the string into two equal halves.

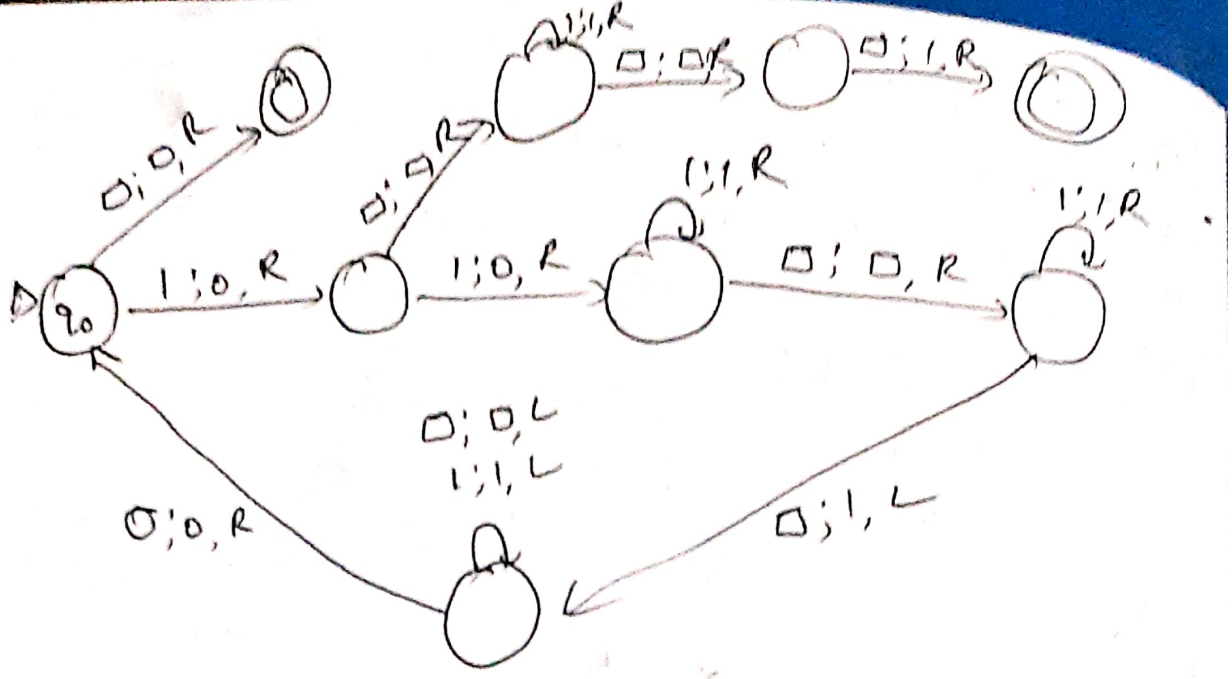


Copy a given string, that is the tape should contain a second copy of the input string separated by a single blank cell from the given string.

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



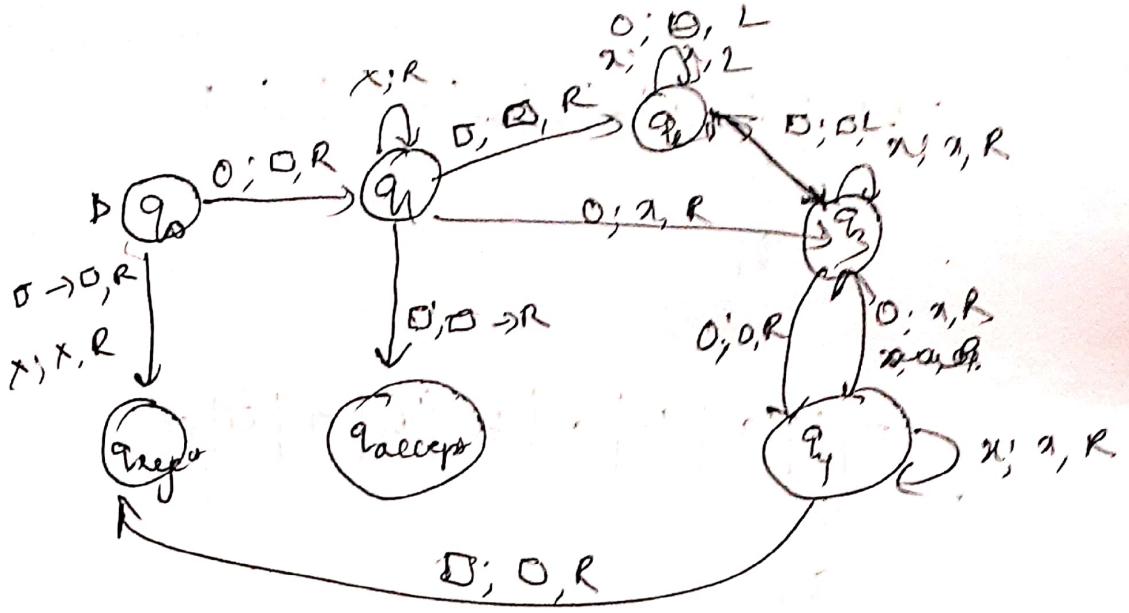
Divide a given number by two in the unary number system. The quotient and the remainder (separated by blank) should be written on the tape when the machine halts. For example given 1111 on the tape, the output should be 11 i.e $4/2 = 2$ & there is no remainder. As another example, given 1111111, the output should be 111 1 i.e $7/2 = 3$ and the remainder is 1



Turing machine M that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0's whose length is a power of 2.

On input string w

1. Sweep left to right across the tape, crossing off every other 0
2. If in stage 1 the tape contained a single 0, accept
3. If in a stage 1 the tape contained more than a single 0 and the number of 0's was odd, reject
4. Return the head to the left hand end of the tape
5. Go to stage 1



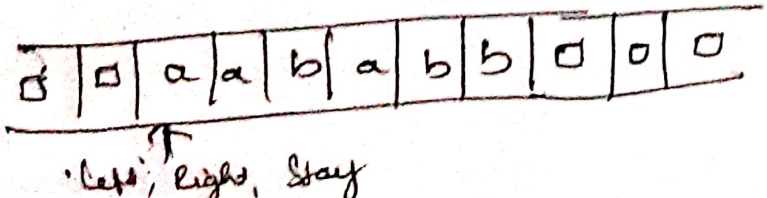
The Church Turing Thesis

Alan Turing and Alonzo Church independently showed that anything that can be computed can be computed by a Turing machine. This statement has no proof and is not a theorem. However no one so far has been able to find a computing problem that Turing machines cannot compute but some other machine, mechanism or formalism can. Hence this claim is known famously as The Church Turing Thesis.

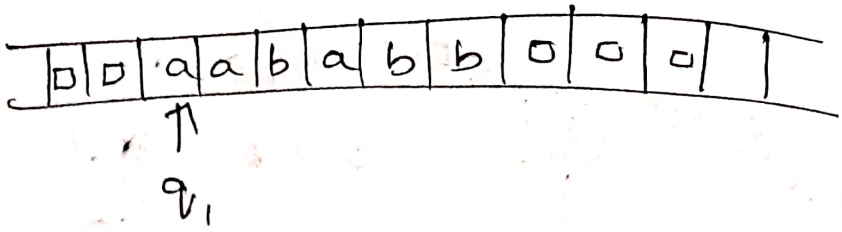
Variations of Turing machines

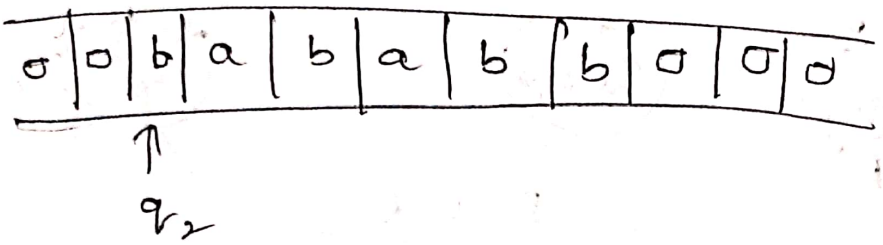
1. Turing machines with stay option

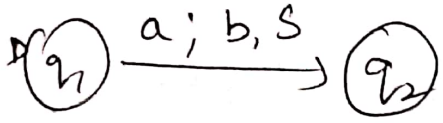
The head can stay in the same position



Ex:

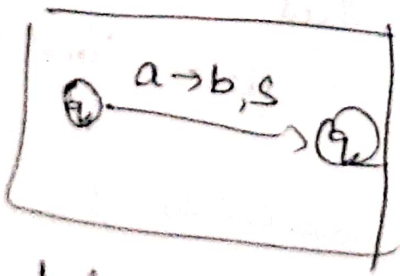




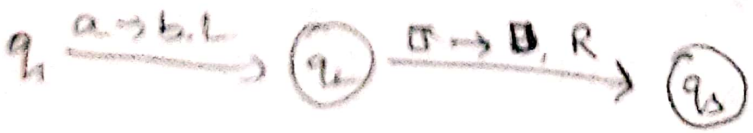


Stay option turing machine is still equivalent to standard turing machine because staying in the same cell is the same as moving once to the right (or left) and moving back left (or right) in additional dummy transition

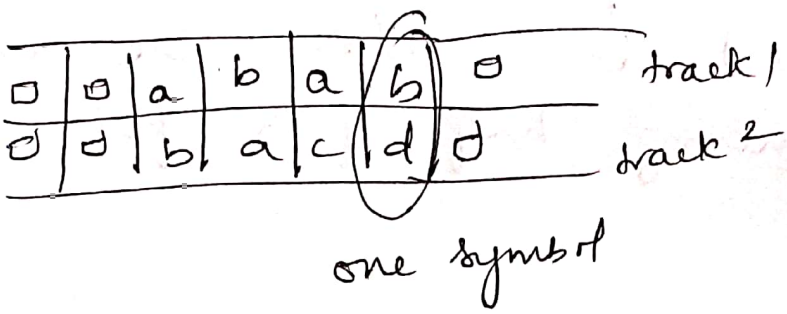
example Stay option machine

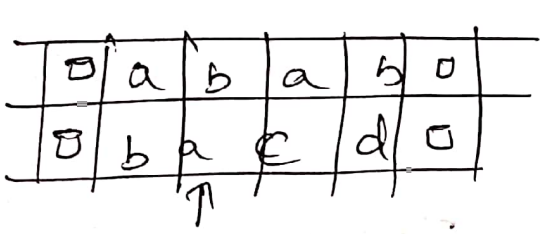


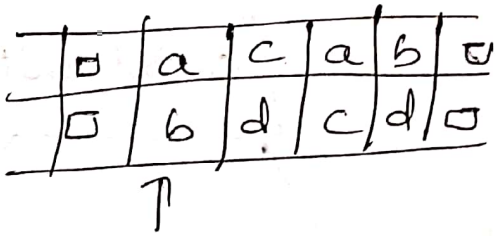
Simulation in Standard turing machine

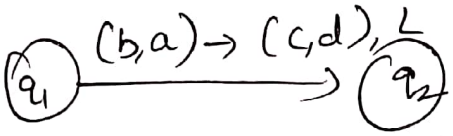


2. Multi-Track Tape









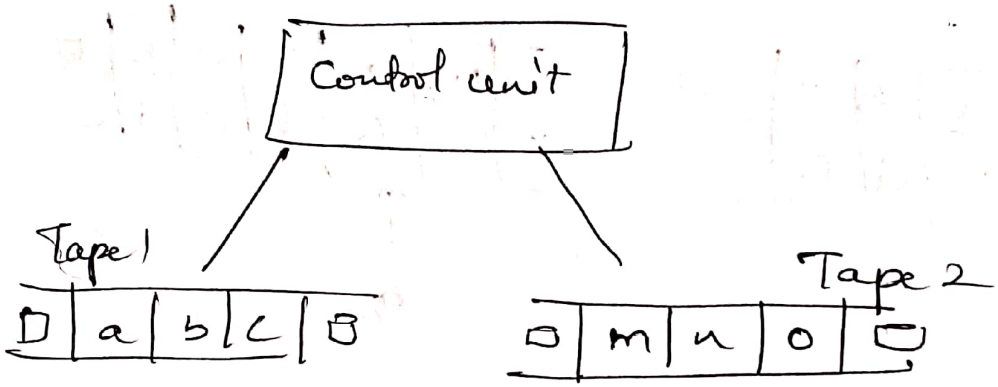
This is still equivalent to the standard machine because each such sequence of four 2 consecutive cells on the standard tape become equivalent to a single cell on the 2 track tape.

3. Semi-infinite tape

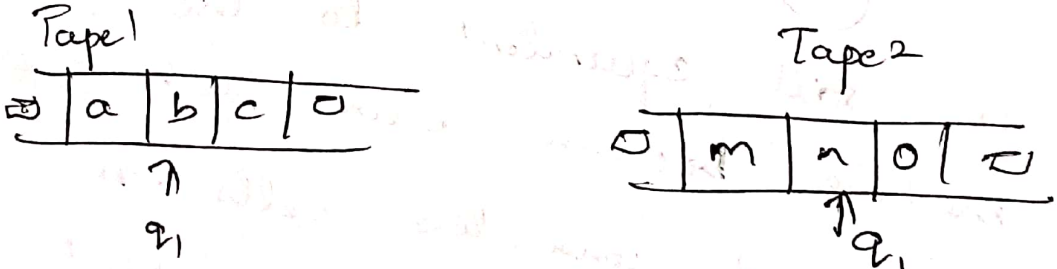
What if the tape is infinitely long only on one side instead of at both ends? This too can be shown to be equivalent to the standard turing machine. Using the idea of multi track m/c

if we split the semi infinite tape into a two track tape, the second track can be used to simulate the contents of the missing half of a full infinite tape.

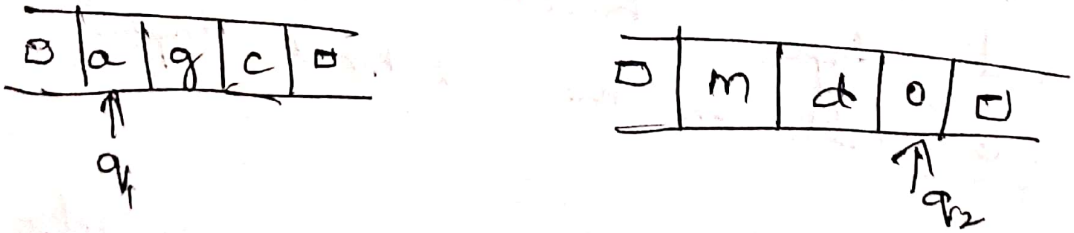
4. Multi tape Turing machines

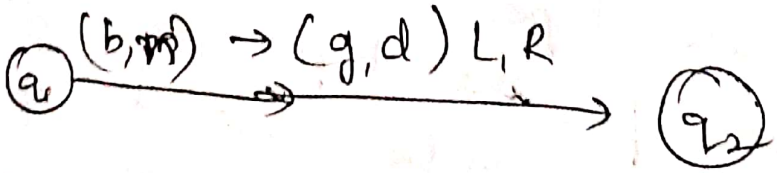


Time 1



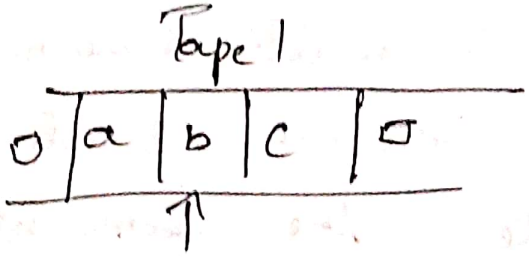
Time 2

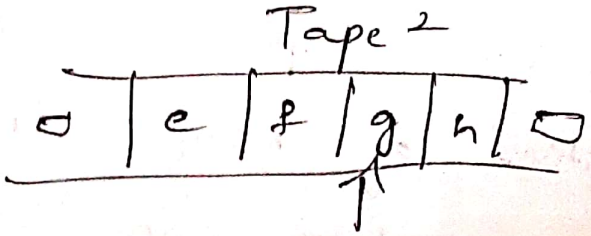




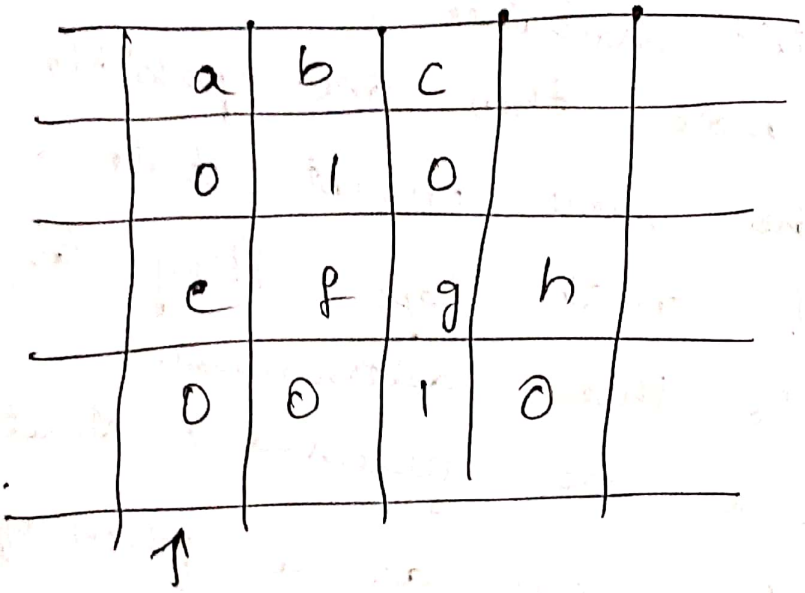
Standard Turing machine simulate multitape machines

Multitape machine

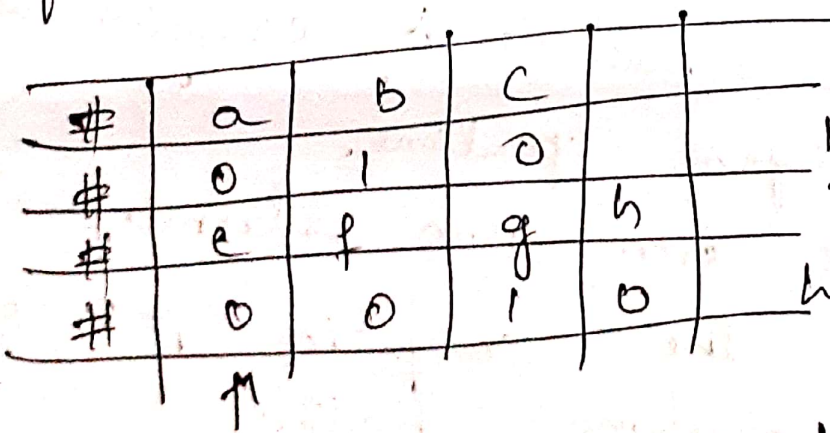




Standard machine with four track tape



Reference point



Repeat for each state transition
- Return to reference point
- find current symbol in Tape 1
- find current symbol in Tape 2
- Make transition.

5. Multi dimensional tape machine: Even if the tape is 2, 3 or multi dimensional instead of being linear sequence of cells, the machine can compute the same set of functions

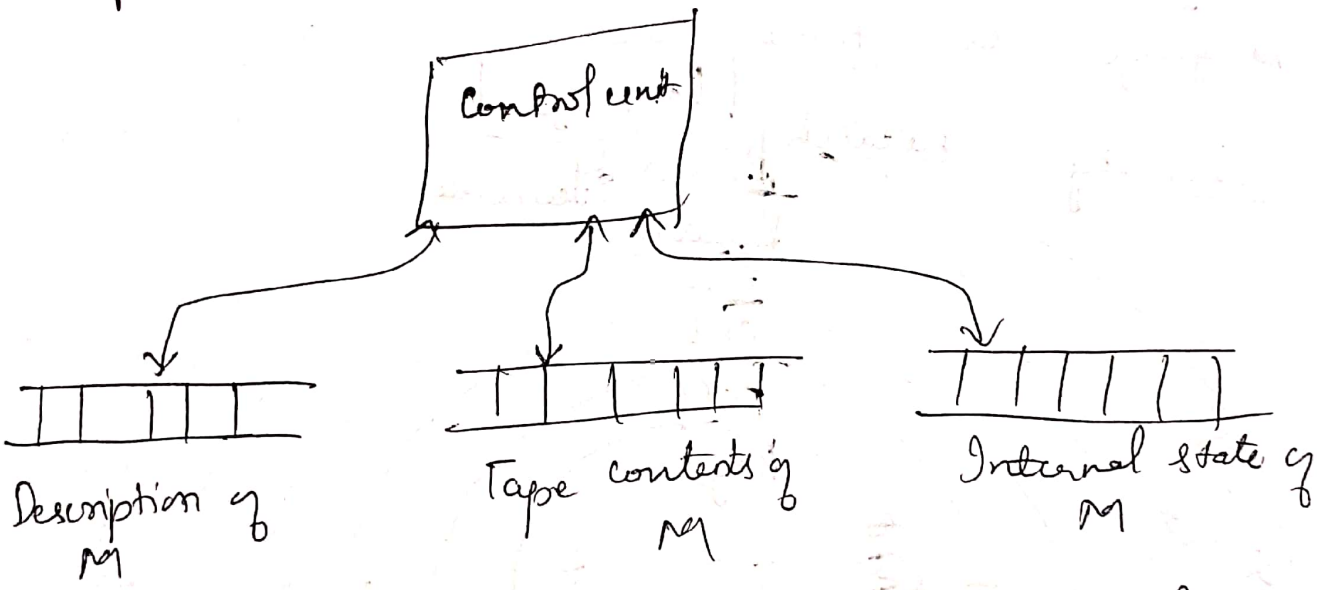
6. Non deterministic machines.

It has more than one possible transition for the same current state and the same symbol on the tape. They too are equivalent to their deterministic counterparts because the guesswork that they do can be seen as a form of backtracking search which can be implemented in a deterministic machine by carrying out appropriate book keeping operations.

Showing that even non determinism does not increase the computing power of standard Turing machines further confirms the Church Turing thesis that anything that can be computed by any kind of computing machine can also be computed by the standard Turing machine.

A Universal Turing Machine

A universal turing machine is a reprogrammable turing machine which given as input the description of a TM M and a string w, can simulate the computation of M on w



A universal turing machine has the structure of a multitape machine.

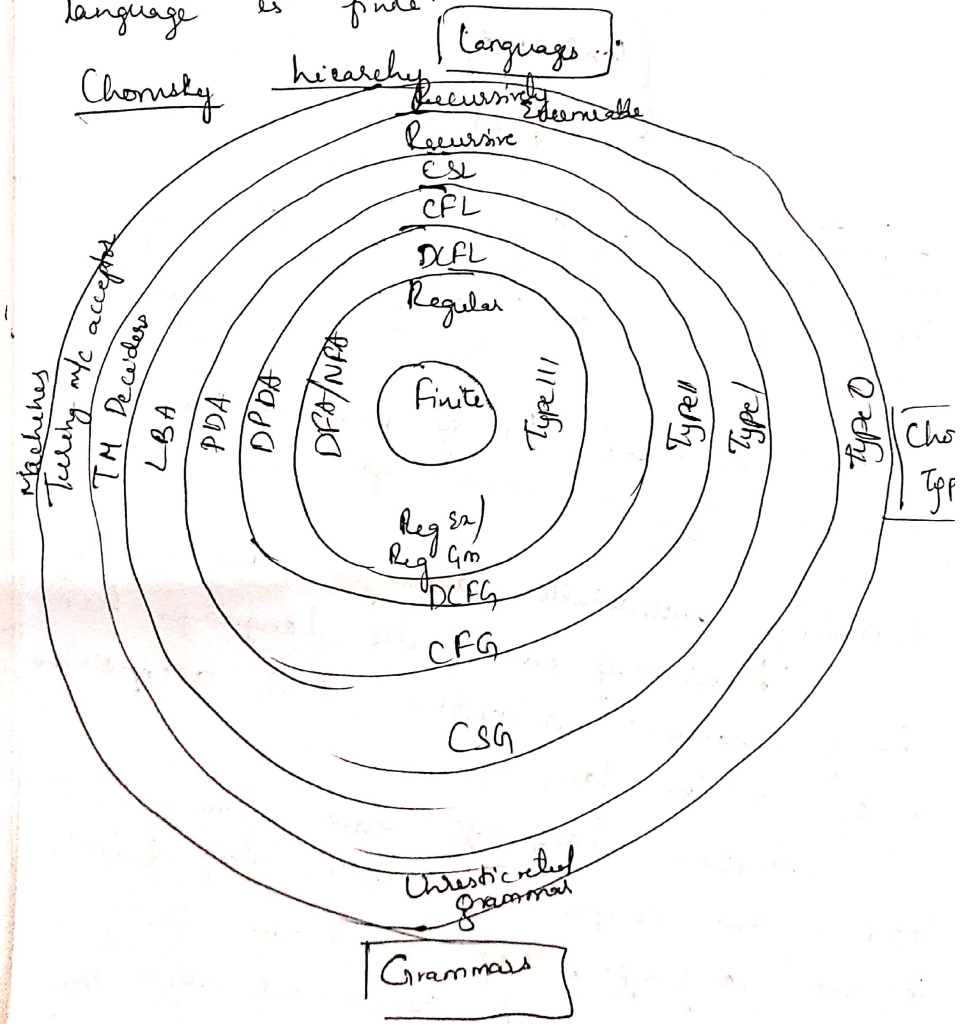
Recursively enumerable languages & Recursive lang

Recursively enumerable languages have Turing machine acceptors that accept ag string in the language.

Recursive language have Turing machine & deciders that compute the membership function for the language. They accept all strings in the language and reject all the strings that are not in the language.

Recursively enumerable languages is the largest class of formal languages that are computable.

Finite languages are a proper subset of regular languages since every finite language is regular but not every regular language is finite.



The Post correspondence problem

This is one of the simple undecidable problem concerning simple manipulation of strings. We can describe this problem easily as a type of puzzle. We begin with a collection of dominos each containing two strings, one on each side. An individual domino looks like

$$\left[\frac{a}{ab}\right]$$

and collection of dominos looks like

$$\left\{\left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right]\right\}$$

Chomsky Types

The task is to make a list of these dominos (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the string of symbols on the bottom. This list is called match. For example

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{abc}{c}\right]$$

Reading off the top string we get abcaabc, which is the same as reading off the bottom.

For some collections of dominos, finding a match may not be possible. For

example, the collection

$$\left\{\left[\frac{abc}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{acc}{ba}\right]\right\}$$

cannot contain a match because every top string is longer than the corresponding bottom string.

Consider $W = \{bb, baa, bbb\}$

$V = \{bbb, aab, bb\}$

In this case, if we concatenate the strings in the same order in which they are given to us we get

$$\left\{\left[\frac{bb}{bbb}\right]\left[\frac{baa}{aab}\right]\left[\frac{bbb}{bb}\right]\right\}$$

Look at one more example

$W = \{a, aa, b\}$

$V = \{aa, ab, bb\}$

$\left[\frac{a}{aa}\right]\left[\frac{aa}{ab}\right]\left[\frac{b}{bb}\right]\left[\frac{b}{bb}\right]$, the bottom string having too many b's at the end.

The other way

$\left[\frac{b}{bb}\right]\left[\frac{b}{bb}\right]\left[\frac{b}{bb}\right]$ - - - this also leads to too many b's at the bottom which leads to infinite loop. There appears

to be no solution at all to this instance of the PCP.

Find the PCP solution in the following cases?

1. $X = \{a, abaaa, ab\}$ $Y = \{aaa, ab, b\}$

Soln 2 1 1 3

$$\left[\frac{abaaa}{ab}\right], \left[\frac{a}{aaa}\right], \left[\frac{a}{aaa}\right], \left[\frac{ab}{b}\right]$$

2. $X = \{ab, baa, abb\}$ $Y = \{aba, bb, baa\}$

No possible solution.

Show that if a language L and its complement are recursively enumerable, then both are actually recursive.

Soln: L is recursively enumerable. It has an acceptor say M.

L is complement L', is recursively enumerable. It has an acceptor say M'.

It is required to make decider for both. Given a string w new automaton would test its membership in both M and M' simultaneously. Sooner or later one of the machine halts accepting it. Then

this new automaton stops as soon as one of them halts.

Case 1: If M halts accepting it, w belongs to L.

If M' halts accepting it, w does not belong to L.

So we have a membership for M. So membership is decidable for L. L is recursive

Case 2: If M' halts accepting it, w belongs to L'.

If M halts accepting it, w does not belong to L'.

So we have a membership for M'. So membership is decidable for L'. L' is recursive.

Show that not all languages are recursively enumerable by Cantor diagonalization

Soln: Assume that every language is recursively enumerable. Therefore we can make a Turing machine acceptor. Every $L(TM)$ is subset of Σ^*. The number of language

is equal to the number of elements in the power set of Σ^* i.e 2^{Σ^*}

Let language be numbered as $L_1 L_2 \ldots L_i$

Let us enumerate all possible strings over Σ^* as $S_1 S_2 \ldots S_j$. Let us make a matrix with strings as columns and languages as rows. The entry at i^{th} row and j^{th} column indicates whether S_j belongs to L_i. We will assume that every language is recursively enumerable. We try to construct a language which cannot be in this table.

Consider the diagonal language L_{diag}. Make the complement of this language L'_{diag}. If L_i contains S_i, then L'_{diag} does not contain it. If L_i does not contain S_i, L'_{diag} does contain it. So we end up with a contradiction. So L'_{diag} cannot be any of the rows of the new L_{diag} this table. Therefore not every language is recursively enumerable.

Strings in Σ^*

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8			
L_1	1	0	0	1	0	0	1	0	0	1	
L_2	0	0	1	0	1	0	0	1	1	1	
L_3	1	1	1	0	0	1	0	1	0	1	
L_4	1	0	1	0	1	0	1	0	0	1	
L_5	0	1	1	1	0	1	0	1	1	1	
L_6	0	0	0	1	1	1	0	0	1	1	
L_7	0	1	0	1	1	0	0	1	0	1	
L_8	0	1	1	0	1	0	1	0	0	1	
L_9	1	0	0	1	0	0	0	0	1	0	

Diagonalization $L_{diag} = \{S_1, S_3, S_6, S_9 \ldots\}$