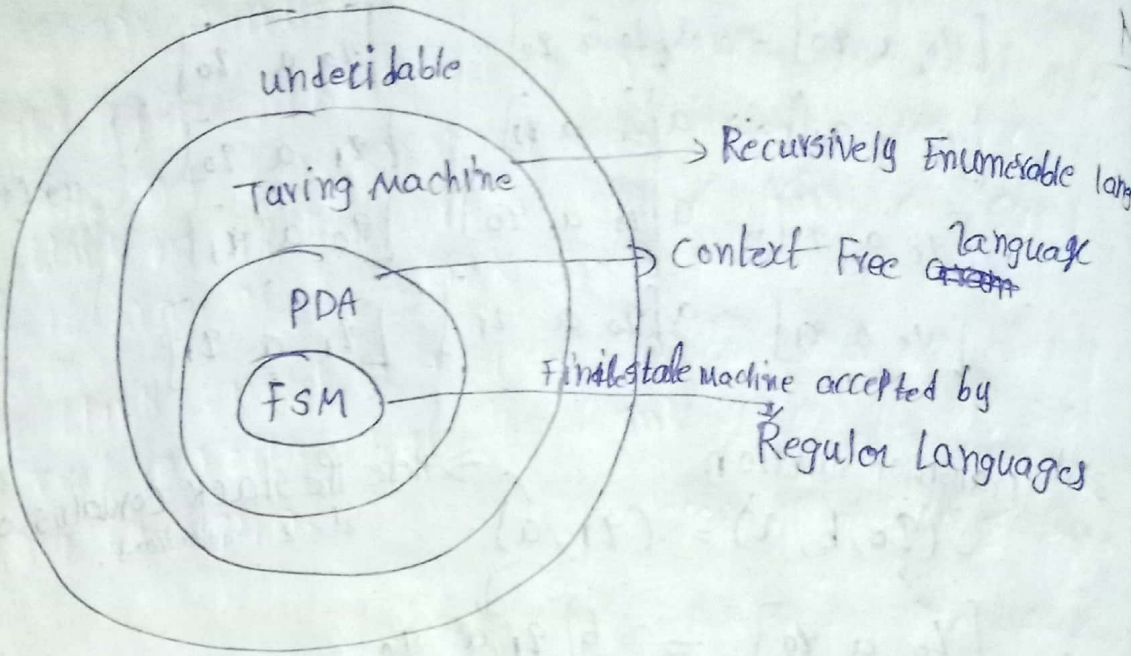
4. Turing Machine



* The Turing machine more powerful than finite state machines and also more powerful than the Pushdown Automata

* All the class of languages that is accepted by Turing machines is known Recursively Enumerable languages

No See what were the data structures that we have?

1) Final state Machine
2) Push down Automata

⟹ lets see what is the kind of data structure that we will have in Turing machines. so if you remember in case of our finite state machines which accepted the regular languages the data structure that we had was input string

FSM: Input string [a|a|a|b|b]

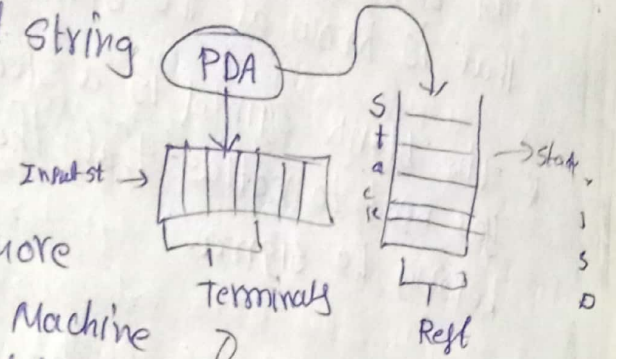
we have a control which can move only to one direction and that is forward direction

* so we saw the finite state Machines and we saw that they had a very limited Power and also a very limited memory.

* the class of language is accepted by finite state machines under regular languages.

* comming to Pushdown Automata

PDA : → The input string
→ A stack



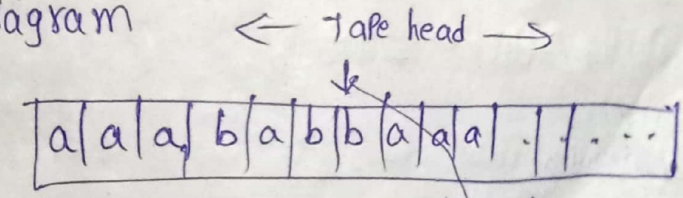
* pushdown Automat was more powerful than finite state Machine and class of languages accepted by context free language as compared to regular language.

diagram which represent input string, stack which added more power to PDA. making it different from finite state machines.

* Now Turing Machine

Turing Machine the data structure that we have is something known as a tape

* so a tape is something that looks like this as it is show in this diagram



* the tape is a sequence of infinite symbols over here as we have something known as tape head

* so the tape head is positioned on the symbol which the current control is present

* Here tape head move 1 left (or) one step to the right depending on the type of computating that we have.

* Here we have input symbols and as I told you this is an infinite sequence.

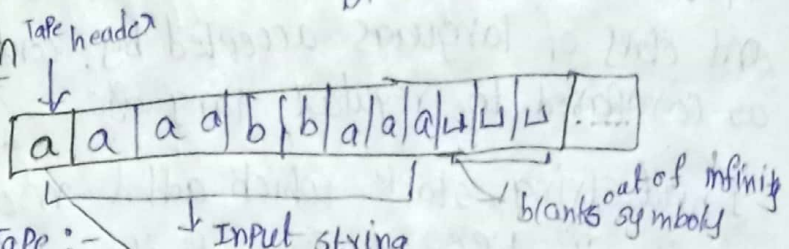
* So the input symbols that we have they are filled into these cells and then remaining cells instead of leaving them just blank we fill them with a special kind of symbol that is know as the blank symbol(␣)

* The blank symbol is a special symbol and something you need to notice that the blank symbol it does not belong to sigma.

Tape Alphabets; $\Sigma = \{a, b\}$

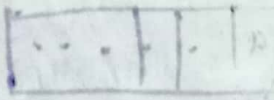
Blank ␣ symbol. $␣ \notin \Sigma$

Initial configuration

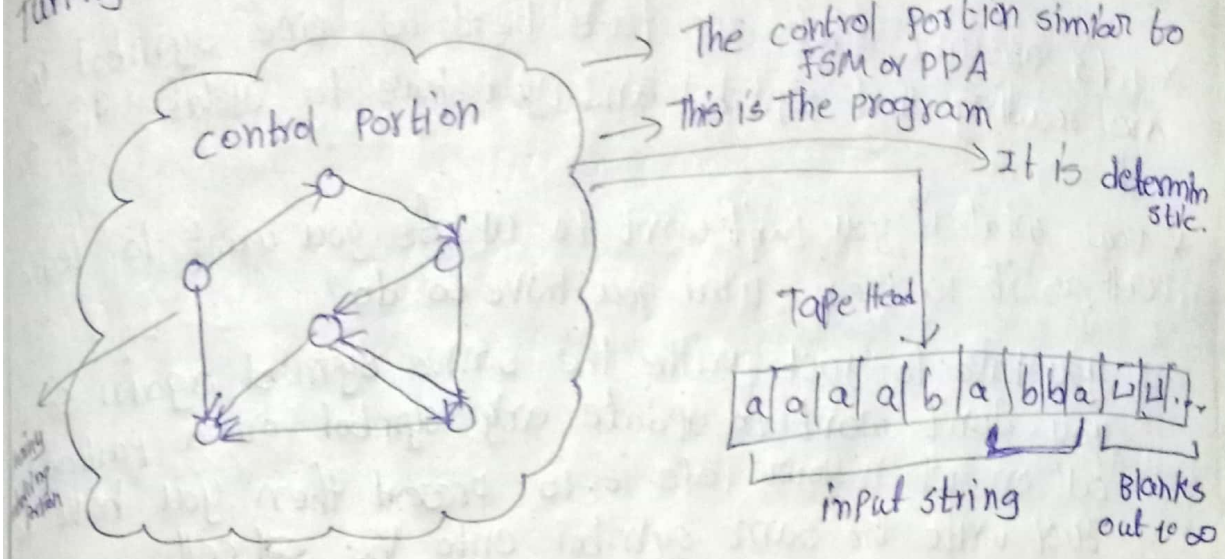


Operations on the Tape:-

* Read / scan symbol below the Tape Head (a)
* update / write a symbol below the Tape Head → overwritten with other symbol like 'B'.
* Move the Tape Head one step LEFT
* Move the Tape Head one step RIGHT.



Here a diagram represents or shows how a Turing Machine actually looks



* control portion : the control portion similar to the finite state machines over a pushdown automata
* this is the portion that is controlling your Turing machine so this is the program or the control portion

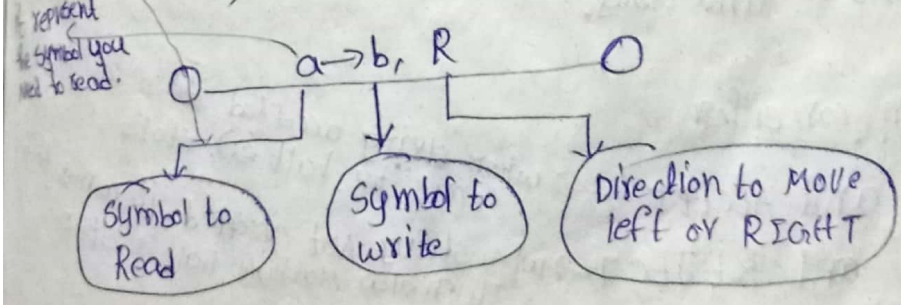
Rules of operation - 1

We have 2 sets of Rule operations This is 1.

At each step of the computation: → what we do is

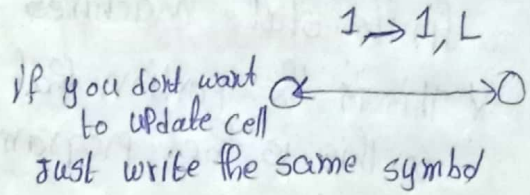
→ Read the current symbol
→ update (i.e write) the same cell
→ move exactly one cell either left or RIGHT.

* If we are at the left end of the tape, and trying to move LEFT, then do not move. stay at the left end.



if you do not update the cell then what do u do?

* let's assume that our tape head is one symbol you are reading that symbol and you have to update the same cell
* Now what if you don't want to update you want to leave the cell as it is then what you have to do?
  * you have to just write the same symbol again so if you don't want to update any symbol on a particular cell on which your tape it is present then you have to just write the same symbol onto the cell.
  * so this represents that situation is circles represents the state and you see that your tape head is on the symbol 1 and the symbol you are going to read



Rules of operation - 2

→ control is with a sort of FSM
→ Initial state
→ Final state: (there are 2 final states)
  1) the ACCEP STATE → if something is accepted (or) string accepted then Turing Machine
  2) the REJECT STATE → if something (or) string is Rejected then Turing Machine will go to the Reject state

which is another kind final state
So, These 2 kinds of final states.

→ Computation can either
  1) HALT and ACCEPT → when string accepted then the Machine comes to halt (or) stops (means stop)
  2) HALT and REJECT → when string Not accepted, it comes to Reject Then also Machine halt (or) stop
  3) Loop (the Machine fails to HALT)
     ↳ some Input the machine could go into a loop that means the Machine keeps on compatation and it never comes to accept neither

keeps on computation and it fails to stop. so that condition is known as loop.

so in loop then machine fails to halt so these are the conditions that we can get when we do competition in our Turing Machine.

Turing Machine (Formal Definition)

A turing machine can be defined as a set of 7 tuples $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$

$Q \rightarrow$ Non empty set of states.

$\Sigma \rightarrow$ Non empty set of symbols.

$\Gamma \rightarrow$ Non empty set of Tape symbols. → It is different from previous

$\delta \rightarrow$ Transition function defined as

$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$

in initial we apply input → They stored in tap

$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$ → then go to next state

Right or left on the tape

$q_0 \rightarrow$ Initial state

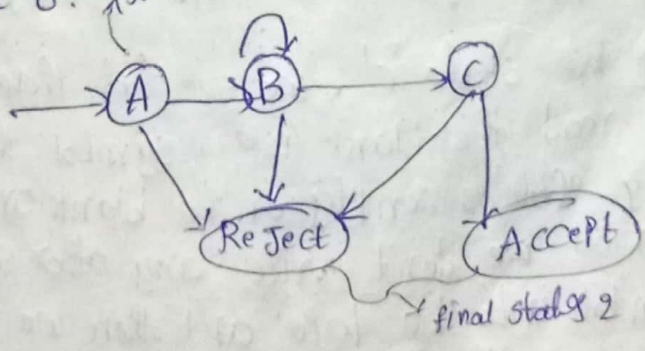
$b \rightarrow$ Blank symbol → special symbol

Turing Machine - Example 1

1) Design a Turing Machin which recognizes the language $L = 01^*0$ → it means a one 0 zero followed any No. of 1's followed by 0 end.

→ it Regular lang

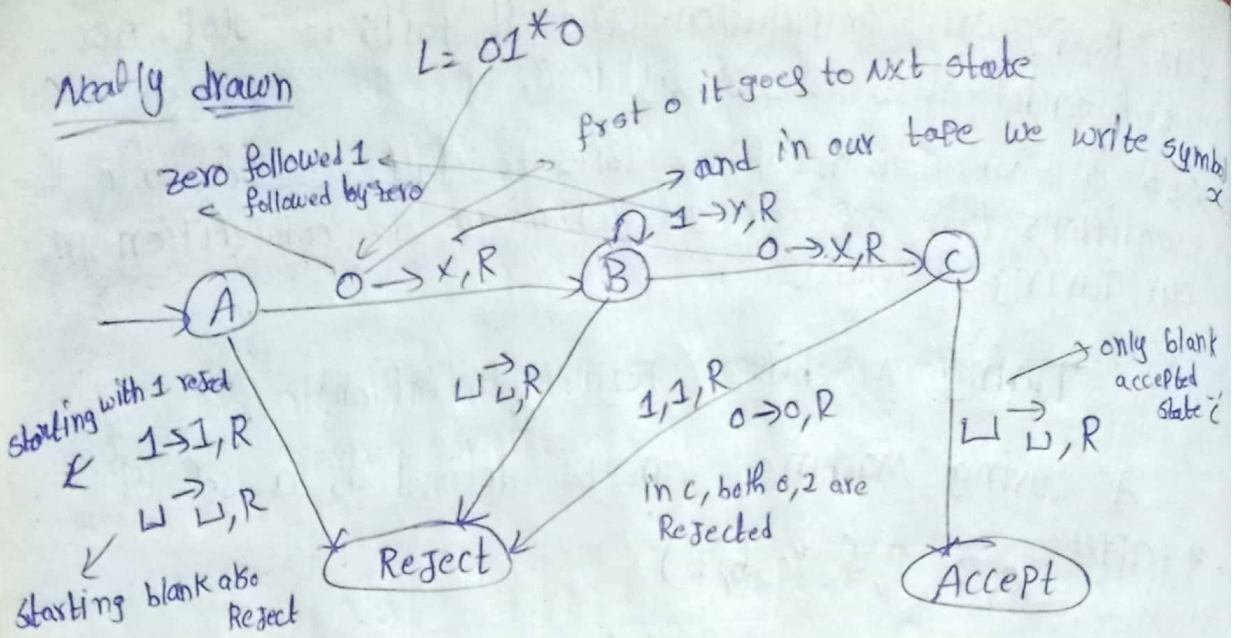
* we are going to design based on this regular language $L = 01^*0$.



Scanned with OKEN Scanner

Neatly drawn

$L = 01^*0$



* A is initial state B nxt B state C state
* They are 2 final states Reject state, Accept state
* $0 \rightarrow X, R$. 0 it goes to next state because in ours first '0'. $L = 01^*0$

~~we write sym~~ in our tape we write symbol X, Move Right on our tape and we come to state B

* we know that any No. of ones after we accepted this zero
* So In state B any no. of 1's. you always stay is state be so ↻ self loop
* $1 \rightarrow Y, R$ and write the symbol in our tape Y and then you Move right on the tape.
* After getting any no. of 1's we stay B. if we get 0 you go to the next state which is 'C' and then you write X on the and the you Move Right on the tape $0 \rightarrow X, R$.
* After we reach the state 'C'. we see that there is Nfg more to take. so we read the blank (␣ ⃗ ␣) symbol that means we read something that is empty or the blank and if we see the blank symbol we don't write any other symbol other than the blank symbol into the tape and then we Move right on the tape and it goes to the accept state
* So this is how it works

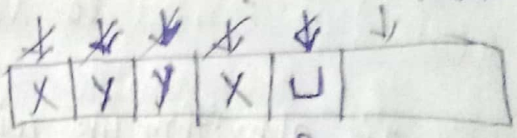
we take the string and check how it works

: 0 1 1 0 → in state B 1, self loop, so Y on tape

↓ if u have zero write X

*first we come to tape

string is ended and goes C to accept state so blank ⊔



Tape

why we writing X, YYX on tape

regular lang $L = 01^k0$ is very simple lang

why do not need the full power of a turing Machine

In order to impliment this, but we have (X, YYX) to do

using Turing Machine and particulary for this language this tape symbols don't have any particular use they not any particular use of accepting this language

We are doing this in order to show you how Turing Machine works and how the tape actually works.

you don't actually need the tape but we have use the tape because it is a turing Machine.

And also Turing Machine is deterministic means when you draw the transition diagram there should be a transistion for each and every input symbol that we have for each and every state

*so lets see it is deterministic (or) not.

$\Sigma = \{0, 1\}$ → input symbols

$b = ⊔$ → special symbol blank.

*zero followed by 1, followed by zero.

*Another thing reember about Turing Machine

anyother thing remembout about Turing Machine is that sometimes when we draw the turing Machine you may notice that some edges may be missing

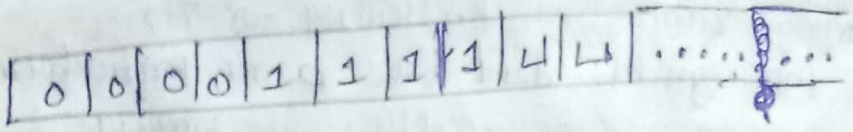
*so if some edges missing then you should understand that edge by default it leads to the Reject state.

2) Design a Turing Machine which recognizes the language $L = 0^N 1^N$?

↳ we will design a algorithm which will be able to accomplish accepting th

↳ we solve this the no of zeros exactly equal to No. of 1's.

↳ And this algorithm we will see how we can design the equivalent Turing Machine for it alright so here we have the tape



Algorithm

→ change "0" to 'X'

→ Move RIGHT to First "1"

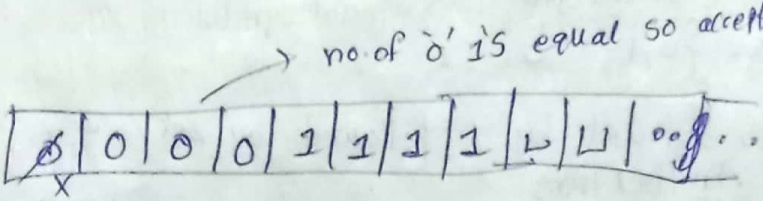
If None : REJECT

→ change "1" to "Y"

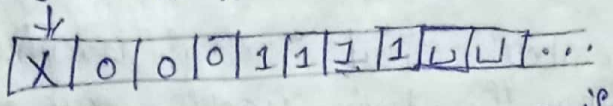
→ Move LEFT to leftmost "0"

→ Repeat the above steps until no more "0"s

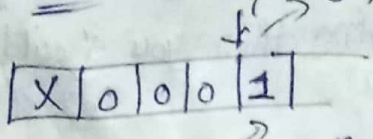
→ Make sure no more "1" is remain.



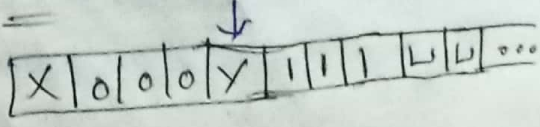
1) first chang "0" to "X"



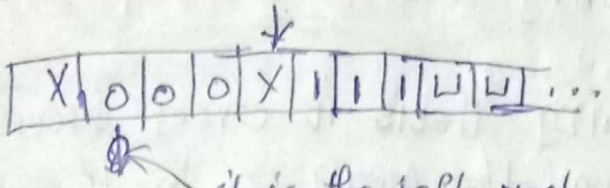
2) Move Right to First 1



3) Change "1" to "Y"

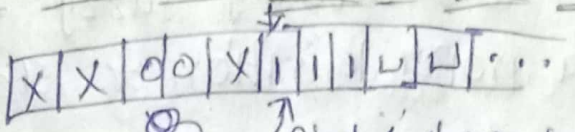


4) Move left to leftmost "0"

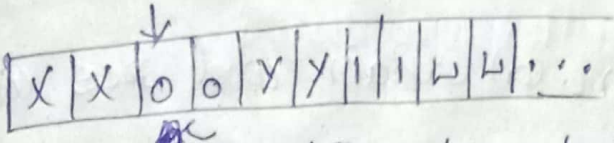


it is the left most zero '0'

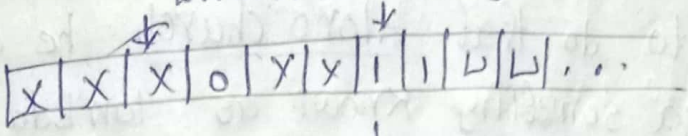
5) Repeat above steps until No More 0s. No More 1's

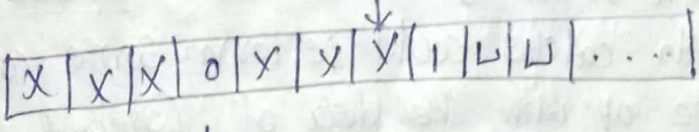


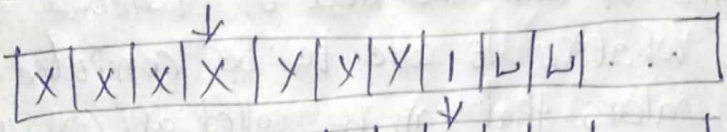
first 1 change to Y

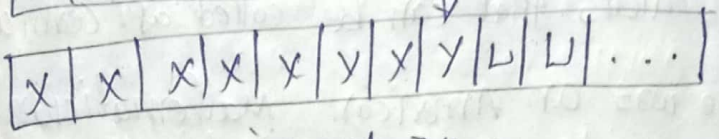


leftmost Zero change to X



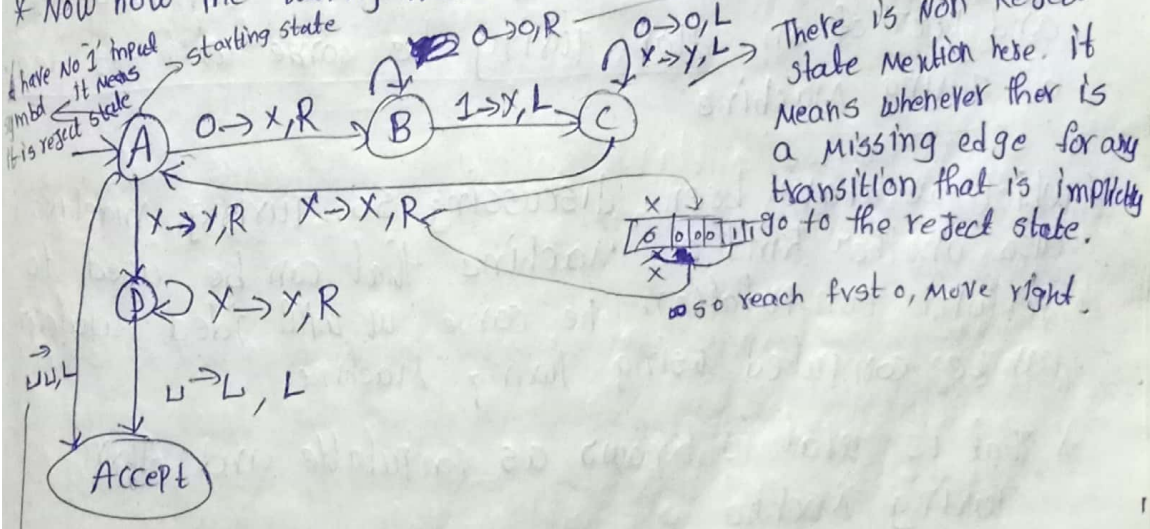


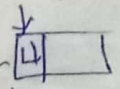




So we have No more 1s and zeros.

* Now how the turing machine designs





empty string ⊔ → ⊔,L → move leftmost
but it is in leftmost. so No change it

Church - Turing Thesis

The church Turing Thesis it comes from the question what does computable mean so in the olden days the term computable was actually fuzzy and we could not give a proper definition or standard in order.

* in order to define (or) explain what does computable actually mean
* so in order to do that Alono church he come up with this idea something known as lambda calculus so the lambda calcus could perform some computations and he came up with the idea of Lambda calculus saying that whatever is able to be computed by our lambda calcus that can be called as computable

* so Alonzo he was an American Mathematician and Magician who made major contributions to mathematical logic and the foundations of theoretical computer science and later on Alan Turing he came with his idea of Turing Machine

* which we have been discussing so Turing machine is also another kind of Machine that can be used to perform computation he came up with idea whatever can be computed using Turing Machine.
* That is what is known as computable and Alan Turing Machine

Several Variations of Turing Machine :

1) one Tape (or) many. → we know each Turing Machine ha 1 tape. if more tapes it is ve powerful.

2) Infinite on both ends. → in our Turing Machin ... but it has both ends.

3) Alphabets {0, 1} or More?

4) can the Head also stay in the same place?

5) Allow Non-Determission.

* by looking thes ~~it have more power in~~ variations (or) features give us more power in our Turing Machine

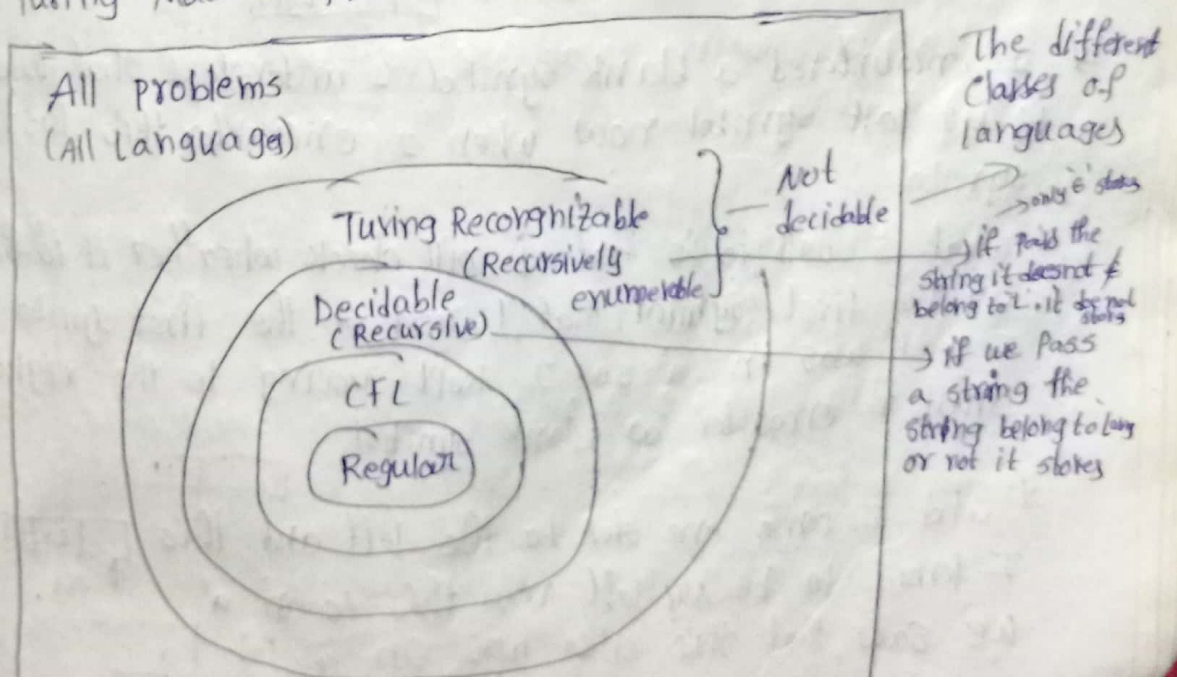
* but surprisingly all variations are equivalent in computing capability.

All variations are equivalent in computing capability

* All the variations of Turing Machine whatever features you add to them it is shown that all those variations are equivalent in computing capability.

* wheather you have these features or not their computing capability or the power remains the same

* I am NOT not prove this but This Mode one kind of Turing Machine from the other.



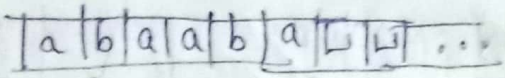
1) Design a Turing Machine that accepts even palindrome over the alphabet?

(leng of string is even no.. ; if we read rev. or first to last The string is sam.)

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

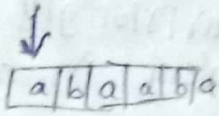
Eg: abaaaba → leng

* our task to design Even Palindrom Turing Machine will work and after that we will see how we will design it using the transition diagram.



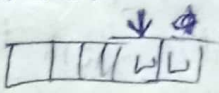
⇒ what we will do is?

* first we have over head over here



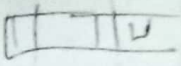
* And we will read the First symbol. so we see that the first symbol it's an 'a'. so I will replace this 'a' with blank symbol and after I replace this 'a' with a blank sym I will keep moving to the right until I reach the end of string

* now how will I know that I reaches the end of the strin I will know that I reach the end of string when I encount a symbol on the rightmost.

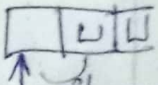


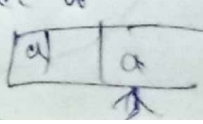
* I encountered a blank symbol I understood that this is my last symbol now when I encounter this last symbol

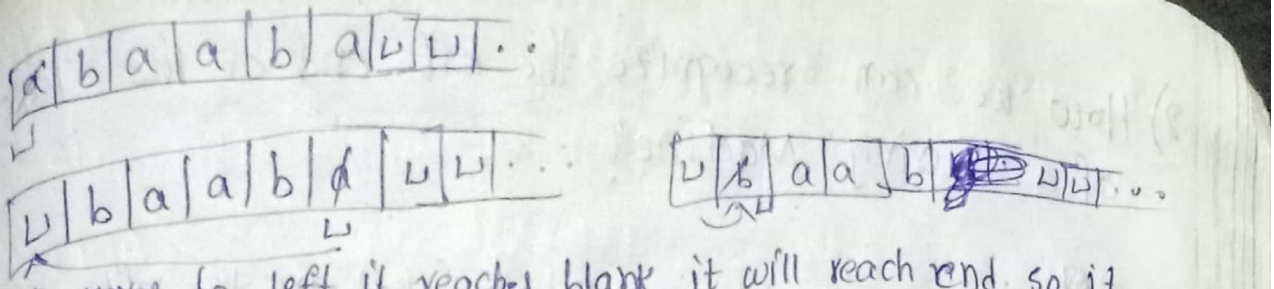
* what I will do is that i will check wheather it is same as the first symbol that i read so the first symbol that I read was in 'a' so I kept moving to the right until I encounter a blank symbol.



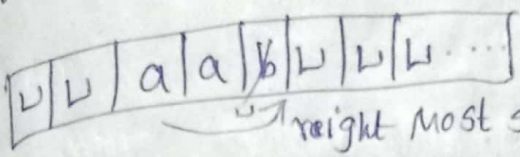
* and I came one step to the left and this I know to be my last now this is an 'a' we saw that this also was an 'a'



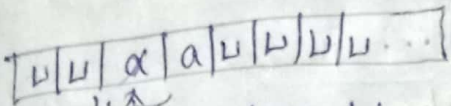




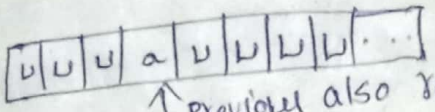
it moves to left it reaches blank it will reach end so it will do just move one step to the right. and replac 'b' with '␣'



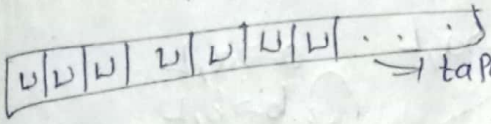
right most symbol. and check previous replaced and prsnt are same symbol. Then replace it



left most symbol.

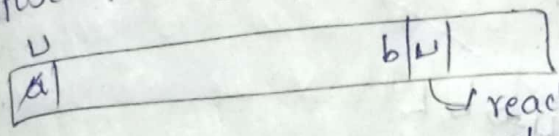


previous also replaced 'a' with blank. so prsnt also replace



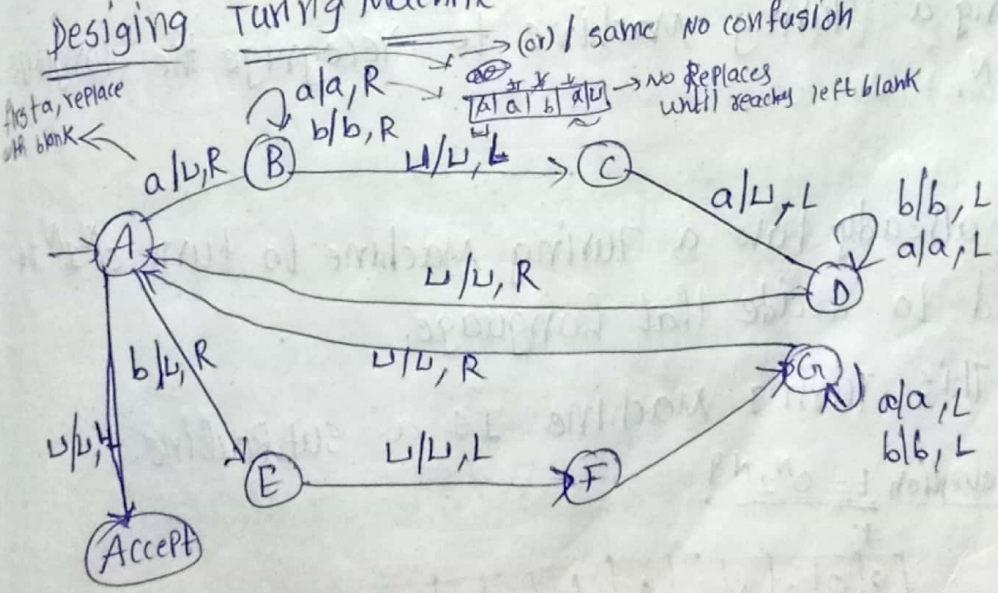
tape have all blank symbols. so it accepted ✓

suppose first symbol 'a' last 'b'



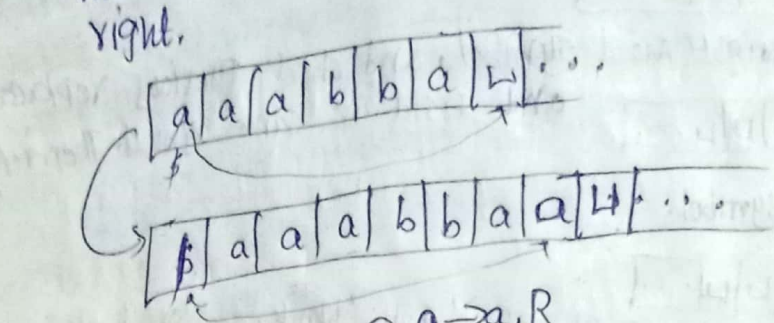
reached end before is 'b'. but previous replaced 'a'. These two symbols are not same then this string not accepted. In this case it is not even palindrome.

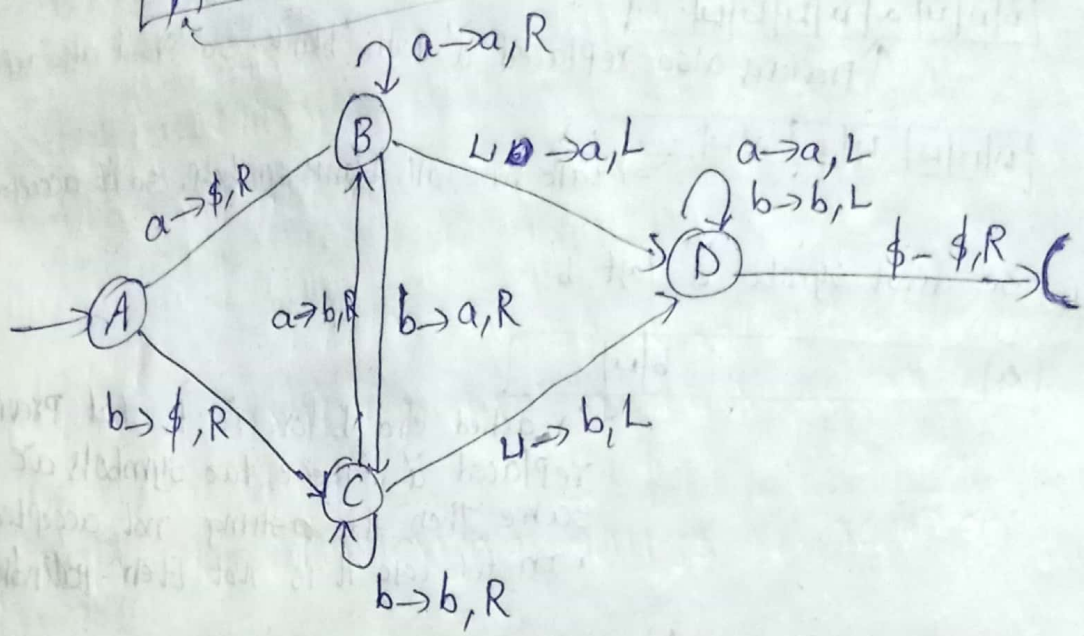
Desiging Turing Machine



2) How we can recognize the left end of the Tape of a Turing Machine?

Sol: Put a special symbol $ (dollar) on the left end of the tape and shift the input over one cell to the right.





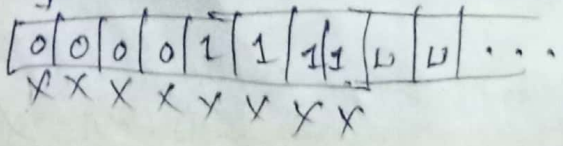
3) Building a Turing Machine to recognige the language $0^N 1^N 0^N$.

Idea

We already have a Turing Machine to turn $0^N 1^N$ to $x^N y^N$ and to decide that Language.

Use this Turing Machine is a subRoutine.

Previous question $L = 0^N 1^N$?



Step 1: 0^N 1^N 0^N

0 0 0 0 1 1 1 1 0 0 0 0

⇓

X X X X X X Y Y 0 0 0 0

Step 2: Build a similar Turing Machine to recognize $y^N 0^N$.

Step 3: Build the final Turing Machine by combining these two smaller Turing Machines together into one Larger Turing Machine.

Comparing two strings

3) A Turing Machine to decide $\{w\#w \mid w \in \{a,b,c\}^*\}$

(1st half should to equal to 2nd half)

Solution

* use a new symbol such as 'X'
* Replace each symbol into an X after it has been examined.

how we can do this?

Here is an example that explains our solution

a b b a c # a b b a c (we see that 1st half of string equal to 2nd half of string. How turing decide 1st half ≠ 2nd half?)

(a, a encountered so replacing 'X')

⇓

X b b a c # X b b a c

⇓

X X b a c # X X b a c

⇓

X X X a c # X X X a c

⇓

X X X X c # X X X X c

⇓

X X X X X X X X X X → we see that all are 'X'. so we say that 1st half equal to 2nd half.

The disadvantage of this problem

The disadvantage is we lost the original string.

Problem

Can we do it non-destructively? i.e without loosing the original strings?

Sol

Replace each unique symbol with another unique symbol instead of replacing all with same symbol.

So eg:- a → P → in place of substitute P
b → Q
c → Y

abbac # abbac
⇓
PQQPY : PQQPY → so we follow this technique

instead of replacing everything with X we can retain the original string

* Restore the original strings if required.

P → a, R
Q → b, R
Y → c, R

Multiple

Multiple Turing Machine

we discussed that all those turing machines they have just a single tape

we know that turing machines they have a tape and we already know what is a function of the tape

but we have seen that in all a Turing Machines They are having just single tape.

Now we see the Multi-tape turing Machine.

* it means this turing machine have more than one tape and this multi-tap turing maching powerful than the single tape turing machine

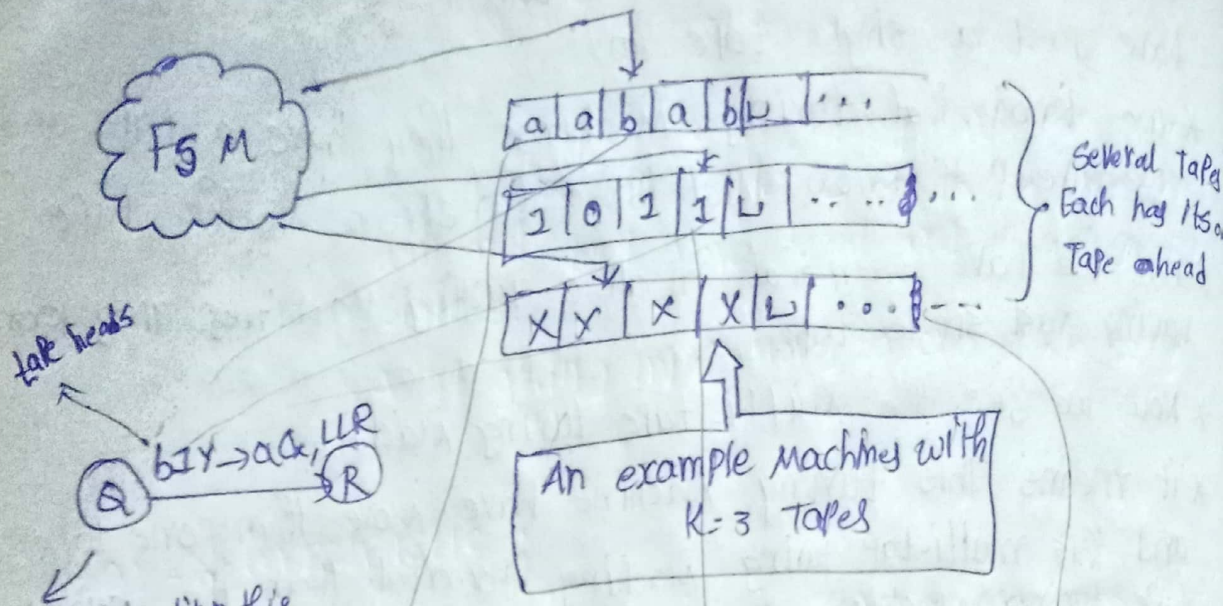
* The single tape as powerful as multi-tape turing Machines we should be able to design a equivalent single tape turing machine for every multi-tape turing machine.

Theorem :- Every multiple Turing Machine has an equivalent single tape Turing Machine.

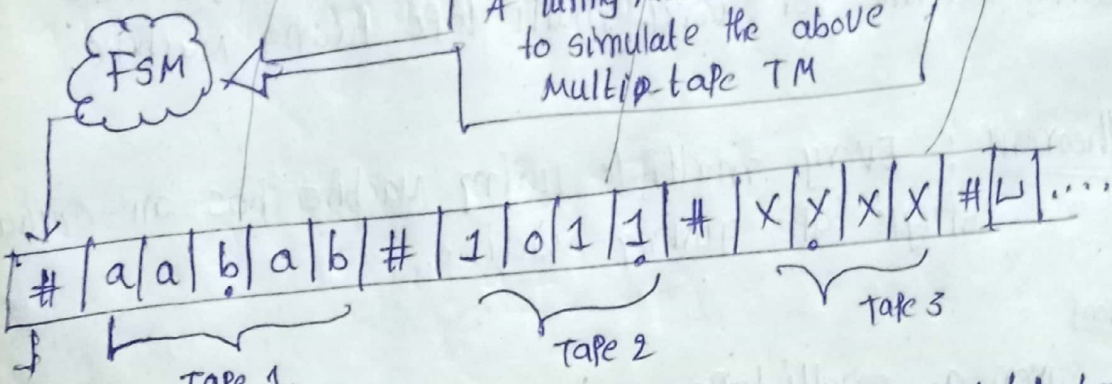
Proof

Given a multi-tape Turing Machine show how to build a single tape Turing Machine.

* Need to store all tapes on a single tape
  * show data representation
* Each tape has a tape head
  * show how to store that information.
* Need to transform a Move in Multitape TM into one or Moves in the single Tape TM.

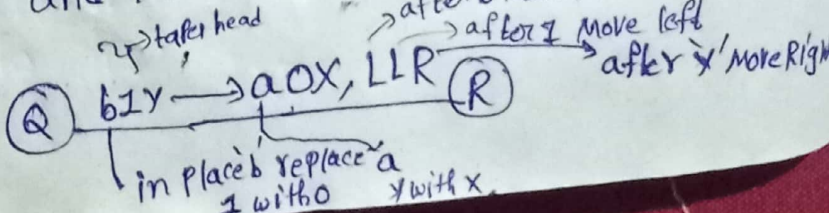


Single Tape Turing Machine



Individual tape heads

- Add "dots" (•) to show where Head "K" is
- To simulate a transition from state q, we must scan all Tape to see which symbols are under the K Tape heads. *So (•) → dots. using the marking topic put a dot it means tape he
- Once we determine this and are ready to make the transition, we must scan across the tape again to update the cells and move the dots.



whenever one head moves off the right end, we must shift our tape so we can insert a blank "␣"

* in muti turing Machine we have blank. but in the single tape we have No 'blank'
* so we have a dot here
* so if we encounter a right end of any tape and if we want to move further to right and we have to shift the entire thing to the right.
* so that we can have a blank symbol, so with that we have completed the conversion over multi-tape turing Machine to a single tape turing Machine
* so we prove that our theoroem multi-tape equivalent to single tape.

Non deterministic Turing Machine

Non deterministic Transition Function

$$\delta : Q \times \Sigma \rightarrow P\{\Gamma \times (R/L) \times Q\}$$

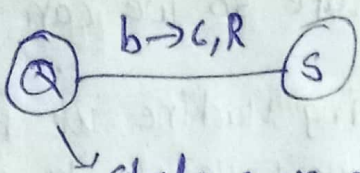
* The only difference of determintic TM and non-deter TM where 'P' is there

P → stands for power set.

* The Transition Function means
  - The state on getting a particular input can go not only to one state, but it can go to many states so instead of one state we have power set of combinations that is possible.

* Taking an Example and how it works?

Deterministic :



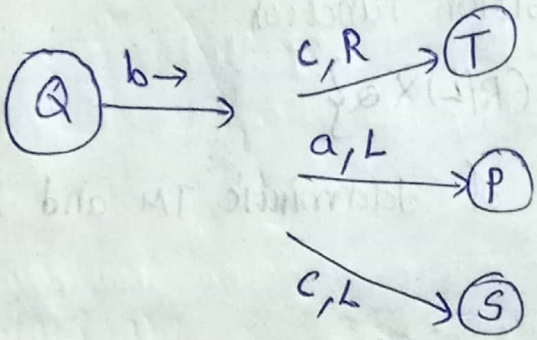
state q on getting particular input symbol B will write C on to the tape and will and will Move Right and it will go to state S.

* It getting input Moves to one state only

Non-deterministic :

in case of non-deterministic Turing Machines let us see what is the difference so here is an example of a Non-deterministic Turing Machine.

* so this is the transition diagram . we have state Q, T, P, S



* on getting input of a state input symbol 'b' .it can go either state T by writing 'c' to the tape and Moving to Right

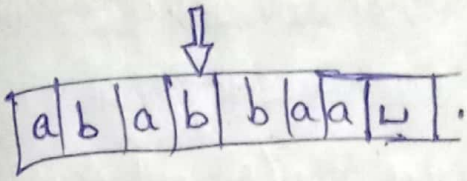
* (OR) it can go to state 'P' by writing 'a' to the tape and Moving to the left

* (OR) it can go to state 's' by writing 'c' to the tape and Moving to the left.

* we see that on getting input we have more states so this is the property of Non-deterministic

Configuration

- A way to represent the entire state of a (TM) at a moment during computation.
- A string captures.
  - The current state
  - The current position of Head
  - The entire Tape contents

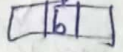


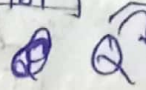
* The string configuration is below

aba q b b a a

This string tells (TM) at any particular moment.

This represent current state which we are present in - so

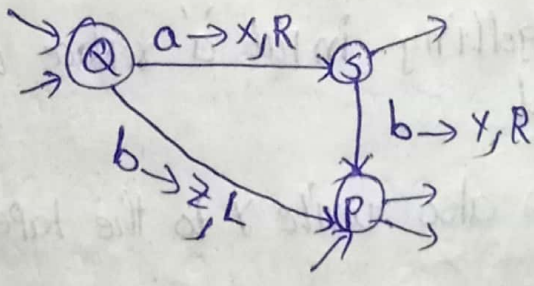




like This

* How it will differs in deter turing Machine and Non-Determinstic turing Machine

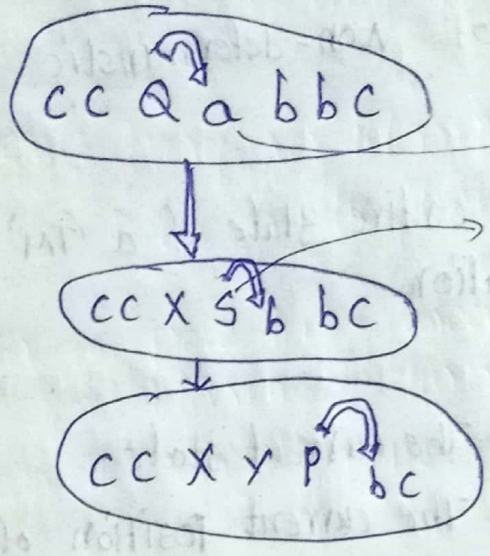
Deterministic TM :



→ This is one snip of Turing Machine.

* Now we see the computation History.

computation history

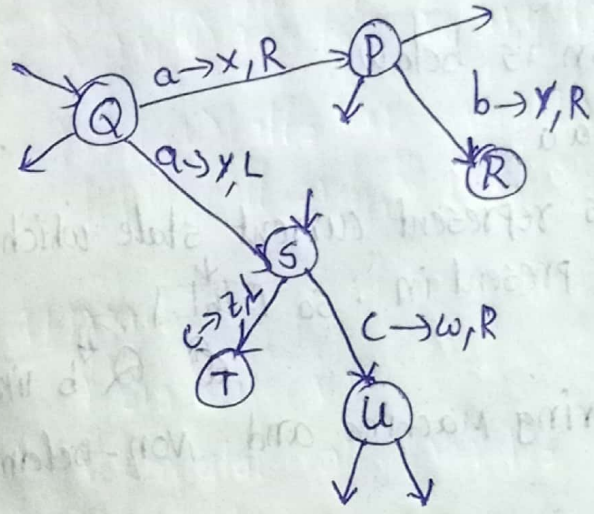


$a \to X, R$ (1)

first take head in 'a' and moves Right

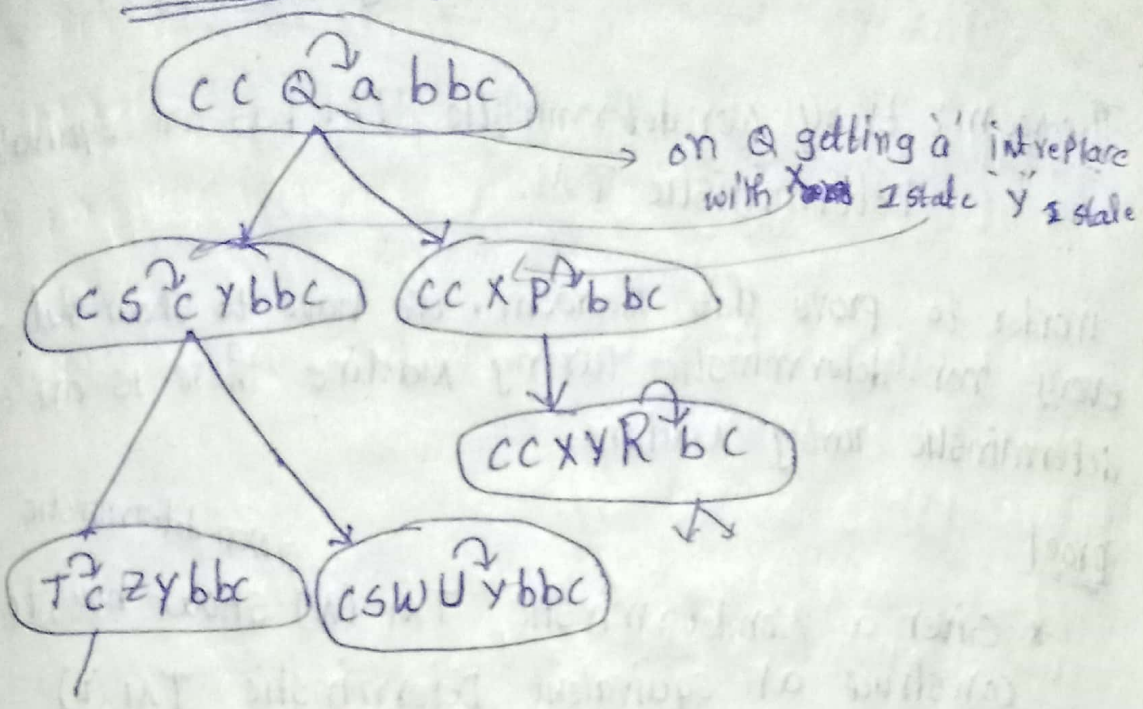
in state 'S', tape head 'b' $b \to Y, R$ and in place of b replace y.

Nondeterministic TM :



* The states are Q P R S T U
* we can see that 'Q' on getting input 'a' replace with X and the tape move to Right
* on getting 'a' it can also write Y to the tape and Move Left.
* we see that particular symbol 'a' have different type of M

computation history



* so we see that here it is not a linear change anymore but it is a tree because for even one single input there are many possibilities
* so this is how the computation history for non-deterministic

outcome of a Nondeterminstic computation

1st outcome

Accept if any branch of computation accepts, then the Nondeterministic TM will accept.

2nd outcome

Reject → if all branches of the computation HALT and REJECT (i.e no branches accept, but all computations HALT) then the Nondeterministic TM Rejects.

3rd outcome

Loop → computation continues but Accept is never encountered. some branches in the computation history are infinite.

Nondeterminism in Turing Machine (part-2)

Theoroem: Every Nondeterminstic TM has an equivalent Deterministic TM.

inorder to prove this theoroem. we have to show that for every non-determinstic Turing Machine there is an equivalent deterministic Turing Machine.

proof

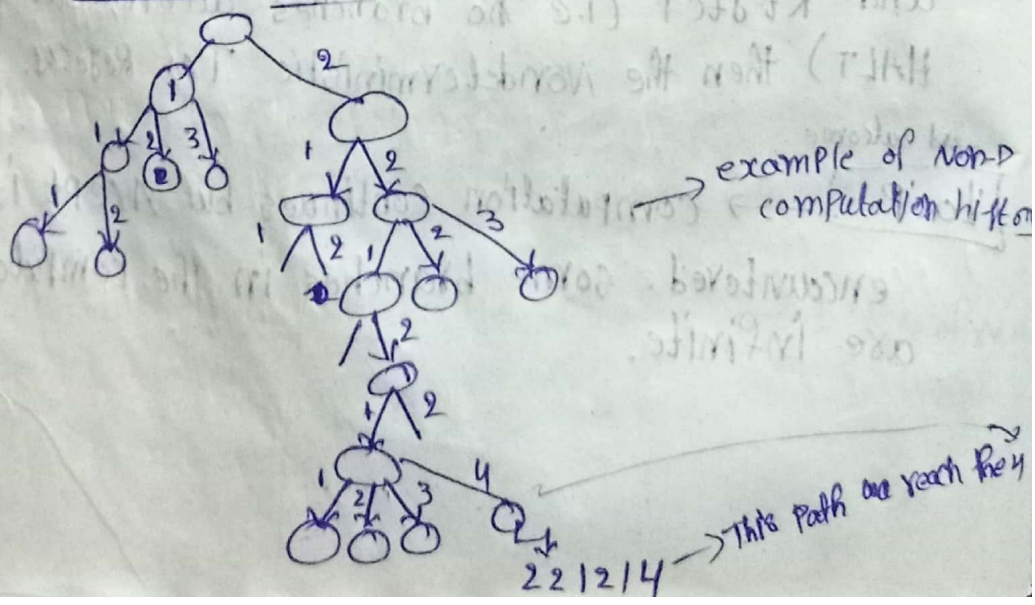
* Given a Nondeterminstic TM (N) → Non-deterministic show how to construct an equivalent Determinstic TM (D)
* If (N) → Non-deter TM accepts an any branch, the (D) → Deterministic will Accept.
* If (N) halts an every branch without any Accept, then D will halt and Reject.

Approach → Search all

* simulate (N)
* simulate all branches of computation
* search for any way (N) can Accept

* Here is the computation history of Non-deterministic TM

computation history



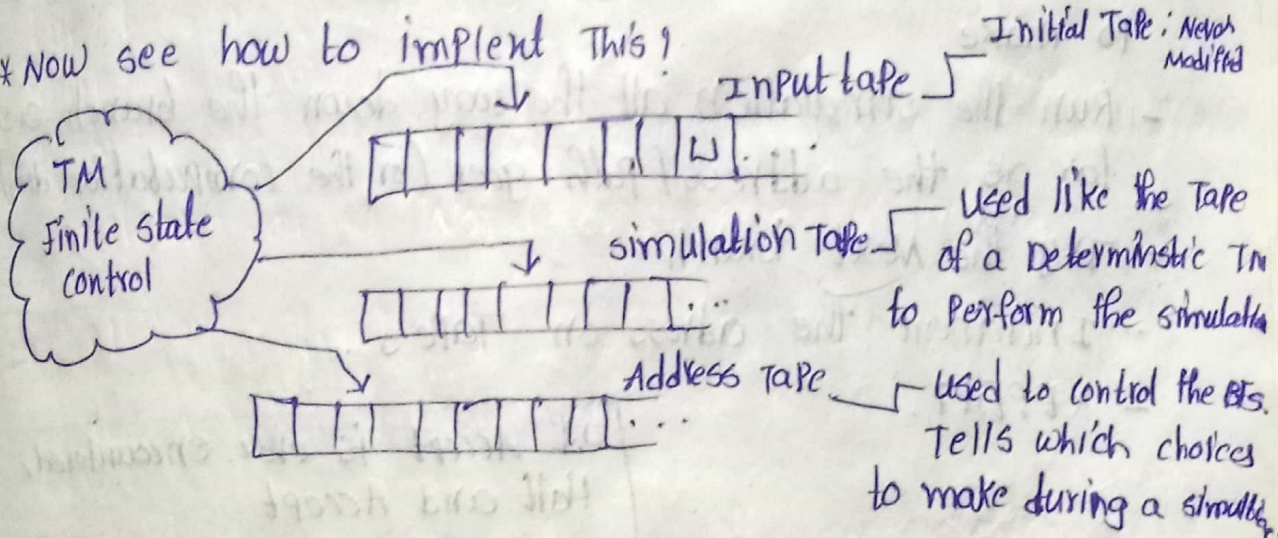
every node have particular Path number like this.

→ A Path to any Node is given by a number
→ Search the tree looking for ACCEPT

Search order
→ Depth First Search x → infinite loop so not suitable
– Breadth First Search ✓

To examine a node
- Perform the entire computation from scratch
- The path number tells which of the many nondeterministic choices to make.

* Now see how to implent this?



* Now we understood how all this technique works
  * so write the Algorithm.

Algorithm

Initially: Tape 1 contains the input (Input tape) Tape 2 and Tape 3 are empty (simulation tape, Address tape)

- copy Tape 1 to TAPE 2
- Run the simulation
- use TAPE2 as "the Tape"
- when choices occur (i.e when Nondeterminstic branch point are encountered) consult TAPE3.
- ~~Run the simulation all~~
- TAPE 3 contains a Path. Each number tells which choice to make
- Run the simulation all the way down the branch as far as the address / Path goes (or the computation dies)
- Try the Next branch
- Increment the adress on Tape 3
- REPEAT.

If Accept is ever encountered, Halt and Accept
If all branches Reject or die out then Halt and Reject

* So we Prove The Theoroem.

Turing Machine as Problem solvers

* we already discussed turing machine are powerful than other machines (Finate Automat, CFG, Pushdown A)

* first we do is Any arbitary problem can be expressed as a language.

* we can formulate a problem into language so that we can design the turing machine inorder to solve the problem.

→ Any instance of the problem is encoded into string.

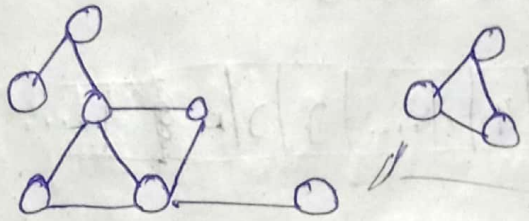
The string is in the language ⟹ The Answer is YES

(It mean our TM accepts string)

The string is not in the language ⟹ The Answer is No. (↓ rejects)

* let us take an example to more clear.

Ex: Is this undirected graph connected? (find connected or Not connected)



* by looking the graph is not connected gap is there
* there is No link b/w 2 graphs. so this is not connect graph.
* let see how can we formulate this problem into a language
* And how can we design a Turing Machine inorder to solve this problem.

First we do is

=> we Must encode the problem into language.

$$A = \{\langle G \rangle \mid G \text{ is a connected graph}\}$$

* language A equal to G. G is a connected graph

⇒ we would like to find a TM to decide the languag

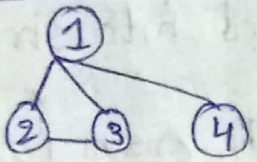
→ TM accept yes it is connected graph

ACCEPT = "YES" This is a connected graph

REJECT = "No", This is not a connected graph / or this is not a valid Representation of a graph.

LOOP = This problem is decidable. our TM will always ha

Representation of Graph

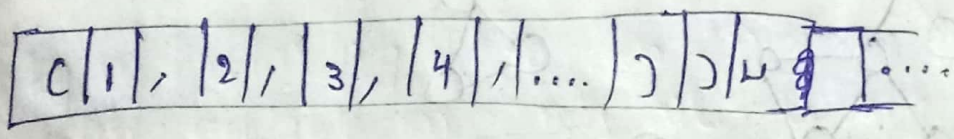


$\langle G \rangle$ = (1,2,3,4) ((1,2) (2,3) (1,3) (1,4))

List of nodes — Edges

alphabets with

$\Sigma = \{(,), ,, 1, 2, 3, 4, \ldots : a\}$



High level Algorithm

Select a Node and Mark it

REPEAT

  For each Node N

    If N is unmarked and there is an edge from N to an already marked Node

  Then

    Mark Node N

  End

until no more nodes can be Marked

For each Node N

  If N is unmarked

End

Accept Then

Implementation level Algorithm

- check that input describes a valid graph
- check Node List
  - scan "(" followed by digits.
  - check that all nodes are different i.e no repeats
  - check edge lists...
  e.t.c
- Mark First Node
  - Place a dot under the first node in the node list..
  - scan the node list to find a node that is not marked e.t.c

* This is actually a brief idea of how you can do it the whole thing take a lot of time
* so we are not going to solve entire thing but we see that using these technique we are able to use Turing Machines as problem solvers
* Turing machines can also be used as problem solvers and hence you can understand how powerful it is as compared to other machines

Decidability and undecidability

First we know Recursive language and Recursive Enumerable languages

Recursive Language:

A language 'L' is said to be recursive if there exists a Turing Machine which will accept all the strings in 'L' and reject all the strings not in 'L'.

* The Turing Machine will halt every time and give an answer (accepted or rejected) for each and every string input.

the turing Machine will accept the string if it belong to the language

* and if the string does not belong to the language it will reject the string.
* Accept (or) reject it can hold the string.

Recursively Enumerable language :

* A language 'L' is said to be a recursively enumerable language if there exists a Turing Machine which will accept (and therfore halt) for all the input strings which are in 'L'.
* But may or may not halt for all input strings which are not in 'L'.
* in recursive enumerable language the turing machine will hold only when it is accepting the string in the language in other cases it may not hold.

* by understanding these two languages how can we defin decidable languages and partially decidable languages and undecidable languages.
* understanding decidable, language position to defin and understand these 3 terms which are decidable language partially decidable language, undecidable languages

Decidable language :

A language 'L' is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa.

Recursive Mean Turing Machine should by either accepting (or) rejecting then that language is said to be a decidable language.

Partially Decidable Language:

A language 'L' is a Partially decidable. if 'L' is a recursively enumerable Language.

Undecidable Language

TM sometimes hold and sometimes will not hold so those language known as Partially decidable lang.

- A language is undecidable if it is not decidable.
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing Machine for that Languages.

Recursive language	TM will always Halt
Recursively Enumerable Language	TM will halt sometimes & may not halt sometimes
Decidable language	Recursive language
Partially Decidable lang	Recursively Enumerable Lang
UNDECIDABLE	NO TM for that language.

Universal Turing Machine

Taking example

The Language — M could be any TMachine, w is a string.

$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w \}$

* it is Turing Recognizable

* The language A_{TM} has two elements where M could any Turing machine and w is string.

* which if we pass into TMachine M will be accepted so if we have a string w which even passed to this Turing Mach M is accepted. Then we can say that this whole thing lies in the language hit TM ∞

* And says that this language is Turing recognizable

* So here we are calling it recognizable but not decidable and we know what is the meaning of decidable and recognizable previously discussed

* why we are not calling it decidable is because we don't what this Turing Machine 'm' is actually doing some specific tasks we don't know.

* if we change this w or depending on the strings that we pass to it this turing machine may somet accept it may sometimes reject it (or) sometimes go into Loop.

* we are not sure. And we call it recongnizable but not decidable

⇒ The question can we have Algorithm?

Given the description of a TM and some input, can we determine wheather the Machine accepts it?

- Just simulate / Run the TM on input.

* we will pass the description of the turing machine M along with input w into our universal Turing Machine and we will run it

How can determine machine accepts it?

* very simple Just simulate or run the Turing Machine on the input then wheather it will accept (or) reject (or) it will loop

* So This universal TM will behave exactly like the other Turing Machine that we have (or) like the Turing machine whose description we passed to the universal Turing Machine

M accepts w: our Algorithm will Halt & Accept

M Rejects w: our Algorithm will Halt & Reject.

M loops on w: our Algorithm will not Halt.

Input: M = the description of some TM
w = an input string for M

Action: Simulate M
- Behave Just like M would (May accept, reject or loop)

The UTM is a recognizer (but not a decider) for

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.

Halt problem

To given a program make sure that this program will halt (or) not and what do we mean by halting.

* Halting means that the program will either accept and halt or reject and halt and it will never go into a loop so that is what we mean by halting.

=> Given a Turing machine, will it halt when run on same particular given input string?

=> Given some program written in some language (Java/c/etc.) will it ever get into an infinite loop or will it always terminate?

Answer

- In General we can't always know
- The best we can do is run the program and see whether it halts.
- For many programs we can see that it will always halt or sometimes loop.

* we said that halting problem is undecidable
* How can prove halting problem is undecidable are we se

=> can we design a machine if given a program can find out or decide if that program will alway halt or not halt on a particular input?

* Inorder to prove this we use contradiction. first we can assume that we can design such a machine.

Let us assume that we can:

H (P, I) → assume Machine (H)
→ Input
→ Program

- Halt
- Not Halt

* on taking particular program (P) input (i) as the argument this Turing machine 'H'
* So It tells us the program halts are Not halts.
* so we assume the Machine exsists the Turing Machine 'H' pass program (P) and input (i). These will tell us it will halt or not halt.

⇒ This allows us to write another program

```
C(X)
if (H (X,X) = = Halt)
    Loop forever;
else
    Return;
```

→ C takes an argument 'x'

If we run 'c' on itself

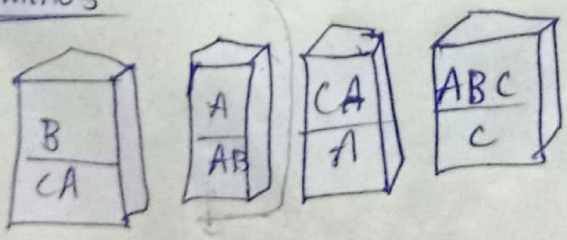
C(C)

H(C, C) = Halt → Not Halt
H(C, C) = = Not Halt → Halt

Post correspondence Problem

The Post correspondance Problem is an undecidable decision Problem that was introduced by Emil Post in 1946.

* Here we see instance (or) example of Post correspondence Problem
* In post correspondence problem something called Dominos
* Dominos



→ it is sometimes called dominos (or) tiles

* In dominos we have 2 parts Numerator and Denominator (or) the top part and bottom part

* you can call it anyway so here we can see that in this first Domino (or) first tile $\frac{B}{CA}$

* B on the top and CA are the bottom and then in this Domino we have A at the top and AB at bottom $\frac{A}{AB}$

* And we have 2 dominos

* let us see what is the task that wee need to do so

=) our task is we need to find a sequence of dominos such that the top and bottom strings are the same.

* wee need to arrange these Domino's in such a way that the top part and the bottom part of these 2 strings are the same

* that means if you combine these symbols on top and then the symbols on the bottom and if you write the top symbols and the bottom symbols should be the same that means they should form the same string.