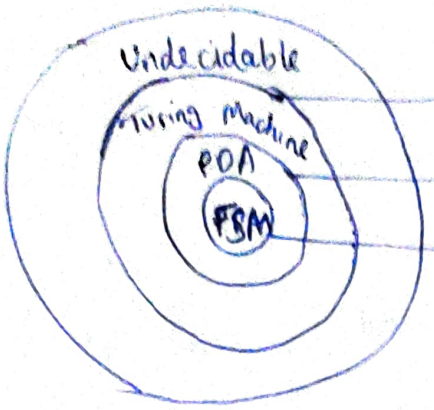
Turing Machine

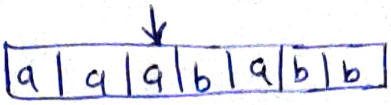
[Neso academy] + Net.



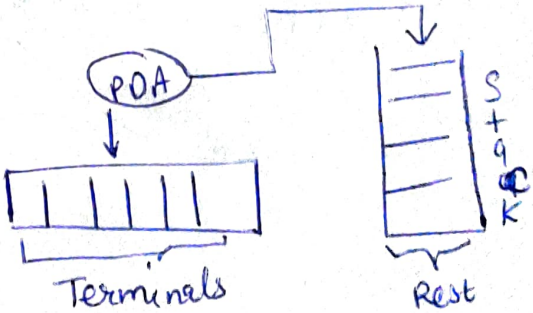
→ Recursively Enumerable Languages
→ Content free languages
→ Regular languages

Data structures used

1) FSM : Input string

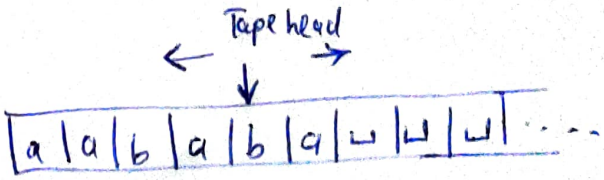


2) PDA : Input string + stack



3) Turing machine

A tape (an infinite sequence)

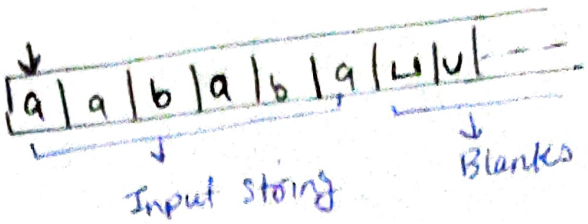


Tape Alphabets : $\Sigma = \{0, 1, a, b, X, Z_0\}$

The Blank ␣ is a special symbol $␣ \notin \Sigma$

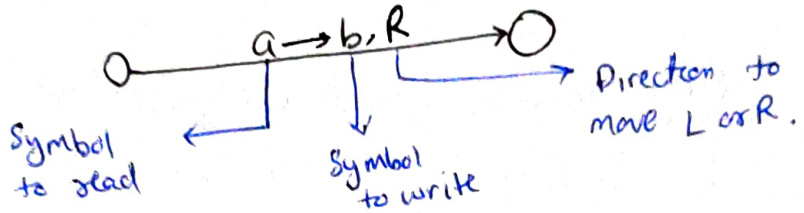
The blank is a special symbol used to fill infinite space.

- Initial Configuration



- Operations on Tape

→ Read / Scan symbol below tape head
→ Update / write a symbol below tape head
→ Move tape head one step left
→ Move tape head one step right



if you dont wanna update (write same symbol)

$1 \rightarrow 1, R$

Definition 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
$q_0 \rightarrow$ Initial state
$\delta \rightarrow$ Transition function defined as
$b \rightarrow$ Blank Symbol

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

$F \rightarrow$ set of final states (Accept state & reject state)

- Thus production rule of Turing machine will be written as

$$\delta(q_0, q) \rightarrow (q_1, y, R)$$

- **Turing Thesis** :- It states that any computation that can be carried out by mechanical means can be performed by some Turing Machine.

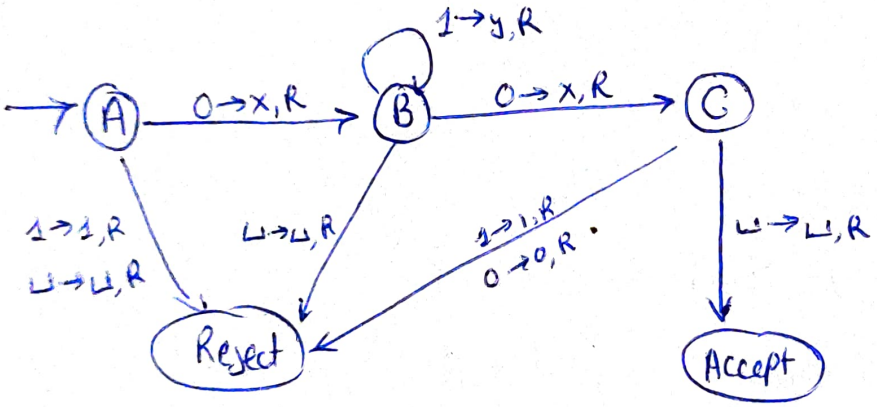
- No one has yet been able to suggest a problem solvable by what we consider an algorithm, for which a turing machine program can not be written.

- it accepts Recursively Enumerable Language.
A language L and Σ is said to be recursively Enumerable if there exists a turing machine that accepts it.

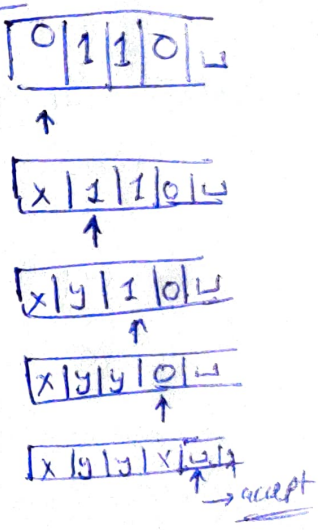
Q1 Design a turing machine which recognizes the language

$$L = 0\,1^*\,0$$

means → 0 (any no. of 1) 0.



eg



- here it is deterministic
(as for all symbol there is one transaction)

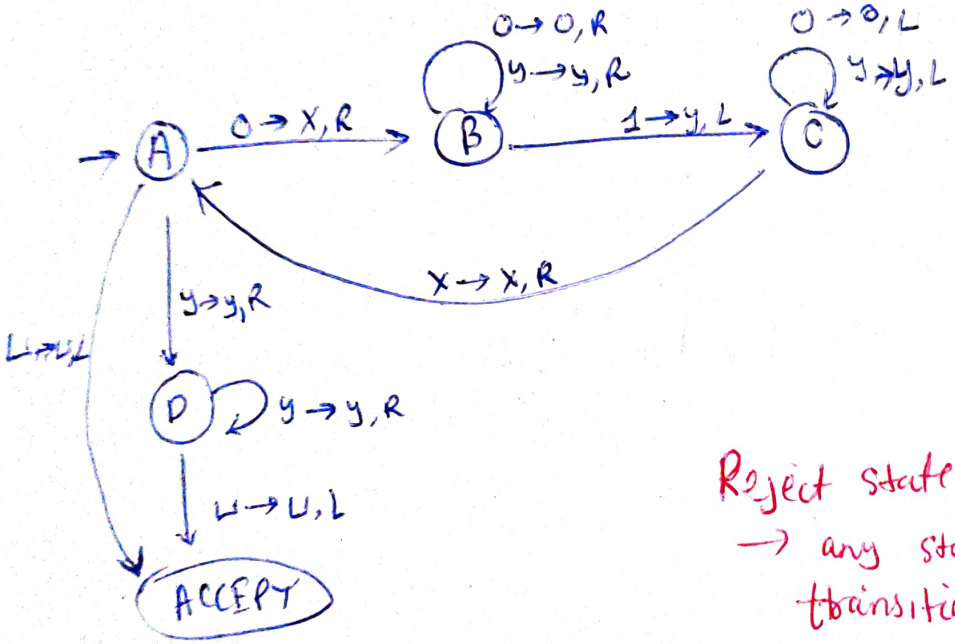
* if we don't make some transition then it means it go to reject state.

Q2 Design Turing Machine that recognizes $L = 0^N 1^N$

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's remain

→ for each 0 ~~to~~ x make 1 → y



Reject state
→ any state with missing transition

eg here A 1 → Reject

CHURCH - TURING THESIS

→ What does computable mean?

→ All varition of turing machines are equivalent in computing capability.

- Turing Recognizable (Recursively enumerable)
  - If accept → accept & halt
  - If reject → may go in loop.

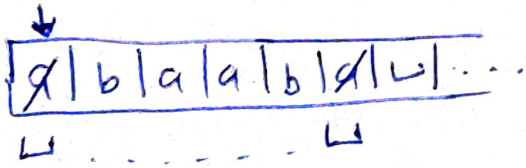
Turing Decidable (Recursive)
- If accept → halt
  If reject → reject & halt

- Undecidable language → never halts

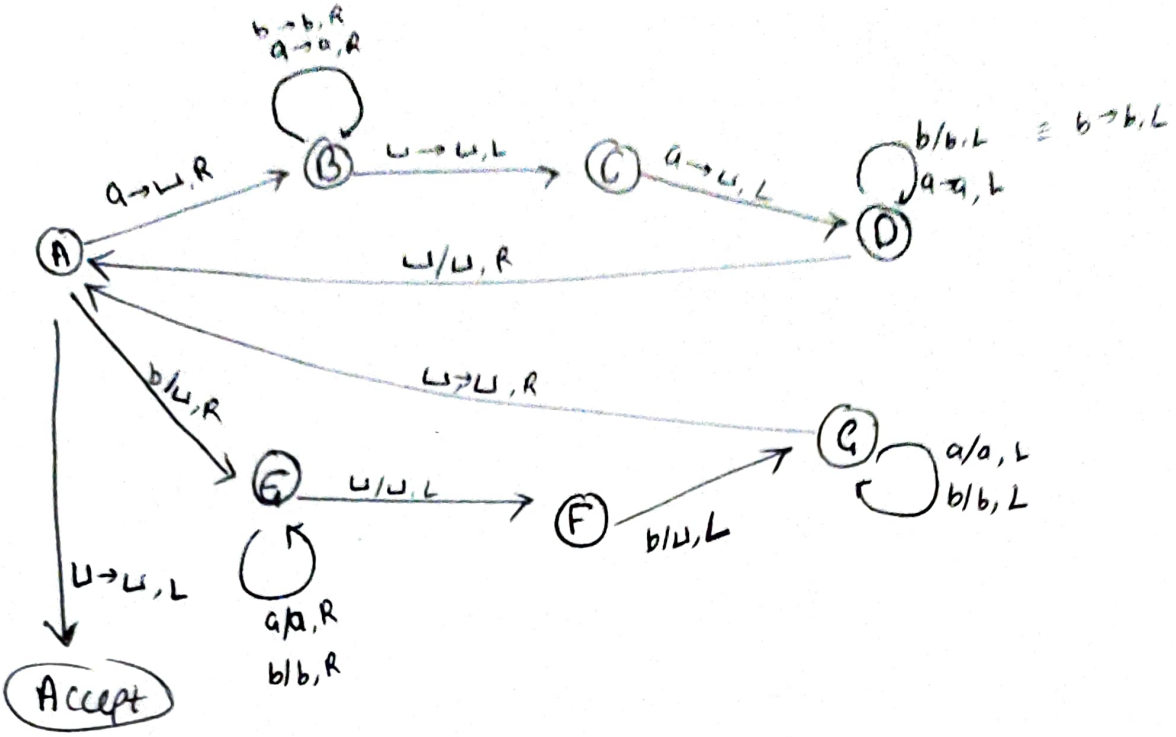
Q3 Design a Turing Machine that accepts Even palindrom over the alphabet $\Sigma = \{a, b\}$

↓ length even

eg a b a a b a



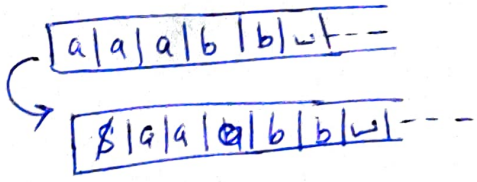
- we start from left replace it with ␣, & go to right most symbol & if it is same as previous left most we replace it with ␣ & again go to left most symbol.
- at end if tape has only ␣ then accept.
- if leftmost & rightmost are different so REJECT.



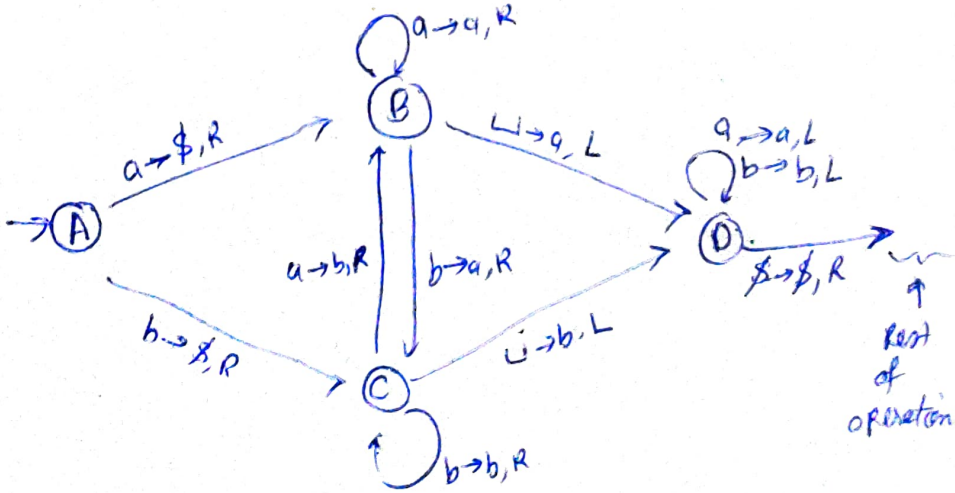
• Turing Machine Programming techniques

1) Problem : How can we recognize left end of Tape of turing machine

Solution : Put a special symbol $ on left end of Tape & shift input over one cell to right.



Turing machine for this



Q4 Build a Turing Machine to recognize $L = 0^N 1^N 0^N$

Idea :- we already have turing machine to turn $0^N 1^N$ to $x^N y^N$ & to decide that Language

- Use this Turing Machine as a subroutine

Step 1 :- 0000 1111 0000

⇓

xxxx yyyy 0000

Step 2 → build a similar Turing machine to recognize $y^N 0^N$

Step 3 → Combine this 2 turing machine to make final turing machine

Turing for Comparing Two Strings

Q5 A Turing Machine to decide $\{ \omega \# \omega \mid \omega \in \{a, b, c\}^* \}$

- Use new symbol `x`
- Replace each symbol with `x` after it has been examined

abc#abc ⇒ xbc#xbc

⇓

xxx#xxx ⇐ xxc#xxc

accepted

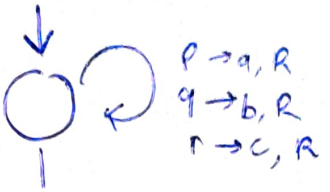
- but we lost original string

to do this without loosing original string

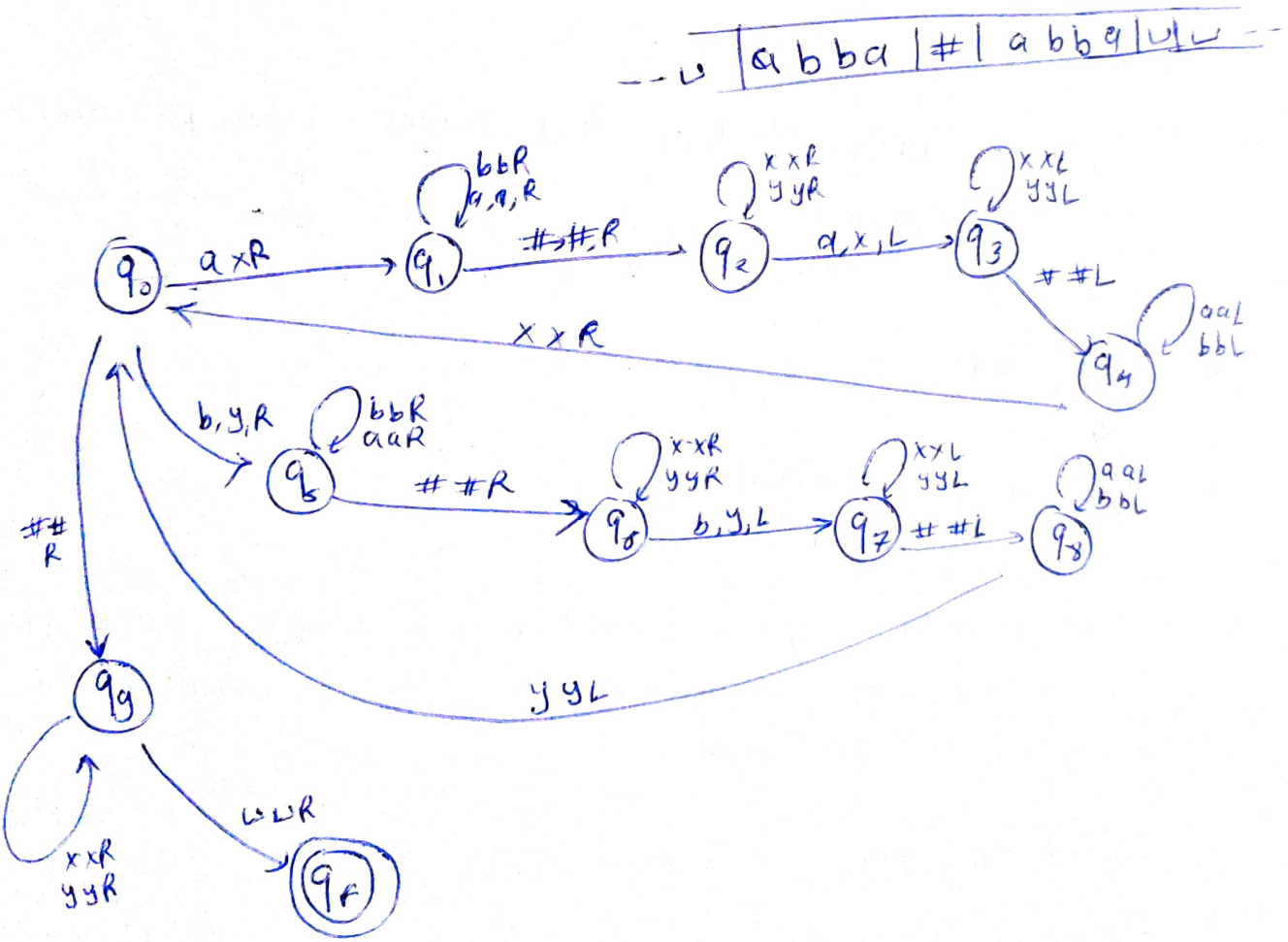
soln Replace each unique symbol with another unique symbol instead of replacing all with same symbol.

eg $a \to p$
$b \to q$
$c \to r$

& restore original by



- New eg $L = \{ w \# w \mid w \in (a,b)^* \}$



Multi tape turing Machine

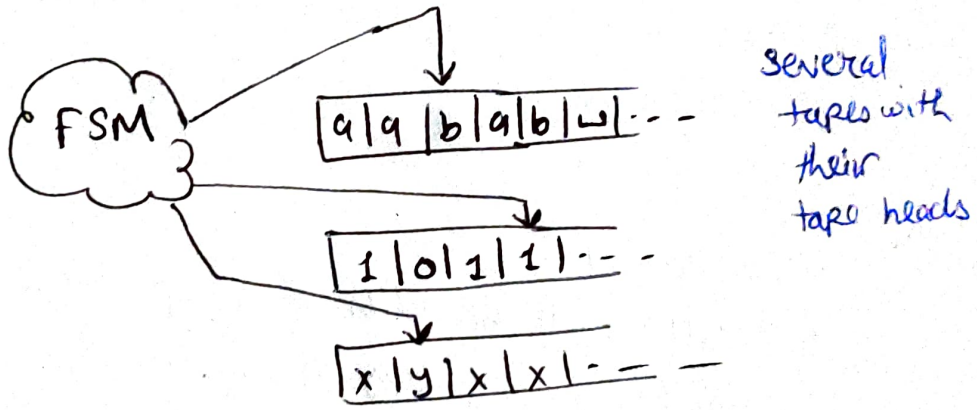
Theorem :- Every Multitape turing Machine has an equivalent Single tape turing Machine.

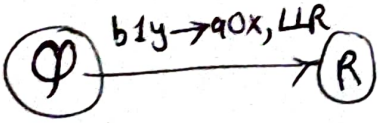
Proof → Given a multitape turing Machine show how to build a single tape turing Machine.

- Need to store all tapes on single tape
  Show data representation
- Each tape has tape head
  Show how to store that info
- Need to transform a move in multitape TM into one or more in single tape TM.

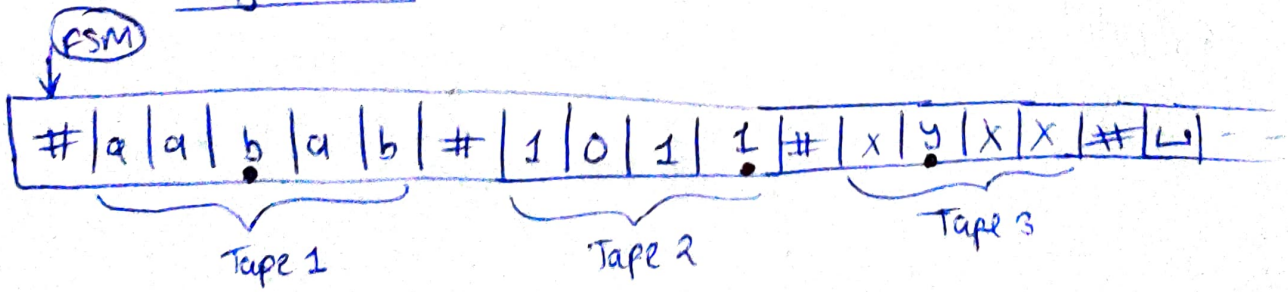
Multitape

k=3





↳ into Single tape



- Add dots where head 'k' is.
- To simulate a transition from state φ, we must scan our tape to see which symbols are under K Tape heads.

- Whenever one head moves off the right end, we must shift our tape so we can insert a ⊔.

Non determinism in Turing Machine

- $TM = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$

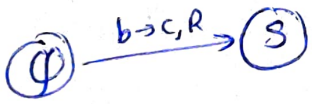
here only δ changes

for deterministic $\delta: Q \times \Sigma \to \Gamma \times (R/L) \times Q$

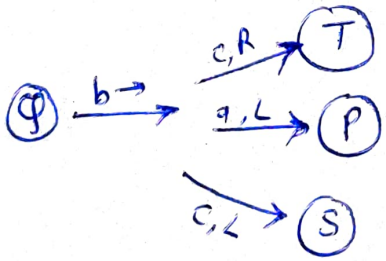
for non deterministic $\delta: Q \times \Sigma \to P\{\Gamma \times (R/L) \times Q\}$

↑ Power set

Deterministic



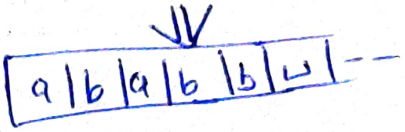
Non-Deterministic



for both configuration varies.

CONFIGURATION

- A way to represent entire state of TM at a moment during computation
- A string which computes:
  - > The current state
  - > current position of head
  - > entire tape contents

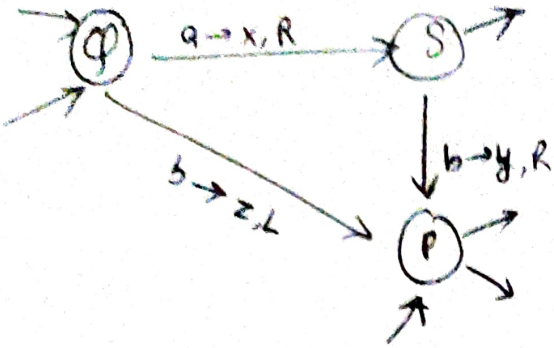


configuration

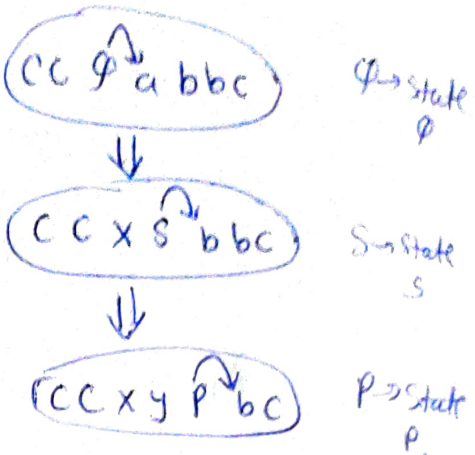
a b a Q b b

Q → current head State

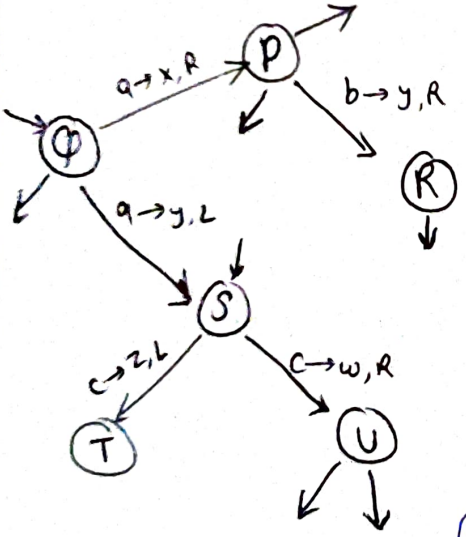
Deterministic TM (configuration)



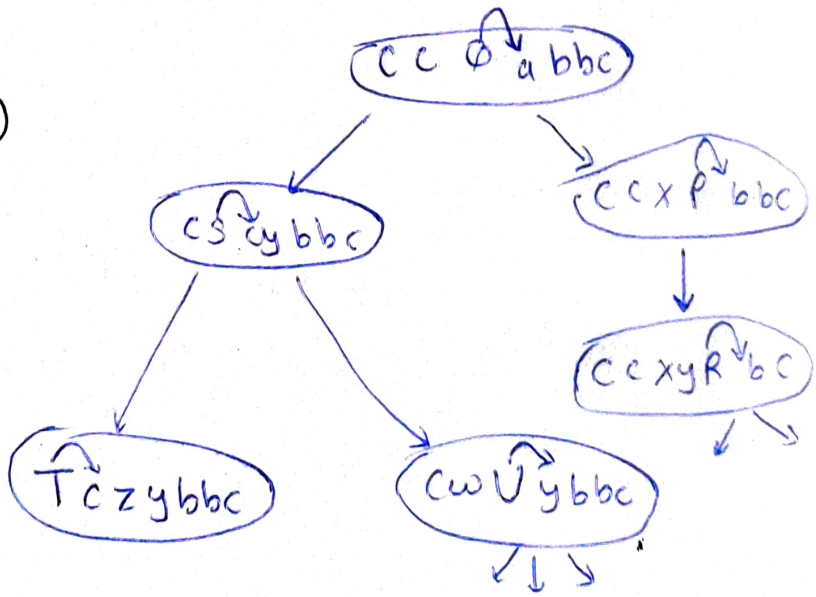
Computation History



Non-Deterministic TM (configuration)



Computation History



Outcomes of a Nondeterminism Computation

Accept → If any branch of computation accepts, then non deterministic TM will accept.

Reject → If all branches of computation HALT & REJECT then Nondeterministic TM rejects.

Loop → Computation continues but Accept is never encountered. Some branches in computation history are infinite.

Theorem : Every Non-deterministic TM has equivalent deterministic TM.

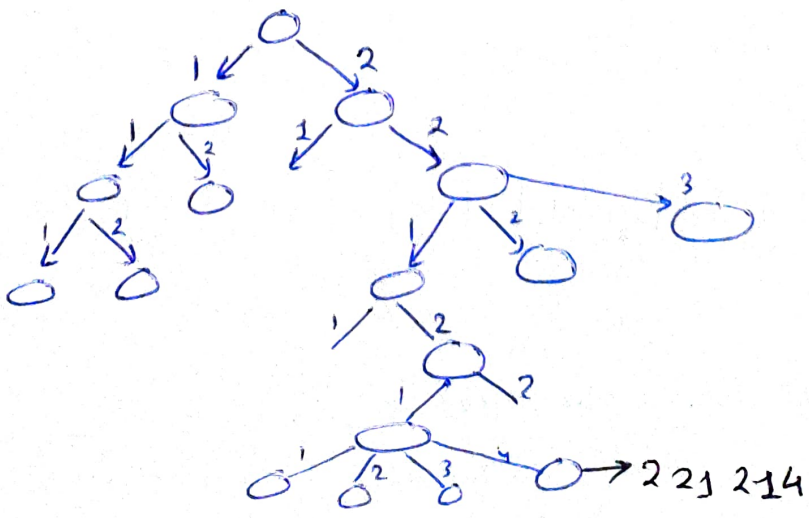
Proof :

- Given a Non-deterministic TM (N) show how to construct an equivalent Deterministic TM(D)
- If N accepts on any branch, then D will Accept
- If N halts on every branch without any ACCEPT, then D will halt & reject.

Approach :

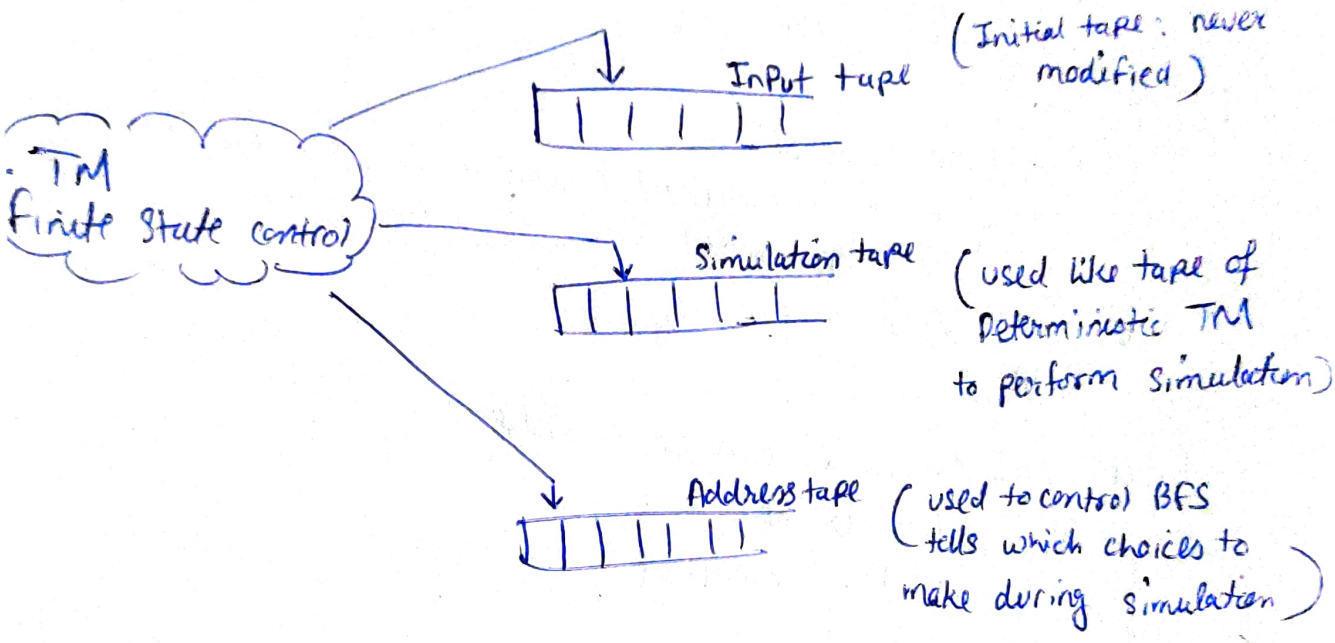
- Simulate N
- Simulate all branches of computation
- Search for any ways N can accept.

Let Computational History ⇒



So accepted state = Q2 1 Q14

- Search for tree looking to accept
- Search order
  → Breadth first search (BFS)
- The path no. tells which of many non-deterministic choices to make.



Algorithm

Initially : TAPE 1 contains INPUT
TAPE 2 & TAPE 3 are empty

- Copy Tape 1 to TAPE 2
- Run the simulation
- Use Tape 2 as 'the tape'
- When choices occur (when non-deterministic branch points are encountered)
- Tape 3 contains a Path. Each no. tells which choice to make

- Run simulation all the way down the branch as far as address / path goes (or computation dies)
- Try next branch
- Increment address on Tape 3
- Repeat

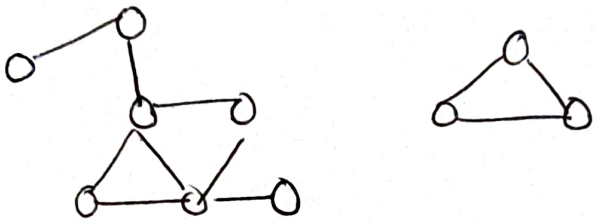
If ACCEPT is ever encountered → Halt & accept
If all branches reject or die out → Halt & reject

Turing Machine as Problem Solver

Any arbitary Problem can be expressed as a language
Any instance of problem is encoded into a string.

The string is in language ⇒ answer is YES
The string is not in language ⇒ answer is NO.

eg Q Is this undirected graph connected?



- We must encode problem into a language

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

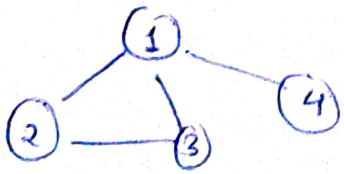
We would like to find TM to decide this languages

ACCEPT :- Yes graph is connected

REJECT → No graph not connected / or invalid graph

LOOP → this problem is decidable. Our TM will always halt

eg



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

List of nodes — Edges

$\{(,), 1, 2, 3, 4 \dots 0\}$

gh Level Algorithm

Select a node & mark it

REPEAT

for each node N

IF N is unmarked & 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

Then REJECT

END

ACCEPT

DECIDABILITY & Undecidability

Recursive Language

- A Language 'L' is said to be recursive if there exist a Turing machine which will accept all the strings in 'L' & rejects all strings not in 'L'.
- Turing machine will halt every time & give answer (accept or reject) for each & every string input.

Recursively Enumerable Language :

- A Language 'L' is said to be recursively enumerable if there exists a TM which will accept (& halt) for all input strings which are in 'L'.
- But may or may not halt for all input strings which are not in 'L'.

- Recognizable → (accept, reject, loop)
- Decidable (accept, reject).

⇒ **Decidable Language**

A Language 'L' is decidable if it is recursive language. All decidable Languages are recursive languages & vice-versa

⇒ **Partially Decidable language**

If it is Recursively Enumerable lang.

⇒ **Undecidable Language**

- it may sometimes be partially decidable but never decidable,
- There exist no TM for that language.

Universal Turing Machine

The language

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

is Turing Recognizable.

- Given description of TM & some input, can we determine whether machine accepts it?
  - Just simulate / Run the TM on input.

M accepts w : Our algorithm will halt & accept
M Rejects w : Our algo will halt & reject
M loops on w : our algo will not Halt.

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

Action → Simulate M
→ Behave just like M would (may accept, reject or loop)

- Like computer we input program & computer acts like that program so computer can be assumed as universal turing machine however TM has tape so computer is equivalent to universal TM.

Halting Problem

Given a Program, WILL IT HALT?

Can we design a TM which inputs a Program & tell will it halt or not?

Ans :- No, we can not make such generalized TM.

This is undecidable problem.

it is equivalent to.

- Given some program in some language (Java/c etc) will it ever get into infinite loop or will it always terminate?
  → undecidable.

→ Best thing we can do is run & program & see whether it halts.