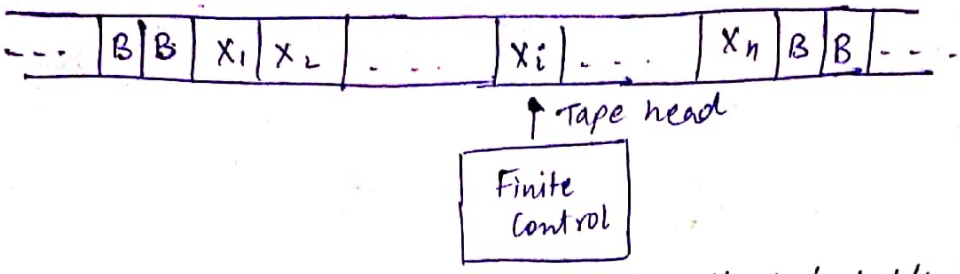
TURING MACHINE

Notation:



→ Initially, the tape is at the leftmost cell that holds the input.

In one, move, the T.M. will:

1) Change state [Maybe same].
2) Write a tape symbol in the cell symbol [Maybe the same symbol].
3) Move the head Left or Right. [Necessary]

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Q: set of states (finite)

Σ: Finite set of input symbols

Γ: complete set of tape symbols $(\Sigma \subseteq \Gamma)$

δ: Transition function:-

$$\delta(q, X) = (p, Y, D)$$

- q: state
- X: Tape symbol
- p: Next state
- Y: symbol ∈ Γ written
- $D \in \{L, R\}$

q_0: Start state

B: Blank symbol

F: Set of final or accepting states.

$F \subseteq Q$.

I.D:

$$X_1 X_2 X_3 \ldots X_{i-1}\, q\, X_i X_{i+1} \ldots X_n$$

1. q is the state of the TM
2. Tape head is scanning the i^{th} symbol from the left.
3. $X_1 X_2 \ldots X_n$ is the portion of the tape between the leftmost and the rightmost non blank.

Move a) $\delta(q, X_i) = (p, Y, L)$

$$X_1 X_2 \ldots X_{i-1}\, q\, X_i X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-2}\, p\, X_{i-1}\, Y\, X_{i+1} \ldots X_n$$

special case-1

1) $i = 1$

$$q X_1 X_2 \ldots X_n \vdash_M p B Y X_2 \ldots X_n$$

2) $i = n$ and $Y = B$

$$X_1 X_2 \ldots X_{n-1}\, q\, X_n \vdash_M X_1 X_2 \ldots X_{n-2}\, p\, X_{n-1}$$

b) $\delta(q, X_i) = (p, Y, R)$

$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$

Special Case:

1. $i = n$ $\quad X_1 X_2 \cdots X_{n-1} q X_n \vdash_M X_1 X_2 \cdots X_{n-1} Y p B$

2. $i = 1$ and $Y = B$

$q X_1 X_2 \cdots X_n \vdash_M p X_2 \cdots X_n$

e.g. $\{0^n 1^n \mid n \geq 1\}$

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1,X,Y,B\}, \delta, q_0, B, \{q_4\})$

δ :-

state \ symbol	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

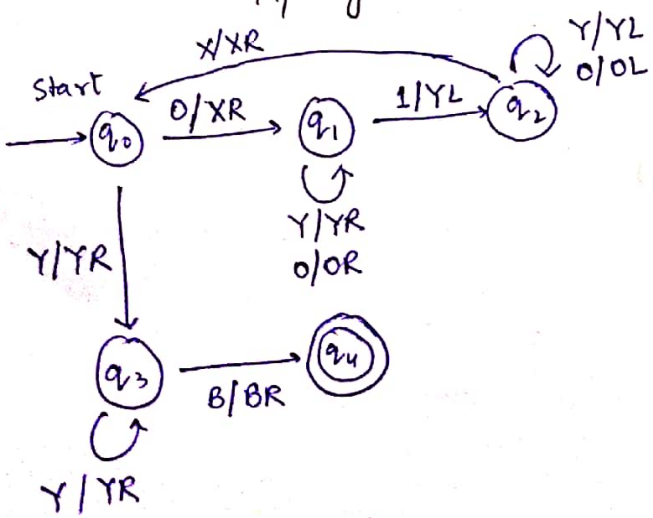
Transition Diagrams

Consists of :-

1) Set of Nodes corresponding to the states of the TM.

2) An arc from state q to state p is labelled by one or more items of the form X/YD whenever: $\delta(q, X) = (p, Y, D)$

X/YD — X: Tape symbols, Y: Tape symbols, D: Direction.



To use TM over integer-valued functions, the integers were represented in unary, as blocks of a single character, and the machine computed by changing the lengths of the blocks or by constructing new block elsewhere on the tape.

e.g. monus or proper subtraction:-

$$m \dot{-} n = \max(m-n, 0)$$

$M = (\{q_0, q_1, q_2 \dots, q_6\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B)$

we have omitted the 7th component (accepting states), as we are changing the tape for the output and not accepting some input.

$$..B\ 0^m\ 1\ 0^n\ B.. \longrightarrow ..B\ 0^{m \dot{-} n}\ B..$$

δ:-

State	symbol 0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	–	–	–

Language of a TM:-

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$L(M)$ is the set of strings $w \in \Sigma^*$ such that $q_0 w \vdash^* \alpha p \beta$ for some $p \in F$ and $\alpha, \beta \in \Gamma^*$

$L(M) = \{ w \in \Sigma^* \mid q_0 w \vdash^* \alpha p \beta,\ p \in F \text{ and } \alpha, \beta \in \Gamma^* \}$

↳ Recursively enumerable languages / RE Lang.

Turing Machines and Halting

→ acceptance by halting.

We say a TM halts if it enters a state q, scanning a tape symbol X, and there is no move in this situation, i.e. $\delta(q, X)$ is undefined.

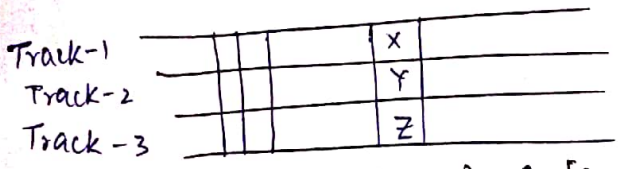
- TM always halts when it is in an accepting state.
- ★ TM which halt eventually, regardless of whether or not they accept are called Recursive.

Programming Techniques for TM

1) Storage in a state

→ finite control can also be used to hold a finite amount of data.

State	q		
Storage	A	B	C



e.g. $M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], B, \{[q_1, B]\})$

$L(M) = 01^* + 10^*$

Set of states Q is $\{q_0, q_1\} \times \{0, 1, B\}$. Considered as a pair.

a) $\{q_0, q_1\}$ → Remembers what the TM is doing.

q_0 :- M has not yet read its first symbol.

q_1 :- M has read the symbol, and is checking that it does not appear elsewhere, by moving right and hoping to reach a blank cell.

b) A Data portion, which rembers the first symbol seen.

$\{0,1\}$ → symbol

B → No symbol has been read.

δ:

1) $\delta([q_0, B], a) = ([q_1, a], a, R)$ for $a \in \{0,1\}$

2) $\delta([q_0, a], \bar{a}) = ([q_1, a], \bar{a}, R)$

3) $\delta([q_1, a], B) = ([q_1, B], B, R)$

2) Multiple Tracks

→ common use of multiple tracks is to treat one track as holding the data and a second track as holding a mask.

$L_{wcw} = \{wcw \mid w \in (0+1)^{*+}\}$

$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$

Q: $\{q_1, q_2, \ldots q_9\} \times \{0, 1, B\}$

Γ: $\{B, *\} \times \{0, 1, c, B\}$

B → Blank, * → checked

Σ: $\{[B,0], [B,1], [B,c]\}$

δ : $\exists a, b \in \{0,1\}$

1. $\delta([q_1, B], [B, a]) = ((q_2, a), [x, a], R)$
2. $\delta([q_2, a], [B, b]) = ([q_3, a], [B, b], R)$
3. $\delta([q_3, a], [B, c]) = ([q_3, a], [B, c], R)$
4. $\delta([q_3, a], [x, b]) = ([q_3, a], [x, b], R)$
5. $\delta([q_3, a], [B, a]) = ([q_4, B], [x, a], L)$
6. $\delta([q_4, B], [x, a]) = ([q_4, B], [x, a], L)$
7. $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$
8. $\delta([q_5, B], [B, a]) = ([q_6, B], [B, a], L)$
9. $\delta([q_6, B], [B, a]) = ([q_6, B], [B, a], L)$
10. $\delta([q_6, B], [x, a]) = ([q_1, B], [x, a], R)$
11. $\delta([q_5, B], [x, a]) = ([q_7, B], [x, a], R)$
12. $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$
13. $\delta([q_8, B], [x, a]) = ([q_8, B], [x, a], R)$
14. $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], R)$

3) Subroutines

→ A set of states that perform some useful process.
- Includes a start state and another state that temporarily has no moves, and serves as the "return" state to pass control to whatever other set of states.

the call of a subroutine occurs whenever there is a transition to its initial state.

e.g. Multiplication.

$0^m 1 0^n 1 \longrightarrow 0^{mn}$

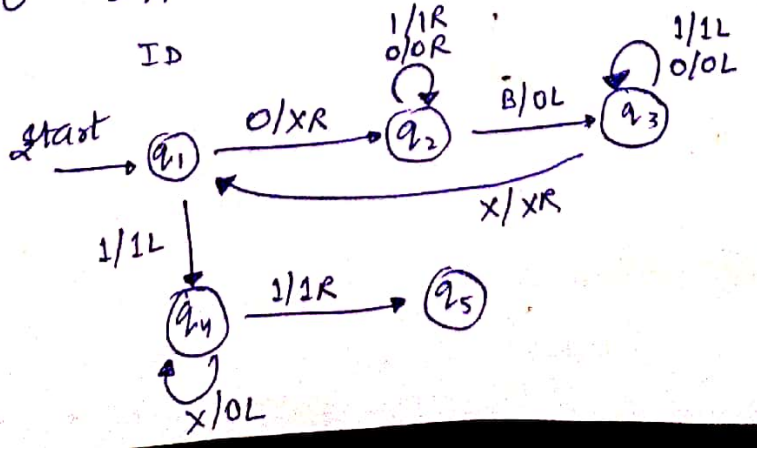
#1 $0^i 1 0^n 1 0^{kn} \longrightarrow 0^{i-1} 1 0^n 1 0^{(k+1)n}$

#2 $10^n 1 \longrightarrow$ Blanks.

we will implement #1 as a subroutine.

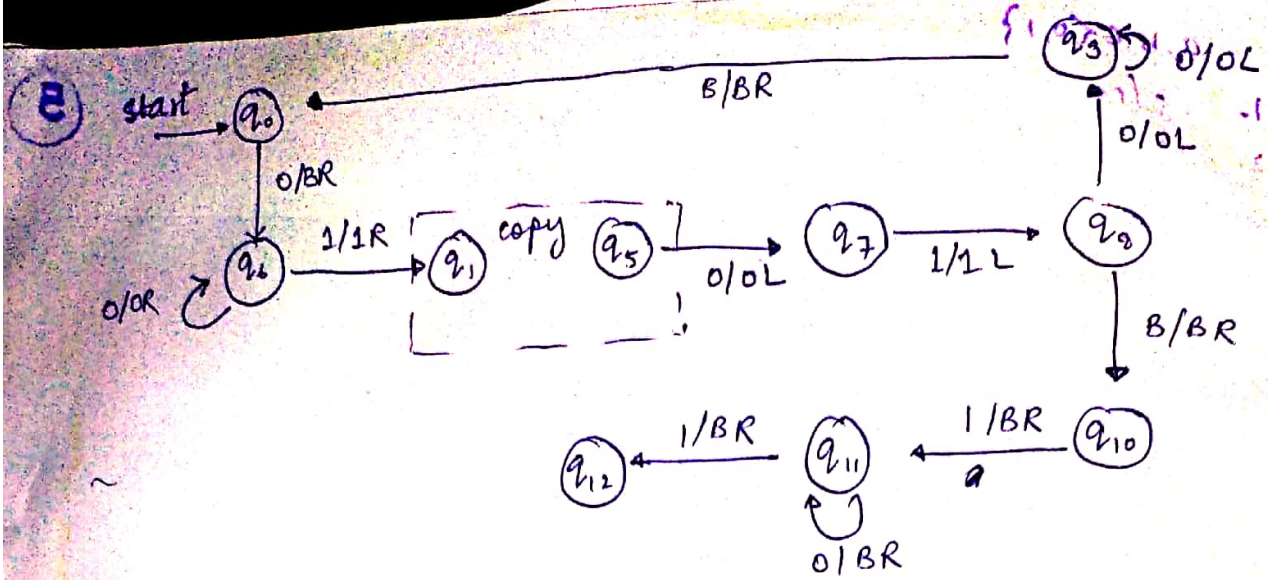
Copy:- $0^{m-k} 1 q_1 0^n 1 0^{(k-1)n} \longrightarrow 0^{m-k} 1 q_5 0^n 1 0^{kn}$

ID



ID

Marks first 0 with an X, moves right in state q_2 until it finds a blank, copies 0 there, and moves left in state q_3 to find the marker X.
↳ Repeats this and finally all X to 0



→ q_0, q_6 converts ID $q_0 0^m 10^n$ to ID $0^{m-1} 1 q_1 0^n$.

→ then the subroutine is called

→ q_7, q_8, q_9 take control after copy to convert ID $0^{m-k} 1 q_5 0^n 10^{kn}$ to ID $q_0 0^{m-k} 10^n 10^{kn}$.

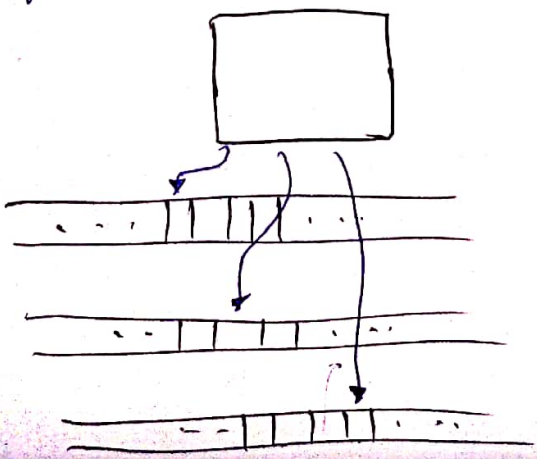
→ $q_8, q_{10}, q_{11}, q_{12}$ converts $10^n 1$ to Blanks.

in state q_{12}, the TM halts with ID $q_{12} 0^{mn}$

Extensions to the Basic Turing Machine

→ Multitape TM

1) Input is finite sequence of input symbols placed on the first tape.
2) All other cells of all the tapes hold the blank.
3) finite control is in the initial state.
4) head of the first tape is at the left end of the input.
5) All other tape heads are at some arbitrary cell; all cells of these tapes "look" the same.



Move of Multitape TM

1) The control enters a new state [maybe same as previous].
2) on each tape, a new symbol is written on the cell scanned. [Maybe same as previous].
3) Each of the tape heads make a move $\in$ {Left, Right, stationary} ★
   Independent movement.

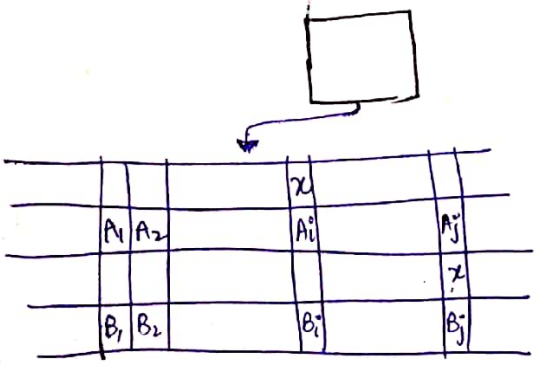
— accept by entering an accepting state.

Equivalence of One-Tape and Multitape TM

Theorem: Every lang. accepted by a multitape TM is recursively enumerable.

Proof: suppose a lang. L is accepted by a k-tape TM M.
We simulate M with a one-tape TM N whose tape we think of as having 2k tracks.
- Half of these hold the tapes of M
- other half of the tracks each hold only a single marker that indicates where the head for the corresponding tape of M is currently located.

e.g.



simulation of a two-tape TM by a one-tape TM.

Simulate a move of M :-

→ N's head must visit the k head markers. So that N not get lost, it must remember how many head markers are to its left at all times; that count is stored as a component of N's finite control.

→ After visiting each head marker and storing the scanned symbol in a component of its finite control, N knows what tape symbols are being scanned by each of M's heads.

→ N also knows the state of M, which it stores in N's own finite control.

→ thus N knows what move M will move.

→ N now revisits each of the head markers on its tape, changes the symbol in the track representing the corresponding tapes of M, and moves the head markers Left or right, if necessary.

→ Finally N changes the state of M as recorded in its own finite control. At this point, N has simulated one move of M.

→ We select as N's accepting states all those states that record M's state as one of the accepting states of M.

→ Thus, whenever the simulated M accepts, N also accepts, and N does not accept otherwise.

Running Time and the Many-Tapes-to-One Construction

Running Time of TM M on input w is the number of steps that M makes before halting.

If M doesn't halt of w, then the running time of M on w is ∞.

Time Complexity of TM M : T(n) :- maximum, over all inputs w of length n, of the running time ~~time~~ of M on w.

Argument : Running times of a one-tape and multitape TM's are within a square of each other.

Theorem : The time taken by the one-tape TM N to simulate n moves of the k-tape TM M is $O(n^2)$.

Proof :- After n moves of M, the tape head markers cannot have separated by more than 2n cells.

Thus if N starts at the left-most marker, it has to move no more than 2n cells right, to find all the head markers.

→ It can then make an excursion leftward, changing the contents of the simulated tapes of M, and moving head markers left or right as needed.

Doing so require no more than 2n moves left, plus at most 2k moves to reverse direction and write a marker X in the cell to the right [If tape head of M moves Right]

Thus, to simulate one more of M, we require $4n+2k$ moves [k is

Hence to simulate n moves — $O((4n+2k)n) = O(n^2)$

Nondeterministic TM

→ Differs from Deterministic TM by having a transition function δ such that for each state q and tape symbol X,

$$\delta(q,X) = \{ (q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots (q_k, Y_k, D_k) \}$$

$$L(M) = \{ w \in \Sigma^* \mid q_0 w \vdash^* \alpha p \beta, \quad p \in F \quad \alpha, \beta \in \Gamma^* \}$$

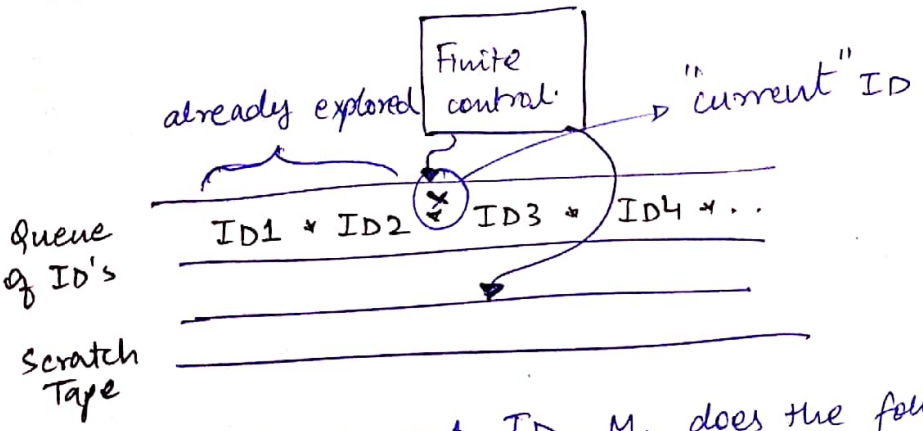
NTM ≃ DTM

→ Proof: for every NTM M_N, we can construct DTM M_D that explores the ID's that M_N can reach by any sequence of choices.

If M_D finds one that has an accepting state, then M_D enters an accepting state of its own. M_D must be systematic, putting a new ID's on a queue, rather than a stack.

Theorem: $L(M_N) = L(M_D)$

Proof: M_D : multitape TM



To process the current ID, M_D does the following:-

1. M_D examine the state and scanned symbol of the current ID. Built into the finite control of M_D is the knowledge of what choices of move M_N for each state and symbol. If the state in the current ID is accepting then M_D accepts and simulates M_N no further.

2. If the state is not accepting, and the state-symbol combination has k moves, then M_D uses its second tape to copy the ID and then make k copies of that ID at the end of the sequence of ID's on tape-1.

3. M_D modifies each of those k ID's according to a different one of the k choices of move that M_N has from its current ID.

4. M_D returns to the marked, current ID, erases the mark, and moves the mark to the next ID to the right.

Then repeat with step (1).

⇒ if M_N enters an accepting state than M_D also enters into an accepting state.

Suppose that m is the maximum number of choices M_N has in any configuration.

then in n moves, M_N can reach at most $1 + m + m^2 + .. + m^n$ ID's.

$$1 + m^1 + m^2 + .. + m^n \leq n.m^n + 1$$

→ M_D explores ID's of M_N is a "breadth first" manner.

thus M_D will considered the accepting ID of M_n among the first $n.m^n + 1$ ID's that it considers.

Since this is a finite bound, and M_D considers this hence M_D accepts the string if M_n accepts.

and we have already observed that if M_D accepts only because M_n accepts.

⇒ $L(M_N) = L(M_D)$. H.P.

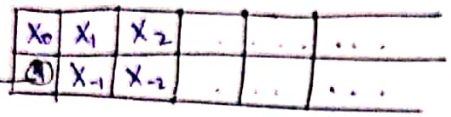
Restricted TM

1) TM with Semi-Infinite Tapes :-

→ can simulate our original TM model, infinite in both directions.

★ Trick :- ① Use two tracks
- → Upper track represents the cell of the original TM that are at or to the right of the initial head pos.
- → Lower track represents the positions left of the initial position, but in reverse order.

serves as endmarker and prevents the head to fall off the left end.



② It never writes a blank.

Theorem: Every lang. accepted by a TM M_2 is also accepted by a TM M_1 with the following restrictions:

1. M_1's head never moves to the left of its position
2. M_1 never writes blank.

PROOF :- For ②, create a new symbol B' that acts as a blank but is Not blank. i.e.:-

a) If M_2 has $\delta_2(q, X) = (p, B, D)$, change this to $\delta_2(q, X) = (p, B', D)$

b) $\delta_2(q, B')$ be same as $\delta_2(q, B)$ $\forall q \in Q$.

For ①, Let $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$, construct M_1 as

$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B, B], F_1)$

where :-

$Q_1 \rightarrow \{q_0, q_1\} \cup (Q_2 \times \{U, L\})$

↳ tells us whether the Upper or lower track is being scanned by M_2.

similar to saying that, U means head of M_2 is at or to the right of its initial position and L means it is to the left of that position.

Γ_1 : $\Gamma_2 \times \Gamma_2$ → tape symbols and input symbols :- $[a, B]$ where $a \in \Sigma$

Blank in M_1 is $[B, B]$. Additionly for $\forall X \in \Gamma_2$, $\exists [X, \#]$ in Γ_1 ← Not in Γ_2 ← New Symbol

serves to mark the left end.

δ_1:

1. $\delta_1(q_0, [a, B]) = (q_1, [a, *], R)$ for any $a \in \Sigma$.
   → First move of M, puts * marker in the lower track of the leftmost cell and change state to q_1 and move right.

2. $\delta_1(q_1, [X, B]) = ([q_2, U], [X, B], L)$ for $\forall X \in \Gamma_2$
   ($[q_2, U]$: initial state of M_2 with attention focused on upper track of M_1; Returning the head to its initial position)

3. If $\delta_2(q, x) = (p, Y, D)$, then for $\forall Z \in \Gamma_2$
   (a) $\delta_1([q, U], [X, Z]) = ([p, U], [Y, Z], D)$
   (b) $\delta_1([q, L], [Z, X]) = ([p, L], [Z, Y], \overline{D})$ ↳ Direction opposite to D.

4. If $\delta_2(q, x) = (p, Y, R)$ then
   $\delta_1([q, L], [X, *]) = \delta_1([q, U], [X, *])$
   $= ([p, U], [Y, *], R)$

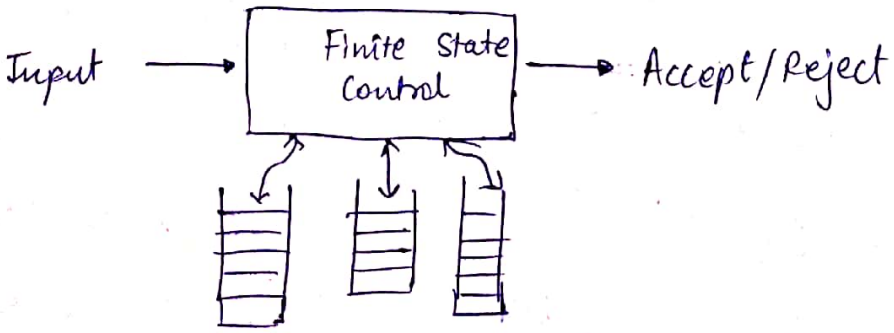
5. If $\delta_2(q, x) = (p, Y, L)$ then
   $\delta_1([q, L], [X, *]) = \delta_1([q, U], [X, *]) = ([p, L], [Y, *], R)$

F_1: $F_2 \times \{U, L\}$

Proof can be observed further. [As stated in TextBook :(]

2) Multistack Machines

↳ with two ~~stacks~~ or more stacks



★ A k-state machine is a deterministic PDA with k-stacks.

A move on multistack machine is based on:-

1. The state of the finite control
2. The input symbol read. Can make a move using ε alphabet but to make it deterministic there cannot be any ε-move or a non-ε-move in any situation.
   ↓ choice of an
3. the Top stack symbol of each

In one move, the multistack machine can:-

a) Change to a new state

b) Replace the top symbol of each stack with a string of zero or more stack symbols.

Thus a typical transition rule of a k-stack machine looks like:-

$$\delta(q, a, X_1, X_2, \ldots X_k) = (p, \gamma_1, \gamma_2, \ldots \gamma_k)$$

→ we assume that there is a special symbol \$, called as the endmarker, that appears only at the end of the input and is not part of that input. → Allows us to know when we have consumed all the available input.

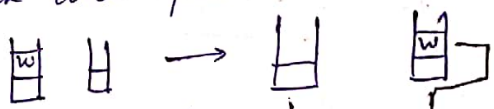
Theorem: If a lang. L is accepted by a TM, then L is accepted by a 2-stack machine.

Proof:- **Idea:** one stack contains what is to the left and other contains what is to the right of the head.

1) S begins with a bottom-of-stack marker on each stack. a "stack is empty" when it contains only the bottom-of-stack marker.

2) Suppose the input is w\$; S copies w into its first stack, ceasing to copy when it reads the end-marker on the input.

3) S pops each symbol from first stack and pushes it into 2nd stack. thus if input:- w\$



4) S enters the (simulated) start state of M.

M has nothing but blanks to the left of the cell scanned by its tape head

w appears at end to the right of the cell scanned by M's head.

5) S simulates a move of M as follows

(a) S knows the state of M, say q, due to finite control

(b) S knows the symbol X scanned by M's tape head; it is the top of the second stack. If it contains bottom-of-stack (only) marker then M moved to a blank.

(c) thus, S knows the next move of M.

(d) The next state of M is recorded in a component of S's finite control, in place of the previous state.

(e) if M replaces X by Y and moves right, then S pushes Y onto its first stack. X is popped off the second track of S.

However there are 2 exceptions:-

(1) If 2nd stack has only bottom-of-stack marker, then the second stack is not changed. (M has moved to get another blank further to the right).

(2) If Y is blank, and the first stack is empty, then the stack remains empty. the reason is that there are still blanks to the left of M's head.

(f) if M replaces X by Y and moves Left, S pops the top of the first stack, say Z, then replaces X by ZY on the second ~~track~~ stack. if Z is bottom-of-stack then M must push BY onto the second stack and not pop the first stack.

(g) S accepts if the new state of M is accepting, otherwise S simulates another move of M in the same way.

3) Counter Machines

Way-1:- ~~Same~~ Same structure as Multistack Machine, but in place of each stack is a counter. Counters hold any non negative integer, but we can only distinguish b/w zero and non-zero counters. That is, the move of the counter machine depends on its state, input symbol, and which, if any, of the counters are zero. In one move, the counter machine can:-

(a) Change state

(b) Add or subtract 1 from any of its counters, independently. value of counter ≥ 0.

Way-2:- Restricted Multistack Machine:-

(a) There are only 2 stack symbols Z_0 (the bottom-of-stack) & X.

(b) Z_0 is initially on each stack.

(c) $Z_0 \leftrightarrow X^i Z_0$ for some $i \geq 0$

(d) $X \leftrightarrow X^i$ for some $i \geq 0$.

$X^i Z_0 \not\equiv i$

$Z_0 \not\equiv 0$

cannot distinguish $X^i Z_0$, $X^j Z_0$.

Power of Counter Machines

1) Recursively enumerable
2) Every Lang. accepted by a one-counter machine is a CFL. the Lang. of one counter machines are accepted by DPDA's, Proof is extremely difficult.
3) two counters are enough to simulate a TM and therefore to accept every recursively enumerable Lang.

Theorem: Every recursively enumerable Lang. is accepted by a 3-counter machine.

Proof:- Begin with:- every recursively enumerable Lang is accepted by a two-stack machine.

Suppose there are $r-1$ tape symbols used by a stack machine. we may identify the symbols with the digits 1 through $r-1$, and think of a stack $X_1 X_2 \dots X_n$ as an integer in base r.

(X_1: top, X_n: bottom)

$$\Rightarrow X_n r^{n-1} + X_{n-1} r^{n-2} + \dots + X_2 r + X_1 \quad \star$$

We use two counters to hold the integers that represent each of the two stacks. the third counter is used to adjust the other two counters.
↳ in particular when we multiply or divide a count by r.

1) **Pop the stack:** $i \longrightarrow i/r$.
starting with the third counter at 0, we repeatedly reduce count i by r, and increase the third counter by 1. when the counter reaches 0, we stop. then, we repeatedly increase the original counter by 1 and decrease the third counter by 1, until the third counter becomes 0 again. At this time, i got converted to i/r.

2) to change X to Y on the top of a stack that is represented by count i :- we increment or decrement i by a small amount, surely no more than r. if $Y > X$, as digits, increment i by $Y - X$; if $Y < X$ then decrement i by $X - Y$.

3> To push X onto a stack that initially holds i :-
we need to replace i by $ir + X$.

→ repeatedly decrement the count i by 1 and increase the counter (which starts from 0, as always), by r. When the original counter becomes 0, we have ir on the third counter. Copy the third counter to the original counter. Finally, we increment the original counter by X.

* Initialize the counter to simulate the stacks in their initial condition. this step is accomplished by incrementing the two counters involved to small integer, whichever integer from 1 to r-1 corresponds to the start symbol.

Theorem: Every recursively enumerable lang is accepted by a 2-counter machine.

Proof: idea: Represent the three counters say i, j, k by a single integer :- $m = 2^i 3^j 5^k$. One counter will hold this number, while the other is used to help multiply or divide m by one of the three primes: 2, 3 or 5.

1> Increment i, j, and/or k.
To increment i by 1, we multiply m by 2. we already saw how to multiply by any constant r. similarly for j→3 and for k→5.

2> Tell, which, if, any of i or j or k = 0.
To tell if i=0, we must determine whether m is divisible by 2, Copy m into the second counter, using the state of the counter machine to remember whether we have decremented m an even or odd number of time. if we have decremented m an odd number of times when it becomes 0, then i=0. We then restore m by copying the second counter to the first. Similarly for j=0 (use 3 states) and for k=0 (use 5 states).

3> Decrement i, j and/or k.
We have already seen how to divide a count m by r previously. so to decrement i, j, k use r = 2, 3, 5. if $m \% r \neq 0$ then the 2-counter machine halts without accepting.

TURING MACHINES AND COMPUTERS

1) A computer can simulate a TM.
2) A TM can simulate a computer, and in some polynomial time,

Simulating a TM by computer:-

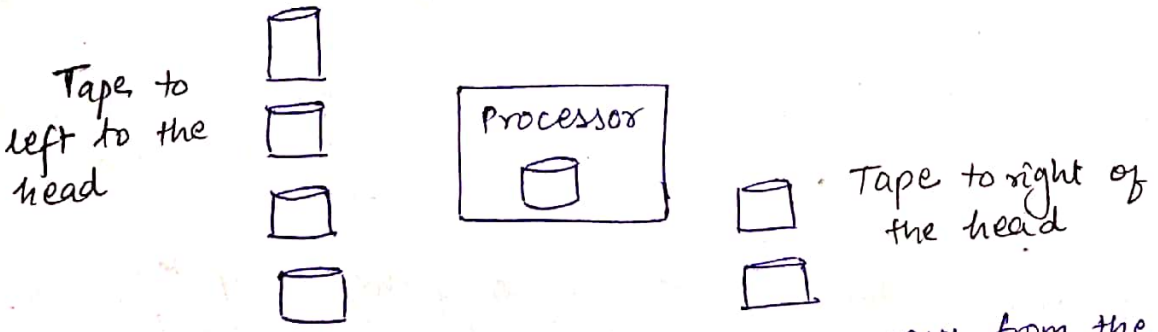
→ We must write a program that acts like a TM.

Finite control: since these are finite states and finite transitions, we can encode states as strings and use a table of transitions. Likewise the Tape symbols can be encoded as character strings of a fixed length.

Main Question: Since the Tape can grow infinitely long, can we simulate the TM using a computer which has finite memory, disk and other storage devices.

If not, then the computer would be a finite automaton and the Lang. that it accepts are regular. Since the disks used to store are replaceable, let us assume that the computer has ~~a~~ no limit on how many disk it can use.

Then we can arrange the disks in two stacks:-



The further down the stacks, the further away from the tape head. If the tape ~~of~~ head of the TM moves sufficiently far to the left that it reaches cells that are not represented by the disk currently mounted in the computer, then it prints a message "swap left". The currently mounted disk is removed by a human operator and placed on the top of the right stack. The disk on top of the left stack is mounted on the computer, and computation resumes.

Similarly for "swap right".

If either stack is empty, then the TM has entered an all-blank region of the tape. In that case, the human operator has to buy another disk.

Simulating a computer by a TM

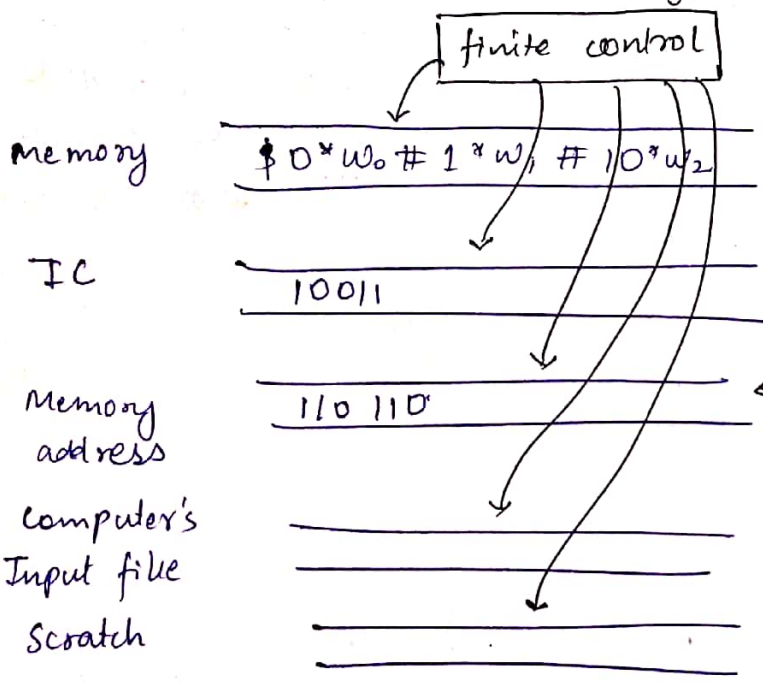
a) First, we shall suppose that the storage and the number of words have no limit.

b) we assume that the program of the computer is stored in some of the words of memory. And "indirect ~~addressing~~ addressing" is permitted.

c) we assume that each instruction has finite number of words, and the instruction can change the value of atmost one word.

d) No restriction on ocurrences of an operation in some specified memory. The relative speed of operations on different words will not be taken into account.

A TM that simulates a typical computer :-



$:- beginning of sequence

*, # are used to make it easy to find the ends of addresses and contents, and to diff b/w binary string has addres or contents.

Our TM will simulate the instruction cycle of the computer.

1) search the first tape for an address that matches the instruction number on tape 2.
2) examine the value
3) Gather any value, if required, from tape 1 to scratch-tape.
4) Execute instruction like copying it to some other address, Adding two values, 'Jump' instructions.

The Scratch tape can be used to perform arithmetic operations.

The actual proof is quite long since it considers every possible operation a computer can do.

Comparing Running Time of a Computer and TM

→ We need to assure that if a problem can be solved on a computer in polynomial time then it can be solved in some polynomial time on a TM too.

Theorem: If a Computer:

1.) Has only instructions that increase the maximum word length by at most 1, and

2.) Has only instructions that a multitape TM can perform on words of length k in $O(k^2)$ steps or less,

the the TM can simulate n steps of the computer in $O(n^3)$ of its own steps.

Proof: there is a constant 'c' that is the largest of the computer's words and addresses appearing in the program. There is also a constant 'd' that is the number of words occupied by the program.

Thus after executing n steps, the computer cannot have created any words longer than $c+n$, and therefore, it cannot have created or used any addresses that are longer than $c+n$ bits either. the total No. of addresses created is at most $d+n$, since in each step the computer can create atmost one address. Since each address-word combination requires at most $2(c+n)+2$ bits. ($2(c+n)$: address, the contents; $+2$: two marker symbols to separate.)

The total number of TM tape cells occupied after n instructions have been simulated is at most $2(d+n)(c+n+1) \longrightarrow O(n^2)$.

We now know that each of the fixed number of lookups of addresses involved in one computer instruction can be done in $O(n^2)$ time [$O($length of Tape$)$].

Since there are words and each word is $O(n)$ length, by second assumption, the instruction themselves can be carried out in $O(n^2)$ time.

The only significant, remaining cost of an instruction is time it takes the TM to create more space on its tape to hold a new or expanded word which involves shifting and copying that can be done in $O(n^2)$ time using scratch-tape.

Thus, we conclude that only $O(n^2)$ time is required for one step and for n steps we need $O(n^3)$ time.

Hence - Proved.

Theorem: A computer described above can be simulated for n steps by a one-tape TM; using at most $O(n^6)$ steps of the TM.