UNIT-4

UNIT-4
Lecture Notes

Turing Machines

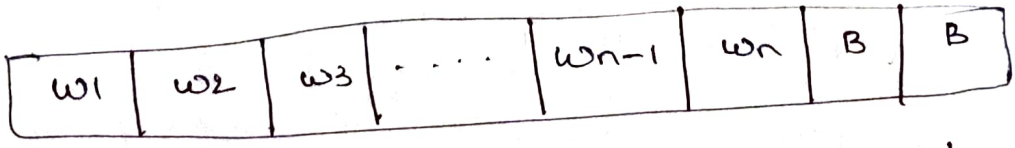
Turing machines are introduced by Alan Turing in 1937

These are simple abstract computational devices.

→ A task is computable if one can specify a sequence of instructions which when followed result in the completion of the task. Such a set of instructions is called an Effective procedure or algorithm for the task.

→ Allan modelled a TM as a machine with a finite no. of control states and an infinite tape bounded at the left and stretching off to the right. The tape is divided into the cells each of which can hold one symbol.

The input of the machine is a string $w = w_1 w_2 \cdots w_n$ initially written on the left most portion of the tape, followed by an infinite sequence of blanks B.



The machine is able to move read/write head to left or right over the tape as it performs its computation. It can read and write the symbols on the tape.

formal Definition:

A turing machine is 7tuple $(Q, \Sigma, T, \delta, q_0, B, F)$

where

Q – is a finite set of states

Σ → input Alphabet set

T → tape symbols, every symbol of Σ is in T and blanks also

δ → $Q \times T \rightarrow Q \times T \times \{L, R\}$ is the translation function, where L and R are directions.

$q_0 \in Q$ → starting state

B → is special symbol indication of a black cell.

F → is a finite set of final states, where TM halts on reaching these states.

when a transition $\delta(q_i, a) = (q_j, b, d)$ is made by a TM, M then the following happens.

* M writes "b" to the current tape cell, over writing "a"
* The current state changes from q_i to q_j
* The tape head moves either to left or to right depending on whether d is L or R.

If the given number is 11110, then 1's complement is 00001.

From the above examples, it is clear that the TM has to be designed such that while moving to right, if the input is 1, it should be changed to 0 and if the input is 0 it should be changed to 1 and halt on B.

fig: TM for 1's complement

	a=0	a=1	a=B
$\rightarrow q_0$	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_A, B, L)
q_A	—	—	—

Techniques for Turing Machine construction:- Programming

A TM has a finite number of states in its CPU. However the number of states is not always small.

We use the finite control not only to represent a position in the "program" of the Turing machine, but to hold a finite amount of data.

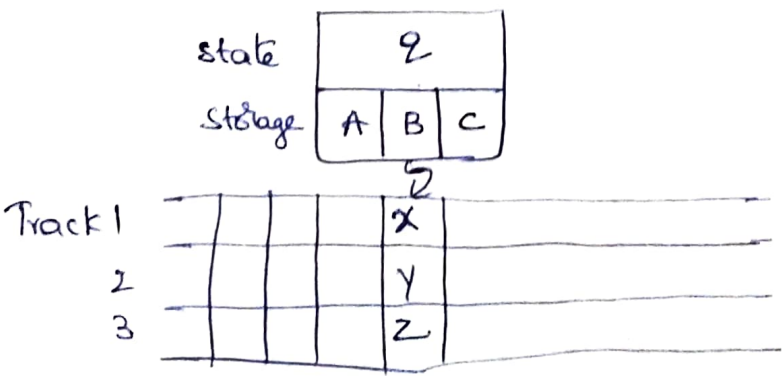


fig:- A TM viewed as having finite control storage and multiple tracks.

In the figure, we see the finite control consisting of only a "control" state q, but three data elements A, B, C.

The technique requires no extension to the TM model; we merely think of the state as a tuple.

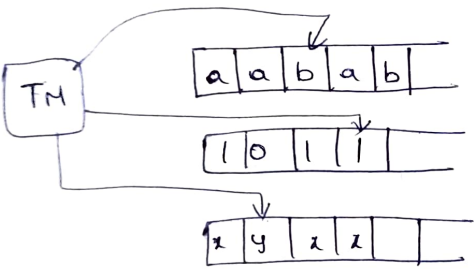
$[q, A, B, C]$ → tuple state

Equivalence of Multi tape and single tape Turing Machine

Theorem :- Every Multitape Turing Machine has an Equivalent single Tape Turing Machine.

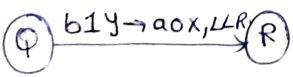
Proof: Given a Multitape TM show how to build a single tape TM.

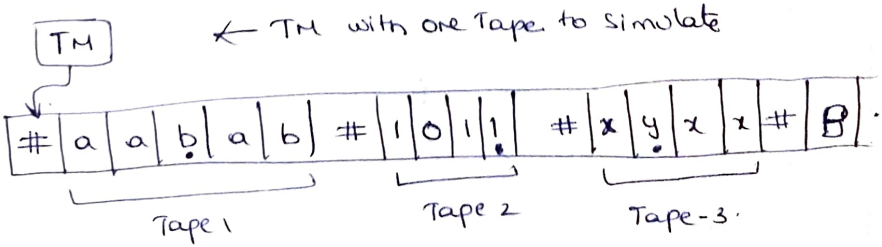
* Need to store all tapes on a single tape
  show data representation
* Each tape has a tape head
  show how to store that information
* Need to transform a move in the Multitape TM into one or moves in the single Tape TM



K = 3 Tapes.

fig: Multi tape TM.





* Add "dots" to show where "k" is

(using marking we can put dots on tape)

* To simulate a transition from state q, we must scan our tape to see which symbols are under K tape heads.
* once we determine this and are ready to make the transition, we must scan across the tape again to update the cell and move the dots.
* when ever one head moves off the right end, we must shift our tape so we can insert a Blank symbol

Techniques for Turing Machine

1. Storage in finite control
2. Multiple tracks
3. Subroutines

(1) ⇒ Storage in finite control

It holds finite information, to do so state consists of pair of elements: 1) Exercising control
2) Storing a symbol.

If we consider $w_1 = 0111111\cdots$

Single 0 is followed by all 1's.

If the string is 011011 → (X) invalid.

Consider $[q_0, 1]$ $[q_1, 1]$

$[q_0, 0]$ $[q_1, 0]$

$[q_0, B]$ $[q_1, B]$

(1) $\delta([(q_0, B), 0]) = ([q_1, 0], 0, R)$ $([q_1, 0], B) = ((q_A, 0), B, L)$.

$\delta([(q_1, 0), 1]) = ((q_1, 0), 1, R)$

Multi Track Turing Machine

* Multi-track Turing machines, a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks
* Here, a single tape head reads n symbols from n tracks at one step.
* It accepts like a normal single-track single-tape TM accepts

A Multi track Turing machine can be formally defined as

6 Tuple $(Q, T, \Sigma, \delta, q_0, F)$ where

Q - set of finite states

T - Tape alphabet set

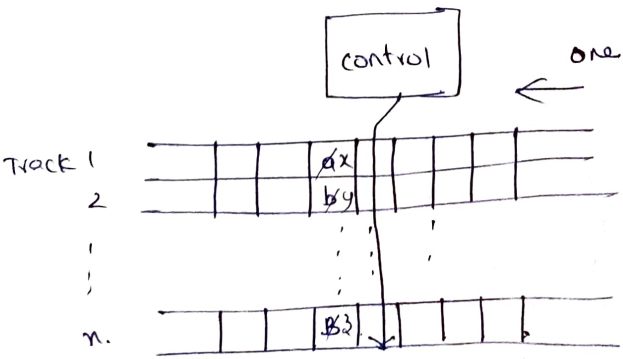
Σ - Input Alphabet set

δ - is a relation on states and symbols where

$\delta(q_i, [a_1, a_2, a_3 \cdots]) = [q_j, [b_1, b_2, b_3 \cdots], \text{Left-shift or Right-shift}]$

q_0 - Initial state

F - set of final states



One tape head to read K symbols from the K track at one step.

sample δ transition

$(q, a, b, B) = (q_{next}, x, y, z, L/R)$

fig:- Multi track TM.

Subroutines

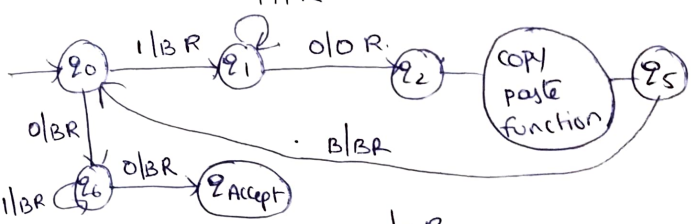
→ A Turing Machine can simulate any type of subroutine found in programming languages, including recursive procedures and any of the known Parameter passing mechanisms.

→ We can design a TM program to serve as a subroutine. It has a designated initial and a return state. These states are ~~is~~ used to indicate the call to a subroutine and return to caller subroutine.

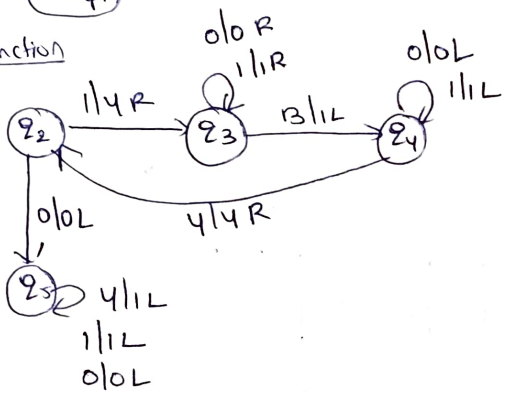
Prob: Design multiplication in TM using subroutines

Input – 2 X 3 ⇒ 1 1 0 1 1 1 performing repeated addition. (copy - paste function)

Output – 1 1 1 1 1 1.



COPY function

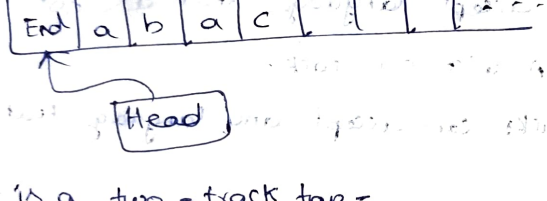


Restricted Turing Machine.

Now, we shall consider some examples of restrictions on the TM that also give exactly the same language recognizing power.

(a) Turing machines with semi-infinite Tapes:

TM with a semi-infinite tape has a left end but no right end. The left end is limited with an end marker.



It is a two-track tape -

* Upper track - It represents the cells to the right of the initial Head position.
* Lower track - It represents the cells to the left of the initial Head position in reverse order.

The infinite length input string is initially written on the tape in the contiguous tape cells.

The machine starts from the initial state, q_0 and the Head scans from the left end marker 'End'.

The exact arrangement is the upper track represents the cells $a_1b_1c_1d_1e_1\ldots$ where 'a' is the initial position of the Head.

$b_1, c_1, d_1 \ldots$ and so on, are the cells to the right.

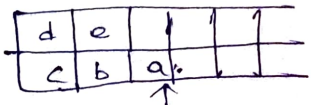


fig: semi-infinite tape with 2 track

* Theorem:

Every language accepted by a Turing Machine M_2 is also accepted by a TM M_1 with the following restrictions:

* M_1's head never moves left of its initial position, and
* M_1 never writes a blank.

(b) Multi stack Machines

Consider several computing models that are based on generalizations of the pushdown automaton.

→ Turing Machines can accept languages that are not accepted by any PDA with one stack.

→ But PDA with 2 stacks can accept any language that a TM can accept.

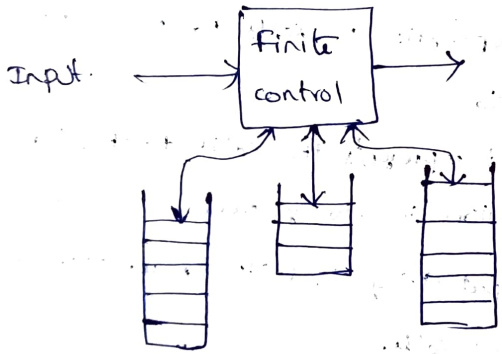


fig:- A Machine with three stacks

→ A K-stack machine is a deterministic PDA with K-stacks

→ It obtains its input from an input source rather than having the input placed in a tape

→ It has a finite control, which is in one of a finite set of states

It has a finite stack alphabet, which it uses for all its stacks.

A move of a multi stack machine is based on:

* the state of the finite control
* the input symbol read, which is chosen from the finite input alphabet.
* the top stack symbol on each stack.

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

In state q, with X_i on top of the i^{th} stack.

for $i = 1, 2 \cdots k$, the machine may consume input a, go to state p, and replace X_i on the top of the i^{th} stack by string γ_i for $i = 1, 2 \cdots k$.

In multi stack machines, a special symbol called the End marker represented by $, it appears only at the End of the input.

(E) Counter Machines

(1) A counter machine has the same structure as the multi-stack machine, but in place of each stack is a counter. Counters can hold any non negative integer, between zero and zero counters.

The move of a counter machine depends on its state,

input symbol and which if any counters are zero.

In one move the counter machine can:

(a) change state

(b) Add or subtract 1 from any of its counters, independently.

(c) A counter machine may also be regarded as a restricted multi stack machine.

The restrictions are as follows:

a) There are only two stack symbols, which we shall refer to as Z_0 (bottom of stack), and X.

b) Z_0 is initially on each stack.

c) we may replace Z_0 only by a string of the form $X^i Z_0$, for some $i \geq 0$.

The power of counter machine.

1. Every language accepted by a counter machine is Recursively Enumerable. (RE).
   → counter machines are special cases of stack machines, which are special cases of multitape TMs, which accept only RE languages.

2. Every language accepted by a one-counter machine is a CFL.
   → A one-counter machine is a special case of a one stack machine; i.e., a PDA.

Theorem Statement:-

Every recursively enumerable language is accepted by a three-counter machine.

~~Here, 2 counters~~

Proof: Use a stack machine, then develop a constructive algorithm for a 2 stacks to 3 counters conversion

Two counters are used to hold the integers that represent each of the two stacks.

The third counter is used to adjust the other two counters.

In particular, to perform the division or multiplication of a count by Y.

→ Every recursively enumerable language is accepted by a two-counter machine

Turing Machines and Computers.

→ In one sense, a (real) computer has a finite number of states, and is thus weaker than a TM.

→ we have to assume an infinite supply of tapes, disks or some peripheral storage device to simulate an infinite tape TM.

→ Assume a human operator can mount disks, keeping them stacked neatly on the sides of the computer.

Simulating a TM by a computer

A computer can simulate finite control, and mount one disk that holds the region of the tape around

the tape head.

→ when the tape head moves off this region, the computer outputs the following request:

* move the current disk to the top of the left or right pile; and
* mount the disk at the top of the other pile.

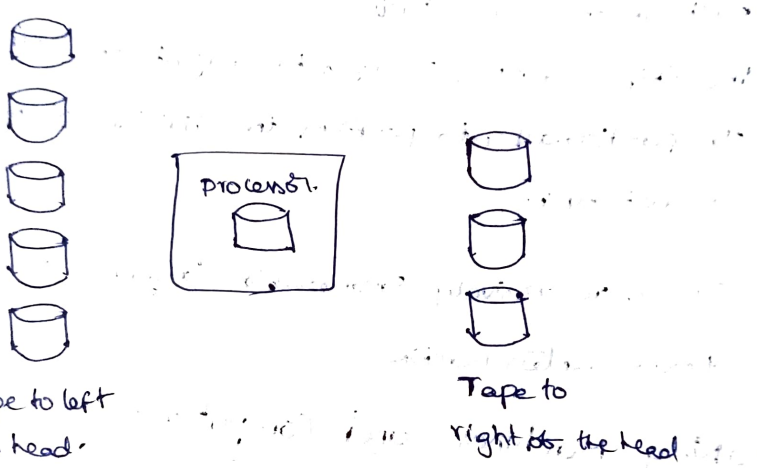


fig:- Simulating a TM with a common computer.

Simulating a computer by a Turing Machine.

→ Simulation is at the level of stored instructions and words in memory

→ TM has one tape that holds all the used memory locations and their contents.

→ other TM tapes hold the instruction counter, memory address, computer input file, and "scratch"

→ Instruction cycle of computer simulated by

* Find the word indicated by the instruction counter on the memory tape
* Examine the instruction code (a finite set of options) and get the contents of any memory words mentioned in the instruction, using the "scratch" tape.
* perform the instruction, changing word values as needed, and adding new address, value pairs to the memory tape, if needed.

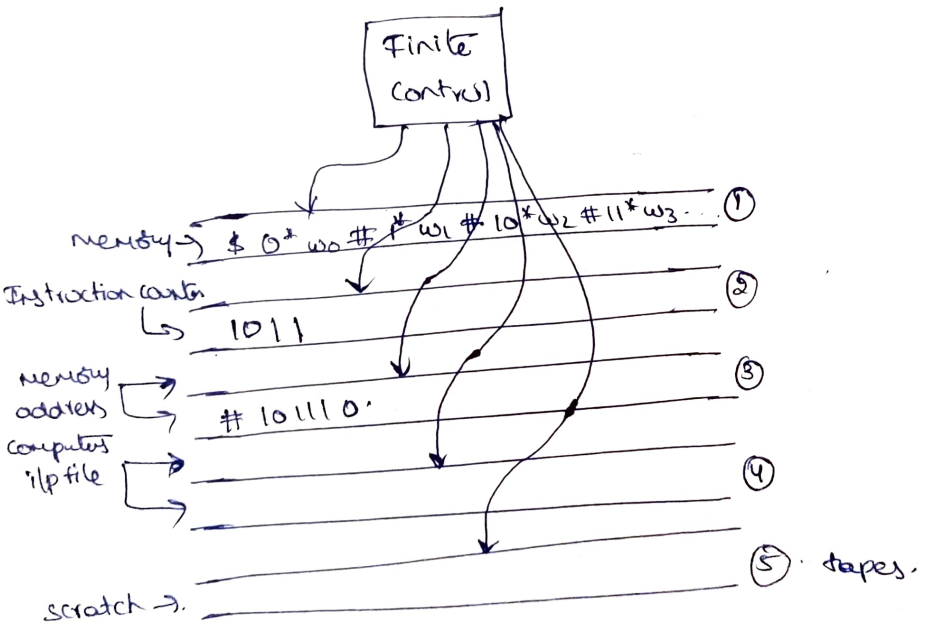


fig:- A TM that simulates a typical computer

Universal Turing Machine

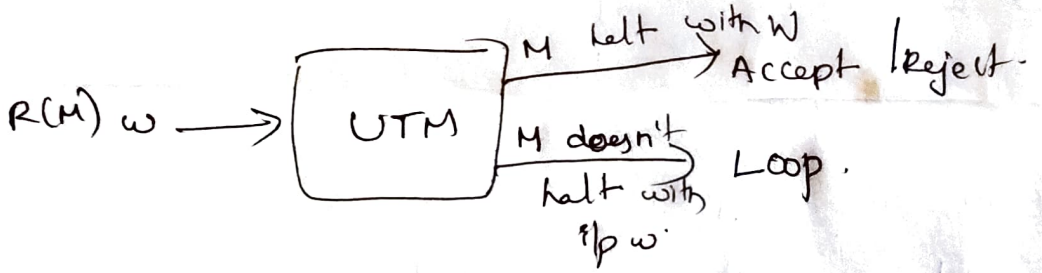
Computer running other computers

It is a general purpose digital computer

UTM is a Turing machine that simulates an arbitrary TM on arbitrary input

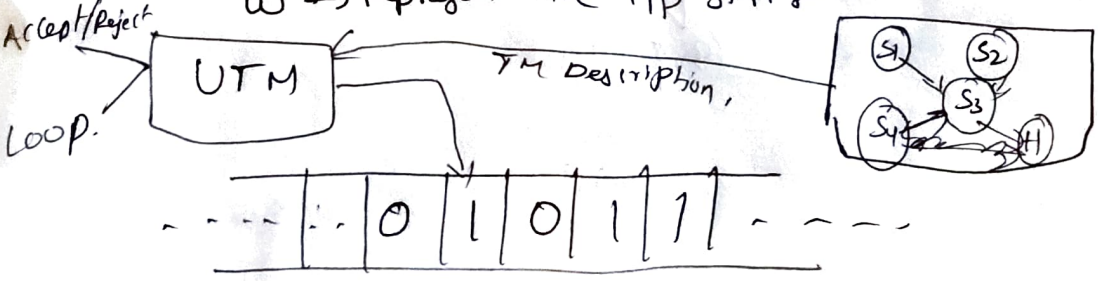
A TM is said to be UTM if it can accept

- the i/p data
- An algorithm (description) for computing the purpose



R(M) → represent a Turing Machine 'M' that accept by halting

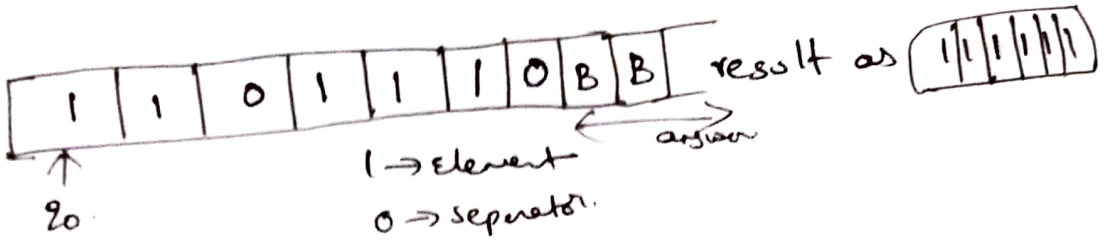
w → Represent the i/p string

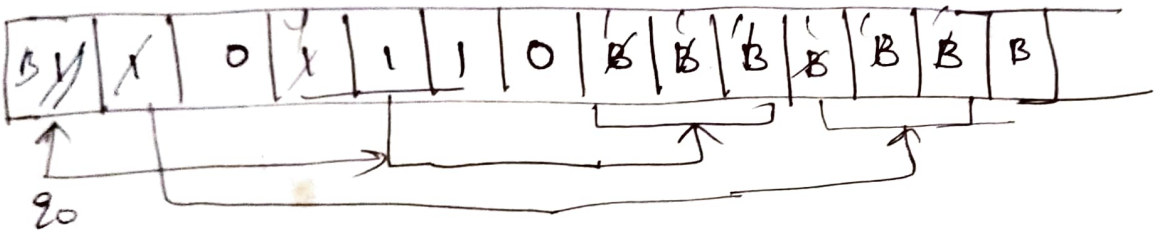


Construction of a Turing machine for multiplication operation

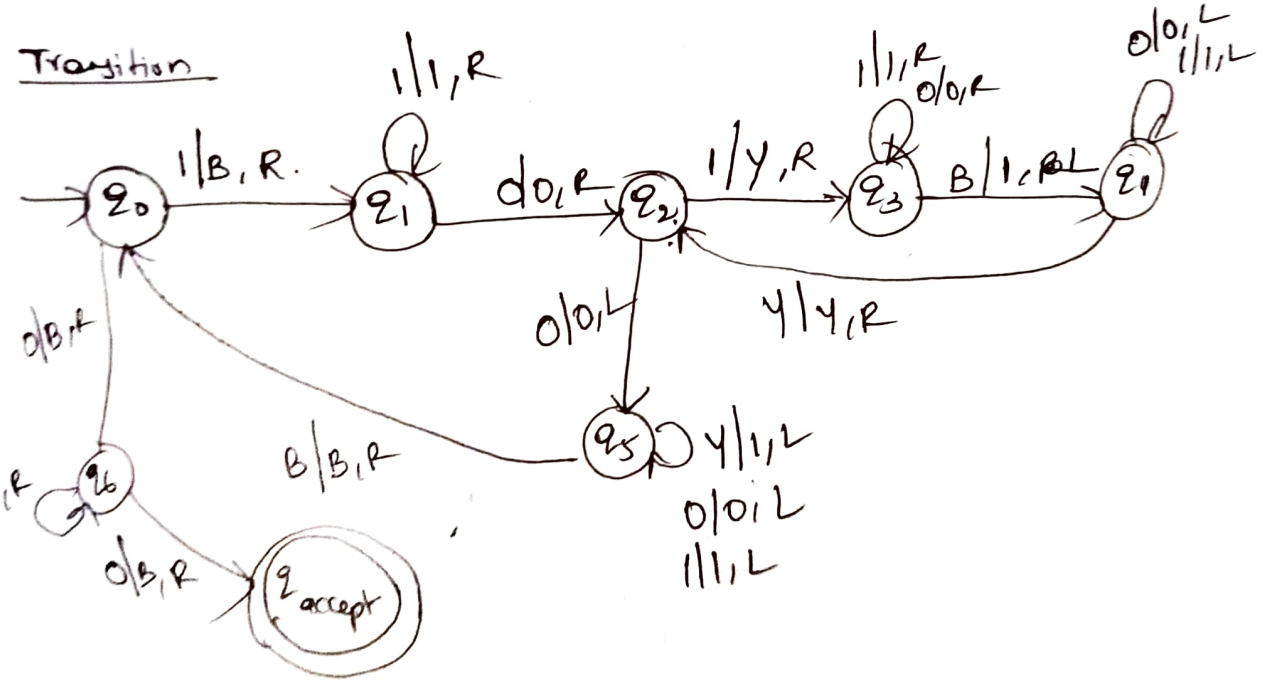
$f = x \times y$ eg 2×3

$$\begin{array}{r} 3 \\ +3 \\ \hline 6. \end{array} \qquad \begin{array}{r} 2 \\ +2 \\ +2 \\ \hline 6. \end{array}$$





Transition



Instantaneous description:

It is triple $< l, q, r >$

where 'q' is the current state, 'l' is a string denoting the tape contents to the left of tape head and 'r' is a string to the right of tape head which does not includes blanks.

The left most symbol of 'r' is the input tape symbol.

A turing machine starts with a initial configuration of $< \epsilon, q_0, w >$

Suppose if the current ID of the turing machine is $\alpha a q_i a \beta$

If the transition is defined as $\delta(q_i, a) = (q_j, b, L)$ then the next ID is $\alpha q_j ab \beta$

If the transition is defined as $\delta(q_i, a) = (q_j, b, R)$ then the ID is $\alpha ab q_j \beta$

Turing machine as language accepter

The language of a turing machine $L(M)$ is the set of strings accepted by M.

$L(M) = \{ x | M \text{ halts and accepts } x \}$

Turing machines are defined so that they can

accept by halting on given input or by entering into recursive loop for invalid input string.

M accepts w iff $<\epsilon, q_0, w> \vdash <l, z, r> q$

declared as accept state

M rejects w iff $<\epsilon, q_0, w> \vdash <l, z, r> \forall q$

declared as nonaccepting state

'M' doesnot accept w iff M rejects w or M loops on w.

Turing Machine as a computational machine :-

A turing machine M computes a function f if when given input w in the domain of f the machine halts in the accepting state with f(w) written to tape.

Design a TM to accept strings belonging to the language $(0+1)^*$

$\delta(q_0, 0) = (q_0, 0, R)$

$\delta(q_0, 1) = (q_0, 1, R)$

$\delta(q_0, B) = (q_A, B, L)$

It can be represented as transition table also

	a=0	a=1	a=B
→ q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_A, B, L)
q_A	—	—	—

Design a TM for finding 1's complement of a given binary number

If the given number is 100110, then its one's complement is 011001.

and if the no. is 11110 then its 1's complement is 00001.

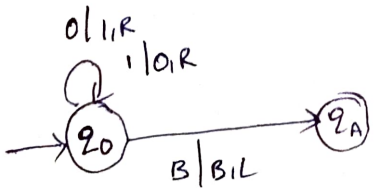
from the above examples, it is clear that the TM has to be designed such that while moving to right, if the input is 1, it should be changed to '0' and if the i/p is '0', it should be changed to 1 and halt on B.

→ $\delta(q_0, 0) = (q_0, 1, R)$

$\delta(q_0, 1) = (q_0, 0, R)$

$\delta(q_0, B) = (q_A, B, L)$.

	a=0	a=1	a=B
→q_0	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_A, B, L)
q_A	—	—	—



→ Design a TM to add two numbers a and b

Soln: let the numbers be 2 and 3. The addition of these numbers using simple logic is explained. The numbers are placed as $B0^2 10^3 B$. After processing the tape content would be $B0^5 B$.

Simple logic is: replace the occurrence of 0 by B and move to right and replace 1 to 0, so that it is in required form as $B0^5 B$. sequence of steps is given for understanding in fig.

$$\delta(q_0, 0) = (q_1, B, R)$$

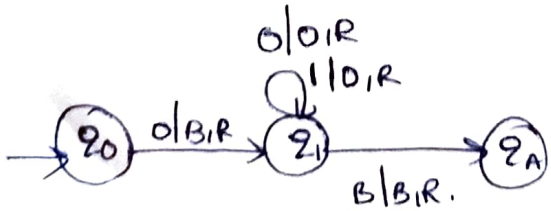
$$\delta(q_1, 0) = (q_1, 0, R)$$

$$\delta(q_1, 1) = (q_1, 0, R)$$

$$\delta(q_1, B) = (q_A, B, R)$$

$B \cdot 001000B \vdash B \cdot 01000B \vdash B0 \cdot 1000B \vdash B00 \cdot 000B \vdash B000 \cdot 00B$

$\vdash B0000 \cdot 0B \vdash B00000 \cdot B.$



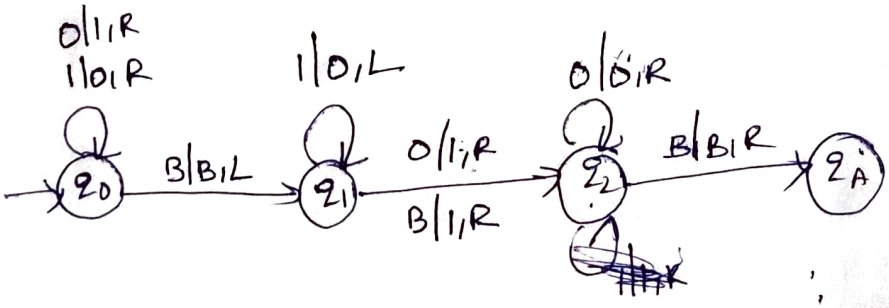
Design a TM that gives two's complement for a given binary representation.

→

Two's complement is computed by first computing one's complement and then adding 1 to it.

Binary num.	1's complement	2's complement
1001	0110	0111
0011	1100	1101
1000	0111	1000

first complements every bit from left to Right ↓ travels back



→ Draw a TM for the function $f(m,n) = m-n$ if $m > n$

and zero for $m < n$

12/3/16

C6, [redacted], D4, D2, D8, D0, E4, E2, F0, F2, [redacted] H0, [redacted]

[redacted] 00 T-66, L6- [redacted] 3V,

Equivalence of counter Machine and Turing machine

Every turing machine can be simulated by a two counter machine. The TM consists of FSM and infinite tape that can be divided and used as a stack.

1. Simulate turing machine by FSM Equipped with two stacks
2. simulate two stacks by four counters.
3. simulate four counters by two counters.

counter machine:

A counter machine can be thought of in one of two ways.

1) It has the same structure as multi stack machine

[Multistack machine is PDA with multiple stacks Equivalent to TM] But inplace of stack counter is used.

A counter can hold any non-negative integer but we distinguish blw zero and non-zero counters.

The move of counters machine depends on its state input symbol and if any of the counters are zero.

In one move counter can do the following

a) change state

b) Add or substract 1 from any of its cou... independently.

A counter is not allowed to become negative, so it cannot subtract 1 from a counter that is currently 0.

Every turing machine can be simulate by a counter machine.

The Turing machine consists of FM i/p tape that can be divided and used as a stack.

The proof is explained in 3 steps

1. Simulate TM by a FSM equipped with 2 stacks
2. Simulate 2 stacks by four counters
3. Simulate 4 counters by two.

Simulating Turing machine by two stacks

Tape is initially filled with zeros, which can be modified with ones and zeros. At any time read, write head points to one cell on the tape This tape is conceptually divided into half and each half is treated as a stack.

According TM is simulated by FSM plus two stacks

moving the head left & 

Right is popping the element from one stack and putting it another.

writing is equivalent to changing the bit before pushing.

Simulate two stacks by four counters:

A stack containing 0's and 1's can be simulated by 2 counters.

$\# 0 * V_0 \# 1 * V_1 \# 2 * V_2 \# 3 * V_3 \ldots \# i * V_i \ldots$

V_i is an integer in binary of $(i+1)^{th}$ word

RAM has a finite no. of registers, one tape is used to hold each register contents, one tape is used to store location counter that indicates the no. of next word to be taken.

Recursive and Recursively enumerable languages

There are three possible outcomes of executing a TM over a given input. The TM may

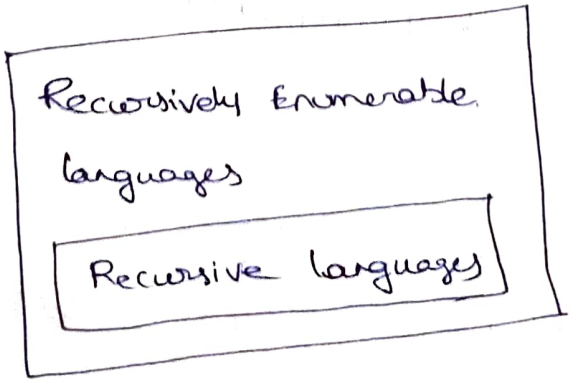
* halt and accept the input;
* halt and reject the input; or
* never halt.

A language is said to be recursive if there Ex... a TM that accepts every string of the language and rejects every string (over the same alphabet) tha... is not in the language.

Note: If a language L is recursive, then its complement L̄ must also be recursive.

→ A language is said to be Recursively Enumerable if there exists a TM that accepts every string of the language, and doesnot accept strings that are not in the language.
for the strings that are not in the language the TM may or may not halt.

Note :- Every Recursive language is also Recursively Enumerable language. It is clear whether every RE language is also Recursive.



Types of Turing Machines:

There are number of other types of Turing machines inaddition to the one which we have seen, such as

TM's with 1) multiple tapes
2) one tape but with multiple heads
3) two dimensional tapes
4) Non deterministic TM's etc.)

→ It turns out that computationally are these TM's are Equally powerful. How fast they can compute may vary.

Non-deterministic Turing machines

A NDTM is a machine for which like NDFA, at any current state and for the tape symbol it is reading, there may be different possible actions to be performed.

Hence an action means:
→ a combination of writing a symbol on the tape
→ moving the tape head
→ going to the next state.

→ one action could be just changing the state without modifying the cell content.
→ one action could be not changing the state and changing the cell content.
→ In all actions, it may move left or Right.

For example, let us consider the language $L = \{w \in \{a|b\}^*\}$. Given a string x, a non deterministic that accepts this language L would first guess the mid point of x, which is the place where the second half of x starts. Then it would compare the first half of x with the second half by comparing i^{th} symbol of the first half with the i^{th} symbol of second half for $i = 1, 2 \ldots$

Turing machine with Two dimensional Tapes:

TM with two-dimensional tape is a kind of TM that has one finite control, one read-write head and one two-dimensional tape. The cells in the tape is two dimensional, i.e., the tape has the top end and the left end, but extends indefinitely to the right and down.

The head of 2 dimensional tape moves one square up, down, left or right. As powerful As TM with one dimensional tape. Since for every two dimensional tape can be represented as one dimesional tape.

Two-Dimensional Tape:

V	V	V	V	V	V	V	. .	. .
h	1	2	6	7	15	16	. .	. .
h	3	5	8	14	17	26	. .	. .
h	4	9	13	18	25	. .	. .	. .
h	10	12	19	24	. .	. .	. .	. .
h	11	20	23	. .	. .	. .	. .	. .
h	21	22	. .	. .	. .	. .	. .	. .

Here the numbers indicate the correspondence between the squares of the two tapes: square numbered 'i' in the 2 dimensional tape is mapped to square numbered 'i' in the one dimensional tape.

Equivalent one-dimensional tape:

V	1	V	2	3	V	4	5	6	V	7	8	9	10	h	11	. .

3. TM with ~~Mdp~~ Multiple Tapes:

This kind of Turing machine has only one finite control more than one input tape with its own read, write head. It is denoted by a 7-tuple $(Q, \Sigma, \delta, T, q_0, B, F)$

Its transition function is a partial function.

$$\delta: Q \times (T \cup \{B\})^n \rightarrow (Q \cup \{h\}) \times (T \cup \{B\})^n \times \{R, L, S\}^n$$

A configuration for this kind of TM must show the current state the machine is in and the state of each tape. It can be proved that any language accepted by a n-tape TM can be accepted by a one-tape TM and that any function computed by a n-tape TM can be computed by a one-tape TM

They are Equally powerful.

4. Turing Machine with multiple heads:

These type of machines have one finite control and one tape, but more than one read/write heads.

In each state only one of the heads is allowed to read and write.

It is denoted by a 7 tuple $(Q, \Sigma, T, \delta, q_0, B, F)$

The transition function

$$\delta: Q \times \{H_1, H_2 \cdots H_n\} \times (T \cup \{B\}) \rightarrow (Q \cup \{h\}) \times (T \cup \{B\}) \times \{R, L, S\}$$

where $H_1, H_2 \cdots H_n$ denote the tape heads. These are powerful as one-tape TMs.

5. Turing machine with Infinite Tape:

This is a kind of TM that have one finite control and one tape which extends infinitely in both directions

Church's Thesis

The church- Turing thesis says that any real world computation can be translated into an equivalent computation involving a TM.

This can be applied to any kind of computations those involving cellular automata, combinatory, register machines and substitution systems

These can also be used in quantum computing probabilistic computing.

RAM can be simulated using Turing machine.

RAM consists of an infinite no. of words numbered from 0,1... where each word can hold one integer

To simulate RAM using TM, we use multi tapes, one of the tapes holds the words of the RAM where each word is seperated by # as shown below

$$\#0*V_0\#1*V_1\#2*V_2\#3*V_3\cdots\#i*V_i\cdots$$

Church Thesis

The church turing thesis says that any real world computation can be translated into an equivalent computation involving a TM. This can be applied to any kind of computations those involving cellular automata, combinatory, register machines and substitution systems.

These can also be used in quantum computing probabilistic computing.

RAM can be simulated using TM.

RAM consists of an infinite no. of words numbered from 0, 1 ... where each word can hold one integer

To simulate RAM using TM, we use multitapes, one of the tapes holds the words of the RAM where each word is seperated by #. One tape is used to store location counter that indicates no. of next word to be taken.

Counter machine

A CM is an abstract m/c used in formal logic and theoretical computer science to model computation. A CM comprises of a set of one or more unbounded register, each of which can hold a single non-negative integer, and a list of arithmetic and control instructions for the machine to follow. A CM consists of → A finite set of registers $r_0 \cdots r_n$, where each register is labelled and can hold any single non-negative integer. A special register to identify the current instruction to be executed is maintained and is called state register.

Linear Bounded Automata and context sensitive language

A Non-deterministic TM is called linear Bounded Automata (LBA) if

- Its input alphabet includes two special symbols ¢ and $ as left and right end markers
- It has no moves beyond these end markers, i.e., no left move from ¢ and no right move from $.
- It never changes the symbols ¢ and $

A linear bounded automata is defined using 8-tuple form by $M = (Q, \Sigma, T, \delta, q_0, ¢, \$, F)$, where $Q, \Sigma, T, \delta, q_0, F$ are same as non-deterministic TM, and ¢ and $ are left and right end markers.

The language accepted by M is defined as $L(M)$ and is given by

$\{w \mid w \in (\Sigma - \{¢, \$\})^*$ and $q_0 ¢ w \$ \xrightarrow[M]{*} aqb$ for some q in $F\}$

Equivalence of LBA's and CSL's

we can show that if 'L' is a context sensitive language (CSL), then there Exists a LBA M such that

$L(M) = L - \epsilon$.

Theorem 3: If L is a CSL, then L is accepted by some LBA.

Proof: Let us construct a LBA M with two-track tape to recognize L. The first track holds the i/p string w as ¢w$. The second track is used while the i/p is processed.

* LBA initializes the second track with S just below the left most symbol of w.
* If $w = \epsilon$, then the system halts without accepting.
* Otherwise, it repeatedly guesses a production and a position in the sentential form which is on the second track
* If the sentential form expands, then it shifts the portion of string from the current position to right.
* If the new sentential form α is longer than w, then the system halts without accepting

Since the right side of all the productions are atleast as long as left side, there would not be any derivation as $S \overset{*}{\Rightarrow} \alpha \overset{*}{\Rightarrow} w$, where α is longer than w. Hence, the LBA accepts a string if and only if $S \overset{*}{\Rightarrow} w$, where w is a word generated by CSG

Pumping Lemma for context free language

If L is a CFL, there is a pumping length 'p' such that any string $w \in L$ of length $\geq p$ can be written as $w = uvxyz$, where $vy \neq \epsilon$, $|vxy| \leq p$, and for all $i \geq 0$, $uv^ixy^iz \in L$

Applications of lemma

Pumping lemma is used to check whether a grammar is context free or not.

Problem:

find out whether the language $L = \{x^n y^n z^n \mid n \geq 1\}$ is context free or not.

Sol:

Let L is context free. Then L must satisfy pumping lemma.

At first, choose a number n of the pumping lemma.

Then take z as $0^n 1^n 2^n$

Break z into $uvwxy$, where

$$|vwx| \leq n \text{ and } vx \neq \epsilon.$$

Hence vwx can not involve both 0's and 2s, since the last 0 and the first 2 are atleast $(n+1)$ position apart. There are 2 cases

Case 1: vwx has no 2s. Then vx has only 0's a...

Then uwy, which would have to be in L, h...

n 2s, but fewer than n 0s or 1s

Case 2: vwx has no 0s

Hence contradiction occurs.

Hence, L is not a CFL.

The Turing Machine

what is computable?

A task is computable if one can specify a sequence of instructions which when followed will result in the completion of the task.

Such set of instructions is called an effective procedure or algorithm for task.

Alan Turing modelled a Turing Machine (TM) as a machine with a finite number of control states and an infinite tape, bounded at the left and stretching off to the right.

The tape is divided into cells, each of which can hold one symbol.

The I/p of the machine is a string $w = w_1 w_2 w_3 \cdots w_n$ initially written on the leftmost portion of the tape, followed by an infinite sequence of blanks B.

$$w_1 \; w_2 \; w_3 \; \cdots \; w_{n-1} \; w_n \; B \; B$$

The machine is able to move a read/write head left and right over the tape as it perform its computation.

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

Q — is a finite set of states

Σ — is the i/p alphabet, which never includes blanks.

Γ — is the tape alphabet, which always includes blanks.

$\delta \rightarrow Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function, where L and R are directions, telling the machine head in which direction to go in a step.

$q_0 \in Q$

B is a special symbol indication of a blank cell

'F' is a set of final states, where the TM halts on reaching these states

$\delta(q_i, a) = (q_j, b, d)$ is made by a TM M:

* M writes b to the current tape cell, overwriting a.
* The current state changes from q_i to q_j
* The tape head moves to the left or right by one cell, depending on whether d is L or R.

Instantaneous Descriptions (ID)

The ID of a TM is the configuration of the system in triple $< l, q, r >$ where q is the current state, l is a string denoting the tape contents to the left of the tape head, and r is a string to the right of the tape head.

The TM starts with the initial configuration $< \varepsilon, q_0, w >$ and repeatedly makes transitions until it halts

At any given time, the move of a TM is dependent on the input and the current state.

Let us suppose that the current ID of the TM is "$\alpha q_i a \beta$"

* If the transition is defined as $\delta(q_i, a) = (q_j, b, L)$ then the next ID is $\alpha q_j a b \beta$.
* If the transition is defined as $\delta(q_i, a) = (q_j, b, R)$ then the ID is $\alpha a b q_j \beta$.

- TM will terminate execution on reaching to accepting configuration $<u, q, w>$ $\forall q \in$ halting state.

Turing Machine as Language Accepter:

The language of a TM M, L(M) is the set of strings accepted by M.

$L(M) = \{x \mid M \text{ halts and accepts } x\}$.

Turing Machines are defined so that they can accept by halting on given input or by entering into recursive loop for invalid input string.

→ M accepts w iff the execution of M on w is terminating and Ends in the accept state:

$<\varepsilon, q_0, w> \vdash <l, q, r>$ $\forall q$ declared as accepting state

→ M rejects w iff the execution of M on w is terminating and Ends in the non accepting state:

$<\varepsilon, q_0, w> \vdash <l, q, r>$ $\forall q$ declared as non-accepting state.

M does not accept w iff. M rejects w or M loops on w.

Turing Machine as a computational Machine:-

A turing machine M computes a function f if, when given input w in the domain of f, the Machine halts in its accept state with f(w) written on the tape.

→ Design a TM to accept strings belonging to the language $(0+1)^*$.

Sol: To design a TM for the given language,

we have to identify

(1) the required states

(2) inputs to be processed

(3) steps of the process and

(4) when to halt

$Q = \{q_0, q_A\}$, $\Sigma = \{0, 1\}$, $\Gamma = \Sigma \cup \{B\}$.

$\delta(q_0, 0) = (q_0, 0, R)$

$\delta(q_0, 1) = (q_0, 1, R)$

Once the machine hits a blank (B) it moves one cell to the left and stops

$$\delta(q_0, B) = (q_A, B, L).$$

fig (a) Transition diagram for $(0+1)^*$.
(simple TM)

Transition Table

	a=0	a=1	a=B
→q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_A, B, L)
q_A	–	–	–

(5) Design a TM for finding 1's complement of a given binary number.

Sol: If the given number is "100110" and its 1's complement is '011001'

Deterministic Pushdown Automata:-

while, PDA's are by definition allowed to be non-deterministic, the deterministic sub case is quite important

In particular, parsers generally behave like deterministic PDA's.

Definition: A PDA is deterministic if there is never a choice of move in any situation.

These choices are of two kinds.

If $\delta(q, a, X)$ contains more than one pair, then surely the PDA is non deterministic because we can choose among these pairs when deciding on the next move.

However even if $\delta(q, a, X)$ is always a singleton, we could still have a choice between using a real i/p symbol or making a move on ϵ.

Thus we define a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ to be deterministic (DPDA), if and only if the following conditions are met:

1. $\delta(q, a, X)$ has atmost one member for any q in Q, a in Σ or $a = \epsilon$ and X in Γ.
2. If $\delta(q, a, X)$ is non empty, for some a in Σ, then $\delta(q, \epsilon, X)$ must be empty.