Turing Machine

Topics:

- Turing Machine – Definition, Representation
- Variants of Turing Machine
- Non-Deterministic and Deterministic TM
- Closure properties of TM
- Languages of TM- Recursive and Recursively enumerable lang
- Turning Machine codes

Turing Machine

- **Finite state automaton**
  - Limitation: finite amount of memory
  - Prevents recognizing languages that are not regular $\{0^n1^n | n = 0,1,2,\ldots\}$
- **Pushdown automation**
  - Unlimited memory : stack
  - Equivalent to context free grammers
  - Can recogninze $A = \{0^n1^n | n = 0, 1, 2,\ldots.\}$
  - Can't recognize context sensitive grammer ($B = \{a^nb^nc^n | n = 0, 1, 2,\ldots.\}$)
- **Linear bounded automata**
  - More powerful than pushdown models.
  - Recognize context sensitive grammer.
  - Cannot recognize all the languages generated by phrase –structure grammers.

TURING MACHINE-DEFINITION

- A Turing Machine accepts the recursively enumerable language generated by type 0 grammars.
- It was invented in 1936 by Alan Turing.
- A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given.
  - It consists of a head which reads the input tape.
  - A state register stores the state of the Turing machine.
  - After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left.
  - If the TM reaches the final state, the input string is accepted, otherwise rejected.

Unrestricted Grammars:

Productions

$$u \rightarrow v$$

String of variables and terminals

String of variables and terminals

Example unrestricted grammar:

$$S \rightarrow aBc$$
$$aB \rightarrow cA$$
$$Ac \rightarrow d$$

A language L is Turing-Acceptable if and only if L is generated by an unrestricted grammar

TURING MACHINE-FORMAL DEFINITION

A TM can be formally described as a 7-tuple

$(Q, \sum, X, \delta, q_0, B, F)$ where –

- **Q** is a finite set of states
- **X** is the tape alphabet
- $\sum$ is the input alphabet
- δ is a transition function;
- $\delta : Q \times X \rightarrow Q \times X \times D$ where D is L or R
- q_0 is the initial state
- **B** is the blank symbol
- **F** is the set of final states

COMPARISON OF HOW A TURING MACHINE DIFFERS FROM FINITE AUTOMATON AND PUSHDOWN AUTOMATON.

Machine	Stack Data Structure	Deterministic?
Finite Automaton	N.A	Yes
Pushdown Automaton	Last In First Out(LIFO)	No
Turing Machine	Infinite tape	Yes

EXAMPLE OF TURING MACHINE

Turing machine $M = (Q, X, \sum, \delta, q_0, B, F)$ with

$Q = \{q_0, q_1, q_2, q_f\}$

$X = \{a, b\}$

$\sum = \{1\}$

$q_0 = \{q_0\}$

B = blank symbol

$F = \{q_f\}$

δ is given by –

	a	b
q0	q1 X R	q1 X R
q1	q1 a R	q2 b R

δ (q0,a)=(q1 X R) Here the transition on state q0 on reading a in tape moves to state q1 , replaces a by X in tape and moves right to read next symbol in the tape

TURING MACHINES ARE USEFUL IN SEVERAL WAYS

- As automation
- As computing function
- Mathematical model for Partial Recursive function
- determining the undecidability of certain languages
- measuring the space and time complexity of problems

TIME AND SPACE COMPLEXITY OF A TURING MACHINE

- the time complexity: the measure of the number of times the tape moves when the machine is initialized for some input symbols
- the space complexity: the number of cells of the tape written.
- Time complexity all reasonable functions –

  T(n) = O(n log n)
- TM's space complexity –

  S(n) = O(n)

TM

- A TM accepts a language if it enters into a final state for any input string w.
  - A language is recursively enumerable (generated by Type-0 grammar) if it is accepted by a Turing machine.
- A TM decides a language if it accepts it and enters into a rejecting state for any input not in the language.
  - A language is recursive if it is decided by a Turing machine.
- There may be some cases where a TM does not stop.
  - Such TM accepts the language, but it does not decide it.

DESIGNING A TURING MACHINE

Example 1

Design a TM to recognize all strings consisting of an odd number of α's.

Solution

The Turing machine **M** can be constructed by the following moves –

Let $\mathbf{q_1}$ be the initial state.

If **M** is in $\mathbf{q_1}$; on scanning α, it enters the state $\mathbf{q_2}$ and writes **B** (blank).

If **M** is in $\mathbf{q_2}$; on scanning α, it enters the state $\mathbf{q_1}$ and writes **B** (blank).

. . .

From the above moves, we can see that **M** enters the state $\mathbf{q_1}$ if it scans an even number of α's, and

it enters the state $\mathbf{q_2}$ if it scans an odd number of α's. Hence $\mathbf{q_2}$ is the only accepting state.

Hence,

$$M = \{\{q_1, q_2\}, \{1\}, \{1, B\}, \delta, q_1, B, \{q_2\}\}$$

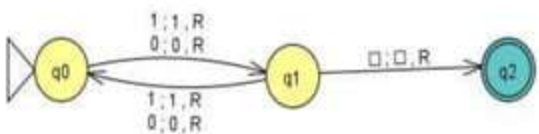
• • •

δ is given by –

Tape alphabet symbol	Present State 'q_1'	Present State 'q_2'
α	BRq_2	BRq_1

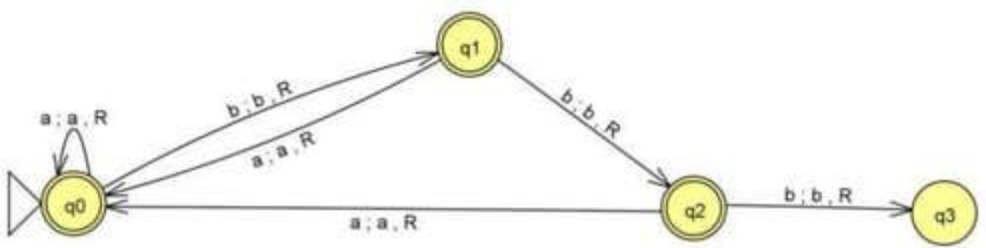
EXAMPLE 2

Design a Turing Machine that accepts a strings of odd length



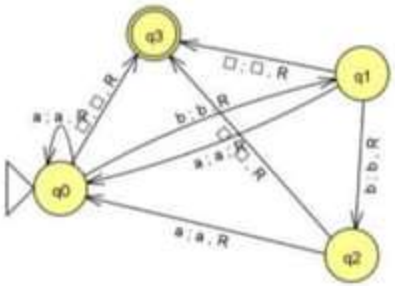
EXAMPLE 3

Design a Turing Machine that reads a strings without bbb



May receive an error that there are transitions out of final states.

ALTERNATE SOLUTION.. TM NOT HAVING STRINGS 'BBB'



EXAMPLE 4

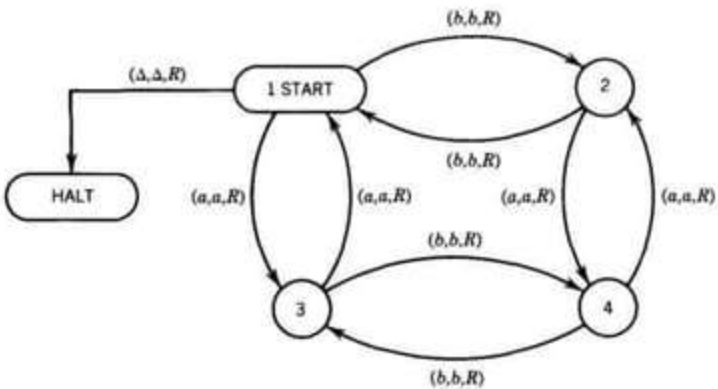
Let us define the four states:

State 1 We have read an even number of a's and an even number of b's.

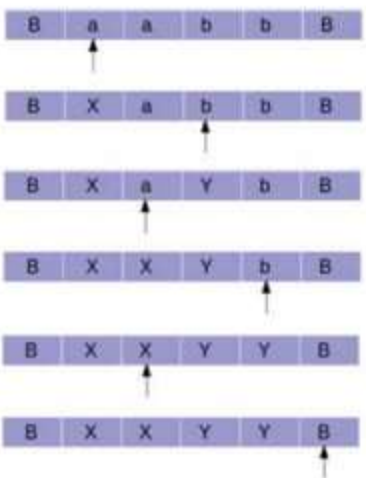
State 2 We have read an even number of a's and an odd number of b's.

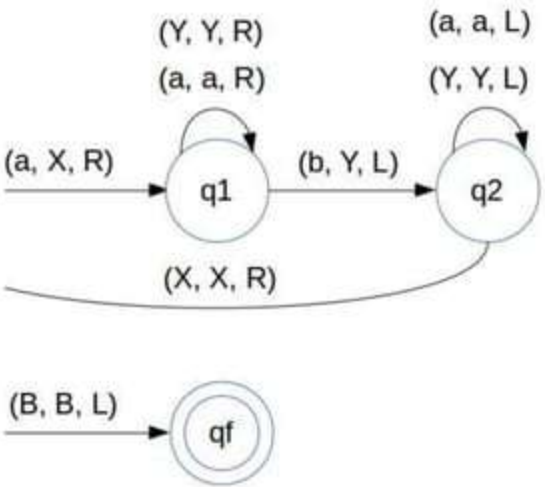
State 3 We have read an odd number of a's and an even number of b's.

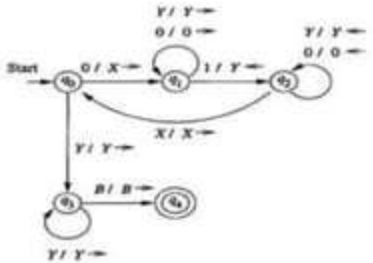
State 4 We have read an odd number of a's and an odd number of b's.



EXAMPLE 5
TURING MACHINE FOR $A^N B^N \mid N \geq 1$;







state input	0	1	x	y	b
->q0	(q1,x,R)	-	-	(q3,Y,R)	-
q1	(q1,0,R)	(q2,Y,L)	-	(q1,y,R)	-
q2	(q2,0,L)	-	(q0,x,R)	(q2,y,L)	-
q3	-	-	-	(q3,y,R)	(q3,B,R)
*q4	-	-	-	-	-

Check String: 1 0011 is accepted or not

q00011b |- xq1011b |-x0q111b |-xq20y1b |-q2x0y1b |-xq00y1b
|- xxq1y1b |-xxyq11b |-xxq2yyb |-xq2xyyb |-xxq0yyb
|- xxyqq3yb |-xxyyq3b |-xxyybq4

It reaches q4 final state, so the given string is accepted

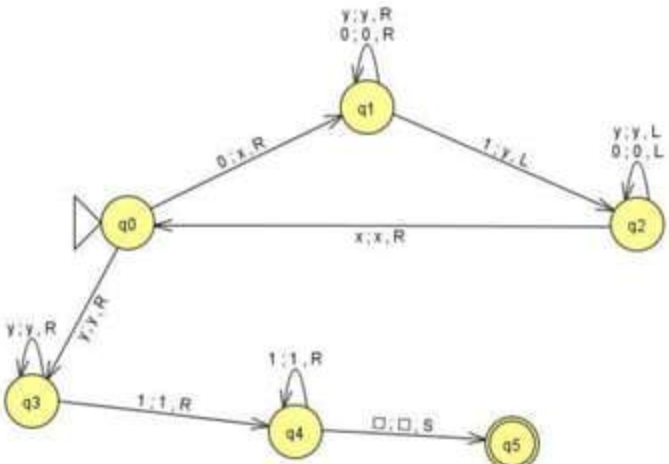
check String: 2 011

q0011 -xq111 |-q2xy1 |-xq0y1 |-xyq3

It reaches q3 which is non-final state, so the given string is rejected

EXAMPLE 6

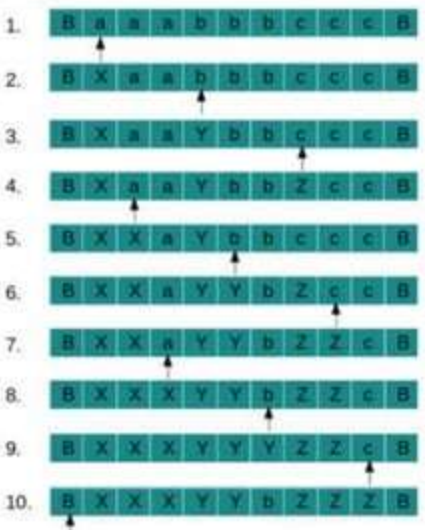
$L=\{0^N1^M \mid N<M\}$



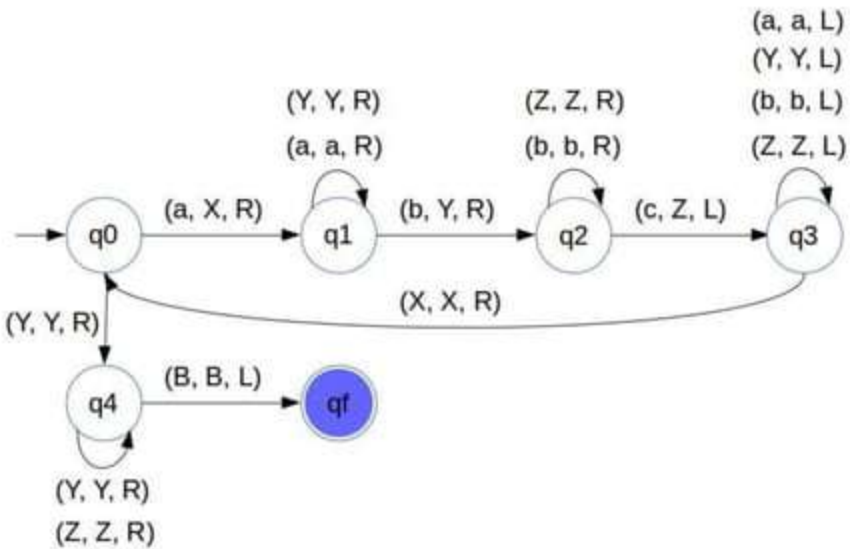
EXAMPLE 7

Turing machine for $a^n b^n c^n \mid n \geq 1$

TAPE movement for string "aaabbbccc":



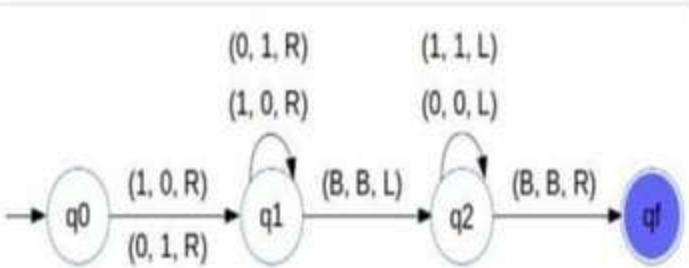
STATE TRANSITION DIAGRAM FOR $A^NB^NC^N \mid N \geq 1$ AS FOLLOWS:



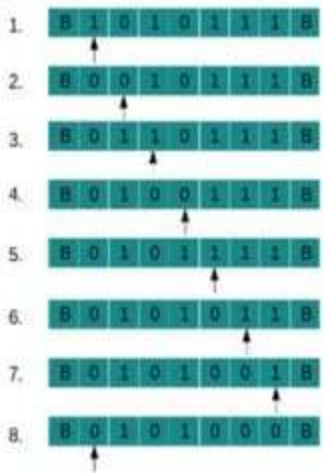
EXAMPLE 8
TURING MACHINE AS TRANSDUCER FOR 1'S COMPLEMENT

Approach for 1's complement

1.Scan input string from left to right
2.Convert '1' into '0'
3.Convert '0' into '1'
4.When BLANK is reached move towards left (start of input string).



TAPE movement for string "1010111":

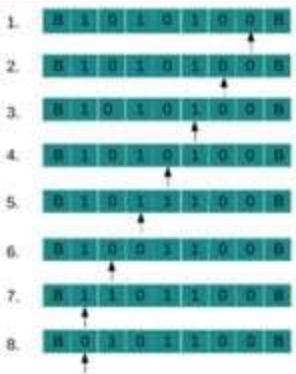


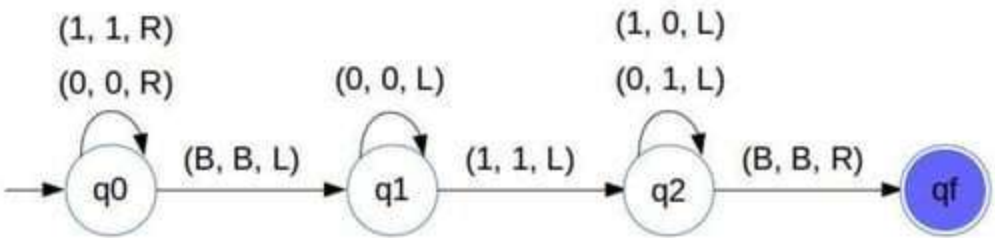
EXAMPLE 9
TURING MACHINE AS TRANSDUCER FOR 2'S COMPLEMENT

Approach for 2's complement

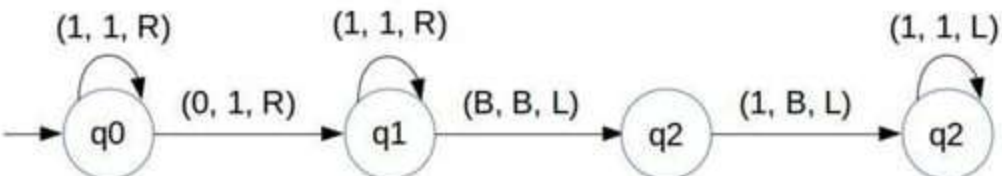
1.Scan input string from **right to left**
2.Pass all consecutive '0's
3.When '1' comes do nothing
4.After that take complement of every digit.

TAPE movement for string "1010100":

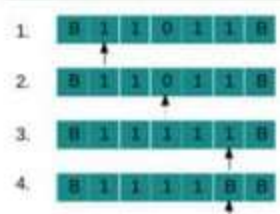




EXAMPLE 10
UNARY ADDITION



TAPE movement for string "110111":

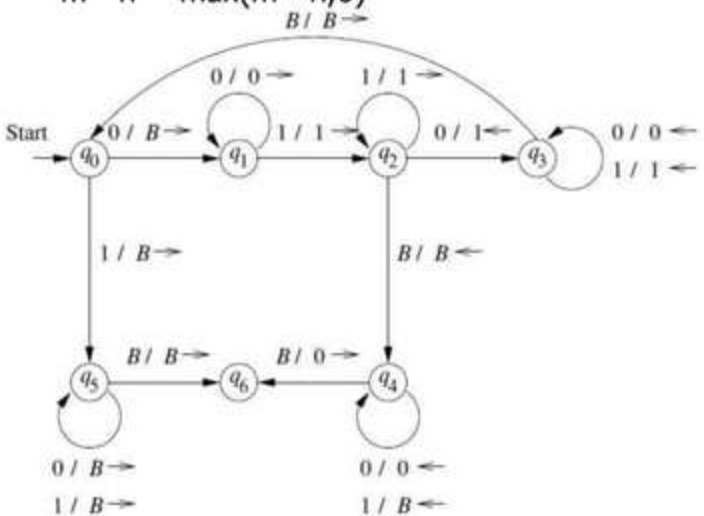


Unary addition is like string concatenation.

EXAMPLE 11

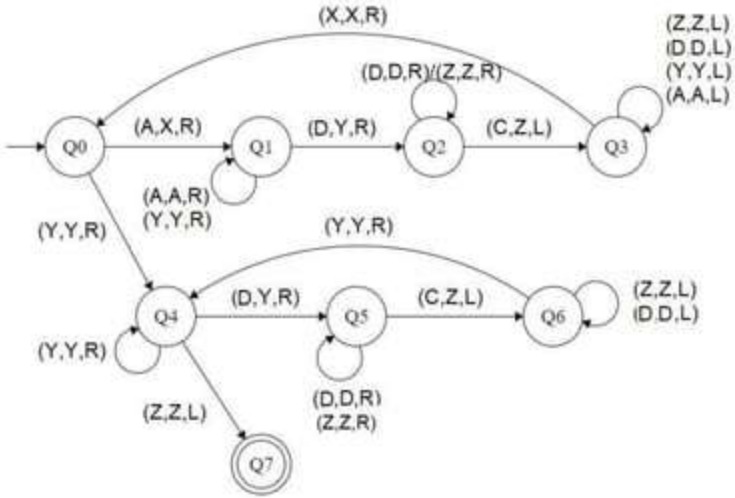
Construct a turing machine to perform proper subtraction defined as

$$m - n = \max(m - n, 0)$$



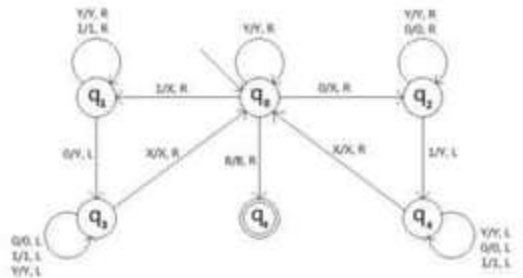
EXAMPLE 12

Construct a Turing Machine for $L= a^i b^j c^k \mid i< j< k;\ i \geq 1\}$



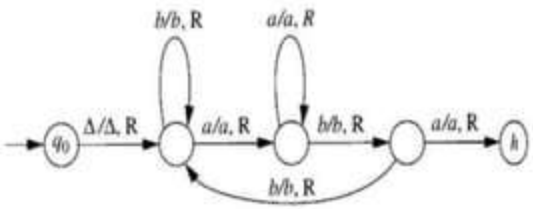
EXAMPLE 13

Construct a Turing Machine for accepting strings with an equal number of 0's and 1's



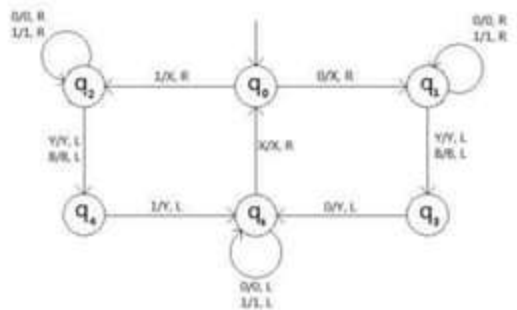
EXAMPLE 14

Construct a Turing Machine for the language with substring aba for the alphabet {a,b}



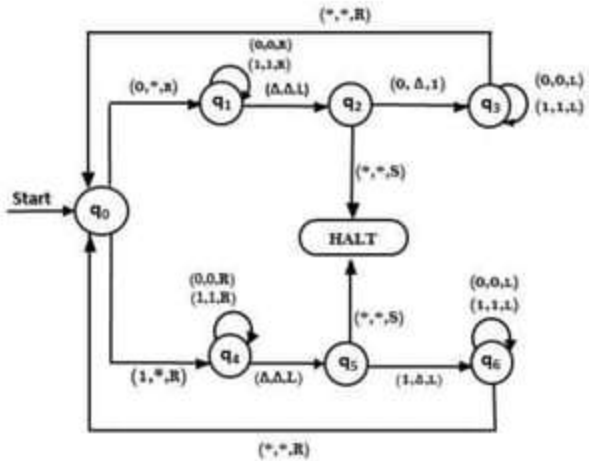
EXAMPLE 15

Construct a turing machine for L={ww^r|w belongs to{0,1}}



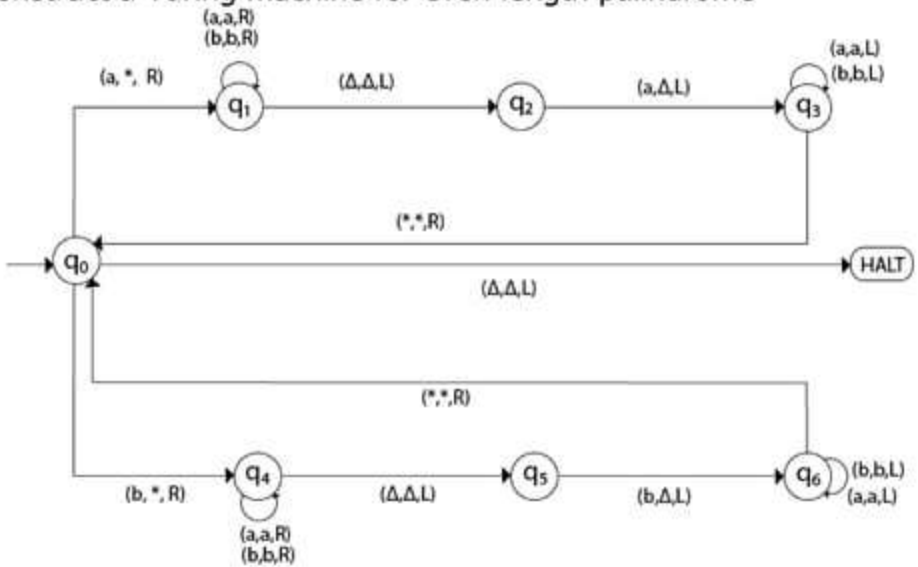
EXAMPLE 16

Construct a Turing machine for odd length palindrome



EXAMPLE 17

Construct a Turing machine for even length palindrome



S2 *VARIANTS OF TURING MACHINES, SIMPLE EXAMPLES*

Multi-track Turing machine

Multi-tape Turing machine

Multi-tape Multi-head TM

Two-way Infinite Tape TM

Non-Deterministic Turing Machine

Multi-dimensional Tape Turing Machine

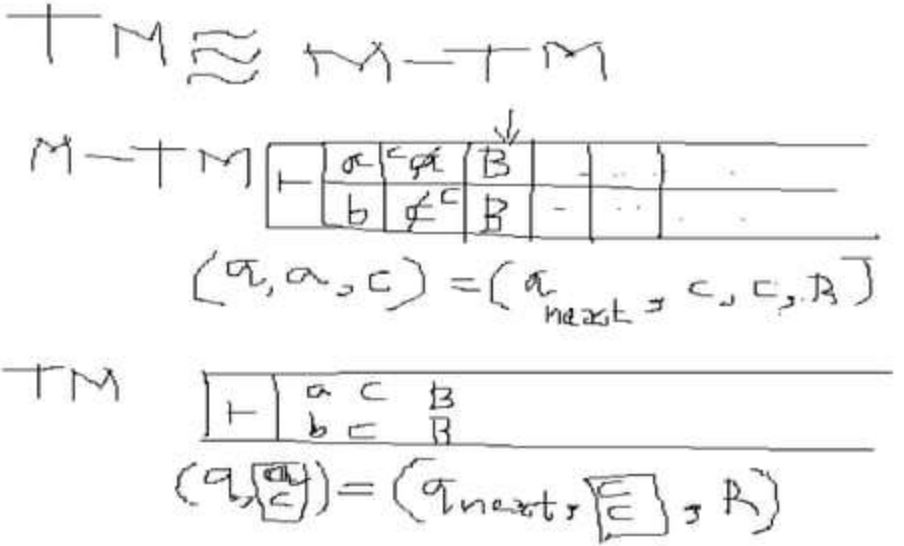
Multi-head Turing Machine

Note: If any language is accepted by TM that means any of its variant also accepts that language.

MULTI-TRACK TM

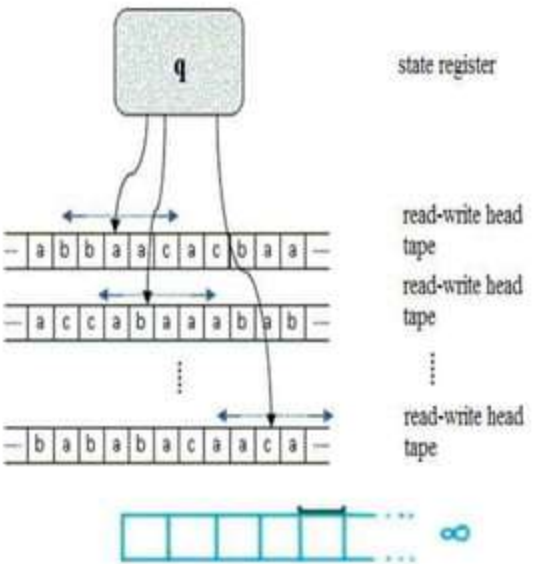
- ✓ In a standard n-tape Turing machine, n heads move independently along n tracks.
- ✓ A k-tack Turing machine(for some k>0) has k-tracks and one R/W head that reads and writes all of them one by one.
- ✓ A k-track Turing Machine can be simulated by a single track Turing machine
- ✓ In multi-track TM, we have semi-infinite tape with multiple tracks.
- ✓ Like a TM, This also has a finite control (or head) pointing to a particular cell.

BOTH TM AND M- TM ARE HAVING THE SAME POWER



MULTI-TAPE TURING MACHINE

- ✓ It has multiple tapes and controlled by a single head.
- ✓ This multi-tape Turing machine has sane power like multi-track Turing machine.
- ✓ Move multiple heads independently.
- ✓ Multi-tape Turing machine can be simulated by single-tape Turing machine.



g Machines

asition Function

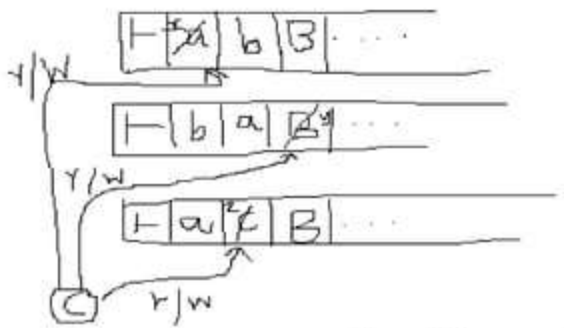
a TM with k tapes

$$\Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R,S\}^k$$

MULTI-TAPE MULTI-HEAD TURING MACHINE

- ✓ The multi-tape Turing machine has multiple tapes and multiple heads.
- ✓ Each tape controlled by separate head.
- ✓ Multi-Tape Multi-head Turing machine can be simulated by standard Turing machine.

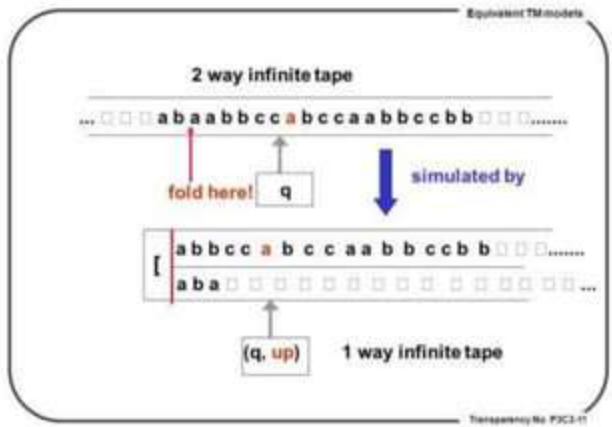
EXAMPLE



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

TWO-WAY INFINITE TAPE TURING MACHINE

- ✓ Infinite tape of two-way infinite tape Turing machine is unbounded in both directions left and right.
- ✓ Two-way infinite tape Turing machine can be simulated by one-way infinite Turing machine(standard Turing machine).



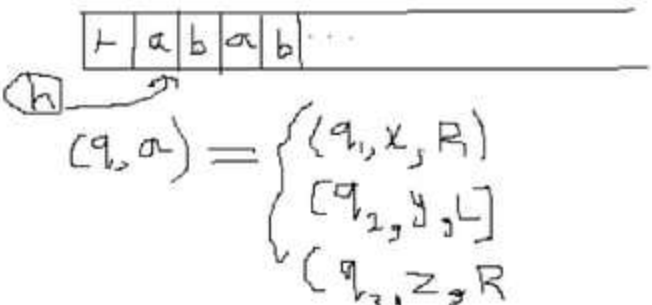
NON-DETERMINISTIC TURING MACHINE

- ✓ A non-deterministic Turing machine has a single, one way infinite tape.

- ✓ For a given state and input symbol has atleast one choice to move (finite number of choices for the next move), each choice several choices of path that it might follow for a given input string.

- ✓ A non-deterministic Turing machine is equivalent to deterministic Turing machine.

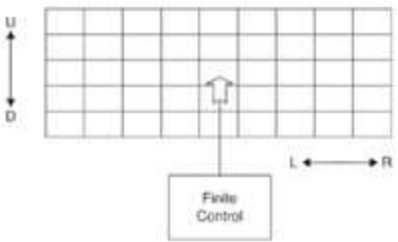
NON-DETERMINISTIC TURING MACHINE-EXAMPLE



MULTI-DIMENSIONAL TAPE TURING MACHINE

- It has multi-dimensional tape where head can move any direction that is left, right, up or down.

- Multi dimensional tape Turing machine can be simulated by one-dimensional Turing machine



MULTI-HEAD TURING MACHINE

- ✓ A multi-head Turing machine contain two or more heads to read the symbols on the same tape.
- ✓ In one step all the heads sense the scanned symbols and move or write independently.
- ✓ Multi-head Turing machine can be simulated by single head Turing machine.

NONDETERMINISTIC TURING MACHINES

- In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have.
  - So, here the transitions are not deterministic.
  - The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.
- An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted.
  - If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

NONDETERMINISTIC TMS-DEFINITION

A non-deterministic Turing machine can be formally defined as a 6-tuple $(Q, X, \sum, \delta, q_0, B, F)$ where –

Q is a finite set of states

X is the tape alphabet

$\sum$ is the input alphabet

δ is a transition function;

$$\delta : Q \times X \rightarrow P(Q \times X \times \{\text{Left\_shift, Right\_shift}\}).$$

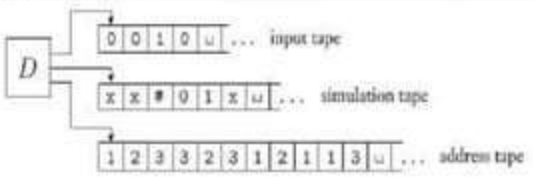
$\mathbf{q_0}$ is the initial state

B is the blank symbol

F is the set of final states

NONDETERMINISTIC TMS AND EQUIVALENCE WITH DETERMINISTIC TMS

- Let *D* be a three tape Turing machine
  - Tape 1 contains the input string *w*
  - Tape 2 contains a copy of *N*'s tape on some branch of computation
  - Tape 3 maintains information on where *D* is in *N*'s tree
    - A string of numbers represents following that path in the tree – e.g., 123 means the root's first child, then that node's second child, then that node's third child
    - Note that some strings may not correspond to real computations in *N*



NONDETERMINISTIC TMS AND EQUIVALENCE WITH DETERMINISTIC TMS

1. Initially, tape 1 will contain *w* and tapes 2 and 3 will be empty
2. Copy the contents of tape 1 to tape 2
3. Simulate *N* on tape 2 along a particular branch using input *w*.
   a) Before each step of *N*, look at the next symbol on tape 3 to determine which choice of transition to make.
   b) If tape 3 has no more symbols, a reject state is found, or the string on tape 3 is invalid, go to stage 4
   c) If an accept state is found, accept
4. Replace the string on tape 3 with the next string in lexicographical order. Simulate this new branch by going to stage 2

COROLLARY

- A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it
- Proof:
  - A deterministic TM is a nondeterministic TM that only ever has one element in the set returned by δ
    - Means that NTMs can recognize all languages TMs recognize
  - Other direction follows from prior proof, i.e., TMs can simulate NTMs
    - Means that TMs recognize all languages NTMs recognize

Let us consider some variants of the TM and argue about their computational power.

1. **Multitape TM:** The TM is allowed to have k tapes. It has a separate head in each of the k tapes. In each transition step, each tape replaces the bit under its respective head and moves left/right.

We can simulate all the k tapes using a single tape. The contents of the different tapes are written side by side, separated by some marker symbol. Two issues arise in this case.

What if the contents of a tape increases?
For each additional symbol added to a tape we shift the contents on the right side, to the right by one cell.

How do we keep track of the head positions for each individual tape?
For each tape symbol, say a, in the original TM we add another symbol 'a'.This new symbol is used to keep track of the head position in each tape's portion.

2. **Two stacks:** The two stacks store the contents of the tape on the left side and right side of the tape head respectively.

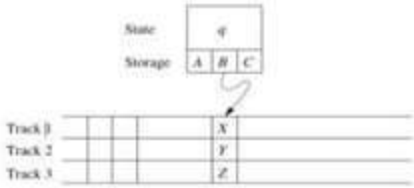
3. **Queue**: Can be simulated using two stacks.

PROGRAMMING TECHNIQUES OF TURING MACHINE

1.Storage in the Finite Control
2.Multiple tracks on a single tape
3.Checking of Symbols
4..Subroutines

1.STORAGE IN THE STATE

- This technique use the finite control of a turing machine to hold a finite amount of data, in addition to the state. The finite control can also be used to hold a finite amount of information along with the task of representing a position in the program
- The finite automata stores the information in pair of elements such as current state and the current symbol pointed by the tape head.



Problem : 1

Use the technique of storage in the finite control to design a Turing machine which takes the leftmost symbol in the input and prints it immediately to the right of the input.

Solution:

M=(Q,{0,1},{0,1,B},&,[q0,B],B,[q0,B])

Set of states {q0,q1,q2} and {0,1,B}

& is

1) δ ([q0,B],q)=([q1,a], a, R)

Q0 is the control state , the data portion of state is B. The symbol scanned is copied into the q0 component of the state, M moves right, entering control state q, as it does so.

2) δ ([q1,a],b)=([q ,a], b, R)

In state q1, M skips over each non blank symbol and continuous moving right.

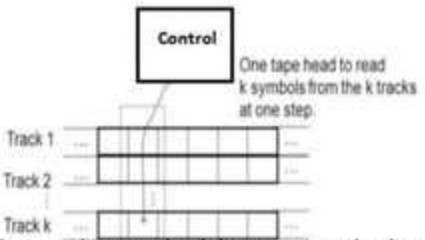
3) δ ([q1,a],B)=([q1,B], a, R)

If M reaches the first blank, it copies the symbol to from its finite control to the tape and enters the accepting state (q1, B).

[NOTE: If M encounters a second occurrence of the symbol it stored initially in its finite control, it has without having entered the accepting state.]

2. MULTIPLE TRACKS

- The turing machines input tape can be divided into several tracks. Each track can hold one symbol ad the tape alphabet of the ™ consists of tuples with one component for each track.



TM has several tracks. Each track can hold one symbol and the tape alphabet of the TM consists of tuples , with one component for each " track ". Example - checking of symbols.

PROBLEM : 2

Design a TM to check whether the given input is a prime or not using multiple tracks:

Solution:

The binary input greater than two is placed on the first track and also the same input is placed on the third track. The TM writes the number two in binary form on the second track. Then divide the third track by the second as follows.

The number on the second track is subtracted from the third track as many times as possible, till getting the remainder, if the remainder is 0, then number on the first track is not prime.

If the remainder is a non zero number, then increase the number on the second track by one.

I f the second track equals the first, the number given is a prime number

because it should be divide by one and itself.

Problem (i) check 8 is prime or not

TRACK 1	8	8	8	8	8
TRACK 2	2	2	2	2	2
TRACK 3	8	6	4	2	0

Thus, the given number 8 is not prime

Problem (ii) check 5 is prime or not

TRACK 1	5	5	5	5	5	5	5	5
TRACK 2	2	2	2	3	3	4	4	5
TRACK 3	5	3	1	5	2	5	1	5

Thus, the given number 5 is prime

PROBLEM : 3

Construct a TM that takes an input greater than 2 and checks whether it is even or odd:

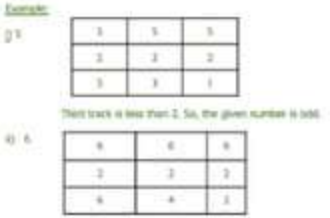
Solution:

The input is placed on the first track and the integer 2 is placed on the second track. The input on the first track is copied into third track.

Initial state:

The number on the second track is subtracted from the third track. If the remainder is same as the number in second track, then the given number is even.

If it is greater than 2, then continue the process until the remainder in the third track is less than or equal to 2. If it is equal to 2, the number is even or else the number is odd.



Third track is less than 2. So, the given number is odd.

SUBROUTINES:

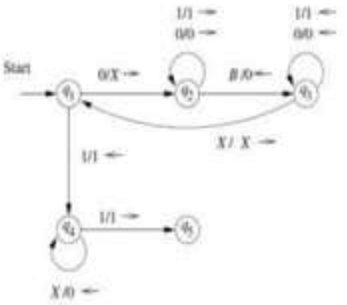
TM can also be built from a **collection of interacting components**, or "subroutines". A TM subroutine is a set of states that perform **some useful process**.

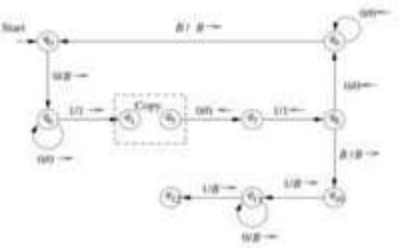
This set of states includes a start state and another state that temporarily has no moves, and that serves as the "return" state to pass contest to whatever other set of states called the subroutine.

Call :- Occurs when there is a transition to its initial state. TM has no mechanism to remember "**return address**".

Therefore, the "calls" are made to start states of different copies of the subroutine, and each copy returns to a different state.

CONSTRUCT A TURING MACHINE FOR MULTIPLE OF INTEGERS USING SUBROUTINE





LANGUAGES OF TURING MACHINES

Recall, an input x on TM M. M can

- Halt and accept x, $x \in L(M)$
- Halt and reject x, $x \notin L(M)$
- Crash, $x \notin L(M)$
- Run forever, $x \notin L(M)$

This defines 2 different classes of languages:

TM M **accepts** language L if $L = L(M)$.

- M accepts x if and only if $x \in L$
- May loop forever

TM M **decides** language L if $L = L(M)$ and if $x \notin L$, M rejects or crashes on x.

- M always stops
- No infinite looping

LANGUAGES OF TURING MACHINES

A language is **recursive (or decidable)** if there exists a TM M that decides L.

A language is **recursively enumerable** if there exists a TM M that accepts L.

If L is *recursive* then L is also *recursively enumerable*.

- A TM that *decides* L also *accepts* L.

If L is recursive then the complement L' is also recursive.

TM for L': Run x on M (the TM that decides L)

- If M accepts x then reject x.
- If M rejects or crashes, then accept x.

CLOSURE PROPERTIES OF TURING MACHINES

CLOSURE PROPERTIES OF TURING MACHINES

Union and Intersection

Recursive

If L_1 is recursive and L_2 is recursive then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursive.
Use a multitape TM:

- Copy input to tape 2 and tape 3
- Execute M_1 on tape 2 and M_2 on tape 3 (neither will run forever; i.e. we get a result)
- They will decide whether x is in L_1 and/or L_2
- Test if both M_1 and M_2 accepted (intersection)

- Test if one of M_1 and M_2 accepted (union)

If L_1 and L_2 are recursive then the difference $L_1 - L_2 = L_1 \cap L_2'$ is recursive.

CLOSURE PROPERTIES OF TURING MACHINES

Recursively Enumerable

If L_1 and L_2 are recursively enumerable then $L_1 \cup L_2$ and $L_1 \cap L_2$ are recursively enumerable.

- Similar to the recursive case but need to handle the case where M_1 and M_2 can run forever.
- Simulate M_1 and M_2 running simultaneously – alternate one step from each machine.

For example, union

- If either machine ever accepts then accept
- If either machine ever rejects or crashes then continue to work on the other machine.

If L is recursively enumerated and L' is recursively enumerable then L is recursive.

- Let M and M' be TMs that accept L and L', respectively.
- Run M and M' simultaneously.
- For any word x, it must be accepted by one of M or M'
- So, either M or M' will halt and accept
- If M halts and accepts then halt and accept
- If M' halts and accepts then halt and reject

The TM that runs M and M' simultaneously always halts and accepts or rejects so it decides L and L is recursive.

If L is recursively enumerable and L is not recursive then L' is not recursively enumerable.

TURING MACHINE CODES

- May assume

$$M = (Q, \{0,1\}, \{0,1,\#\}, \delta, q_1, \#, \{q_2\})$$

- Unary encoding

$$0 \mapsto 0, 1 \mapsto 00, \# \mapsto 000, L \mapsto 0, R \mapsto 00$$

- $\delta(q_i, X) = (q_j, Y, R)$ encoded by

$$0^i 1 \underbrace{0 \ldots 0}_{X} 1 0^j 1 \underbrace{0 \ldots 0}_{Y} 1 \underbrace{0 \ldots 0}_{R}$$

- δ encoded by

$$111\,\text{code}_1\,11\,\text{code}_2\,11 \ldots 11\,\text{code}_r\,111$$

- Encoding of Turing machine M denoted by $\langle M \rangle$.

CODES OF TM

PROBLEM :

Let the TM in question be

M= ({q1, q2, q3}, {0,1}, {0,1, B}, δ, q1, B, {q2})

where δ consists of the rules:

δ (q1, 1) = (q3,0, R)

δ (q3,0) = (q1, 1, R)

δ (q3,1)=(q2,0,R)

δ (q3,B)=(q3,1,L)

The codes for each of these rules, respectively, are:

0100100010100

0001010100100

00010010010100

0001000100010010

For example, the first rule can be written as δ (q1,X2)=(q3,X1,D2), since 1=X2 , 0=X1 and R=D2.Thus, its code is 01102103101102 , as was indicated above.

A code for M is:

000101010010000010010010100000010001000100010

Note that there are many ether possible codes for M. In particular, the codes for the four transitions may be listed in any of 4! orders, giving us 24 codes for M.

Thank you