UNIT-5

Turing Machine in TOC

Turing Machine was invented by Alan Turing in 1936 and it is used to accept Recursive Enumerable Languages (generated by Type-0 Grammar).

A turing machine consists of a tape of infinite length on which read and writes operation can be performed. The tape consists of infinite cells on which each cell either contains input symbol or a special symbol called blank. It also consists of a head pointer which points to cell currently being read and it can move in both directions.

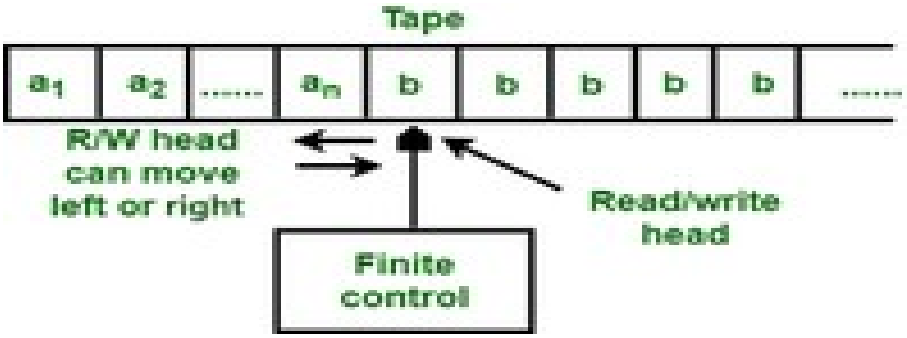


Figure: Turing Machine

A TM is expressed as a 7-tuple (Q, T, B, ∑, δ, q0, F) where:

- **Q** is a finite set of states
- **T** is the tape alphabet (symbols which can be written on Tape)
- **B** is blank symbol (every cell is filled with B except input alphabet initially)
- **∑** is the input alphabet (symbols which are part of input alphabet)
- **δ** is a transition function which maps Q × T → Q × T × {L,R}. Depending on its present state and present tape alphabet (pointed by head pointer), it will move to new state, change the tape symbol (may or may not) and move head pointer to either left or right.
- **q0** is the initial state
- **F** is the set of final states. If any state of F is reached, input string is accepted.

Let us construct a turing machine for L={0^n1^n|n>=1}

- Q = {q0,q1,q2,q3} where q0 is initial state.
- T = {0,1,X,Y,B} where B represents blank.
- ∑ = {0,1}
- F = {q3}

Transition function δ is given in Table 1 as:

	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)	**Halt**

Illustration

Let us see how this turing machine works for 0011. Initially head points to 0 which is underlined and state is q0 as:

B	0	0	1	1	B

The move will be δ(q0, 0) = (q1, X, R). It means, it will go to state q1, replace 0 by X and head will move to right as:

B	X	0	1	1	B

The move will be δ(q1, 0) = (q1, 0, R) which means it will remain in same state and without changing any symbol, it will move to right as:

The move will be δ(q1, 1) = (q2, Y, L) which means it will move to q2 state and changing 1 to Y, it will move to left as:

Working on it in the same way, the machine will reach state q3 and head will point to B as shown:

Using move δ(q3, B) = halt, it will stop and accepted.

Note:

- In non-deterministic turing machine, there can be more than one possible move for a given state and tape symbol, but non-deterministic TM does not add any power.
- Every non-deterministic TM can be converted into deterministic TM.
- In multi-tape turing machine, there can be more than one tape and corresponding head pointers, but it does not add any power to turing machine.
- Every multi-tape TM can be converted into single tape TM.

Question: A single tape Turing Machine M has two states q0 and q1, of which q0 is the starting state. The tape alphabet of M is {0, 1, B} and its input alphabet is {0, 1}. The symbol B is the blank symbol used to indicate

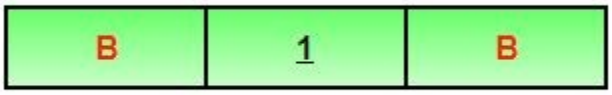
end of an input string. The transition function of M is described in the following table.

	0	1	B
q0	q1,1,R	q1,1,R	Halt
q1	q1,1,R	q0,L,R	q0,B,L

The table is interpreted as illustrated below. The entry (q1, 1, R) in row q0 and column 1 signifies that if M is in state q0 and reads 1 on the current tape square, then it writes 1 on the same tape square, moves its tape head one position to the right and transitions to state q1. Which of the following statements is true about M?

1. M does not halt on any string in (0 + 1)+
2. M does not halt on any string in (00 + 1)*
3. M halts on all string ending in a 0
4. M halts on all string ending in a 1

Solution: Let us see whether machine halts on string '1'. Initially state will be q0, head will point to 1 as:



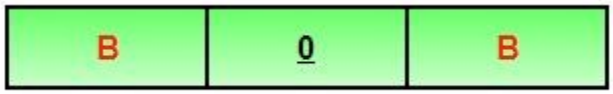
Using δ(q0, 1) = (q1, 1, R), it will move to state q1 and head will move to right as:

Using δ(q1, B) = (q0, B, L), it will move to state q0 and head will move to left as:

It will run in the same way again and again and not halt.

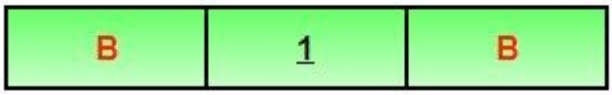
Option D says M halts on all string ending with 1, but it is not halting for 1. So, option D is incorrect.

Let us see whether machine halts on string '0'. Initially state will be q0, head will point to 1 as:



Using δ(q0, 0) = (q1, 1, R), it will move to state q1 and head will move to right as:

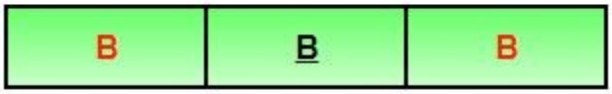
Using δ(q1,B)=(q0,B,L), it will move to state q0 and head will move to left as:



It will run in the same way again and again and not halt.

Option C says M halts on all string ending with 0, but it is not halting for 0. So, option C is incorrect.

Option B says that TM does not halt for any string (00 + 1)*. But NULL string is a part of (00 + 1)* and TM will halt for NULL string. For NULL string, tape will be,



Using δ(q0, B) = halt, TM will halt. As TM is halting for NULL, this option is also incorrect.
So, option (A) is correct.

This article is contributed by **Sonal Tuteja**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recursive and Recursive Enumerable Languages in TOC

Recursive Enumerable (RE) or Type -0 Language

RE languages or type-0 languages are generated by type-0 grammars. An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language and may or may not enter into rejecting state for the strings which are not part of the language. It means TM can loop forever for the strings which are not a part of the language. RE languages are also called as Turing recognizable languages.

Recursive Language (REC)

A recursive language (subset of RE) can be decided by Turing machine which means it will enter into final state for the strings of language and rejecting state for the strings which are not part of the language. e.g.; L= $\{a^nb^nc^n|n>=1\}$ is recursive because we can construct a turing machine which will move to final state if the string is of the form $a^nb^nc^n$ else move to non-final state. So the TM will always halt in this case. REC languages are also called as Turing decidable languages.

The relationship between RE and REC languages can be shown in Figure 1.

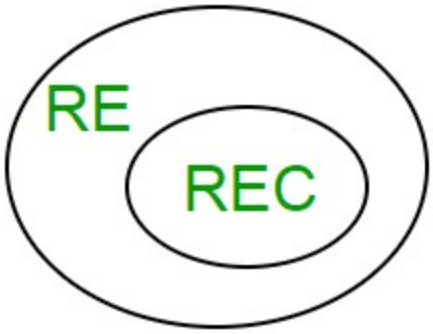


Figure 1

Closure Properties of Recursive Languages

- **Union**: If L1 and If L2 are two recursive languages, their union L1∪L2 will also be recursive because if TM halts for L1 and halts for L2, it will also halt for L1∪L2.
- **Concatenation:** If L1 and If L2 are two recursive languages, their concatenation L1.L2 will also be recursive. For Example:

  L1= $\{a^n b^n c^n | n>=0\}$

  L2= $\{d^m e^m f^m | m>=0\}$

  L3= L1.L2

  = $\{a^n b^n c^n d^m\ e^m f^m | m>=0$ and $n>=0\}$ is also recursive.
- L1 says n no. of a's followed by n no. of b's followed by n no. of c's. L2 says m no. of d's followed by m no. of e's followed by m no. of f's. Their concatenation first matches no. of a's, b's and c's and then matches no. of d's, e's and f's. So it can be decided by TM.
- **Kleene Closure:** If L1is recursive, its kleene closure L1* will also be recursive. For Example:

  L1= $\{a^n b^n c^n | n>=0\}$

  L1*= $\{ a^n b^n c^n || n>=0\}^*$ is also recursive.
- **Intersection and complement**: If L1 and If L2 are two recursive languages, their intersection L1 ∩ L2 will also be recursive. For

Example:

L1= {$a^n b^n c^n dm | n>=0$ and $m>=0$}

L2= {$a^n b^n c^n d^n | n>=0$ and $m>=0$}

L3=L1 ∩ L2

= { $a^n b^n c^n d^n$ |n>=0} will be recursive.

L1 says n no. of a's followed by n no. of b's followed by n no. of c's and then any no. of d's. L2 says any no. of a's followed by n no. of b's followed by n no. of c's followed by n no. of d's. Their intersection says n no. of a's followed by n no. of b's followed by n no. of c's followed by n no. of d's. So it can be decided by turing machine, hence recursive. Similarly, complement of recursive language L1 which is ∑*-L1, will also be recursive.

Note: As opposed to REC languages, RE languages are not closed under complementation which means complement of RE language need not be RE.

Turing Machine Construction

Turing Machines can broadly be classified into two types, the Acceptors and the Transducers. Acceptor Turing Machine is an automaton used to define Turing-acceptable languages. Such a machine can be used to check whether a given string belongs to a language or not. It is defined as a 7-tuple machine.

Coming to Transducers: In general, transducers are the devices used to convert one form of signal into another. The same can be told about Turing Machine Transducers.

A transducer is a type of Turing Machine that is used to convert the given input into the output after the machine performs various read-writes. It doesn't accept or reject an input but performs series of operations to obtain the output right in the same tape and halts when finished.

Few examples of Turing Machine transducers are:

- Turing Machine for addition
- Turing Machine for subtraction
- Turing Machine for multiplication
- Turing Machine for 1's and 2's complement

Implementation:

Now we will be proposing a Java program that was written to simulate the construction and execution performed by turing machine transducers. There are two inputs that must be given while executing it: A .txt file to define the automaton (an example for unary multiplication machine is given after the code), and a string to be entered via the console window which will be the input on tape for the automaton to execute.

The .txt file's path must be given as input. This was done so that the same program can be used for various types of machines instead of hard-coding the automaton. We just need to write a different txt file for generating a different automaton.

Java was chosen specifically because of the OOP structure, using which a class was defined for State, Transition, Machine, etc, to be able to encapsulate various aspects of an entity within an object. For example, a transition is defined as a class whose members are three characters – read, write and shift which store read symbol, write a symbol, and shift direction respectively, along with the index of the next state to which the machine should transition to. The same applies to a State object, which stores a list of possible outgoing transitions.

Variation of Turing Machine

1. Multiple track Turing Machine:

- A k-track 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

2. 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).

3. Multi-tape Turing Machine:

- It has multiple tapes and is controlled by a single head.
- The Multi-tape Turing machine is different from k-track Turing machine but expressive power is the same.
- Multi-tape Turing machine can be simulated by single-tape Turing machine.

4. Multi-tape Multi-head Turing Machine:

- The multi-tape Turing machine has multiple tapes and multiple heads

- Each tape is controlled by a separate head
- Multi-Tape Multi-head Turing machine can be simulated by a standard Turing machine.

5. Multi-dimensional Tape Turing Machine:

- It has multi-dimensional tape where the head can move in any direction that is left, right, up or down.
- Multi dimensional tape Turing machine can be simulated by one-dimensional Turing machine

6. Multi-head Turing Machine:

- A multi-head Turing machine contains 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 a single head Turing machine.

7. Non-deterministic Turing Machine:

- A non-deterministic Turing machine has a single, one-way infinite tape.
- For a given state and input symbol has at least one choice to move (finite number of choices for the next move), each choice has several choices of the path that it might follow for a given input string.
- A non-deterministic Turing machine is equivalent to the deterministic Turing machine.

Introduction to Linear Bounded Automata :
A Linear Bounded Automaton (LBA) is similar to Turing Machine with some properties stated below:

- Turing Machine with Non-deterministic logic,
- Turing Machine with Multi-track, and
- Turing Machine with a bounded finite length of the tape.

Tuples Used in LBA :
LBA can be defined with eight tuples (elements that help to design automata) as:

$M = (Q , T , E , q0 , M_L , M_R , S , F)$,

where,

Q -> A finite set of transition states

T -> Tape alphabet

E -> Input alphabet

q0 -> Initial state

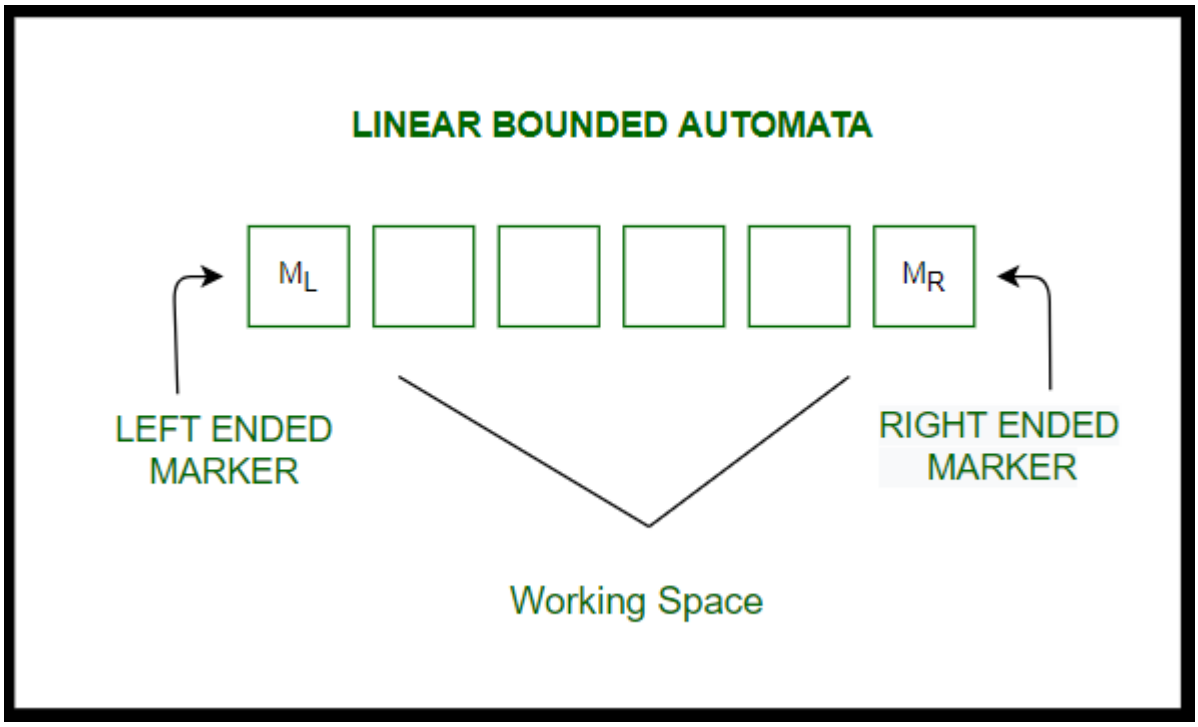
M_L -> Left bound of tape

M_R -> Right bound of tape

S -> Transition Function

F -> A finite set of final states

Diagrammatic Representation of LBA :



Examples:

Languages that form LBA with tape as shown above,

- L = {$a^{n!}$ | n >= 0}
- L = {wn | w from {a, b}+, n >= 1}
- L = {wwwR | w from {a, b}+}

Facts :

Suppose that a given LBA M has

--> q states,

--> m characters within the tape alphabet, and

--> the input length is n

1. Then M can be in at most f(n) = q * n * mn configurations i.e. a tape of n cells and m symbols, we are able to have solely mn totally different tapes.
2. The tape head is typically on any of the n cells which we have a tendency to are typically death penalty in any of the q states.