FORMAL LANGUAGE AND AUTOMATA THEORY.

UNIT – V: Turning Machine

Turing Machine, Definition, Model, Representation of Turing Machines- Instantaneous Descriptions, Transition Tables and Transition Diagrams, Language of a Turing Machine, Design of Turing Machines, Techniques for Turing Machine Construction, Types of Turing Machines, Church's Thesis, Universal Turing Machine, Restricted Turing Machine.

Unit-V

Turing Machine, Definition, Model:

A Turing Machine is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars. It was invented in 1936 by Alan Turing.

Definition

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.

A TM can be formally described as a 7-tuple (Q, X, Σ, δ, q_0, B, F) where –

- **Q** is a finite set of states
- **X** is the tape alphabet
- **Σ** is the input alphabet
- **δ** is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{Left\_shift, Right\_shift\}$.
- **q_0** is the initial state
- **B** is the blank symbol
- **F** is the set of final states

Comparison with the previous automaton

The following table shows a 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

Important and Previous JNTU Questions:

1Q. Show how a turing machine tape processes the input string "abababab".

2Q. In which direction does the head of a turing machine move?

3Q. What does input tape contain in a turing machine? 2Q. What does symbol 'B' represent in an input tape?

4Q. Give the formal definition of TM? What are the different types of TMs? Explain. [May / June 2015, Set 3, 6M]

5Q. Differentiate between PDA and TM with respect to: halt state and final state [May / June 2015, Set 3, 4M]

6Q. Define universal Turing machine and universal language. [April/May 2016, Set 1, 3M]

Representation of Turing Machines-Instantaneous Descriptions, Transition Tables and Transition Diagrams

Example of Turing machine

Turing machine M = (Q, X, Σ, δ, q_0, B, F) with

- Q = {q_0, q_1, q_2, q_f}
- X = {a, b}
- Σ = {1}
- q_0 = {q_0}
- B = blank symbol
- F = {q_f }

δ is given by −

Tape alphabet symbol	Present State 'q₀'	Present State 'q₁'	Present State 'q₂'
A	$1Rq_1$	$1Lq_0$	$1Lq_f$
B	$1Lq_2$	$1Rq_1$	$1Rq_f$

Here the transition $1Rq_1$ implies that the write symbol is 1, the tape moves right, and the next state is q_1. Similarly, the transition $1Lq_2$ implies that the write symbol is 1, the tape moves left, and the next state is q_2.

Time and Space Complexity of a Turing Machine

For a Turing machine, the time complexity refers to the measure of the number of times the tape moves when the machine is initialized for some input symbols and the space complexity is 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)

Important and Previous JNTU Questions:

1Q. Draw Transition tables for the 2 forms of diagrams shown above. 2Q. In how many ways can a move be specified for a turing machine?
2Q. How to draw a transition table for a turing machine?

3Q. In how many tuples can the move of a turing machine be described?

4Q. Give the formal definition of TM? What are the components of TM? What is id of TM?
[May / June 2015, Set 2, 6M]

5Q. Give the formal definition of TM? Give the block diagram of TM
[May / June 2015, Set 1, 3M]

6Q. Draw a transition diagram for Turing machine and explain it in detail.
[April/May 2016, Set 2, 6M]

Language of a Turing Machine, Design of Turing Machines

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

The basic guidelines of designing a Turing machine have been explained below with the help of a couple of examples.

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 q1 be the initial state.
- If M is in q1; on scanning α, it enters the state q2 and writes B(blank).
- If M is in q2; on scanning α, it enters the state q1 and writes B(blank).
- From the above moves, we can see that M enters the state q1 if it scans an even number of α's, and it enters the state q2 if it scans an odd number of α's. Hence q2 is the only accepting state.

Hence,

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

where δ is given by −

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

Example 2

Design a Turing Machine that reads a string representing a binary number and erases all leading 0's in the string. However, if the string comprises of only 0's, it keeps one 0.

Solution

Let us assume that the input string is terminated by a blank symbol, B, at each end of the string.

The Turing Machine, M, can be constructed by the following moves −

- Let q0 be the initial state.

- If M is in q0, on reading 0, it moves right, enters the state q1 and erases 0. On reading 1, it enters the state q2 and moves right.
- If M is in q1, on reading 0, it moves right and erases 0, i.e., it replaces 0's by B's. On reaching the leftmost 1, it enters q2 and moves right. If it reaches B, i.e., the string comprises of only 0's, it moves left and enters the state q3.
- If M is in q2, on reading either 0 or 1, it moves right. On reaching B, it moves left and enters the state q4. This validates that the string comprises only of 0's and 1's.
- If M is in q3, it replaces B by 0, moves left and reaches the final state qf.
- If M is in q4, on reading either 0 or 1, it moves left. On reaching the beginning of the string, i.e., when it reads B, it reaches the final state qf.

Hence,

$M = \{\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{0,1, B\}, \{1, B\}, \delta, q_0, B, \{q_f\}\}$

where δ is given by –

Tape alphabet symbol	Present State 'q_0'	Present State 'q_1'	Present State 'q_2'	Present State 'q_3'	Present State 'q_4'
0	BRq_1	BRq_1	ORq_2	-	OLq_4
1	$1Rq_2$	$1Rq_2$	$1Rq_2$	-	$1Lq_4$
B	BRq_1	BLq_3	BLq_4	OLq_f	BRq_f

Important and Previous JNTU Questions:

1Q. Give the formal definition of TM? Give the block diagram of TM.
[May / June 2015, Set 1, 3M]

2Q. What does the symbol 'h' represent in a turing machine?

Techniques for Turing Machine Construction

A Turing Machine can work as any of the followings:

Turing Machine as a Language Acceptor
Turing Machine as a Language Generator
Turing Machine as a Transducer (Function Solver)

The basic strategy for designing a TM is given below:

a. The objective of scanning a symbol by the tape head is to know about the future status. The machine must remember the symbols scanned previously, by going to the next unique state.
b. The number of states must be minimised. This can be achieved by changing the states:
    - Only when there is a change in the written symbol or
    - When there is a change in the movement of the tape head.

There are different ways to describe the task of Turing machine:
The transition diagram:
The Turing machine can be represented using the transition diagram. For a directed graph, an arc going from the vertex (which corresponds to the state P) to the vertex that corresponds to the state q, and the also the edge label, can be represented in different forms as follows:

FORM-1:

Read symbol(a) → write symbol (b), move Left(L) or move right or No move (N)

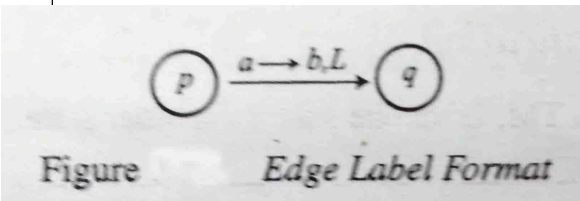


Figure Edge Label Format

FORM-2:

Read symbol(a)/ write symbol (b), move Left(L) or move right(R) or Do not move (N)

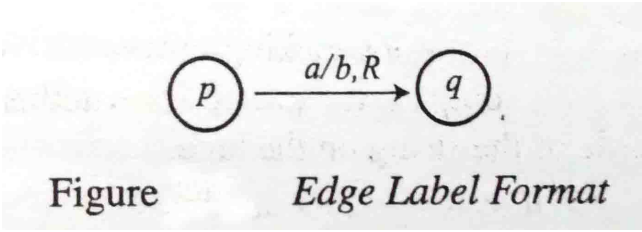


Figure Edge Label Format

Important and Previous JNTU Questions:

1Q. Design a Turing Machine that replaces all 0's with 1's.

2Q. What are the possible moves in a turing machine?

3Q. What is the objective of scanning a symbol by the tape head? 2Q. How can we minimize the number of state in a turing machine?

4Q. Design Turing machine to compute the function n! (Factorial of a number)
[May / June 2015, Set 4, 12M]

5Q. What is an infinite loop in TM? Explain with an example.
[May / June 2015, Set 4, 3M]

6Q. Design a Turing Machine for L = {wcwR/w∈(0+1)*}
[May / June 2015, Set 3, 10M]

7Q. When do you say that a Turing machine accepts a string?
[April/May, Set 3, 3M]

Types of Turing Machines, Church's Thesis

Intuitive notion of computation equals **Turing-machine model of computation.**

The thesis is not a mathematical statement and therefore it is not possible to prove it (in the usual mathematical sense). Instead we should view the thesis as a scientific hypothesis. Nevertheless, the thesis makes interesting mathematical predictions that we can prove or disprove rigorously. (A disproof of a prediciton of the Church–Turing thesis would falsify the hypothesis.)

Computing on Configurations

We will discuss examples of other computational models (variants of Turing machines) and verify that these models are no more powerful than ordinary Turing machines. (In this way, we verify predictions of the Church–Turing thesis.)

In order to simulate another computational model by Turing machines, we will follow a simple strategy:

- Identify a notion of configuration for the competing computational model.
- Encode these configurations as strings.
- Argue that Turing machines can manipulate these strings in a way that simulates the

computation of the competing model.

Multi-tape Turing machines

Suppose we augment Turing machine with multiple tapes and multiple independent heads. As before, the finite-state control determines the behavior of the heads depending on what the heads read in the current step. Formally, we have a transition function $\delta: Q \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k$ to control k-independent heads.

Theorem: Single-tape Turing machines can simulate multi-tape Turing machines.

Proof Sketch: For simplicity we consider the case k=2. We can encode the configuration of a 2-tape machine M as a string C of the form

$$C = u_1 q a_1 v_1 \# u_2 q a_2 v_2.$$

Here, q Q is the state of the control, u_i Γ* is the content of tape i before the head, v_i Γ* is the content of tape i after the head, and a_i Γ is the symbol that head i is reading in the current step. (The hash symbol doesn't belogs to Γ delimits the contents of the two tapes.)

The following single-tape Turing machine M' simulates the computation of the 2-tape machine M. For simplicity, we can choose the tape alphabet of M' such that it contains all elements of Γ Q {#} (but we could also choose to encode everything as binary strings).

Operation of M' on input w:

- Write the start configuration $C_0=q_0w\#q_0$ of M on the tape of M'.
- Repeat the following steps:
  - Read the configuration C of M on the tape of M'.
  - Accept if C is an accept-configuration for M and reject if C is a reject-configuration for M.
  - Manipulate the tape content such that it becomes the subsequent configuration C' of M.

Each of the steps can be implemented by a constant number of passes over the tape. After i full iterations of the loop, the tape of M' contains the configuration of M after i computaton steps. Therefore, M' accepts or rejects on input w if and only if M accepts or rejects on input w.

Non-deterministic Turing machines

Suppose we allow non-deterministic transitions in the finite-state control of a Turing machine M. Formally, we have a transition function δ such that δ(q,a) Q×Γ×{L,R} describes the possible transitions if the control is in state q and the head reads a. Then, a configuration C of M can yield multiple configurations C' in one step, denoted C B $_{\delta}$C'.

We say that a non-deterministic Turing machine **accepts** on input w if there *exists* a sequence of configurations C0,…,Cm such that C0 is the start configuration, C_i B δC_{i+1} for all i {0,…m−1}, and Cm is an accept-configuration. The language L(M) of a non-deterministic Turing machine M is the set of strings w Σ* such that M accepts on input w. (Unlike for deterministic machines, we don't define what it means for a non-deterministic machine to reject.)

Theorem: A language L is recognizable by non-deterministic Turing machines if and only if it is recognizable by deterministic Turing machines.

Proof: Let M be a non-deterministic Turing machine. We are to construct a deterministic Turing machine M' with the same language L(M')=L(M). The idea is that M' enumerates all possible computation paths of M.

Operation of M′ on input w:

- Enumerate all binary strings x (e.g., in lexicographic order) and perform the following operations:
  - Check that x encodes a sequence of configuration $C_0,\ldots,C_m$.
  - Check that $C_0=q_ow$ is the start configuration.
  - Check that $C_i \vdash \delta C_{i+1}$ for all i 0,…,m−1.
  - Check that Cm is an accept configuration.
  - Accept if all checks are passed.

The machine M′ can perfom each step of the loop by doing a finite number of passes over the tape content. If M′ accepts, then there exists an accepting computation path for M and therefore M accepts. On the other hand, if M accepts, then there exists an accepting computation path and an encoding of the path as a binary string x. The machine M′ will accept when the enumeration reaches the string x.

Computing on Turing Machines

Going the beyond idea of encoding configurations as strings and computing on them, we can encode whole machines as strings and compute on them. For a Turing machine M, we write

$$\langle M \rangle$$

to denote an encoding of it as a binary string. We assume that the formatting of the encoding is so that it is easy for another Turing machine to manipulate it. For convenience, we also assume that every binary string encodes some Turing machine. (For example, we can say if a binary string doesn't respect the usual formatting, then it decodes to a dummy machine M_0.)

Another property of such encodings that is sometimes useful is that for every machine M, there are infinitely many binary strings that decode to M or to a machine that behaves exactly in the same way as M. We can assume this property because by adding unreachable dummy states to M, we can get infinitely many machines that behave exactly like M.

In the following, we will use the notation to encode arbitrary mathematical objects as binary strings.

Universal Turing machines

We will show that there exists a Turing machine U that can simulate arbitrary Turing machines. Here, it is interesting that the number of states of U and its tape alphabet are fixed, but it can still simulate machines with many more states

and many more tape alphabet symbols. A Turing machine with this property is called universal. (In contrast, the machines in our previous constructions were allowed to depend on the machine they are simulating.)

The fact that modern computers are multi-purpose devices and can, in principle, execute arbitrary programs stems from this formal idea of universality.

Theorem: There exist a Turing machine U that on input <M,w> simulates the operation of the Turing machine M on input w {0,1}*.

Proof: It's convenient to first build a two-tape machine U_0 that achieves this task. Then, we can simulate U_0 by a single-tape machine U using our previous construction for multi-tape machines.

Operation of U_0 on input <M,w>.

- Write M's start configuration $C_0=q_0w$ as binary string C_0 on tape 1.
- Write M's description as binary string <M> on tape 2.
- Repeat the following steps:
  - *Invariant:* tape 1 holds the binary encoding of a configuration C of M.
  - Locate on tape 1 the state of M (encoded as binary string).
  - Locate on tape 1 the tape symbol under M's head (encoded as binary string).
  - Look up the corresponding part in the transition table of M on tape 2.
  - Accept or reject if the current state is the accept or reject of M.
  - Update the content of tape 1 according to this transition.

It's important to note the finite control of U_0 cannot "remember" the current state of M or the tape symbol that M is currently reading. Nevertheless, U_0 can find the positions of these items on the tapes. (We can implement this process by moving markers across the tape until we find a match.)

After i full iterations of the loop, tape 1 holds the configuration of M after I steps on input w. Therefore, the machine U_0 on <M,w> accepts or rejects if and only if M on w accepts or rejects.

Acceptance problem for Turing machines

Theorem: The acceptance problem ATMATM for Turing machines is recognizable,

$$A_{TM}=\{<M,w>|\ M \text{ accepts on input } w\}.$$

Proof: The universal Turing machine U from the previous theorem recognizes this language, because U accepts on input <M,w>|if and only if M accepts on input w.

Important and Previous JNTU Questions:

1Q. Design a Turing Machine for the {L=wwR/w∈(0+1)*}
[May/June 2015, Set 2, 10M]

2Q. Design a total Turing machine to accept the language: L2 = {w ∈ {a, b, c}*| #a(w)≤ #b(w) ≤ #c(w)} (Note: '#' means number)
[May/June 2015, Set 1, 12M]

3Q. Design A Turing Machine to recognize the language $\{1^n2^n3^n/n>=1\}$.
[April/May 2016, Set 4, 16M]

Universal Turing Machine, Restricted Turing Machine.

Universal Turing Machine

Theorem: There is a Turing machine UTM called the universal Turing machine that, when run on <M, w>, where M is a Turing machine and w is a string, simulates M running on w.

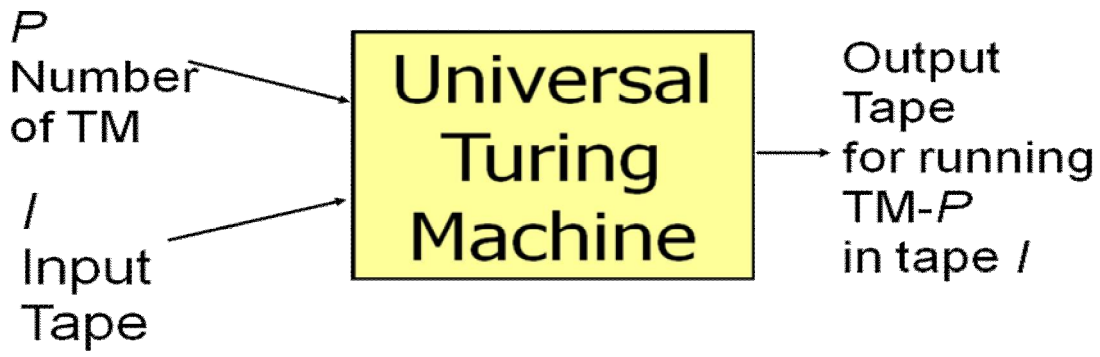
A "Universal Turing Machine," or 'UTM,' can emulate any other specific Turing machine, by defining states and symbols. The UTM is defined with certain capabilities.

- The UTM can define the symbols that the specific Turing machine will use.
- It can define the symbols that encode the states and transition rules for the specific Turing machine.
- It can encode the rules for that specific Turing machine onto the input tape.

A single-tape UTM needs to define a marker to mark the end of the "specific" program and the start of the specific machine's initial tape. It must also shuffle

the read/write head between the specific TM's program and its data. As noted, it is simpler to describe a UTM with multiple tapes.

The Universal Turing Machine is remarkably similar to the Von Neumann model of a computer, where both programs and data can be stored on the same medium. Any modern computer capable of copying a program file from one medium to another, and later running that program, follows this architecture.



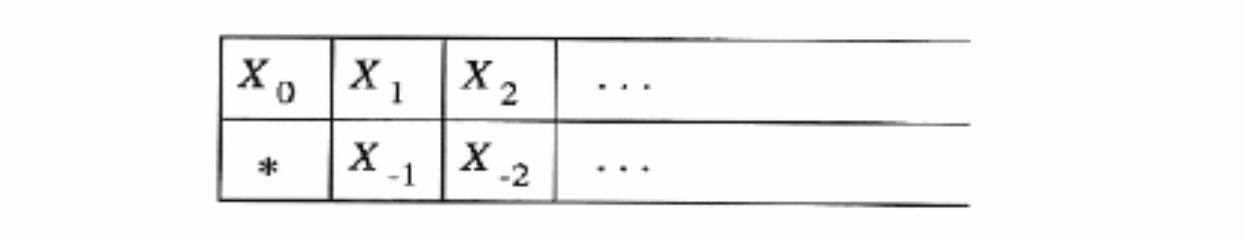
Restricted Turing Machine:

- **Turing machines** are extremely basic abstract symbol-manipulating devices which, despite their simplicity, can be adapted to simulate the logic of any computer that could possibly be constructed.
- They were described in 1936 by Alan Turing.
- Though they were intended to be technically feasible, Turing machines were not meant to be a practical computing technology, but a thought experiment about the limits of mechanical computation; thus they were not actually constructed
- Each move by a *Turing Machine* results in Change of state;
- Writing a tape symbol in the cell just scanned; and Moving the tape head left or right.

But in a Restricted Turing Machine have the following components:

- Semi-infinite Tapes' of the TM

- A TM with a ***semi-infinite*** tape means that there are no cells to the left of the initial head position.
- A TM with a semi-infinite tape simulates a TM with an infinite tape by using a two-track tape.
  The use of two tracks is as follows:
- The upper track represents the cells of the original TM that are at the right of the initial head position.
- The lower track represents the cells at the left of the initial head position, but in reverse order.
- A semi-infinite tape that apparently can simulate a two-way tape:



- Every language accepted by a TM *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.

Important and Previous JNTU Questions:

1Q. Design a Turing Machine which can multiply two positive integers.
[April/May 2016, Set 3, 16M]

2Q. Design a Turing machine to accept the set of al palindrome over {0,1}*. Draw a transition diagram for the Turing machine of the above.
[April/May 2016, Set 2, 10M]

3Q. With an example string demonstrate the action of input tape in a turing machine.

4Q. With an example show how to accept or reject a string using turing machine.