KARNATAK LAW SOCIETY’S

**GOGTE INSTITUTE OF TECHNOLOGY**

UDYAMBAG, BELGAUM-590008

(An Autonomous Institution under Visvesvaraya Technological University, Belgaum)

**(APPROVED BY AICTE, NEW DELHI)**



*Course Activity Report*

**TOPIC: Design and implementation Turing Machine for the following language L= (anbn for n>=1)**

*Submitted in the partial fulfillment for the academic requirement of*

***FIFTH Semester B.E.***

***in***

***FORMAL LANGUAGES and AUTOMATA***

*Submitted by:*

1). Aditya Nandeshwar                     2GI19CS008

2). Aniketh Mahadik       2GI19CS019

3). Atreay Kukanur 2GI19CS028

4). Fakkiresh Noorshettar 2GI19CS041

Karnataka Law Society’s

GOGTE INSTITUTE OF TECHNOLOGY

Udyambag Belagavi -590008

Karnataka, India.

**Department of Computer Science and Engineering**



**Certificate**

This is to certify that the Course Project work titled **“*Design and Implementation of Turing Machine*”** carried out by Aditya Nandeshwar**,** Aniketh Mahadik, Atreay Kukanur, Fakkiresh Noorshetter bearing **USNs:** 2GI19CS008,2GI19CS019, 2GI19CS028, 2GI19CS041is submittedin partial fulfilment of the requirements for 5th semester B.E. in **Computer Science and Engineering,** Visvesvaraya Technological University, Belagavi. It is certified that all corrections/ suggestions indicated have been incorporated in the report. The course project report has been approved as it satisfies the academic requirements prescribed for the said degree.

Date: Signature of Guide

Place: Belagavi Guide Name

***Prof.Mallikarjun Math***

KLS Gogte Institute Technology, Belagavi

Name of the Examiners Signature of the Examiners

1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Course Seminar report and ppt content**

**Marks allocation:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Batch No.: 11 | | | | | |
| 1. | Course Project Title:  Design and implementation Turing Machine for the following language L=(anbn for n>=1) | Marks Range | USN | | | |
| **2GI19CS008** | **2GI19CS019** | **2GI19CS028** | **2GI19CS041** |
| 2. | Abstract (PO2) | 0-2 |  |  |  |  |
| 3. | Application of the topic to the course (PO2) | 0-3 |  |  |  |  |
| 4. | Literature survey and its findings (PO2) | 0-4 |  |  |  |  |
| 5. | Methodology, Results and Conclusion (PO1,PO3,PO4) | 0-6 |  |  |  |  |
| 6. | Report and Oral presentation skill (PO9,PO10) | 0-5 |  |  |  |  |
|  | Total | 20 |  |  |  |  |

**1.Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

**2.Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of

mathematics, natural sciences and Engineering sciences.

**3.Design/Development of solutions:**Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**4.Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**5.Modern tool usage:**Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**6.The engineer and society:**Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**TABLE  OF CONTENTS :**

* INTRODUCTION
* DESIGNING OF TURING MACHINE
* METHODOLOGY
* CODE
* CONCLUSION
* REFERENCE

**Abstract:**

A **Turing machine** is a [mathematical model of computation](https://en.wikipedia.org/wiki/Mathematical_model_of_computation) that defines an [abstract machine](https://en.wikipedia.org/wiki/Abstract_machine) that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, given any [computer algorithm](https://en.wikipedia.org/wiki/Computer_algorithm), a Turing machine capable of implementing that algorithm's logic can be constructed.

A Turing machine is a general example of a [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) (CPU) that controls all data manipulation done by a computer, with the canonical machine using sequential memory to store data. More specifically, it is a machine ([automaton](https://en.wikipedia.org/wiki/Automaton)) capable of [enumerating](https://en.wikipedia.org/wiki/Enumeration) some arbitrary subset of valid strings of an [alphabet](https://en.wikipedia.org/wiki/Alphabet_(formal_languages)); these strings are part of a [recursively enumerable set](https://en.wikipedia.org/wiki/Recursively_enumerable_set). A Turing machine has a tape of infinite length on which it can perform read and write operations.

Assuming a [black box](https://en.wikipedia.org/wiki/Black_box), the Turing machine cannot know whether it will eventually enumerate any one specific string of the subset with a given program. This is due to the fact that the [halting problem](https://en.wikipedia.org/wiki/Halting_problem) is unsolvable, which has major implications for the theoretical limits of computing.

The Turing machine is capable of processing an [unrestricted grammar](https://en.wikipedia.org/wiki/Unrestricted_grammar), which further implies that it is capable of robustly evaluating first-order logic in an infinite number of ways. This is famously demonstrated through [lambda calculus](https://en.wikipedia.org/wiki/Lambda_calculus).

**Introduction to Turing Machine:**

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, ∑, δ, q0, B, F) where −

* **Q** is a finite set of states
* **X** is the tape alphabet
* **∑** is the input alphabet
* **δ** is a transition function; δ : Q × X → Q × X × {Left\_shift, Right\_shift}.
* **q0** is the initial state
* **B** is the blank symbol
* **F** is the set of final states

**Designing Turing Machines:**

● Despite their simplicity, Turing machines are very powerful computing devices.

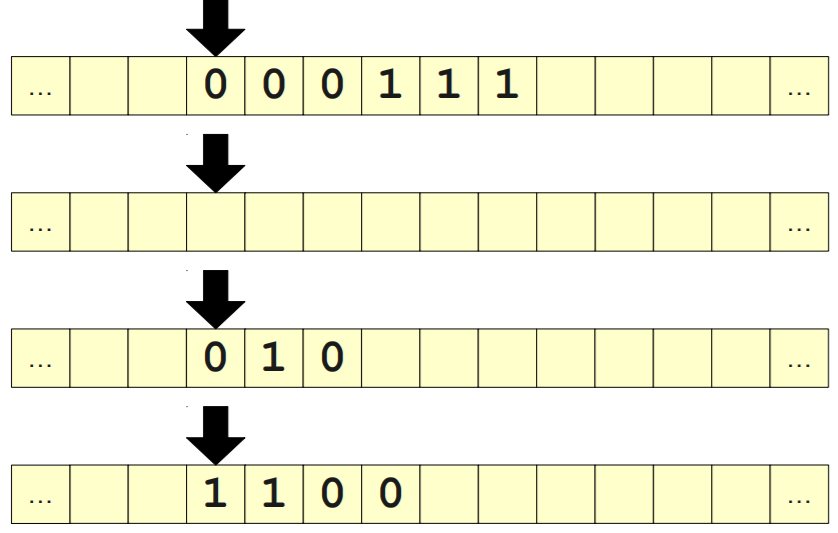
● Today's lecture explores how to design Turing machines for various languages. Designing Turing Machines

● Let Σ = {0, 1} and consider the language L = {0 n1 n | n ∈ ℕ}.

● We know that L is context-free.

● How might we build a Turing machine for it?

L = {0 n1 n | n ∈ ℕ}



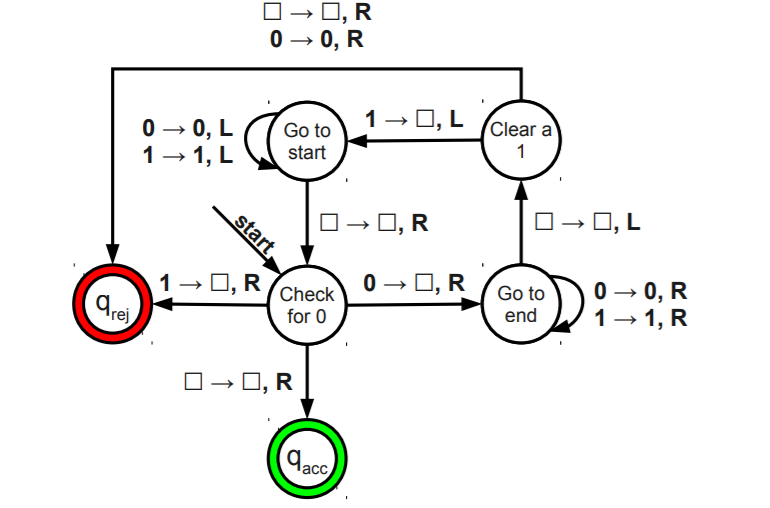
A Recursive Approach

● The string ε is in L.

● The string 0w1 is in L if w is in L.

● Any string starting with 1 is not in L.

● Any string ending with 0 is not in L.



Turing machine as an Enumerator

Simply put, an Enumerator is a Turing machine with an output printer. The Turing machine can use the

printer as an output device to print strings. The following diagram gives a schematic of an enumerator :-

Turing machine as an Enumerator

Simply put, an Enumerator is a Turing machine with an output printer. The Turing machine can use the

printer as an output device to print strings. The following diagram gives a schematic of an enumerator :-

**Given Problem:**

We have to design a Turing machine for anbn where n>=1.

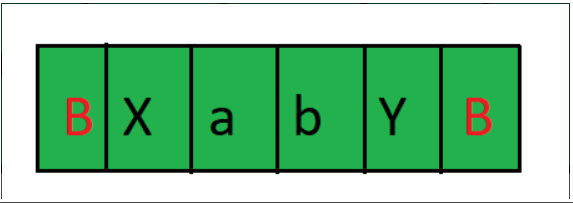
**Analysis:**

We can analyze that we have equal no of a’s and b’s and in some order i.e., first all a’s will come and then all b’s will come.

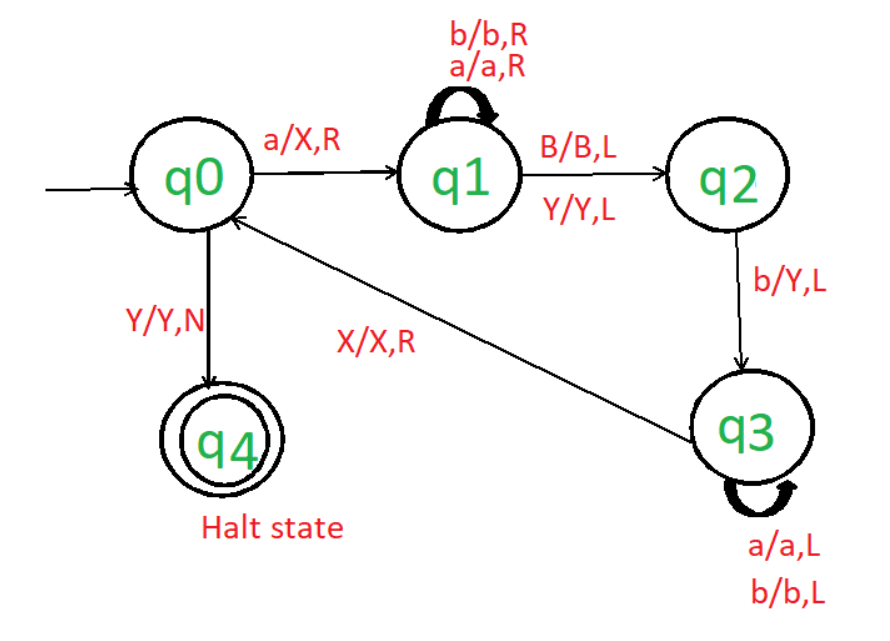
**Approach:**

Let us understand the approach by taking the example “aabb”.

* Scan the input from the left.
* First, replace an ‘a’ with ‘X’ and move right. Then skip all the a’s and b’s and move right.
* When the pointer reaches Blank(B) Blank will remain Blank(B) and the pointer turns left. Now it scans the input from the right and replaces the first ‘b’ with ‘Y’. Our Turing machine looks like this –



* Again the pointer reaches Blank(B) or X. It now scans the input from left to right. The pointer moves forward and replaces ‘a’ with ‘X’.
* Again the pointer reaches Blank(B) or Y. It now scans the input from the right to left. The pointer moves forward and replaces ‘b’ with ‘y’.
* We repeat the same steps until we convert all the a’s to ‘X’ and b’s to ‘Y’.
* When all the a’s converted to ‘X and all the b’s converted to ‘Y’ our machine will halt.



**Code:**

#function to perform action of states

def action(inp, rep, move):

    global tapehead

    if tape[tapehead] == inp:

        tape[tapehead] = rep

        if move == 'L':

            tapehead -= 1

        else:

            tapehead += 1

        return True

    return False

tape = ['B']\*50

string = input("Enter String: ")

i = 5

tapehead = 5

for s in string: #loop to place string in tape

    tape[i] = s

    i += 1

state = 0

a, b, X, Y, R, L, B = 'a', 'b', 'X', 'Y', 'R', 'L', 'B'

oldtapehead = -1

accept = False

while(oldtapehead != tapehead): #if tapehead not moving that means terminate Turing machine

    oldtapehead = tapehead

    if state == 0:

        if action(a, X, R):

            state = 1

        elif action(Y, Y, R):

            state = 3

    elif state == 1:

        if action(a, a, R) or action(Y, Y, R):

            state = 1

        elif action(b, Y, L):

            state = 2

    elif state == 2:

        if action(a, a, L) or action(Y, Y, L):

            state = 2

        elif action(X, X, R):

            state = 0

    elif state == 3:

        if action(Y, Y, R):

            state = 3

        elif action(B, B, L):

            state = 4

    else:

        accept = True

if accept:

    print("String accepted on state = ", state)

else:

    print("String not accepted on state = ", state)

**Conclusion:**

Possible applications are many because of the universal nature of Turing Machines. Theoretically any computational problem can be expressed as a (probably inefficient) Turing Machine, so analysis of a TM to solve the problem by this method could lead to information about the problem in general and perhaps to more efficient computational procedures for it. Functions mapping the set of real numbers to itself could be set up as Turing Machines with the property that the IRR are all right-moving, or are so after some point. The examples in this paper show that this can happen. This property would ensure that a given degree of precision of the input (initial tape) would determine a corresponding degree of precision in the answer, provided the moving-window in which the computation occurs has a left end that advances to the right as the computation proceeds.

**References:**

* Marvin L. Minsky “Computation: Finite and Infinite Machines” PrenticeHall, Inc, Englewood Cliffs, N.J. U.S.A., 1967.
* Nixon, John TM analyser A computer program written in C++ for the analysis of Turing Machines.
* Machlin R. and Stout Q.F. (1990) “The complex behavior of simple machines” Physica D 42, 85-98.
* Yap, Chee Keng , Introduction to Chapter 12 in “Fundamental problems in Algorithmic Algebra” Oxford University Press, 2000.
* Knuth D. E. and Bendix P. B. “Simple Word Problems in Universal Algebras.” In Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967). Pergamon Press, pp. 263-297, 1970.
* Nixon J., “Application of Lie groups and differentiable manifolds to general methods for simplifying systems of partial differential equations” J. Phys. A: Math. Gen. 24 (1991) 2913-2941.
* Nixon J., “Minimization of dimension for functional differential equations and the thermodynamics of the classical one-dimensional fluid” J. Phys. A Math. Gen. 27 (1994) 1407-1426.