Reflector – A Dynamic Manifestation of Turing Machines with Time and Space Complexity Analysis

Behroz Mirza
MS Computing,
Shaheed Zulfikar Ali Bhutto Institute of Science and Technology
90 and 100 Clifton
Karachi -75600

Muhammad Rafi
,
Shaheed Zulfikar Ali Bhutto Institute of Science and Technology
90 and 100 Clifton
Karachi -75600

Abstract—**The Turing Machine model has proven to be capable of simulating every known computational model. Since its inception the model has served as the basis on which computational devices have been constructed. The paper focuses on highlighting the fact that Turing Machine serves as a precise model for defining and executing a specific algorithm based on the Computability Theory. The paper also highlights the fact that when executed on a Turing Machine an algorithm's time and space complexity can be analyzed based on the Complexity Theory. Therefore, this model can serve as a superb abstraction for the computational devices as number of steps and space required to execute an algorithm can be predicted. It is worth mentioning that the simulator engine named Reflector designed in this study is the foremost simulator in the regards to have the distinct capabilities of; firstly dynamically modelling a Turing Machine based on its manifestations and secondly performing Time & Space Complexity Analysis of an algorithm executed on the Turing Machine.**

Keywords— Turing Machine, Computability Theory, Complexity Theory, Time Complexity, Space Complexity, Reflector

I. INTRODUCTION

The Turing Machine model has stood the test of time. It has proven to be capable of simulating every known computational model as was claimed by its inventor, Allen Turing. Complex computational Models like quantum computing, parallel and probabilistic computing, internet and digital computers which were not even conceived when this model was designed have been successfully simulated on it [1]. This model continues to serve as the basis on which computational devices have been constructed and has been an active area of research even after 75 years of its inception.

II. PROBLEM STATEMENT

Although considerable amount of work has been done in understanding of this illustrious model in terms of simulating machines based on their descriptions, however there has been considerably less work in developing simulators which are capable of exhibiting the multiple functionalities namely:

1. Dynamically modeling and simulating Turing Machines described on runtime specifications based on Computability Theory.
2. Performing Analysis of algorithms executed on these models in terms of Time Complexity and Space Complexity based on Complexity Theory.

This Independent Study attempts to design a simulator called Reflector that exhibits the above mentioned functionalities thus complying with the Computability Theory and Complexity Theory. This leads to highlighting the primary fact that the Turing Machine model remains an excellent abstraction for the computational devices because the number of steps and space required to execute an algorithm can be predicted.

III. LITERATURE REVIEW

A. Computation and the Turing Machine Concept

Many now view Computation as a fundamental part of nature, like atoms or the integers. The definition of the word 'computation' continues to evolve with the advancement in the field of Computer Science. In the Rebooting Computing Summit of January 2009, "What is Computation" was identified as the most important question in the computing field [2].

Alan Turing in his seminal paper precisely answered this question by describing his Turing Machine model, which states that "A Turing Machine is capable of computing anything that is computable [3].

Not only the classic Turing Machine model has served as the basis on which computational devices have been constructed but also the definition of computation as given by Turing has proven to be unalterable to date.

In the early part of 20th century the concepts of "computers" and "computation" were used but in different perspectives. Computers were those human beings who performed computations while Computation stood for the manual steps executed in a specific order for evaluating functions of mathematical nature. During this era, Kurt Gödel, Alonzo Church and Alan Turing [3] separately defined computation. Gödel defined it in terms of the recursive functions, Church in terms of the evaluations of "lambda expressions" and Turing defined it in terms of his Turing Machine model.

Moving forward it can be seen that the definition of Computation has progressively elaborated [4]

- 1940's it was termed as an area of study involving automatic computing
- 1950's it was termed as study involving information processing
- 1960's it was termed as study of phenomena around computers
- 1970's it was termed as study of what can possibly be automated
- 1980's it was termed as study of computation
- 2000's it is termed as study of natural and artificial information processes

As cited above the understanding of 'Computation' deepens itself as the time progressed. However it is worth mentioning here computation models that were not present when Turing Machine was conceived have been successfully simulated on it, thus high lighting the fact that the Turing Machine is capable of simulating every known computational model and the association of Computation with Turning machine still stands firm.

B. Computability Theory and Simulation Works

Automata theory and Computability theory forms the foundation of computation and hence are instrumental in developing a deep understanding of Turing Machines and Automata. Based on these two theories, over the last 50 years considerable work has been done in writing Simulators for Turing Machines and Automata. These works can be categorized into two types [5]

- Text based Simulators simulating on the basis of Notational Language
- Visual based Simulators simulating on the basis structured input

1) Text Based Notational Language Simulators: Some of the notational language based simulation works are: Coffin [5] has developed a Notational Language based simulator which can simulate Turing machines. An ordered sequence of quintuples is used to manifest a specific Turing Machine. An infinite and multiple track tapes simulator was developed by Head [5]. It works in a specific notational language format defined by fixed templates. Harris [5] has worked on developing multiple simulators that are used for executing finite automata, pushdown automata and Turing machines. These also work on notational languages where each instruction is represented as a tuple. University of Northern Colorado uses Scott's [6] Turing Machine notational language based simulator for educational and research purposes.

2) Visual Simulators: Some of the works belonging to Visual simulation accepting structured input are: Hannay [5] developed a visual centric simulator which is capable of simulating finite automata, pushdown automata and Turing machines. Vieira's [5] visual centric simulator called Language Emulator is capable of simulating automata, Mealy and Moore machines.

3) Latest Works: The latest works done by Bhattacharyya, M. [7] are also important in this regards. His notational language based simulator formulates a Turing machine with a read/write tape. The symbols are written and read from the tape. The machine's description is loaded into it via a structured text file comprising of the instruction set. The head is capable of moving in both left and right directions.

It is evident from the sources highlighted above that Turing Machine simulation has been an active area of research since last 50 years, which continues to draw in attention.

C. Complexity Theory and Turing Machines

After mentioning the works on Computability Theory we move towards the Complexity Theory. It is observed that Turing Machines can be analyzed from the aspect of analyzing the resources required in terms of time and space to compute a specific problem.

1) Time Complexity of a Turing Machine: The Time Complexity of a Turing Machine can be defined as a function where f(n) is the maximum number of steps that the Turing Machine 'T' uses on an input of size 'n'. If f(n) is the running time of T, we say that T is an f(n) Turing Machine [8].

2) Space Complexity of a Turing Machine: The Space Complexity of a Turing Machine can be defined as a function where f(n) is the maximum number of tape cells that he Turing Machine T scans on any input of length n. If the space complexity of T is f(n) we say that T runs in space f(n) [8].

As said in the abstract the research focuses on developing a Simulator that will comply with Computability Theory by simulating Turing Machines based on their manifestations and with Complexity Theory by analyzing Time and Space complexity of the algorithms executed.

The literature review clearly highlights the fact that the Turing Machine model has served as the fundamental and primary reference for understanding and innovating the computation concept as it has proved capable of simulating every known computational model. It also serves as a superb abstraction for the computational devices as number of steps and space required to execute an algorithm on them can be ascertained.

IV. RESEARCH METHODOLOGY

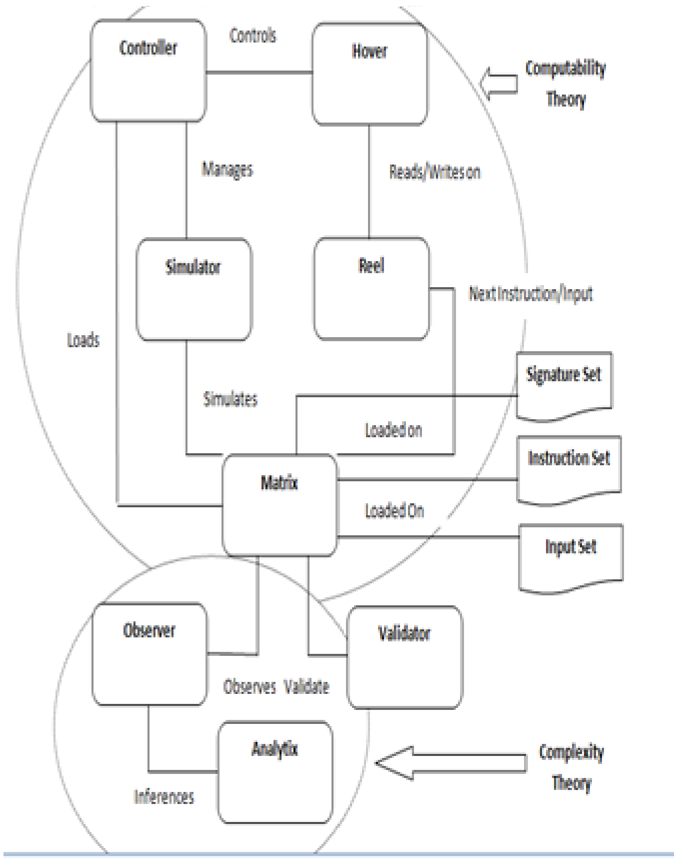
The research methodology adopted for this research is 'Experimental Research'. The Experimental Setup designed for this purpose is the Reflector Engine, designed and developed in java (discussed in the next section). The Independent Variable used is the Turing Machine Manifestations and the Input String loaded in the 'Matrix' Module while the Dependent variable is the Time & Space Complexity values generated via the Observer & Analytix module.

To enhance experimental validity, in each experiment the Input Strings belonging to the language recognized by the specific Turing Machine are manipulated and results are observed

V. EXPERIMENTAL SETUP AND DESIGN

Below is the Reflector – Architecture designed for conducting the research experiments. As is evident from the diagram the Simulator is logically grouped into 2 modules; one focusing on the Computability theory and the other focusing on the Complexity theory

A. Reflector Engine- Architectural Model



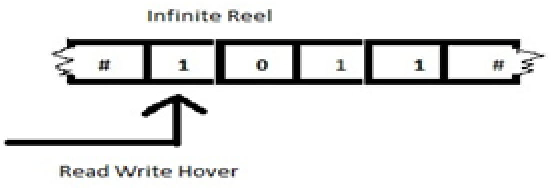
B. Reflector Engine- Module Wise Description

Reflector is designed and developed in Java using Eclipse IDE. The module wise description is as follows:

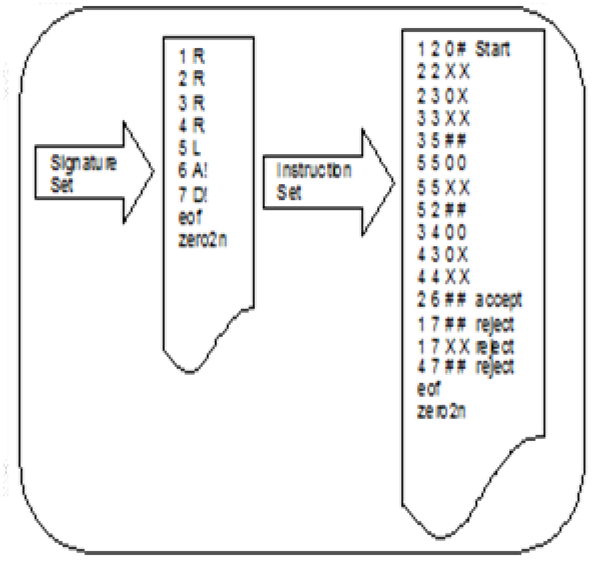
1) Controller: The main module that integrates and coordinates with other modules.

2) Reel: The Reel is the "tape" part of the Turing Machine and is used as reading/writing input/output. To facilitate the read/write operations the Reel is implemented as two separate Heaps; one represents the left and the other represents the right side of the Reel. The 'Input Set' is written to the Reel.

3) Hover: The Hover works as a 'Head'. It loads the input value to be processed into the 'Reel'. The Hover moves one by one over each literal; its movement is controlled by the left hover and right hover functions.



*4) Signature Set: S*ignature represents the name and description of each state of the machine. Together all the states of a machine are called its 'Signature Set'; responsible for differentiating a Turing Machine from another.



5) Instruction Set: The set of instructions a machine executes in order to perform its destined task is called Instruction Set. Similar to the Signature Set, the Instruction Set of a machine is different from other.

6) Matrix: Matrix forms the Control Plane of the Reflector. It is to the Matrix that the relevant 'Signature Set' and 'Instruction Set' is loaded via the 'Matrix Loaded' process. Matrix forms the brain of the Reflector.

7) Simulator: The Simulator together with the Controller is responsible for 'Creating a Turing Machine' based on its Signature Set. It is also responsible for 'Machine Traversal' from one State to another based on the Instruction selected from the 'Instruction Set' and the 'Input Set'.

8) Observer: Observer is responsible for performing Time Complexity Analysis by tracking the number of steps executed in validating the 'Input Set. It is also responsible for Space Complexity Analysis by measuring the number of Reel Cells utilized in determining the output.

9) Analytix: The Analytix is responsible for analyzing and producing the results as generated by the Observer.

B. Reflector Engine- Component mapping to Turing Machine's 7 tuples:

No	Turing Machine's Tuples	Reflector's Modules
1	Q is the set of internal states	Signature Set
2	Σ is the input alphabet	Input Set
3	Γ is the tape alphabet	Alphabets from Input Set and Instruction Set
4	δ is the transition function (δ : Q * Γ)	Instruction Set
5	q_0 ϵ Q, is start state	Row in Instruction Set labeled 'Start'
6	qaccept ϵ Q, is accept state	Row in Instruction Set labeled 'Accept', denoted by A!
7	qreject ϵ Q, is reject state	Row in Instruction Set labeled 'Reject', denoted by D!

VI. EXPERIMENTS AND RESULTS

Multiple experiments were conducted on the Reflector Engine. Below is the list of some Turing Machines simulated on the Reflector Engine:

1) Palindrome Machine
2) 0^{2^n} - Decider Machine
3) $a^n b^n$ - Decider Machine
4) Even parity Recognizer
5) Odd parity Recognizer
6) Unary to Binary Converter
7) Binary Increment Generator
8)

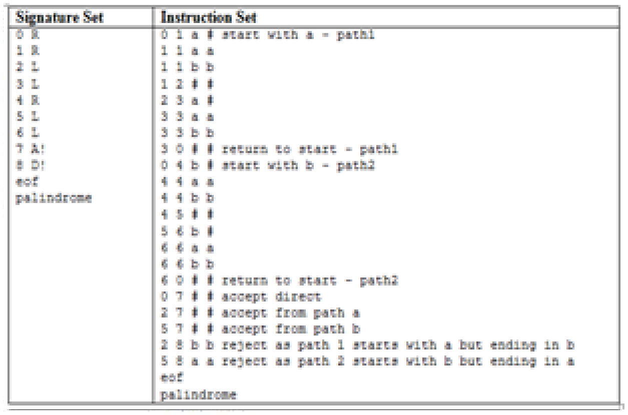
Each experiment conducted comprised of two phases:

1) Simulating a specific Turing Machine as per its Signature Set and Instruction Set.
2) Analyzing Algorithmic Complexity in terms of Time & Space Complexity with multiple input values.

To ensure experimental validity diversified Turing Machines were used and diversified input values on each machine were used. Two experiments of the relevant machines listed are discussed below.

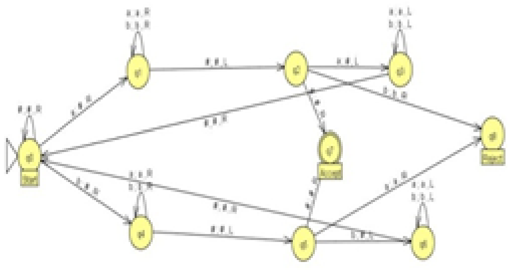
A. Palindrome Machine

1) Machine Description: This machine recognizes strings belonging to the palindrome language. These strings when reversed do not change; valid strings include 'abba', 'bab'. The machine reads and strikes the first letter, then goes to the end, if same letter is found it strikes it but if not the machine 'Rejects'. It continues until similar letters at both ends are found and finally 'Accepts'. The Signature Set and Instruction Set of the machine is shown next.



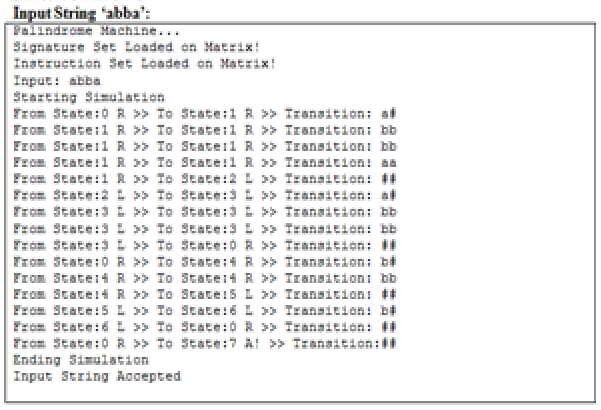
Signature Set	Instruction Set
0 R	0 1 a # start with a - path1
1 R	1 1 a a
2 L	1 1 b b
3 L	1 2 # #
4 R	2 3 a #
5 L	3 3 a a
6 L	3 3 b b
7 A!	3 0 # # return to start - path1
8 D!	0 4 b # start with b - path2
eof	4 4 a a
palindrome	4 4 b b
	4 5 # #
	5 6 b #
	6 6 a a
	6 6 b b
	6 0 # # return to start - path2
	0 7 # # accept direct
	2 7 # # accept from path a
	5 7 # # accept from path b
	2 8 b b reject as path 1 starts with a but ending in b
	5 8 a a reject as path 2 starts with b but ending in a
	eof
	palindrome

3) Machine Design:



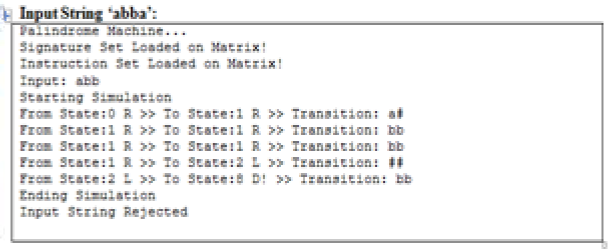
3) Execution: Below is the log from Reflector.

Input String 'abba':



```
Palindrome Machine...
Signature Set Loaded on Matrix!
Instruction Set Loaded on Matrix!
Input: abba
Starting Simulation
From State:0 R >> To State:1 R >> Transition: a#
From State:1 R >> To State:1 R >> Transition: bb
From State:1 R >> To State:1 R >> Transition: bb
From State:1 R >> To State:1 R >> Transition: aa
From State:1 R >> To State:2 L >> Transition: ##
From State:2 L >> To State:3 L >> Transition: a#
From State:3 L >> To State:3 L >> Transition: bb
From State:3 L >> To State:3 L >> Transition: bb
From State:3 L >> To State:0 R >> Transition: ##
From State:0 R >> To State:4 R >> Transition: b#
From State:4 R >> To State:4 R >> Transition: bb
From State:4 R >> To State:5 L >> Transition: ##
From State:5 L >> To State:6 L >> Transition: b#
From State:6 L >> To State:0 R >> Transition: ##
From State:0 R >> To State:7 A! >> Transition:##
Ending Simulation
Input String Accepted
```

Input String 'abba':



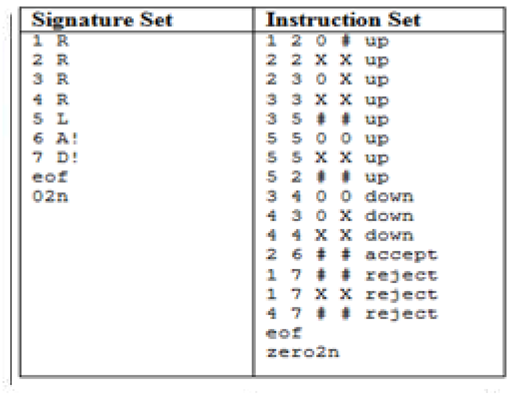
```
Palindrome Machine...
Signature Set Loaded on Matrix!
Instruction Set Loaded on Matrix!
Input: abb
Starting Simulation
From State:0 R >> To State:1 R >> Transition: a#
From State:1 R >> To State:1 R >> Transition: bb
From State:1 R >> To State:1 R >> Transition: bb
From State:1 R >> To State:2 L >> Transition: ##
From State:2 L >> To State:8 D! >> Transition: bb
Ending Simulation
Input String Rejected
```

4) *Results*: The results generated by the Observer and Analytix Module of the Reflector Engine are:

	Machine		Algorithmic Complexity		Stack Operations		Hover Movement	
	Input String	Machine Decision	Time Complexity (Steps)	Space Complexity (Reel size)	Push	Pop	Hover Left	Hover Right
1	abba	Accepted	16	5	18	15	6	8
2	abb	Rejected	6	4	7	5	1	13
3	abbaabba	Accepted	46	9	52	45	20	24
4	abbaabb	Rejected	10	8	15	9	1	7

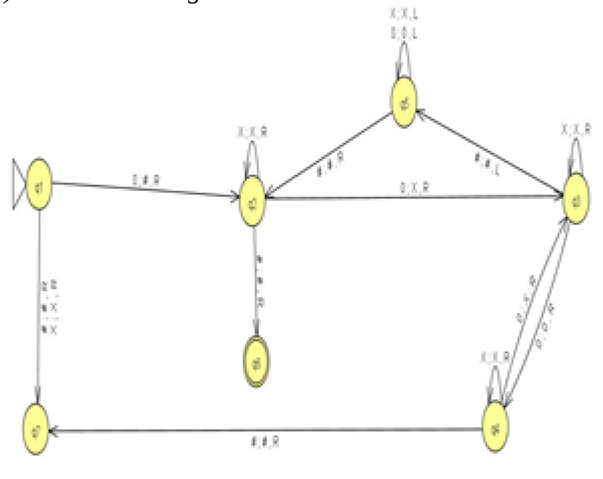
B. 0^{2^n} - Decider Machine

1) Machine Description: The machine works as a decider deciding the strings belonging to the language 0^{2^n}; valid strings include ‘’00’,’0000’. This machine has distinction of exhibiting ‘Computer memory’ concept. In each iteration it strikes of the number of 0’s in half. During the iteration as it moves to the end it keeps track that whether the number of zeroes it has seen is ‘even’ or ‘odd’. If ‘odd’ and greater than 1, it crashes as the original number of ‘zeroes’ could not be a power of 2.

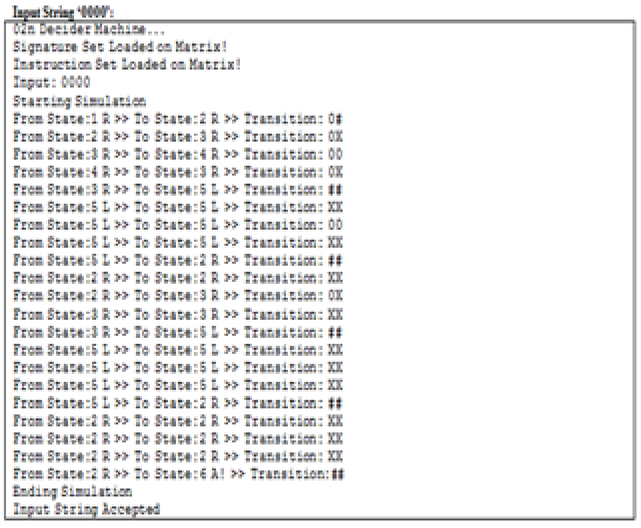


Signature Set	Instruction Set
1 R	1 2 0 # up
2 R	2 2 X X up
3 R	2 3 0 X up
4 R	3 3 X X up
5 L	3 5 # # up
6 A!	5 5 0 0 up
7 D!	5 5 X X up
eof	5 2 # # up
02n	3 4 0 0 down
	4 3 0 X down
	4 4 X X down
	2 6 # # accept
	1 7 # # reject
	1 7 X X reject
	4 7 # # reject
	eof
	zero2n

1) *Machine Design:*

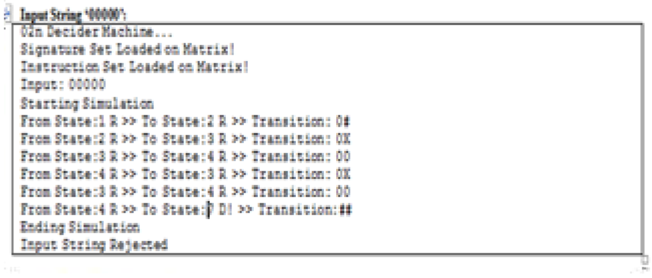


5) *Execution:* Below is the log from Reflector.



Input String ‘0000’:

```
02n Decider Machine...
Signature Set Loaded on Matrix!
Instruction Set Loaded on Matrix!
Input: 0000
Starting Simulation
From State:1 R >> To State:2 R >> Transition: 0#
From State:2 R >> To State:3 R >> Transition: 0X
From State:3 R >> To State:4 R >> Transition: 00
From State:4 R >> To State:3 R >> Transition: 0X
From State:3 R >> To State:5 L >> Transition: ##
From State:5 L >> To State:5 L >> Transition: XX
From State:5 L >> To State:5 L >> Transition: 00
From State:5 L >> To State:5 L >> Transition: XX
From State:5 L >> To State:2 R >> Transition: ##
From State:2 R >> To State:2 R >> Transition: XX
From State:2 R >> To State:3 R >> Transition: 0X
From State:3 R >> To State:3 R >> Transition: XX
From State:3 R >> To State:5 L >> Transition: ##
From State:5 L >> To State:5 L >> Transition: XX
From State:5 L >> To State:5 L >> Transition: XX
From State:5 L >> To State:5 L >> Transition: XX
From State:5 L >> To State:2 R >> Transition: ##
From State:2 R >> To State:2 R >> Transition: XX
From State:2 R >> To State:2 R >> Transition: XX
From State:2 R >> To State:2 R >> Transition: XX
From State:2 R >> To State:6 A! >> Transition:##
Ending Simulation
Input String Accepted
```



Input String ‘00000’:

```
02n Decider Machine...
Signature Set Loaded on Matrix!
Instruction Set Loaded on Matrix!
Input: 00000
Starting Simulation
From State:1 R >> To State:2 R >> Transition: 0#
From State:2 R >> To State:3 R >> Transition: 0X
From State:3 R >> To State:4 R >> Transition: 00
From State:4 R >> To State:3 R >> Transition: 0X
From State:3 R >> To State:4 R >> Transition: 00
From State:4 R >> To State:7 D! >> Transition:##
Ending Simulation
Input String Rejected
```

6) *Results:*

	Machine		Algorithmic Complexity		Stack Operations		Hover Movement	
	Input String	Machine Decision	Time Complexity (Steps)	Space Complexity (Reel size)	Push	Pop	Hover Left	Hover Right
1	0000	Accepted	22	5	24	21	8	12
2	00000	Rejected	7	6	10	6	0	5
3	00000000 00000000	Accepted	146	17	160	145	64	80
4	00000000 00000000 0000	Rejected	102	21	120	101	40	60

VII. CONCLUSION

The research focused on designing a simulator called Reflector. It is the foremost simulator to have both the capabilities; first dynamically modeling a Turing Machine based on its manifestations (Signature Set & Instruction Set) and secondly performing Time & Space Complexity Analysis of an algorithm executed on the Turing Machine. The research emphasized on the fact that Turing Machine serves as a precise model for defining and executing a specific algorithm based on the Computability Theory & analyzing its time and space complexity based on the Complexity Theory. It can thus be concluded that this model has served as the fundamental reference for understanding and innovating the ‘computation’ concept. The Turing Machine model remains an excellent abstraction for the computational devices as the number of steps and space required to execute an algorithm can be predicted.

References

[1] Fortnow, L.. What is Computation. Ubiquity, An ACM publication, 2010

[2] Denning, P. J.. What is Computation. Ubiquity, An ACM Publication, 2010

[3] Turing, A. On computable numbers, with an application to the Etscheidungs problem. Proceedings of the London Mathematical Society. , 42:230–265, 1936

[4] Denning, P. J. Computing Field: Structure . In Wiley Encyclopedia of Computer Science and Engineering (B. Wah, Ed.). Wiley Interscience, 2008

[5] Pinaki Chakraborty, P. S. Fifty Years of Automata Simulation: A Review. ACM INROADS Volume 2, 2011.

[6] Scott, T. A.Turing machine simulation used in a breadth first computer science course. Journal of Computing in Small Colleges, 22(1): , 240-245, 2006

[7] Bhattacharyya, M. Simulating a Turing Machine. ACM XRDS VOL.18 NO.3 SPRING 2012 .

[8] Sipser, M.Introduction To Theory Of Computation by Micheal Sipser MIT. Boston, Massachusetts: Thomson Course Technology, 2004