FORMAL LANGUAGE AND AUTOMATA THEORY

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Candidate Name

AP21110011360

Sai Chakradhar Rao Mahendrakar

Under the Guidance of

Dr. Arnab Mitra

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

May 2023

Certificate

Date: 15-May-23

This is to certify that the work present in this Project entitled **FORMAL LANGUAGE AND AUTOMATA THEORY** has been carried out by **Sai Chakradhar Rao Mahendrakar** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

Dr. Arnab Mitra

Designation,

Affiliation.

Acknowledgements

We are grateful because we managed to complete our Formal Language And Automata Theory [FLAT] project within the time given by our lecturer **Dr. Arnab Mitra**. This project could not have been completed without the effort and cooperation of our group members.

We also sincerely thank our lecturer for the guidance and encouragement in finishing this project and for teaching us in this course

This project involves the implementation of a Turing machine to calculate the logarithm of a number in base 2. Turing machines, developed by Alan Turing, have played a pivotal role in the foundation of computation theory and the study of algorithmic processes. Their ability to simulate any algorithmic computation makes them a fundamental concept in understanding computability and complexity. By leveraging the power of Turing machines, we explore the intricacies of logarithmic calculations in base 2, highlighting the versatility and potential of Turing machine models in solving complex computational problems.

I am grateful to everyone who has contributed to the completion of this project on Turing machine complexity analysis using logarithmic base 2 and its relation to primitive recursive functions. Firstly, I would like to express my sincere appreciation to my project advisor, **Dr. Arnab Mitra** , for their invaluable guidance, expertise, and continuous support throughout this research endeavor. Their insightful feedback and constructive suggestions have significantly shaped the direction and scope of this project.

I would also like to acknowledge the faculty members of the **CSE SRMAP** for creating an environment conducive to academic exploration and learning. Their provision of resources and infrastructure has greatly facilitated my research efforts. Additionally, I extend my gratitude to my peers and colleagues for their valuable insights and engaging discussions, which have enriched the project's content.

Lastly, I want to thank my family and friends for their unwavering support, encouragement, and understanding. Their belief in my abilities and constant motivation have been integral to the successful completion of this project. I am truly indebted to the collective contributions of all those involved, and I extend my heartfelt appreciation to each and every one of them.

Table of Contents

Abstract

The Turing machine is a powerful computational model that forms the foundation of theoretical computer science. It allows us to explore the limits of computability and analyze the efficiency of algorithms. In this project, we investigate the complexity analysis of a Turing machine utilizing the logarithmic base 2 and its relationship to primitive recursive functions.

We begin by introducing the concept of Turing machines and their fundamental operations. Next, we delve into the properties and characteristics of logarithmic base 2. By expressing the input size in terms of its binary representation, we can analyze the logarithmic growth of the machine's computational steps. We explore various algorithms and problems, examining their complexities in terms of log n base 2. This analysis sheds light on the efficiency and scalability of these algorithms, providing insights into their computational properties.

Additionally, we explore the connection between logarithmic base 2 and primitive recursive functions. Primitive recursive functions represent a class of computable functions that can be defined using a limited set of basic operations. By exploring the complexity of Turing machines using logarithmic base 2, we can gain a deeper understanding of the expressive power and limitations of primitive recursive functions. This analysis contributes to the broader understanding of computability and complexity theory.

Through practical examples and applications, we demonstrate how the utilization of logarithmic base 2 and its relationship to primitive recursive functions can lead to more efficient algorithm design. By identifying patterns and structures within problems, we can leverage these insights to optimize computations in various domains. This project provides a comprehensive exploration of the complexity analysis of a Turing machine using logarithmic base 2, and its connection to primitive recursive functions, with practical implications for algorithm design and optimization.

List of Publications

The thesis is mainly based on the results presented in the following articles.

1. [Paul M. B. Vitányi] Turing machine. January 2009,
   DOI:10.4249/scholarpedia.6240
   https://www.researchgate.net/publication/220579932_Turing_machine

2. [Raphael M. Robinson] Primitive recursive functions, October 1947
   https://projecteuclid.org/journals/bulletin-of-the-american-mathematical-society/volume-53/issue-10/Primitive-recursive-functions/bams/1183511140.full?tab=ArticleLink

Statement of Contributions

Sai Chakradhar : Algorithm Design

Research and understand the algorithm for computing logarithms in base 2.

Determine the input and output formats for the Turing machine.

Design the overall structure of the Turing machine based on the algorithm.

Specify the necessary states and transitions to implement the logarithm computation.

Vivekananda: Tape and State Management

Implement the tape data structure and operations.

Design methods for reading and writing symbols on the tape, considering the input and output formats.

Develop the state management system, including defining the states and their meanings.

Create mechanisms for moving the tape head and updating the current state accordingly.

Venkata Ramesh : Transition Function and Testing

Formulate the transition function for the Turing machine based on the algorithm.

Specify the transition rules that determine the next state and tape symbol based on the current state and symbol read.

Test the Turing machine by running different inputs and verifying that the logarithm in base 2 is computed correctly.

Debug and refine the machine's operation based on the testing results.

Throughout the process, the team members should communicate and collaborate to ensure the algorithm is correctly translated into the Turing machine design. Regular meetings, code reviews, and discussions will help address any challenges and ensure a successful implementation of the logarithm computation Turing machine.

List of Tables

Introduction

Turing Machine

The objective of this project is to design a Turing machine that can compute the logarithm of a given input number in base 2 , where n represents the input number. The logarithm function is a fundamental mathematical operation widely used in various fields, such as computer science, mathematics, and engineering. By developing an efficient Turing machine for logarithm computation, we aim to explore the theoretical foundations of computation and analyze the complexity of this specific problem.

In automata theory and the theory of computation, Turing machines are utilized to explore algorithm properties and determine the solvability of problems by computers. They provide a means to represent and analyze the behavior of algorithms, including an examination of their computational complexity. Computational complexity refers to the resources, such as time and memory, required by algorithms to solve problems efficiently.

By studying Turing machines and their characteristics, researchers can delve into the limits of computation. They help define and analyze algorithms, serving as a universal model of computation capable of simulating any algorithm. Turing machines aid in classifying problems based on their computational complexity, such as determining whether a problem is decidable, tractable, or undecidable

Recursive Functions

Recursive functions are a fundamental concept in computer science and mathematics that involve defining a function in terms of itself. A recursive function solves a problem by breaking it down into smaller, simpler subproblems and then combining the solutions of these subproblems to obtain the solution to the original problem.

In a recursive function, the function definition includes a base case that provides a termination condition for the recursion. When the base case is reached, the function does not call itself and instead returns a specific value or performs a specific action. Additionally, the recursive function includes one or more recursive cases, where the function calls itself with a modified input, moving closer to the base case.

It is important to note that recursive functions require careful design to ensure termination and avoid infinite recursion. Each recursive call should progress towards the base case, and the problem size should decrease with each recursion to guarantee that the recursion eventually stops.

Methodology

Turing Machine:

To investigate the complexity analysis of a Turing machine using logarithmic base 2 and its relation to primitive recursive functions, the following methodology can be employed:

1. Research and Literature Review: Conduct an extensive review of existing literature on Turing machines, logarithmic base 2, and primitive recursive functions. Gain a comprehensive understanding of the underlying concepts, theories, and previous research in the field.

2. Formulation of Problem Statements: Clearly define the specific algorithms, problems, and complexity classes that will be analyzed in the project. This step involves identifying suitable examples that demonstrate the relationship between logarithmic base 2 and primitive recursive functions.

3. Implementation of Turing Machine: Develop a Turing machine implementation that operates with logarithmic base 2. This implementation should be capable of executing the selected algorithms and solving the defined problems. Consider the appropriate programming language or simulation environment for the implementation.

4. Data Collection and Analysis: Run the implemented Turing machine on various input sizes and collect data on the computational steps required. Analyze the collected data to determine the growth patterns and complexities in terms of log n base 2. This analysis will provide insights into the efficiency and scalability of the algorithms under study.

Primitive Recursive Functions:

5. Comparison with Primitive Recursive Functions: Analyze the relationship between the complexity analysis using logarithmic base 2 and the properties of primitive recursive functions. Compare the growth rates, limitations, and expressive power of primitive recursive functions with the findings from the Turing machine analysis.

6. Result Interpretation and Discussion: Interpret the results obtained from the complexity analysis and the comparison with primitive recursive functions. Discuss the implications, limitations, and potential applications of the findings. Identify patterns, trends, and possible extensions for further research.

7. Validation and Evaluation: Validate the implemented Turing machine by comparing its results with established theoretical findings and existing literature. Evaluate the accuracy and reliability of the methodology used in the project.

8. Documentation and Reporting: Prepare a comprehensive report documenting the methodology, findings, analysis, and discussions. Clearly present the results and provide a coherent explanation of the entire research process.

By following this methodology, researchers can effectively analyze the complexity of Turing machines using logarithmic base 2 and explore their relation to primitive recursive functions. The systematic approach ensures rigor, accuracy, and reproducibility in the research process.

Discussion

Turing Machine : For $\lceil \log_2 n \rceil$.

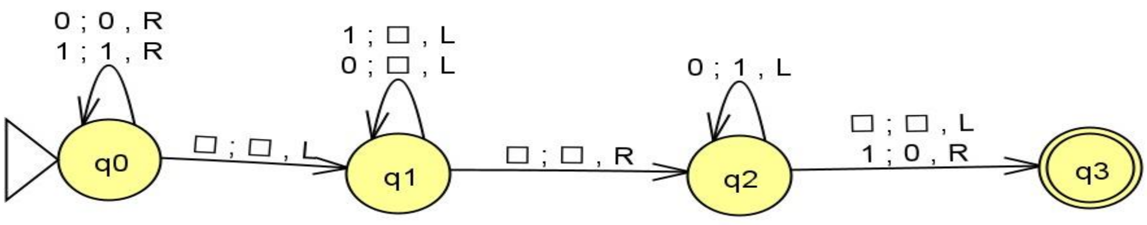
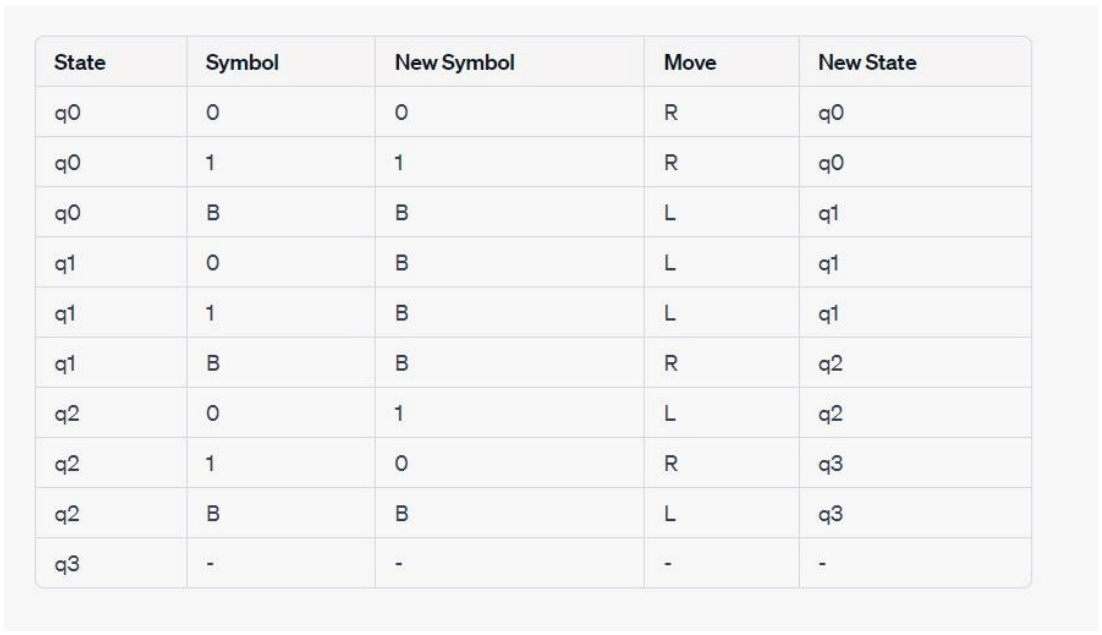


Table 1. [Transition Table]



State	Symbol	New Symbol	Move	New State
q0	0	0	R	q0
q0	1	1	R	q0
q0	B	B	L	q1
q1	0	B	L	q1
q1	1	B	L	q1
q1	B	B	R	q2
q2	0	1	L	q2
q2	1	0	R	q3
q2	B	B	L	q3
q3	-	-	-	-

Every primitive recursive function is a total recursive function

Base Functions:

Primitive recursive functions start with base functions, such as the successor function (S(x) = x + 1) and the zero function (Z(x) = 0). Base functions are simple functions that are defined for all inputs and always terminate.

For example, the successor function takes any number x and adds 1 to it, which is a well-defined operation that always terminates.

Composition:

Composition allows us to combine two primitive recursive functions to create a new one. If we have two primitive recursive functions, f(x) and g(x), their composition h(x) = f(g(x)) is also a primitive recursive function. When we apply composition, the output of the inner function g(x) serves as the input for the outer function f(x). Both f(x) and g(x) are already defined for all inputs and always terminate, so their composition h(x) will also be defined for all inputs and always terminate.

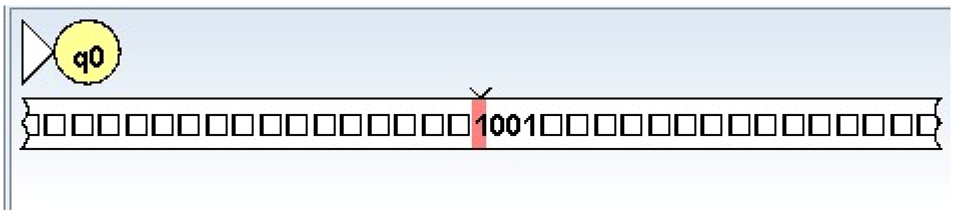
Primitive Recursion:

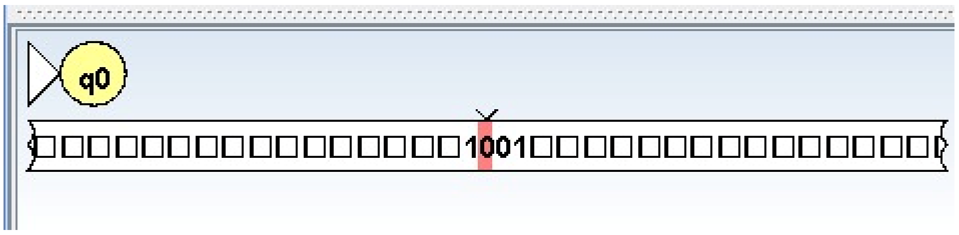
Primitive recursion allows us to define a function by providing a base case and a recursive step. If we have a primitive recursive function f(x, y) and another primitive recursive function g(x), their primitive recursion h(x, y) is also a primitive recursive function. The base case function f(x, y) is defined for all inputs (including the recursive step) and always terminates. The recursive function g(x) is also defined for all inputs and always terminates. When we apply primitive recursion, the resulting function h(x, y) combines the base case and the recursive step. It is defined for all inputs and always terminates due to the properties of both f(x, y) and g(x). By using base functions, composition, and primitive recursion, we can construct any primitive recursive function. Since each component of the construction is defined for all inputs and always terminates, the resulting primitive recursive function will also be defined for all inputs and always terminate.

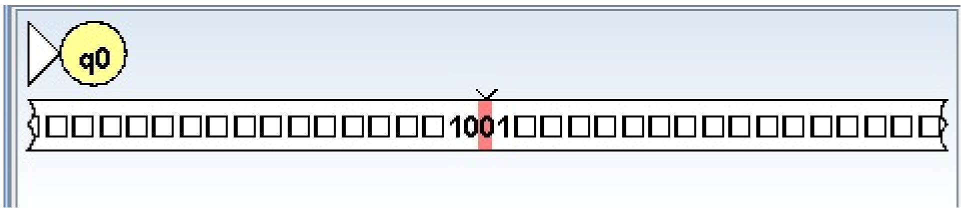
In summary, every primitive recursive function is a total recursive function because it is constructed using base functions, composition, and primitive recursion, which ensure that the functions are defined for all inputs and always terminate.

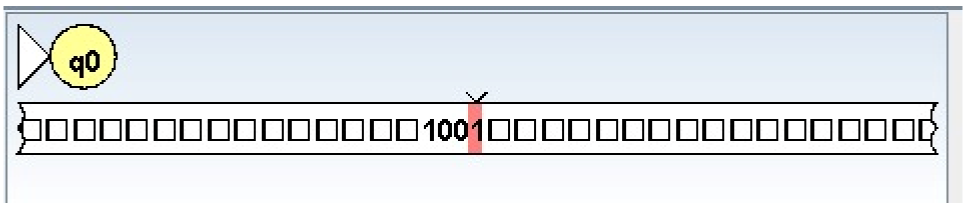
Working of project:

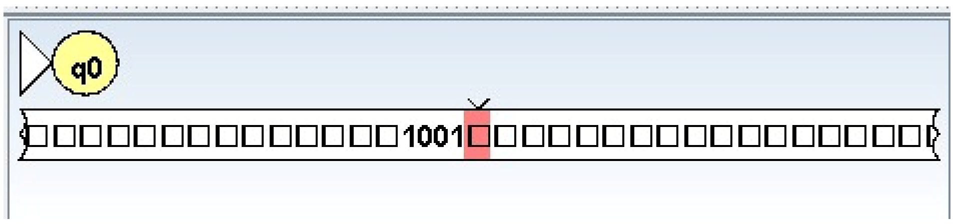
Example Input 1001 – 9

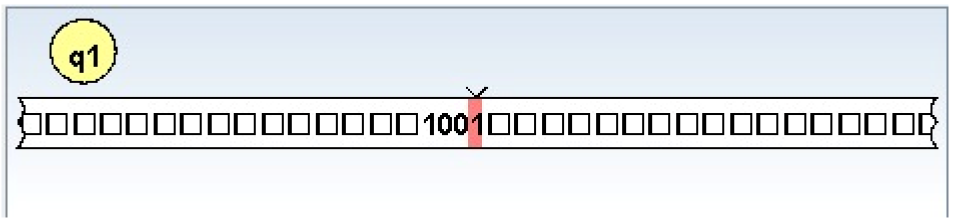


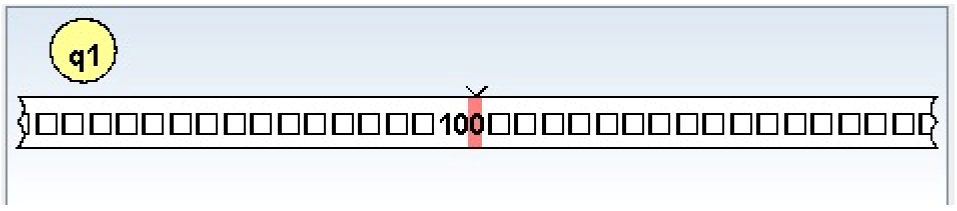


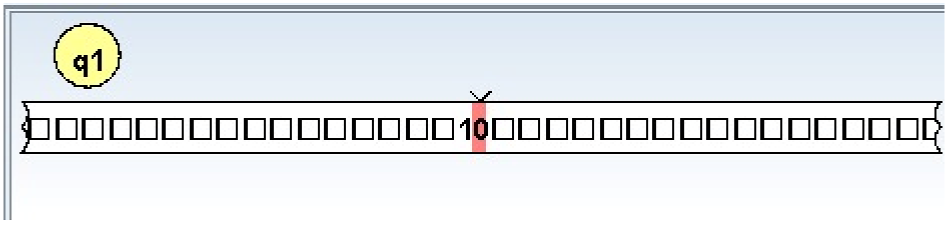


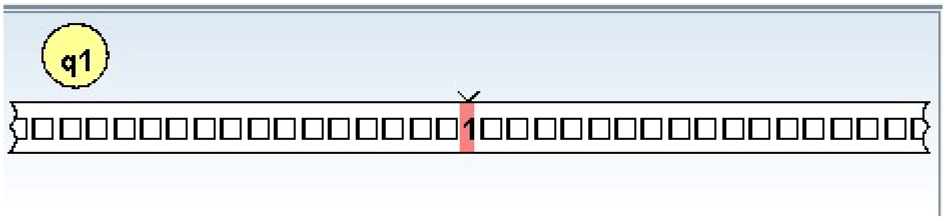


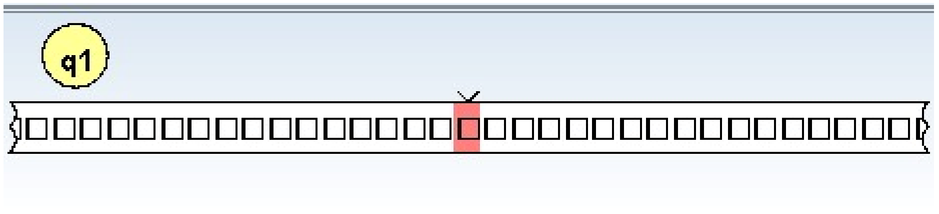


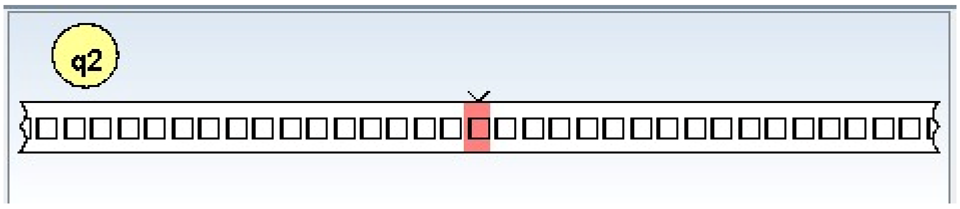


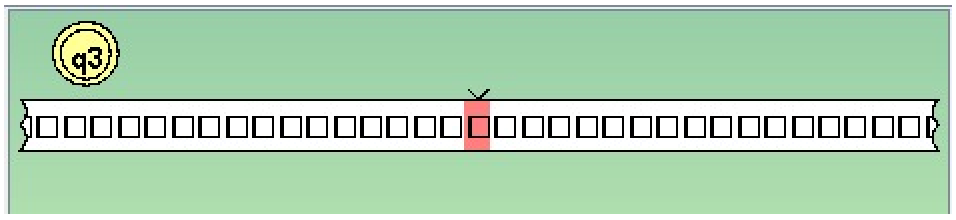



q1
10


q1
1


q1


q2


q3

Primitive Recursive Function:

Let's consider an example of a primitive recursive function and demonstrate that it is a total recursive function.

Example: Factorial Function (n!)

The factorial function is a classic example of a primitive recursive function. Let's denote it as F(n).

Base case:

F(0) = 1, where 0! is defined as 1.

The base case is a simple constant value, which is defined for input 0 and terminates immediately.

Recursive step:

F(n) = n * F(n-1), for n > 0.

In the recursive step, we multiply the input n with the factorial of (n-1).

For example, F(3) = 3 * F(2) = 3 * 2 * F(1) = 3 * 2 * 1 * F(0) = 3 * 2 * 1 * 1 = 6.

The recursive step is well-defined for all inputs greater than 0, and it terminates as each recursive call reduces the input by 1 until it reaches the base case.

By using the base case and recursive step, we have defined the factorial function F(n) as a primitive recursive function. Now, let's demonstrate that it is a total recursive function:

Defined for all inputs:

The base case F(0) is defined.

For any input n > 0, the recursive step F(n) = n * F(n-1) is defined in terms of F(n-1). The recursive calls continue until reaching the base case F(0), which is defined.

Therefore, the factorial function is defined for all non-negative inputs.

Always terminates:

The base case F(0) terminates immediately.

In the recursive step, each recursive call reduces the input by 1. As the input decreases with each recursion, it eventually reaches the base case F(0), terminating the recursion.

Hence, the factorial function always terminates for all non-negative inputs.

Therefore, we have shown that the factorial function, which is a primitive recursive function, is also a total recursive function as it is defined for all inputs and always terminates.

Concluding Remarks

Turing machines have been instrumental in advancing our understanding of computation and algorithms. They have provided a theoretical framework for analyzing complexity, exploring the limits of computation, and designing efficient algorithms. Turing machines continue to shape the field of computer science and inspire further research and innovation in various areas of computing.

Turing machines have revolutionized our understanding of computation and have become the cornerstone of theoretical computer science. Their impact spans both theory and practice, influencing the way we analyze algorithms, design computing systems, and explore the boundaries of what can be computed. The legacy of Turing machines will continue to shape the future of computer science and computation.

Future Work

This project sets the stage for several promising avenues of future research. Firstly, further complexity analysis can be conducted to explore the efficiencies of specific algorithms and problems within the framework of logarithmic base 2. This deeper exploration would enhance our understanding of computational scalability.

Secondly, comparative studies with other complexity classes, such as polynomial or exponential time, can be pursued to establish the relative computational power of logarithmic base 2 analysis. Additionally, practical applications and implementations can be developed to harness the insights gained from this research. Lastly, theoretical extensions can explore the application of logarithmic base 2 and primitive recursive functions in alternative computational models. These future research directions have the potential to advance algorithm design, complexity theory, and computational efficiency in various domains.

References

1. https://cs.stackexchange.com/questions/132571/create-a-turing-machine-for-log-base-2-of-n
2. https://www.ques10.com/p/10531/design-a-turing-machine-to-find-the-value-of-log-1/
3. https://www.youtube.com/watch?v=XSA9EQn__xw&t=602s&ab_channel=JayaDewan
4. https://www.youtube.com/watch?v=9oCVYWhoxVM&ab_channel=Swapnilkadam