National University of Sciences & Technology

School of Electrical Engineering and Computer Science

Department of Computing

CS352 Theory of Automata and Formal Languages: BSCS13(CDE), Spring 2025

Assignment 2	
CLO2: Analyze the computing devices as finite state machines and Turing machines.	
Marks: 30	Instructor: Dr. Fatima Abdullah
Announcement Date: May 1, 2025	Due Date: May 6, 2025 (11:59 pm)

Part-1: Solve the following Questions by utilizing the knowledge of Turing Machine. (15 marks)

Made By:
Muhammad Haseeb Ul Haq **(454512)**
Muhammad Moiz **(464192)**

Question 1

Design a Turing Machine that accepts the language L = {$0^n1^n0^n$ | n ≥ 1}.

Provide a comprehensive explanation of how your Turing Machine operates.

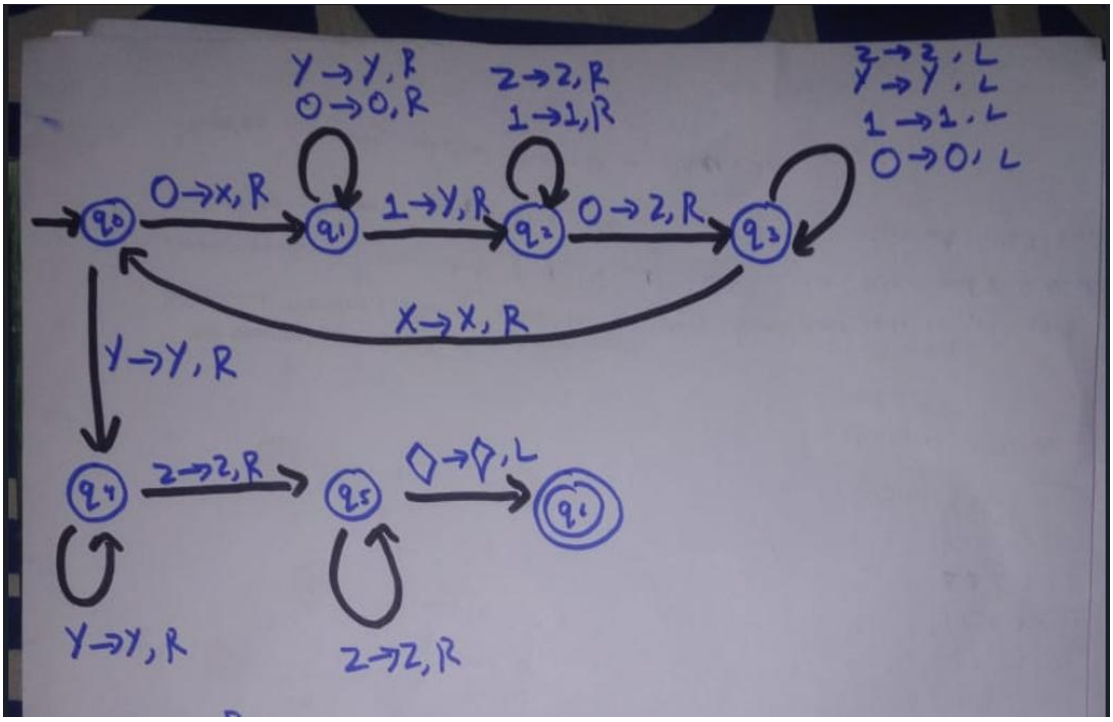
Solution:

The Turing Machine operates through the following methodology:

- It begins by identifying the leftmost 0 and converts it to an X
- It then locates the first 1 in the sequence and transforms it to a Y
- Next, it finds the subsequent 0 and changes it to a Z
- The machine then returns to the beginning of the modified string
- This cycle repeats for each 0 that appears before the first 1 in the original string
- When the machine encounters an X immediately followed by a Y (indicating all initial 0s have been processed), it proceeds through all consecutive Y symbols
- Upon reaching a Z, it continues moving through all Z symbols

- When a blank space is detected after all Z symbols, the string is accepted
- If at any point an unexpected symbol is encountered during transitions, the string is rejected

Design the transitions or logic needed to accept valid strings.



Analyze the time complexity of your Turing Machine in terms of input size.

Solution:

- Each traversal to replace symbols requires O(n) operations
- This traversal happens n times (once for each initial 0), resulting in $O(n^2)$ complexity
- The final verification traversal adds O(n) operations
- Therefore, the overall time complexity is $O(n^2)$

Is there a way to reduce the time complexity? Justify your reasoning, and if possible, suggest or describe an improved design.

Solution:

Yes, the time complexity can be reduced by implementing a multi-tape approach. Using three tapes:

- First tape stores all initial 0s

- Second tape holds all 1s that follow
- Third tape contains the final sequence of 0s

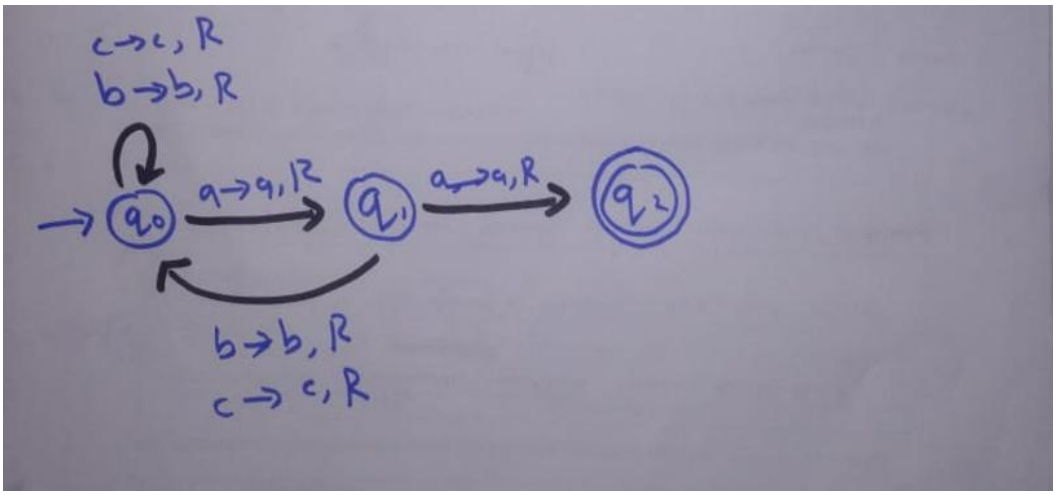
After this distribution, parallel comparison between tapes can be performed, reducing the time complexity to O(n). This is possible because:

- The initial configuration of the three-tape machine requires O(n) time to set up
- Once the input is distributed across the tapes, comparisons can be made simultaneously
- This eliminates the need for repeated back-and-forth traversals across the entire input
- The parallel processing capability of multiple tapes enables linear-time verification of the pattern $0^n1^n0^n$

Question 2

Create a Turing Machine that scans rightward until it identifies two consecutive 'a' characters and then halts. The machine's alphabet consists of {a, b, c}. Provide both a diagram and an explanation of your solution.

Diagram:



Solution:

The Turing Machine employs three states:

- State q_0: Indicates the previously read character was not 'a'
- State q_1: Indicates the previously read character was 'a'
- State q_2: Halt state, activated upon finding two consecutive 'a's

Operating mechanism:

- Upon reading an 'a' from any state except q_2, the machine advances to the next state
- If any other character is read (while not in q_2), the machine reverts to q_0
- The machine halts upon reaching state q_2, signifying successful detection of consecutive 'a's

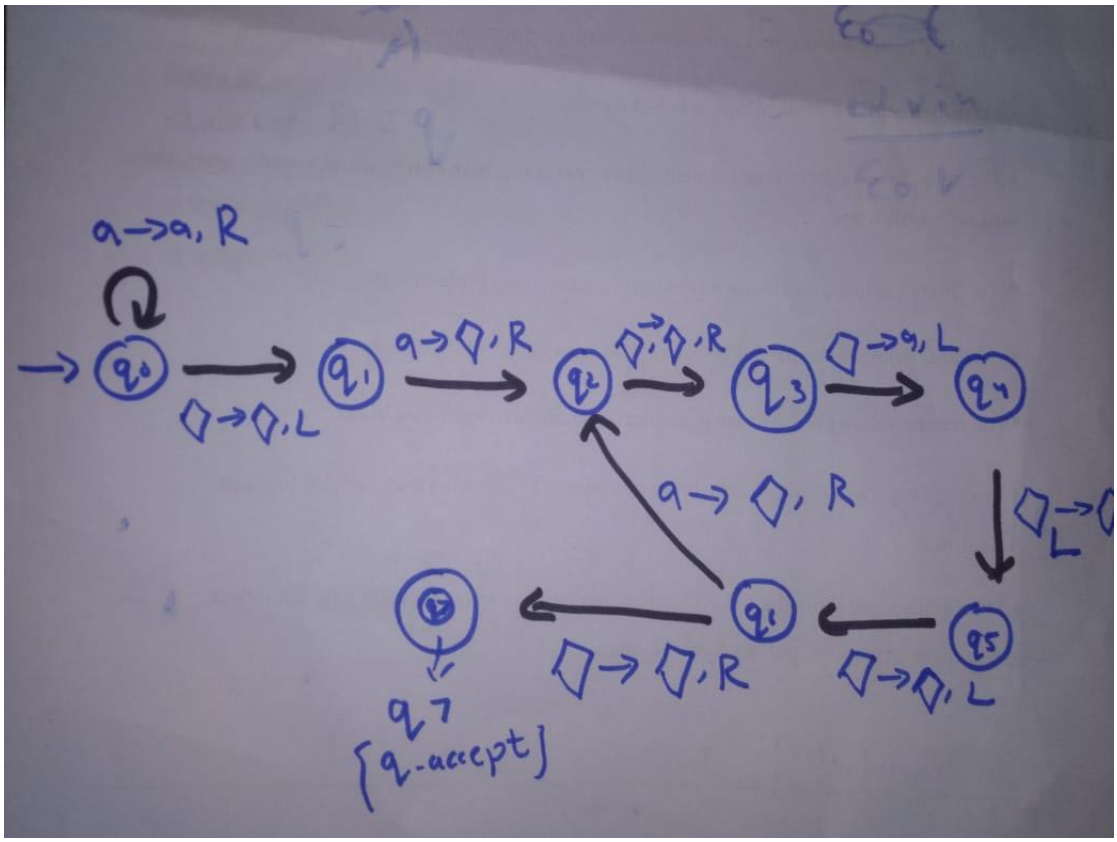
Question 3

Design a Turing Machine that shifts its entire input two positions to the right. Include both a diagram and an explanation of your solution.

Input specifications:

- **Alphabet: {a}**
- **Example input: aaa**
- **Initial configuration: ❑aaa❑**
- **Final output: ❑❑❑aaa❑**

Diagram:



Solution:

The Turing Machine functions as follows:

- Initially, it moves through all 'a' characters until reaching the blank space at the string's end
- It then returns to the rightmost 'a', replaces it with a blank
- The machine then places an 'a' at a position two spaces to the right
- It retreats to find the new rightmost 'a' and repeats this process
- This continues until all characters have been shifted two positions rightward

Part 2: Theoretical Questions

Question 1

Explain the importance of the Church-Turing thesis. Is it possible for all computable functions to be computed by a Turing Machine?

Answer:

The Church-Turing thesis represents a fundamental principle stating that any algorithmically computable function can be calculated by a Turing Machine. Its significance includes:

- Establishing computational boundaries: Functions that cannot be computed by a Turing Machine are generally classified as non-computable
- Creating unity among computational models: It demonstrates that various models (lambda calculus, recursive functions, Turing Machines) possess equivalent computational capabilities
- Providing foundational support for theoretical computer science disciplines, particularly complexity theory and computability studies

According to this thesis, all computable functions are, by definition, capable of being computed by a Turing Machine.

Question 2

Define what makes a language Turing recognizable. Provide an example of a language that is Turing recognizable but not decidable.

Answer:

A language L is considered Turing recognizable (also termed recursively enumerable) if there exists a Turing Machine M where:

- For any string w that belongs to L, M accepts w (halts and accepts)
- For any string w that does not belong to L, M either rejects w or continues running indefinitely

This means we can identify strings within the language, but may not definitively determine when a string is excluded from it.

Example of a Turing recognizable but undecidable language: L = {⟨M,w⟩ | Turing Machine M accepts input w}

This represents the Acceptance Problem (A_TM), which can be recognized (by simulating M on w and accepting if it accepts) but cannot be decided (due to the halting problem). The recognizer accepts strings in L but may never halt for strings not in L.

Question 3

Would a Turing Machine with an infinite number of tapes possess greater computational power than a standard single-tape Turing Machine? What impact would this have on time complexity?

Answer:

Despite intuitive expectations, a Turing Machine with infinite tapes does not gain additional computational power:

- In terms of computability, it cannot solve problems beyond what a standard single-tape machine can solve
- However, it can achieve greater efficiency in computation, solving the same problems in fewer operational steps

Time complexity implications:

- Multi-tape machines can simulate single-tape equivalents with quadratic improvement in efficiency
- For instance, multi-tape Turing Machines can simulate single-tape operations in $O(n^2)$ time
- Complexity classifications (P, NP, etc.) remain consistent across these model variations since the efficiency improvements maintain polynomial relationships