Assignment

Theory of Computation

[CSE3702]

UNIVERSAL TURING MACHINE

By

Lavanya Anand

210285

Ridit Jain

210471

Department of Computer Science and Engineering
School of Engineering and Technology
BML Munjal University

(May 2024)

Table of Content

Content Page No.

1. Problem Representation

1.1 Introduction to Turing Machines and Universal Turing Machines

The Turing Machine (TM), conceptualized by Alan Turing in 1936, is a fundamental model of computation that has profoundly influenced the field of computer science. It abstractly represents the simplest form of a computer that, despite its simplicity, can execute any computable function. A Turing Machine operates on an infinite memory tape divided into discrete cells, each capable of holding a symbol from a finite set. The machine processes input by manipulating symbols on the tape according to a set of rules defined by a transition function, under the control of a state register.

A Turing Machine(TM) is expressed as a 7-tuple $(Q, T, B, \Sigma, \delta, q_0, F)$ where:

1. **Q:** the finite set of states
2. **T:** the tape symbol
3. **B:** a blank symbol used as a end marker for input
4. **∑:** the finite set of input symbols
5. **δ:** a transition or mapping function ($Q \times T$ -> $Q \times T \times \{L,R\}$).
6. **q_0:** the initial state
7. **F:** a set of final states

Components of a Turing Machine:

- **Tape:** The Turing machine tape acts as both input and output, where each cell can contain a symbol. The tape is theoretically infinite, allowing unbounded computation.
- **Head:** The read-write head moves along the tape, enabling the machine to perform operations such as read, write, and erase by accessing individual tape cells.
- **State Register:** Contains the current state of the Turing Machine. The state dictates the next operation based on the symbol under the head and the transition rules.
- **Transition Function:** A set of rules that dictate state transitions and head movements based on the current state and the symbol being read. It effectively directs the operations of the Turing Machine.

A Universal Turing Machine (UTM), also introduced by Turing, is a more sophisticated model that simulates the operation of any other Turing Machine. It takes the description of a TM as part of its input along with the input tape for the TM and then mimics the behavior of that TM. Essentially, a UTM can be considered a programmable machine, where the program it runs specifies another Turing Machine.

The objective of this project is to develop a program that effectively simulates a Universal Turing Machine. The UTM developed will take as input the description of any TM, along with an input string, and will process this string as dictated by the TM's rules. The end goal is to determine whether the TM would accept or reject the string based on its predefined conditions.

1.2 The Functionality of a Turing Machine

To understand the challenge, one must first comprehend how a typical TM functions. A Turing Machine consists of a tape, which is divided into cells. Each cell can contain a symbol, which can be read and written by a device known as the head. The position of the head moves along the tape, guided by a set of rules or a transition function, which dictates the TM's operations based on its current state and the symbol it reads on the tape.

The TM begins in a predefined start state and moves between states according to its transition function, which can change the symbol in the current cell, move the tape head one position to the left or right, or maintain its position. The process continues until the machine reaches an accept or reject state, which terminates the operation, with the accept state indicating that the input string conforms to the specific language or conditions defined by the TM.

1.3 Simulating a Turing Machine with a Universal Turing Machine

The simulation of a TM by a UTM involves several complex components and operations. The UTM must first parse and understand the description of the TM, which includes the tape alphabet, the set of states, the start state, and the transition rules. This description effectively programs the UTM to behave like the specified TM when processing an input string.

Once configured, the UTM uses its own mechanisms to simulate the TM's tape, head, and state transitions. This requires the UTM to manage a dynamic representation of the tape, adjust the head's position as needed, and switch between states according to the simulated TM's rules. The challenge lies in accurately replicating every possible behaviour of the TM, including handling infinite loops and ensuring that the UTM's decisions (accept or reject) match those that the original TM would make under the same conditions.

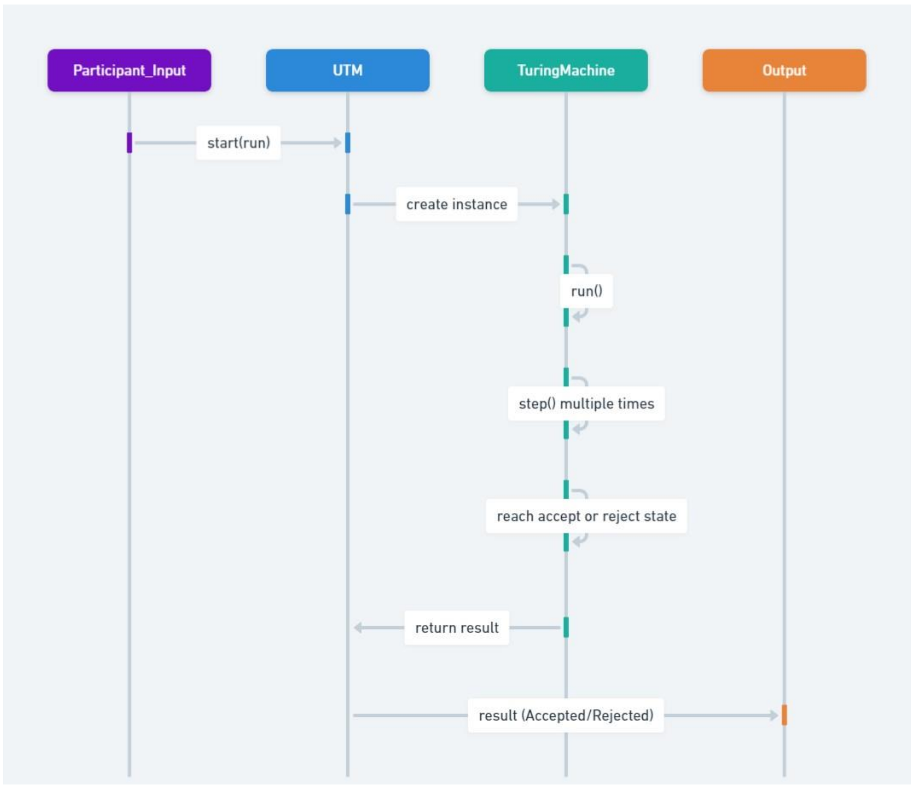


Figure 1.1 Sequence Diagram for the Universal Turing Machine Simulation

1.4 Problem

The specific problem tackled by this project involves the language $L = \{0^{2n}1^{n} | n \geq 0\}$, which comprises strings formed by a sequence of zeros followed by exactly half as many ones. For example, the string "000011" is a valid member of this language as it contains four zeros followed by two ones. This language is interesting because it involves a direct relationship between two segments of the string: the count of zeros, which is always twice the count of ones that follow.

This language represents a straightforward yet non-trivial example for a TM, where the machine must accurately count and compare the number of symbols. The challenge for the Turing Machine—and by extension, the UTM simulating it—is to process each string and determine if it adheres to the specified pattern of zeros and ones. The TM must keep track of the number of zeros and ensure that the following sequence of ones is exactly half the length. If these conditions are met, the string is accepted; otherwise, it is rejected. This problem exemplifies the type of pattern recognition and counting that Turing Machines are equipped to handle, providing a clear and structured task for our UTM simulation.

The Turing Machine (TM), designed to solve this problem, must accurately count and compare the number of symbols in each string. The challenge for the TM—and by extension, the Universal Turing Machine (UTM) simulating it—is to systematically process each string to determine if it adheres to the defined pattern of zeros and ones. Specifically, the TM must:

1. Count the number of zeros.
2. Verify that the subsequent sequence of ones is exactly half the length of the zeros.
3. Accept the string if these conditions are satisfied; otherwise, reject it.

This computational requirement exemplifies the type of pattern recognition and sequential counting tasks that Turing Machines are adept at handling, setting a precise and structured task for our UTM simulation.

Universal Turing Machine Simulation

A Universal Turing Machine is a highly versatile model capable of simulating any other Turing Machine by using the description of a TM and an input string to demonstrate how the described TM would process that string. For our project, we configure a UTM to simulate a TM specifically designed to process strings from the language L, ensuring that the behavior of the UTM precisely matches that of the original TM.

Description and Components of the Turing Machine

Formal Definition of Language L:

- $L = \{0^{2n}1^{n} | n \geq 0\}$
    - **0^n** represents a string of n consecutive 0s.
    - **1^n** represents a string of n consecutive 1s.
    - **n** is an integer greater than or equal to 0.

Examples:

- Strings in L: "", "001", "000011", "000000111", "000000001111", ...
- Strings not in L: "01", "1", "0011", "0101", ...

Turing Machine Components:

1. **Tape Alphabet (Γ):** The set of symbols the machine uses during operations, which includes input symbols and additional symbols needed in the computation.

2. **Input Alphabet (Σ):** The set of elements or symbols that encode the strings in the language L, specifically {0,1}.
3. **States (Q):** Includes the start state (q0), accepting state (q_A), and rejecting state (q_R).
4. **Transition Function (δ):** Specifies how the Turing Machine transitions between states based on the currently-read patterns on the tape. The transition function guides decisions regarding the current tape cell and the state, with instructions to "move left," "move right," or "stay put."
5. **Start State (q_0):** The initial state when the TM receives the input string.
6. **Accept States:** Describes the set of conditions under which strings are accepted as members of language L.
7. **Reject States:** Represents conditions where the input string does not meet the criteria for language L.

Constraints and Assumptions

Before delving into the operational mechanics of the Turing Machine, it is critical to outline the constraints and assumptions under which the machine will operate:

1. **Binary Input:** The tape's alphabet consists solely of zeros and ones, reflecting the binary nature of the language L.
2. **Termination States:** Upon processing a string from the language L, the Turing Machine must halt in an accept state if the string conforms to the language rules. Conversely, it should halt in a reject state if the string does not meet the criteria.
3. **Input Flexibility:** The Turing Machine is designed to handle strings of any length, ensuring flexibility and robustness in processing potentially infinite tape lengths.

Turing Machine Configuration

To address the problem at hand, let us construct a Turing Machine capable of recognizing the language L={02n1n|n≥1} }:

- **States (Q):** $\{A,B,C,D,E\}${A,B,C,D,E}, with AA as the initial state.
- **Tape Alphabet (T):** {0,1,X,Y,_}, where __ represents a blank symbol.
- **Input Alphabet (Σ):** {0,1}.
- **Final States (F):** $\{E\}$, indicating successful processing.

Transition Function Table:

Current State	Read Symbol	Next State	Write Symbol	Move
Q0	0	Q1	B	R
Q1	0	Q2	B	R
Q2	0	Q2	0	R
Q2	1	Q2	1	R
Q2	B	Q3	B	L
Q3	1	Q4	B	L
Q4	0	Q4	0	L
Q4	1	Q4	1	L
Q4	B	Q0	B	R
Q0	B	Q5	B	R

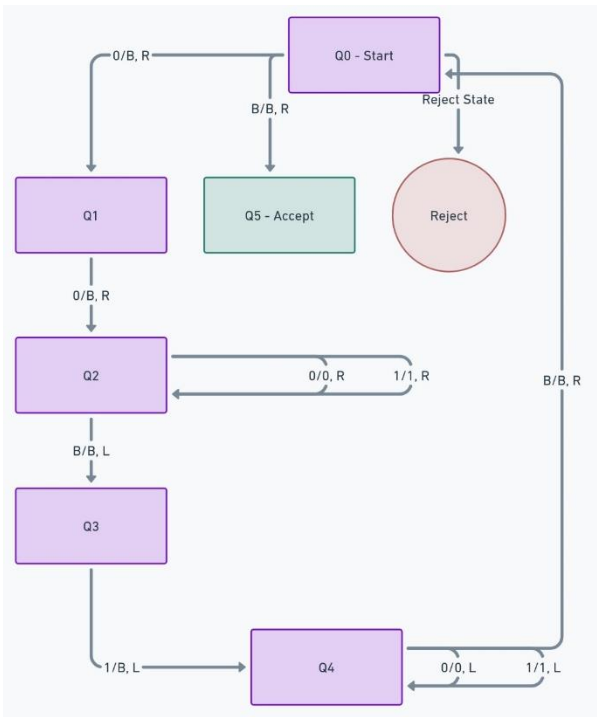


Figure 1.2 State Transition Diagram

This table and the corresponding diagram articulate how the Turing Machine transitions through various states based on input symbols, emphasizing the complexity and precision required in simulating such machines using a UTM. This detailed representation forms the basis for testing and validating the UTM's capability to accurately simulate the specified TM and by extension, solve the problem posed by the language L.

Turing Machine Operation Example

Let's examine how this Turing Machine processes the string "001":

Initial Tape Configuration:

0	**0**	**1**	**_**
[A]	0	1	_

Transition Details:

1. **Move 1:**
    - **Operation:** $\delta(\mathrm{q}0,0) = (\mathrm{q}1,B,R)$
    - **Action:** Replace '0' with 'B' and move right.
    - **Resulting Tape:**

B	**0**	**1**	**_**
B	**[B]**	1	_

2. **Move 2:**
    - **Operation:** $\delta(B,0) = (C,B,R)$
    - **Action:** Replace next '0' with 'B' and move right.
    - **Resulting Tape:**

B	**B**	**1**	**_**
B	B	**[C]**	_

3. **Move 3:**
    - **Operation:** $\delta(C,1) = (C,1,R)$
    - **Action:** Confirm '1' and move right.
    - **Resulting Tape:**

B	**B**	**1**	**_**
B	B	1	**[C]**

4. **Move 4:**
   - **Operation:** $\delta(C,\_) = (D,B,L)$
   - **Action:** Replace blank with 'B', preparing to check completion, move left.
   - **Resulting Tape:**

B	**B**	**1**	**B**
B	B	**[D]**	B

5. **Move 5:**
   - **Operation:** $\delta(D,1) = (D,B,L)$
   - **Action:** Replace '1' with 'B', check if zeros left, move left.
   - **Resulting Tape:**

B	**B**	**B**	**B**
B	**[D]**	B	B

6. **Move 6:**
   - **Operation:** $\delta(D,B) = (A,B,R)$
   - **Action:** Confirmed no zeros left, cycle to check more or halt, move right.
   - **Resulting Tape:**

B	**B**	**B**	**B**
B	B	**[A]**	B

7. **Move 7:**
   - **Operation:** $\delta(A,B) = (E,B,R)$
   - **Action:** No more inputs, move to accept state.
   - **Resulting Tape:**

B	**B**	**B**	**B**
B	B	B	**[E]**

Final State: The machine reaches state E, indicating the string "001" is accepted according to the language L. This step-by-step account illustrates the Turing Machine's ability to meticulously follow its programming to determine whether strings belong to the specified language. Each transition, carefully orchestrated to manipulate the tape and guide the machine through its states, showcases the robust computational architecture designed to support complex decision-making processes inherent in language recognition tasks.

2. Explanation of the Data Structures

The simulation of a Universal Turing Machine (UTM) requires a detailed understanding of both Turing Machine (TM) and UTM components. These models are foundational in computational theory, with the UTM designed to emulate any TM's operational protocol by interpreting its description and executing its transition rules. In this section, we explore the integral data structures and their interactions within these models to clarify their roles in our simulation.

2.1 Turing Machine (TM)

2.1.1 Components

1. **Tape**

    - **Structure:** List
    - **Description:** The tape serves as the primary memory unit of the TM, comprised of an array of cells each containing a symbol from a predefined alphabet. This tape is conceptually infinite, capable of expanding dynamically as necessary when the machine's head reaches near its current boundary. The initial configuration includes the input string followed by an unlimited sequence of blank symbols represented by 'B'.
    - **Functionality:** The head accesses this tape to read and write symbols, crucial for the machine’s processing. The list's ability to dynamically adjust by appending or inserting symbols ensures that no data is lost or overwritten incorrectly, preserving the integrity of the computational process.

2. **Head**

    - **Structure:** Integer
    - **Description:** This component is essentially a pointer or marker that can move left or right across the tape, performing read and write operations based on the machine's transition rules.
    - **Functionality:** Its movement is dictated by simple arithmetic operations on its position index, allowing for efficient transitions and ensuring that the head aligns properly with the tape cells during operations.

3. **State Register**

    - **Structure:** String or Integer
    - **Description:** Holds the current operational state of the TM, acting as a control mechanism that determines the flow of execution based on the transition functions.
    - **Functionality:** This register updates during computation to reflect the machine’s state transitions, guiding the subsequent actions of the read/write head.

4. **Transition Function**

    - **Structure:** Dictionary
    - **Description:** A critical component, the transition function maps pairs of states and read symbols to a set of outputs that include the next state, the symbol to write, and the direction for head movement.
    - **Functionality:** The dictionary structure allows for rapid lookup of instructions, which is essential for the TM to respond correctly and swiftly to its current inputs.

5. **Start, Accept, and Reject States**

    - **Structure:** Strings or Integers, Sets
    - **Description:** These states define the TM's operation bounds: the start state initiates computation; accept states define successful completion criteria; reject states signify failure to comply with the TM's rules.
    - **Functionality:** By evaluating against these states at each step, the TM decides whether to continue computation or halt with a result, ensuring that the machine’s operations are goal-oriented and bounded.

2.1.2 State Transitions for TM

The state transitions within a Turing Machine (TM) are meticulously organized and dictated by the transition function. This function is essentially the "programming" of the TM, providing a set of instructions that the machine follows based on its current state and the symbol it reads from the tape. Below, we explore these transitions in detail, lets focus on how the machine moves from one state to another, changes symbols on the tape, and directs the movement of the head.

- **Start State**: The start state, designated as Q_0, is where the TM begins its computation. It sets the operational baseline before any input is processed. At the start, the TM reads the symbol under the head in this initial state and consults the transition function to determine the subsequent action.
- **Accept States**: These states indicate a successful halt of the TM, where the input string is accepted. For instance, reaching state Q_5 in this TM configuration suggests successful processing according to the machine's specified rules.
- **Reject States**: Conversely, these states signal that the input string does not meet the machine's specifications, resulting in the TM halting and rejecting the input. In your TM, there are defined reject states to handle such outcomes.

Detailed State Transitions

The transitions provided in your TM code define specific behaviors as follows:

- **Transition from Q_0 to Q_1**:
    - **Condition**: If the TM is in state Q_0 and reads the symbol '0'.
    - **Action**: Write a 'B', move the head to the right, and transition to state Q_1.
    - **Purpose**: This may initiate or prepare for complex computation sequences, marking parts of the input as processed.
- **Transition from Q_0 to Q_5:**
    - **Condition**: If the TM is in state Q_0 and reads a blank ('B').
    - **Action**: Write a 'B', move the head to the right, and transition directly to state Q_5, presumably an accept state.
    - **Purpose**: This likely handles cases where the input is already in a final state or meets specific criteria for immediate acceptance.
- **Transition from Q_1 to Q_2**:
    - **Condition**: If in state Q_1 and the symbol under the head is '0'.
    - **Action**: Write a 'B', move the head to the right, and transition to state Q_2.
    - **Purpose**: Further processing of the input, setting conditions for the next computational steps.

- **Transition within Q_2:**
    - **Multiple Conditions**:
        - **Reading '0' or '1'**: Continue to write the same symbol and move right.
        - **Reading 'B'**: Write 'B', move left, and transition to Q_3.
    - **Purpose**: Processes a sequence of symbols, possibly counting or verifying sequence lengths before moving to a decision state.
- **Transition from Q_3 to Q_4:**
    - **Condition**: If in state Q_3 and reads '1'.
    - **Action**: Write a 'B', move the head to the left, and transition to state Q_4.
    - **Purpose**: Prepares for final operations or backtracking to check earlier parts of the input.
- **Transition within Q_4 and back to Q_0**:
    - **Multiple Conditions**:
        - **Reading '0' or '1'**: Continue to write the same symbol and move left.
        - **Reading 'B'**: Write 'B', move right, and return to Q_0.
    - **Purpose**: Cycles through the input to possibly reevaluate or reprocess from the beginning, ensuring all conditions for acceptance are met.

Execution Flow

During its operation, the TM continuously:

1. **Reads the current tape symbol** under the head.
2. **References its current state** and the transition function to decide:
    - **What symbol to write** on the tape, replacing the current symbol under the head.
    - **Which direction to move** the head (either left or right, based on the transition rule).
    - **Which state to transition to** next, based on the input symbol and the existing state.

The process repeats until the TM reaches an accept or reject state, determining the outcome of the computation based on the designed rules. The effectiveness of the TM relies on the precision with which these transitions are designed and implemented, ensuring deterministic and reliable computational outcomes.

2.2 Universal Turing Machine (UTM)

Components

1. **TM Description Parser**
   - Function: Interprets the detailed setup of any TM, including states, tape alphabet, and transition functions. This component is crucial for configuring the UTM to accurately emulate the specified TM.
2. **Simulated Tape and Head**
   - Description: Replicates the tape and head of the TM being simulated. This allows the UTM to perform the same operations as the TM, ensuring that the behavior of the TM is faithfully reproduced on the UTM platform.
3. **Control Unit**
   - Function: Manages the execution processes of the UTM, applying the TM's rules to the simulated tape based on the current state and the symbol read by the head. This unit effectively coordinates the operations of the simulated TM, dictating transitions and ensuring that each step is executed according to the specified rules.

Transitions for UTM

Transitions within the UTM involve interpreting a TM's description and then emulating its operations step-by-step. This begins with initializing the simulated tape with the input string, setting the machine to the start state specified by the TM, and systematically applying the TM’s transition rules. The simulation continues until an accept or reject state is encountered, which determines the outcome of the computation. Through this extensive exploration of the data structures used in TM and UTM simulations, we can appreciate the complexity and precision required to model these foundational computational systems. Each component plays a specific role, working in concert to ensure that both TMs and UTMs perform their tasks accurately and efficiently, demonstrating the practical application of theoretical computational concepts.

3. Program/Code

Following is the code

```
import json

class State:
    def _init_(self, name):
        self.name = name

    def _eq_(self, other):
        return isinstance(other, State) and self.name == other.name

    def _hash_(self):
        return hash(self.name)

class Transition:
    def _init_(self, current_state, current_symbol, next_state, write_symbol,
move):
        self.current_state = current_state
        self.current_symbol = current_symbol
        self.next_state = next_state
        self.write_symbol = write_symbol
        self.move = move

class TuringMachineDescription:
    def _init_(self, transitions, start_state, accept_states, reject_states):
        self.transitions = transitions
        self.start_state = start_state
        self.accept_states = accept_states
        self.reject_states = reject_states

    def to_dict(self):
        return {
            "transitions": [(t.current_state.name, t.current_symbol,
t.next_state.name, t.write_symbol, t.move) for t in self.transitions],
            "start_state": self.start_state.name,
            "accept_states": [state.name for state in self.accept_states],
            "reject_states": [state.name for state in self.reject_states]
        }

    @staticmethod
    def from_dict(description_dict):
        transitions = []
        for t in description_dict["transitions"]:
            current_state = State(t[0])
            next_state = State(t[2])
            transitions.append(Transition(current_state, t[1], next_state,
t[3], t[4]))
        start_state = State(description_dict["start_state"])
        accept_states = {State(state) for state in
description_dict["accept_states"]}
        reject_states = {State(state) for state in
description_dict["reject_states"]}
        return TuringMachineDescription(transitions, start_state,
accept_states, reject_states)

    def _str_(self):
        return str(self.to_dict())


class TuringMachine:
    def _init_(self, initial_tape, description):
        self.tape = list(initial_tape) + ['B']
        self.head = 0
        self.current_state = description.start_state
        self.transitions = description.transitions
        self.accept_states = description.accept_states
        self.reject_states = description.reject_states

    def step(self):
        if self.head < 0:
            self.tape.insert(0, 'B')
            self.head = 0

        current_symbol = self.tape[self.head]
        for transition in self.transitions:
            if transition.current_state == self.current_state and
transition.current_symbol == current_symbol:
                self.tape[self.head] = transition.write_symbol
                self.current_state = transition.next_state
                if transition.move == 'R':
                    self.head += 1
                elif transition.move == 'L':
                    self.head -= 1
                return

        if self.current_state not in self.reject_states:
            self.reject_states.add(self.current_state)
        self.current_state = 'Reject'

    def run(self):
        while self.current_state not in
self.accept_states.union(self.reject_states):
            self.step()
        return self.current_state in self.accept_states
```

```python
class TuringMachineDescriptionEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, State):
            return obj.name
        elif isinstance(obj, Transition):
            return {
                "current_state": obj.current_state.name,
                "current_symbol": obj.current_symbol,
                "next_state": obj.next_state.name,
                "write_symbol": obj.write_symbol,
                "move": obj.move
            }
        elif isinstance(obj, set):
            return list(obj)
        elif isinstance(obj, TuringMachineDescription):
            return obj.to_dict()
        return super().default(obj)
def turing_machine_description_decoder(obj):
    if 'transitions' in obj:
        transitions = []
        for t in obj['transitions']:
            current_state = State(t[0])
            next_state = State(t[2])
            transitions.append(Transition(current_state, t[1], next_state,
t[3], t[4]))
        start_state = State(obj["start_state"])
        accept_states = {State(state) for state in obj["accept_states"]}
        reject_states = {State(state) for state in obj["reject_states"]}
        return TuringMachineDescription(transitions, start_state,
accept_states, reject_states)
    return obj

class UniversalTuringMachine:
    def _init_(self, tm_description, initial_tape):
        self.description = tm_description
        self.tm = TuringMachine(initial_tape, tm_description)

    def run(self):
        return self.tm.run()

    def encode_description(self):
        return json.dumps(self.description,
cls=TuringMachineDescriptionEncoder)

    @staticmethod
    def decode_description(encoded_description):
        return json.loads(encoded_description,
object_hook=turing_machine_description_decoder)

# Description for TM based on the state diagram provided
tm_description = TuringMachineDescription(
    [
        Transition(State('Q0'), '0', State('Q1'), 'B', 'R'),
        Transition(State('Q0'), 'B', State('Q5'), 'B', 'R'),
        Transition(State('Q1'), '0', State('Q2'), 'B', 'R'),
        Transition(State('Q2'), '0', State('Q2'), '0', 'R'),
        Transition(State('Q2'), '1', State('Q2'), '1', 'R'),
        Transition(State('Q2'), 'B', State('Q3'), 'B', 'L'),
        Transition(State('Q3'), '1', State('Q4'), 'B', 'L'),
        Transition(State('Q4'), '0', State('Q4'), '0', 'L'),
        Transition(State('Q4'), '1', State('Q4'), '1', 'L'),
        Transition(State('Q4'), 'B', State('Q0'), 'B', 'R')
    ],
    State('Q0'),
    {State('Q5')},
    {State('Reject')}
)

input_tape = ""
utm = UniversalTuringMachine(tm_description, input_tape)

# Encoding description
encoded_description = utm.encode_description()
print("Encoded description:", encoded_description)

# Decoding description
decoded_description =
UniversalTuringMachine.decode_description(encoded_description)
print("Decoded description:", decoded_description)

# Running UTM
result = utm.run()
print("Accepted" if result else "Rejected")
```

4. Results

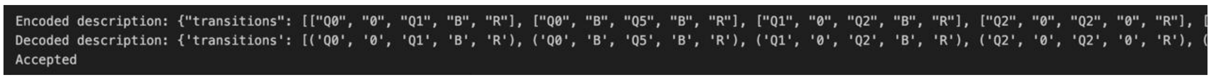
In this section of the report, we present the results obtained from the execution of our Universal Turing Machine (UTM) simulation, designed to process strings based on the language $L=\{0^{2n}1^{n} | n \geq 0\}$. The objective of these tests is to demonstrate the operational effectiveness and correctness of the Turing Machine (TM) we have simulated. We conducted several test cases to evaluate whether the TM accurately accepts or rejects strings in accordance with the predefined language rules.

The test methodology involved running the UTM simulation with a set of pre-determined strings as inputs. Each test case was chosen to represent different scenarios within the scope of the language L. For each input, the UTM executed the TM's transition rules and state definitions, and the output (accept or reject) was recorded. The expected outcomes were predetermined based on the logical structure of the language L, which stipulates that for any string to be accepted, the number of zeros must be exactly double the number of ones.

Test Cases and Outcomes

Here we detail the inputs used, the outputs received, and the rationale for each outcome:

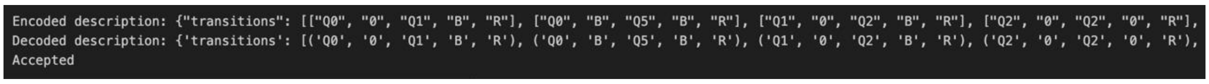
1. **Test Case 1: Empty String**
    - **Input: ""** (an empty string)
    - **UTM Output:** Accepted
    - **Analysis:** The language L definition includes the empty string, representing $0^{2*0}1^{0}$, where n=0. The UTM correctly identified the empty string as valid, aligning with the language rules where the absence of symbols is a valid scenario.



```
Encoded description: {"transitions": [["Q0", "0", "Q1", "B", "R"], ["Q0", "B", "Q5", "B", "R"], ["Q1", "0", "Q2", "B", "R"], ["Q2", "0", "Q2", "0", "R"], [
Decoded description: {'transitions': [('Q0', '0', 'Q1', 'B', 'R'), ('Q0', 'B', 'Q5', 'B', 'R'), ('Q1', '0', 'Q2', 'B', 'R'), ('Q2', '0', 'Q2', '0', 'R'), (
Accepted
```

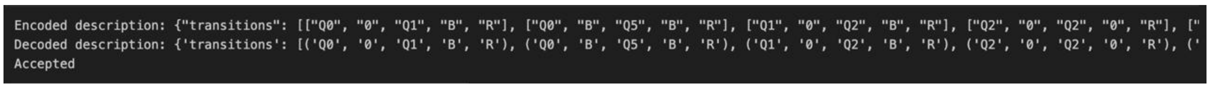
2. **Test Case 2: String "001", that belongs to the language**
    - **Input: "001"**
    - **UTM Output:** Accepted

- **Analysis:** This string follows the pattern $0^{2*1}1^1$, with two zeros followed by one one, perfectly matching the criteria. The UTM processed the string and validated it against the TM's rules, confirming its acceptance as per the defined language.



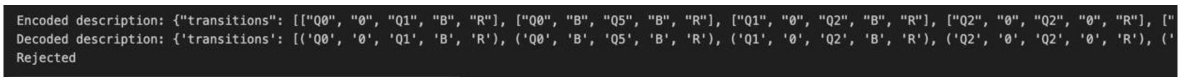
```
Encoded description: {"transitions": [["Q0", "0", "Q1", "B", "R"], ["Q0", "B", "Q5", "B", "R"], ["Q1", "0", "Q2", "B", "R"], ["Q2", "0", "Q2", "0", "R"],
Decoded description: {'transitions': [('Q0', '0', 'Q1', 'B', 'R'), ('Q0', 'B', 'Q5', 'B', 'R'), ('Q1', '0', 'Q2', 'B', 'R'), ('Q2', '0', 'Q2', '0', 'R'),
Accepted
```

3. **Test Case 3: String "000011", that belongs to the language**
   - **Input: "000011"**
   - **UTM Output:** Accepted
   - **Analysis:** Fitting the pattern $0^{2*2}1^2$, this string has four zeros followed by two ones, which is a direct match with the language specifications. The TM successfully recognized and processed the string as valid, demonstrating the accuracy of the UTM's simulation capabilities.



```
Encoded description: {"transitions": [["Q0", "0", "Q1", "B", "R"], ["Q0", "B", "Q5", "B", "R"], ["Q1", "0", "Q2", "B", "R"], ["Q2", "0", "Q2", "0", "R"], ["
Decoded description: {'transitions': [('Q0', '0', 'Q1', 'B', 'R'), ('Q0', 'B', 'Q5', 'B', 'R'), ('Q1', '0', 'Q2', 'B', 'R'), ('Q2', '0', 'Q2', '0', 'R'), ('
Accepted
```

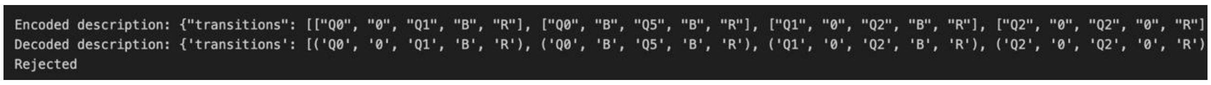
4. **Test Case 4: String "0001", that don’t belong to the language**
   - **Input: "0001"**
   - **UTM Output:** Rejected
   - **Analysis:** With three zeros and one one, this string fails to satisfy $0^{2n}1^n$ since $3 \neq 2 * 1$. The UTM accurately rejected this string, validating the robustness of the transition rules and state evaluations embedded within the TM.



```
Encoded description: {"transitions": [["Q0", "0", "Q1", "B", "R"], ["Q0", "B", "Q5", "B", "R"], ["Q1", "0", "Q2", "B", "R"], ["Q2", "0", "Q2", "0", "R"], ["
Decoded description: {'transitions': [('Q0', '0', 'Q1', 'B', 'R'), ('Q0', 'B', 'Q5', 'B', 'R'), ('Q1', '0', 'Q2', 'B', 'R'), ('Q2', '0', 'Q2', '0', 'R'), ('
Rejected
```

5. **Test Case 5: String "0000011111", that don’t belong to the language**
   - **Input: "0000011111"**
   - **UTM Output:** Rejected

- **Analysis:** Although there are five zeros and five ones, the pattern requires twice as many zeros as ones, which this string does not meet. The TM appropriately identified the discrepancy in the symbol counts and rejected the string, further confirming the precision of our TM's computational logic.



```
Encoded description: {"transitions": [["Q0", "0", "Q1", "B", "R"], ["Q0", "B", "Q5", "B", "R"], ["Q1", "0", "Q2", "B", "R"], ["Q2", "0", "Q2", "0", "R"]
Decoded description: {'transitions': [('Q0', '0', 'Q1', 'B', 'R'), ('Q0', 'B', 'Q5', 'B', 'R'), ('Q1', '0', 'Q2', 'B', 'R'), ('Q2', '0', 'Q2', '0', 'R')
Rejected
```

The test results provide a comprehensive demonstration of the UTM's ability to emulate the specified TM accurately. In each case, the UTM's outputs aligned perfectly with the expected outcomes, underscoring the effectiveness of our simulation in adhering to the theoretical constructs of Turing Machines and their application in language processing. These results not only affirm the operational validity of our UTM but also enhance our understanding of the practical implications of Turing's models in automating complex pattern recognition tasks in computational linguistics.

Through this rigorous testing process, we have confirmed that the Turing Machine, as simulated by the UTM, operates with high reliability and precision, effectively interpreting and processing a range of input strings according to the strict definitions of language L. This validates both the design of the Turing Machine and the effectiveness of the Universal Turing Machine as a powerful tool for studying the properties and behaviors of computable functions.