19. (i) Draw A TM That Loops Forever On All Words Ending In \(A \) And Crashes On All Others.

19. (i) Draw A TM That Loops Forever On All Words Ending In \(A \) And Crashes On All Others.

Introduction to Turing Machines and Their Behaviors

Turing machines (TMs) are foundational models in the theory of computation, used to formalize the concepts of algorithmic processes. They serve as idealized computational devices capable of recognizing languages, solving problems, and modeling algorithms. Understanding how to design a Turing machine with specific behaviors on different input strings is essential for grasping the nuances of computational theory.

In this article, we explore how to construct a Turing machine (TM) that exhibits a particular behavior based on the input string: it should loop forever on all words ending in the symbol \(A\), and crash (enter a non-accepting, halting state) on all other words. This task involves designing a machine that can recognize whether the input ends with \(A\) and then behave accordingly.

Understanding the Problem Statement

Before diving into the construction, let's clarify what the problem entails:

- Input strings: Any string over some alphabet (for simplicity, assume the alphabet includes at least \(A\) and possibly other symbols).
- Behavior requirements:
- If the input string ends with \(A\), the TM should loop forever—meaning it never halts or halts with an infinite loop.
- If the input string does not end with \(A\), the TM should crash, which in Turing machine terms typically means entering a rejecting or halting state with no further transitions.

This problem is an example of designing a machine with conditional infinite behavior based on the input's last symbol, illustrating the capability of TMs to perform complex control flow.

Key Concepts for the TM Design

Designing such a TM involves understanding several core concepts:

1. Recognizing the End of the Input

- Since TMs operate on tapes with potentially unbounded length, they need a method to determine the end of the input.
- Typically, the input is written on the tape, and the head starts at the beginning.
- The TM moves right until it encounters a blank symbol (often denoted by \(\sqcup \)), which signifies the end of the input.

2. Differentiating Behavior Based on the Last Symbol

- Once the TM reaches the end of the input, it must check the last symbol before the blank.
- If that symbol is \(A\), it should initiate an infinite loop.
- Otherwise, it should halt in a crash state.

3. Creating Infinite Loop and Crash States

- An infinite loop can be created by having the TM repeatedly perform some action, such as moving back and forth or looping over a set of states without halting.
- A crash can be modeled by entering a halting state with no further transitions.

Step-by-Step Construction of the Turing Machine

Let's now detail the steps to design the TM:

Step 1: Initialization

- The TM begins at the leftmost position of the input tape.
- It will scan rightward to find the end of the input (the blank symbol).

Step 2: Moving to the End of Input

- The machine reads symbols and moves right:
- For each symbol (say, \(a\)), it moves right.
- When it encounters the blank symbol \(\sqcup \), it recognizes the end of the input.

Step 3: Checking the Last Symbol

- After reaching the blank, the machine moves one step left to read the last symbol.
- It then evaluates whether this symbol is \(A\).

Step 4: Behavior Based on the Last Symbol

- If the last symbol is \(A\):
- The machine enters a looping state, where it repeatedly moves back and forth or performs a cycle of actions with no halting.
- If the last symbol is not \(A\):
- The machine enters a crash state, halting immediately with no further transitions.

Designing the States and Transitions

To implement the above logic, we define specific states and transition rules:

State	Description	Transition Rules
\(q_0\)	Start state	Move right over input symbols until blank \(\sqcup \) is found
\(q_{end}\)	End of input reached	Move left to read last symbol
\(q_{check}\)	Check last symbol	If \(A\), go to looping state; else, crash
\(q_{loop}\)	Loop forever	Repeatedly move back and forth (or stay in a cycle)
\(q_{crash}\)	Crash state	Halt immediately

Implementing the Infinite Loop

The key challenge is to create a state that results in infinite looping:

- Method 1: Moving Back and Forth
- From q_{check}, if the last symbol is A, transition to a state q_{loop}.
- In q_{loop}, define transitions:
- Move right, then move left, repeatedly.
- This cycle never halts, thus creating an infinite loop.

- Method 2: Repeating Actions
- Alternatively, in q_{loop}, perform an action that leads back to itself, such as moving right and left endlessly.

Example Transition Diagram (Conceptual)

Here's a simplified conceptual flow:

1. Start at q_0:
- Read input symbol.
- If symbol is not blank ($ \sqcup $), move right.
- Loop until blank is encountered.

2. At q_{end}:
- Move left to the last symbol of input.
- Transition to q_{check}.

3. At q_{check}:
- Read the last symbol.
- If symbol == A, transition to q_{loop}.
- Else, transition to q_{crash}.

4. At q_{loop}:
- Move right, then move left repeatedly.
- Continue indefinitely, creating a loop.

5. At q_{crash}:
- Halt (reject).

Formal Turing Machine Description

Below is a formal pseudo-implementation:

- States: $Q = \{q_0, q_{end}, q_{check}, q_{loop}, q_{crash}\}$
- Input alphabet: Σ, including A
- Tape alphabet: $\Gamma \supseteq \Sigma$
- Start state: q_0
- Blank symbol: $\sqcup$
- Transition function δ:

Current State	Read Symbol	Next State	Write Symbol	Move Direction
q_0	any	q_0	same	Right
q_0	$\sqcup$	q_{end}	same	Left
q_{end}	any	q_{check}	same	Left
q_{check}	A	q_{loop}	same	-
q_{check}	other	q_{crash}	same	-
q_{loop}	any	same	same	Move right
q_{loop}	blank	same	same	Move left (or stay)

Note: The looping state can be designed to move back and forth or perform a cycle of transitions that never halt.

Implications and Applications

Designing such a TM demonstrates the power of Turing machines to perform complex behaviors based on input characteristics. The ability to create infinite loops conditioned on input patterns has implications in understanding:

- Decidability and undecidability: How certain problems are inherently non-halting or non-computable.
- Language recognition: How machines can recognize specific patterns and behaviors.
- Control flow in computation: How conditional loops and halts can be modeled.

This construction also illustrates the importance of state design and transition logic in creating machines with tailored behaviors.

Conclusion and Summary

Constructing a Turing machine that loops forever on all words ending in \(A\) and crashes on all others involves understanding the process of:

- Moving to the end of the input.
- Checking the last symbol.
- Creating an infinite loop conditionally.
- Halting immediately otherwise.

This exercise deepens our understanding of the flexibility and expressive power of Turing machines, serving as a fundamental example in the theory of computation. Whether for academic purposes or practical modeling, the ability to design such machines underscores the core principles of automata theory, formal languages, and computational logic.

Further Reading and Resources

- "Introduction to the Theory of Computation" by Michael Sipser
- "Automata

Frequently Asked Questions

What is the primary behavior of the Turing Machine described in problem 19. (i)?

The Turing Machine loops forever on all input strings ending with 'A' and halts (crashes) on all other inputs.

How can one design a Turing Machine to recognize whether an input string ends with 'A'?

The TM should scan to the end of the input; if the last symbol is 'A', it enters an infinite loop; otherwise, it halts immediately.

Why does the Turing Machine need to loop forever on certain inputs instead of halting?

Looping forever signifies acceptance or a special processing state, indicating the input belongs to a particular language—in this case, strings ending with 'A'—while halting indicates rejection.

Is the described Turing Machine a decider or a semi-decider? Why?

It is a semi-decider for the language of strings ending with 'A' because it halts on strings not ending with 'A' but loops forever on those that do, thus not deciding all inputs.

What challenges arise when designing a Turing Machine that loops forever on certain inputs?

The main challenge is ensuring the machine correctly distinguishes inputs that should cause indefinite looping from those that should halt, maintaining proper state transitions and tape movements.

Can this TM be used to decide whether a string ends with 'A'? Why or why not?

No, because the TM loops forever on strings ending with 'A', so it does not halt to provide a definitive yes/no answer, making it a recognizer rather than a decider.

How does this problem illustrate the difference between decidability and recognizability in Turing Machines?

It shows that a TM that loops forever on certain inputs is recognizing a language rather than deciding it, highlighting that not all recognizable languages are decidable.

Additional Resources

Drawing a Turing Machine That Loops Forever on All Words Ending in ‘A’ and Crashes on All Others

Creating a Turing Machine (TM) with specific behaviors based on input characteristics is a fundamental exercise in the theory of computation. The task at hand involves designing a TM that exhibits two distinct behaviors:

- Loops forever on all inputs ending with the symbol ‘A’.
- Crashes (rejects or halts without accepting) on all other inputs.

This problem encapsulates core concepts such as input analysis, conditional behavior, and the construction of Turing Machines with non-trivial operational logic. In this detailed review, we will explore the problem comprehensively, breaking down the requirements, approaches, and the design of such a TM step-by-step.

Understanding the Requirements

Before diving into the construction, it's essential to clarify what is meant by looping forever and crashing in the context of Turing Machines.

What Does Looping Forever Mean?

- The TM enters an infinite computation without halting.
- It does not accept or reject; it simply continues running indefinitely.
- For our purposes, the TM should do this only on inputs ending with 'A'.

What Does Crashing Mean?

- The TM halts immediately after starting, without accepting.
- The halting can be considered as rejecting, or simply halting without acceptance.
- The TM should do this for all other inputs (i.e., those not ending with 'A').

Clarifying Input and Tape Alphabet

- Assume the input alphabet includes at least the symbol 'A'.
- The tape alphabet may include 'A' and the blank symbol (say '_').
- Input is provided on the tape starting from the leftmost cell, with the rest of the tape blank.

Key Objectives

- Behavior on words ending with 'A': After reading the input, the machine should enter an infinite loop, for example, by repeatedly moving left and right on the tape, or performing a cycle of states with no halting condition.
- Behavior on words not ending with 'A': The machine should halt immediately, effectively "crashing" or rejecting.

Design Strategy Overview

To construct such a Turing Machine, we need a systematic approach that checks whether the input ends with ‘A’, then branches into different behaviors accordingly.

High-Level Approach

1. Scan the input to the end:
- Move right across the tape until reaching the blank symbol, indicating the end of the input.

2. Check the last symbol:
- Examine the symbol immediately before the blank.
- If it’s ‘A’, proceed to an infinite loop.
- If it’s any other symbol, halt immediately.

3. Implementing the infinite loop:
- Enter a sequence of states that cause the machine to move back and forth or stay in a cycle with no halting condition.

4. Implementing the crash:
- Halt immediately without accepting or rejecting, effectively crashing.

Key Challenges

- Ensuring the machine correctly identifies whether the last symbol is ‘A’.
- Guaranteeing the infinite loop runs only when appropriate.
- Avoiding unintended behaviors, such as infinite loops on words not ending with ‘A’.

Step-by-Step Construction of the Turing Machine

Let's now detail the construction, including states, transitions, and logic.

1. State Definitions

- q_start: Initial state, start scanning the input.
- q_move_right: Move right to find the end of the input.
- q_check_last_symbol: Check the last symbol before blank.
- q_loop: Infinite loop state for inputs ending with ‘A’.

- q_crash: Halting state for inputs not ending with ‘A’.

2. Transition Functions

From q_start

- Read the current symbol:
- If symbol is ‘A’ or any other valid input symbol, move right.
- Transition to q_move_right to scan to the end.

From q_move_right

- Keep moving right until the blank symbol (‘_’) is found:
- On reading a non-blank symbol, move right and stay in q_move_right.
- When the blank symbol is encountered, move left to position the head on the last symbol.

From q_check_last_symbol

- Read the symbol under the head:
- If ‘A’, transition to q_loop.
- Otherwise, transition to q_crash.

From q_loop

- Enter an infinite loop:
- For example, alternate between moving left and right between two states, with no halting transition.
- This can be achieved with states q_loop1 and q_loop2:
- In q_loop1, move right, then switch to q_loop2.
- In q_loop2, move left, then switch back to q_loop1.
- Since these do not lead to halting, the machine loops forever.

From q_crash

- Halt immediately: no further transitions. The machine halts in a rejecting or crash state.

Example Transition Table

Current State	Read Symbol	Write Symbol	Move Direction	Next State

q_start	A or other	Same	Right	q_move_right
q_move_right	Non-blank	Same	Right	q_move_right
q_move_right	Blank ('_')	Same	Left	q_check_last_symbol
q_check_last_symbol	'A'	Same	Stay	q_loop
q_check_last_symbol	other	Same	Stay	q_crash
q_loop1	Any	Same	Right	q_loop2
q_loop2	Any	Same	Left	q_loop1
q_crash	-	-	-	Halt

Note: The actual implementation may vary slightly, but this captures the core logic.

Deep Dive into the Infinite Loop Construction

Creating a loop that runs forever on inputs ending with 'A' involves designing a cycle of states that perform moves without halting.

Strategies for Infinite Looping

- Back-and-Forth Movement:
- States alternate between moving right and left, never reaching a halting state.
- For example:
- q_loop1: move right, switch to q_loop2.
- q_loop2: move left, switch back to q_loop1.
- This creates an endless oscillation.

- Self-Looping States:
- A single state that stays in the same state, moving in a fixed direction repeatedly.
- However, moving back and forth is often more illustrative.

Implementation Details

- After confirming the last symbol is 'A', transition into q_loop1.
- In q_loop1:
- Move right indefinitely.
- In q_loop2:
- Move left indefinitely.
- Transition between these states to sustain perpetual motion.

Ensuring the Loop Only Triggers When Needed

- The transition into q_loop should only occur if the last symbol is ‘A’.
- The check in q_check_last_symbol ensures this.

Handling Edge Cases and Practical Considerations

While the above approach covers the core logic, real-world Turing Machine design must address several nuances:

Empty Input

- If the input is empty (blank at start), the last symbol is undefined.
- Decide whether the machine crashes or loops.
- Typically, in such cases, the machine should crash, as the input does not end with ‘A’.

Inputs Longer Than One Symbol

- The method of moving to the end and checking the last symbol remains valid regardless of input length.

Multiple Symbols and Symbols Other Than ‘A’

- The check must be precise to distinguish ‘A’ from other symbols.
- The transition logic should handle all possible symbols.

Halting Behavior

- The crash state should be a halting state with no outgoing transitions.
- The infinite loop states should also have no halting transitions.

Summary and Final Thoughts

Designing a Turing Machine that loops forever on all words ending in ‘A’ and crashes on all others exemplifies the power and flexibility of Turing Machines in formal language theory. The key elements include:

- Input scanning to identify the last symbol.

- Conditional branching based on the last symbol.
- Infinite looping achieved via cyclical state transitions.
- Immediate halting for other inputs.

This construction highlights several concepts:

- The importance of carefully managing states to produce desired behaviors.
- The utility of cycle states to generate infinite loops.
- The significance of precise input analysis to conditionally control the machine's behavior.

Conclusion

Constructing a TM with such specific behavior demonstrates a deep understanding of the machinery and logical structure of Turing Machines. It requires meticulous state management, transition design, and an appreciation of how infinite loops can be implemented via cyclical state transitions. Such a machine serves as a valuable educational tool, illustrating how machines can be engineered to recognize, react to, or ignore particular input patterns, and underscores the theoretical foundations underlying computability and decidability.

This problem also emphasizes the importance of designing Turing Machines with clear, well-structured state diagrams, which facilitate understanding and verification of complex behaviors. Whether used as a teaching

19 I Draw A Tm That Loops Forever On All Words Ending In A And Crashes On All Others

19 I Draw A Tm That Loops Forever On All Words Ending In A And Crashes On All Others

Related Articles

- Help, I Was Out Sick For Most Of The Week Before And Didnt Learn This Materal, I Really Need Help
- What Safety Measures Must Be Used When Epinephrine Is Administered From The Sterile Back Table
- Unless Otherwise Stated, An Offer Made Face-to-face Or During A Telephone Call Usually ExpiresT/F

Back to Home