THE CHURCH-TURING THESIS

By
Benard Gavuna

Introduction

- So far in our development of the theory of computation we have presented several models of computing devices.
- **Finite automata** are good models for devices that have a small amount of memory.
- **Pushdown automata** are good models for devices that have an unlimited memory that is usable only in the last in, first out manner of a stack.
- We have shown that some very simple tasks are beyond the capabilities of these models. Hence they are too restricted to serve as models of general purpose computers.

Turing Machines

- We turn now to a much more powerful model, first proposed by Alan Turing in 1936, called the Turing machine.
- Similar to a finite automaton but with an unlimited and unrestricted memory, a Turing machine is a much more accurate model of a general purpose computer.
- A Turing machine can do everything that a real computer can do.
- Nonetheless, even a Turing machine cannot solve certain problems.
- In a very real sense, these problems are beyond the theoretical limits of computation.

Turing Machines

- A Turing machine is a mathematical model of computation that defines an abstract machine which manipulates symbols on a strip of tape according to a table of rules.
- Abstract machines that formalize the concept of computation

Components of Turing Machines

- ***Tape***: An infinite sequence of cells, each containing a symbol from a finite alphabet.
- ***Head***: Reads and writes symbols on the tape and can move left or right.
- ***State Register***: Stores the state of the Turing machine, one of a finite set of states.
- ***Finite Table of Rules***: Dictates the machine's actions based on the current state and the symbol it reads.

Difference between Finite Automaton Turing Machines

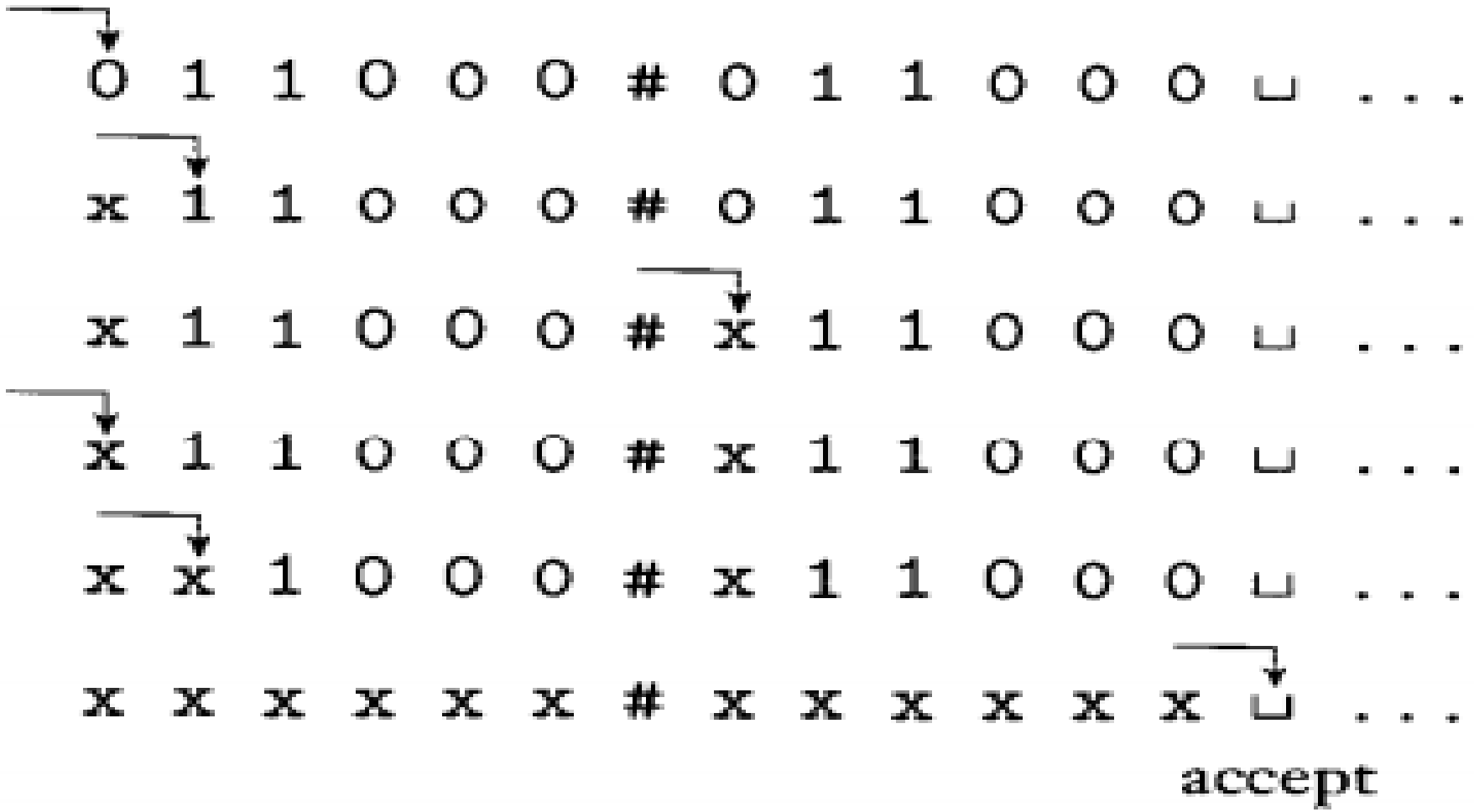
1. A Turing machine can both write on the tape and read from it.
2. The read - write head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

Example: TM for Language L = {w#w | w∈{0,1}*}

M_1 = "On input string w:

1. Scan the input to be sure that it contains a single # symbol. If not, *reject*.
2. Zig-zag across the tape to corresponding positions on either side of the # symbol to check on whether these positions contain the same symbol. If they do not, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
3. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise *accept*."

Example: snapshots for input string 011000#011000



Formal Definition of a Turing Machine

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the special ***blank*** symbol $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Formal Definition of a Turing Machine

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the special ***blank*** symbol $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Example 1

- *Turing machine to recognize the language $L=\{a^n b^n \mid n \geq$.*

1. *Start in state $q0$.*
2. *Scan right to find the first 'a', replace it with 'X', move to state $q1$.*
3. *Move right to find the first 'b', replace it with 'Y', move to state $q2$.*
4. *Move left back to the leftmost 'a' (or 'X'), repeat until no more 'a' or 'b' pairs are found.*
5. *If all 'a's and 'b's are successfully replaced, accept; otherwise, reject.*

Variants of Turing Machines

- Alternative definitions abound, including those with ***multiple tapes*** or with ***nondeterminism***

a) Multitape Turing Machine

- Has multiple tapes and multiple heads.
- Each tape has its own head for reading and writing.
- Equivalent in computational power to a single-tape Turing machine but can be more efficient.

Variants of Turing Machines

b) Nondeterministic Turing Machine

- At each step, the machine can choose from multiple possible transitions.
- Equivalent in computational power to deterministic Turing machines, though it's unknown if they differ in terms of time complexity (related to P vs NP problem).
- The transition function: $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$.

Variants of Turing Machines

c) Universal Turing Machine

- A Turing machine that can simulate any other Turing machine.
- The concept leads to the idea of stored-program computers.

Equivalences with other models

- The essential feature of Turing machines is unrestricted access to unlimited memory.
- This feature makes TM to be stronger computing models compared to finite automata and pushdown automata.
- You might have several programming languages but all describe the same set of algorithms

Algorithms

Informal Definition

- An algorithm is a collection of simple instructions for carrying out some task.
- Sometimes are called **procedures** or **recipes**
- Ancient mathematical literature contains descriptions of algorithms for a variety of tasks, such as finding prime numbers and greatest common divisors.

Algorithms

Formal Definition

Formally, an algorithm is defined as a Turing machine that halts on all inputs. It consists of a finite sequence of well-defined instructions that take an input, process it, and produce an output in a finite amount of time.

The Church-Turing Thesis

- The Church-Turing Thesis is a foundational concept in theoretical computer science and mathematics, proposing a formal definition of what it means for a function to be computable.

The Church-Turing Thesis

Statement

- The Church-Turing Thesis posits that any function that can be computed by an algorithm can be computed by a Turing machine.
- In other words, the concept of "computability" defined by Turing machines is equivalent to the informal notion of an algorithm.

Implications

- If a problem can be solved by any computational means, it can be solved by a Turing machine.
- The thesis is not a formal theorem but a hypothesis about the nature of computation.

Historical Context

- Proposed independently by Alonzo Church and Alan Turing in the 1930s.
- Church used the lambda calculus, and Turing used his eponymous machines to formalize the concept of computation.

Use Cases and Applications

- ***Computability Theory*** - Understanding which problems can be solved algorithmically.
- ***Complexity Theory*** - Studying the efficiency of algorithms and classifying problems based on their computational complexity.
- ***Artificial Intelligence*** - Designing algorithms that mimic human problem-solving abilities.
- ***Cryptography*** - Developing secure encryption methods based on computational hardness assumptions.

Examples

- ***Halting Problem*** - Given a description of a Turing machine and an input, determine if the machine halts on that input. Proven undecidable by Turing.
- ***Prime Checking*** - Designing a Turing machine that determines if a given number is prime.
- ***Sorting Algorithms*** - Implementing well-known algorithms like QuickSort or MergeSort as Turing machines.

Exercise

Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet $\{0,1\}$:

a. $\{w \mid w$ contains an equal number of 0s and 1s$\}$.

b. $\{w \mid w$ contains twice as many 0s as 1s$\}$.

c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$.