CSE331 Automata and Computability

Lecture 1.1: Introduction to the Theory of Computation

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

Introduction: why this course

We are living in a fascinating world led by a mysterious machine called computers. Our life has become so dependent on these machines that nowadays many people even believe that one day may come when computers will be able to do everything as seen in the science fiction. This might seem a legit guess to many but is that true? Is there any proof that a computer will eventually be invincible solving all the problems that can be fed into it? In short, is there any proven limitation of a computer? If any, then what are they? If these questions have been bugging you, then congratulations, you have come to the right place!

Welcome to this exciting course of computer science where we shall explore, for the first time, the extraordinary science behind computers. In this course, you shall explore the fundamental capabilities of a computer as well as its limitations if there is any.

Before diving into the detail, let's explore few basics. In this lecture, we shall discuss a brief overview of the course, its scope as well as some high-level applications.

Theory of Computation

This course broadly deals with what is called the **Theory of Computation** – a joint branch of computer science & Mathematics – that can be divided into **three** major components-

- **Automata Theory:** the study of **conceptual machines/abstract machines/automata** that can solve logical problems or computational problems. The singular form of automata is **automaton**.
- **Computability Theory:** deals with the very question whether a problem can actually be solved by a computer (an automaton) or not.
- **Computational Complexity Theory:** deals with the **efficiency** of the solution of a computational problem provided that the problem is solvable by a computer.

  This efficiency is measured in terms of-
  - ***Time Complexity****:* the time or the number of steps required to solve the problem in terms of input size.
  - ***Space (memory) Complexity***: the amount of memory required to solve the problem in terms of input size.

In addition to these, there is the **Formal Language Theory** that is very closely related to the Automata Theory.

However, in this course, we shall focus on the Automata Theory and a little bit on the Computability Theory.

Rational thinking process and Automata Theory

To understand Automata theory, first, we need to understand how the human mind thinks and take decisions logically. (Recall the example of the Stop sign, finding the number of occurrences of 2's etc. from the main lecture). If you are given an instance (example) of a problem as an **input**, you will, first, **process** the input via proper thinking based on the already-known or given set of instructions or knowledge and then come to a decision in the form of an action or a response, formally known as an **output**.

Automata Theory emerged from this **input-processing-output framework** in the early 20th century when mathematicians and logicians tried to develop the concept of a machine that can replicate this thought process – the way we think.

Automata are abstract models of machines that can perform logical functions on its own based on some predefined rules given a set of inputs.

However, **not** all problems are solvable in a computer and that is where the Computability Theory comes in which helps us to decide which problem can be solved from a computational point of view and which one cannot.

A very famous example of the non-solvable problems is **the halting problem**. It is basically the question- '*Can a system be developed that can decide by itself whether a computer program will stop at some point or run indefinitely given the description of an algorithm of that program and its corresponding input?*' Alan Turing proved that we cannot develop such a machine.

(Finite) State Machine

Abstract machines can imitate the logical thought processes of the human mind. The following example of a state machine shows how we can achieve that.

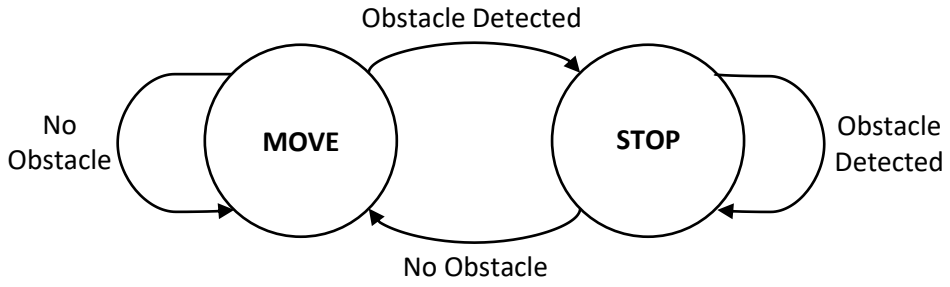

Figure 1: States and corresponding transitions of an obstacle avoiding robot

Consider a simple obstacle avoiding robot. It is built on a very basic principle:

> *If there is nothing blocking its path, it keeps **moving**, however, if there is an obstacle, it **stops** right away.*

We can express the above logical statement by a state diagram (refer to CSE260: Digital Logic Design course) consisting of few states and some transitions among them.

Notice, the robot can only be in two possible states (see Figure 1)-

State-1: it is moving (**MOVE** state)

State-2: it is stopped (**STOP** state)

The following way the state diagram simulates the behavior of the robot-

1. Let's start from the MOVE state.
2. As long as it sees no obstacle in its path, it keeps moving i.e. it remains in the same state (the self-loop in MOVE state).
3. The moment the robot's sensor picks up something blocking its path, it goes from MOVE state to the STOP State (following the top arrow from MOVE to STOP).
4. As long as the obstacle is there it remains in that state (the self-loop in STOP state).
5. As soon as the path is cleared, it heads back to the MOVE state again (following the down arrow from STOP to MOVE).

In this way, a state machine can be used to simulate the logical process of an entity.

Theory of Computation – A look back

- In 1936, Alan Turing first came up with the idea of an abstract machine that had, in theory, all the computational capability of a modern computer.
- It was a simple machine and only had-
    - **A memory tape** of infinite length divided into cells;
    - **A read/write head** that would go back & forth over the tape sequentially and could either write into the associated cell or read from it; and
    - **A control table** that would hold all the predefined rules that would tell the machine what to do in a given situation.
- Turing also tried to define the **boundary** between what a computer potentially could do and what it could not – a subject of discussion in the Computability Theory.

The following points out some important milestones in the history of the theory of computation-

- In 1943, Warren McCulloch and Walter Pitts published a paper that tried to model the nervous system as a finite computational device.
- In the 1950s, G. H. Mealy and E. F. Moore further generalized the theory to more powerful finite machines.
- In the late 1950s, Noam Chomsky started the study of formal grammars. These grammars have close associations with abstract automata and the computability of problems. He also described the famous Chomsky Hierarchy.

- In 1969, S. Cook defined a distinction between the problems that can be solved by computers truly and the problems that can be solved in principle, however, will take so much time – even with sufficient computational power – that computers are essentially useless for those problems.

Interestingly, a lot of the pioneers who contributed highly in this field actually came from other genres-

- The first two people who formalized the finite state machines were neuroscientists.
- Noam Chomsky who pioneered in the field of **formal grammars** – a very important in a number of branches in computer science including Automata theory, Compilers or Natural Language Processing (NLP), and text mining – actually had his background in linguistics and journalism.

Applications of Theory of Computation

The applicability of Theory of Computation is so immense that S Wolfram, a renowned computer scientist, argues that the whole universe can eventually be described as a machine with a finite number of different states and rules. Some applications follow-

- Finite Automata (FA) and some formal grammars help in designing and construction of important software components and systems.
- Concepts of computability help us decide whether a computational problem can be solved with a direct approach or other workarounds needs to be adopted e.g. simplification, approximation, some kind of heuristics etc.
- Text mining & Pattern matching.
- And many, many more…

In this course, we shall study the concepts of Regular Expression (RE) and Regular Language (RL) which are widely used in pattern matching in many fields. I shall also explore Context-free Grammar (CFG) as well as Context-free Language (CFL) which is the key in recognizing your codes used in various compilers.
