CS760 Topics in Computational Complexity **2024-25 Sem I**

Lecture 2: 06-08-2024

Scribe: Anirudh Garg, Pratik Sahoo *Lecturer: Rohit Gurjar*

1 Some undecidable problems

Conway's Game of Life. You can view the game's rules here play the game here. Despite its simplicity, certain questions about the Game of Life are undecidable. For instance, it's undecidable whether a given initial configuration will eventually stabilize, enter a repeating cycle, or continue to change indefinitely (this is related to the "Life universality problem"). This undecidability arises because the Game of Life is Turing complete, meaning it can simulate any computation. As a result,

Claim 2.1. *No general algorithm can predict the long-term behaviour of arbitrary configurations in Conway's Game of Life.*

Hilbert's tenth problem. The problem ask if a given polynomial equation (in multiple variables) has an integer solution. The problem is known to be undecidable. This is the result of combined work of Martin Davis, Yuri Matiyasevich, Hilary Putnam and Julia Robinson, which culminated in 1970. The proof shows that you can encode the behavior of any C++ program into a Diophantine equation in such a way that the program halts on a given input if and only if the corresponding Diophantine equation has an integer solution.

2 The Turing Machine: A Formal Model of Computation

A Turing machine (TM) is a mathematical model of computation that consists of a **tape**, a **head**, and a **finite state control**. It operates on a tape divided into cells, each of which can contain a symbol from a finite alphabet. The head can read and write symbols on the tape and move left or right. The finite state control determines the actions of the head based on the current state and the symbol read.

Formally, a Turing machine is defined by a tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where:

- **Q** is a finite set of states.
- $\boldsymbol{\Sigma}$ is the input alphabet, a finite set of symbols.
- $\boldsymbol{\Gamma}$ is the tape alphabet, a finite set of symbols that includes $\boldsymbol{\Sigma}$ and the blank symbol 'B'.
- δ is the transition function, mapping a state and a symbol to a new state, a symbol to write, and a direction (L or R) for the head to move.
- q_0 is the initial state.
- **F** is a set of final states.

The Turing machine starts with the input string written on the tape, the head positioned at the beginning of the string, and the control in the initial state. The machine then iterates through the following steps:

1. **Read**: The head reads the symbol on the current cell.
2. **Transition**: The transition function is consulted based on the current state and the symbol read. It determines the new state, the symbol to write, and the direction to move the head.

3. **Write**: The head writes the new symbol onto the current cell.

4. **Move**: The head moves left or right according to the transition function.

The machine halts when it enters a final state. The output of the Turing machine is the string on the tape at the time of halting.

2.1 Multi-Tape Turing Machines

A multi-tape Turing machine is an extension of the basic model with multiple tapes. Each tape has its own head and is controlled by the same finite state control. This allows for more complex computations as information can be stored and accessed simultaneously on different tapes.

2.2 Simulation of Multi-Tape by Single-Tape

Surprisingly, any multi-tape Turing machine can be simulated by a single-tape Turing machine. This is achieved by encoding the contents of multiple tapes into a single tape, using special delimiters to separate the data. The single-tape machine can then simulate the actions of the multi-tape machine by carefully traversing the tape and manipulating the encoded data. A multi-tape TM running in time $t(n)$ can be simulated by a single tape TM running in time $O(t(n)^2)$.

2.3 Binary Alphabet

A Turing machine with any alphabet can be converted into a Turing machine with a binary alphabet. This is done by encoding each symbol of the original alphabet with a unique binary string. The binary Turing machine can then simulate the actions of the original Turing machine by reading, writing, and manipulating these binary strings.

2.4 Conclusion

The Turing machine, despite its simple design, is a remarkably powerful model of computation. Its ability to simulate various computational processes, including multi-tape models and different alphabets, demonstrates its universality and its crucial role in understanding the theoretical limits of computation.

> "The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer."
> — Alan Turing

Church-Turing Thesis:
The Church-Turing Thesis posits that the intuitive concept of a computable function corresponds to what we now recognize as a partial recursive function. In simpler terms, it suggests that any computation that can be performed in the real world can be effectively simulated by a Turing machine.

3 Simulating x86 Assembly Instructions with a Turing Machine

To support the hypothesis that any computation done on a reasonable model of computation can be done by a Turing machine, we can check that for the basic RAM model. Converting a program in a high level language like C++ to a TM will be too cumbersome. Instead, we can consider a low level language like Assembly, and see how its basic instructions can be done by a TM.

MOV (Move Data)
This instruction copies the value from one register to another. A Turing machine simulates this by reading the value stored in the memory location corresponding to the source register, then writing this value into the location corresponding to the destination register. The machine moves the tape head to the position that represents the source register, reads the value, then moves the tape head to the position representing the destination register, and writes the value.

ADD (Addition)

This instruction adds a value to the contents of a register. A Turing machine would read the value stored in the tape cell representing the register. It would then go through a series of states to add the specified value. For binary arithmetic, this involves handling carries if necessary. Finally, the machine would write the new value back to the tape in the location representing the register.

CMP (Compare)

This instruction compares the values in two registers and determines their relationship (e.g., equal, greater, or less). The Turing machine moves to the tape cells corresponding to the two registers, reads both values, and enters different states based on whether the values are equal, greater, or less. Instead of setting flags, the machine would transition to states that represent the result of the comparison.

JMP (Jump)

This instruction causes the execution to jump to a different instruction located at a specified label. A Turing machine would simulate this by moving its tape head to the position corresponding to the start of the instruction labeled by the target. The machine transitions to a state where it begins reading and executing the instructions at that new position.

4 Concept of RAM and modern computers

We can defined a simple model for modern computer system as:

- (Countably) infinite RAM (i.e., bits of memory)
- Finite number of registers to store arbitrarily large integers
- A program counter (storing an arbitrarily large integer)
- Basic arithmetic, memory & control flow commands on above components

In this model, any location of the RAM can be accessed/manipulated in constant time. It turns out that this model of computation is also equivalent to Turing Machines (with polynomial overhead). To simulate the RAM model by TM, one can take one tape for every register. Most interesting part is to simulate random access of RAM.

5 Universal Turing Machine

A universal Turing machine (UTM) is a Turing machine capable of computing any computable sequence, i.e., simulating the work of any other Turing machine. Formally, a Universal Turing Machine U has the following properties:

- **Input:** The input to U consists of two parts:
  - A description of a Turing machine M, which includes the machine's states, alphabet, transition functions, and tape configuration.
  - An input for the machine M to process.
- **Simulation:** Given a description of a Turing machine M and an input for M, the Universal Turing Machine U simulates the behavior of M on that input. In other words, U emulates M by interpreting the description of M and performing the same computations that M would perform on the provided input.
- **Operational Structure:** The UTM operates by:
  - Reading the description of M and the input from its tape.
  - Simulating M by using its own set of states and transition functions to mimic the operation of M on the given input.

– Producing the same output as M would produce if M were run on the given input.

- **Universality:** The UTM is called "universal" because it can simulate any Turing machine. By loading a description of any Turing machine and its input, it can execute the corresponding computation.

Smallest Universal Turing Machine: It is known that there is Universal Turing Machine which has 15 states and 2 symbol alphabet.

6 Time Complexity

Let A be a Turing machine that always halts.
Let $t_A : \{0,1\}^* \to \mathbf{N}$ describe the number of steps that A takes on a given string. The time complexity of a Turing machine is defined as:

$$t_A(n) = \max_{x \in \{0,1\}^n} t_A(x)$$

7 Efficient Computation

DTime($t(n)$) (Deterministic Time) is the class of problems where there exists a halting Turing Machine that runs in time $O(t(n))$, where n is the size of the input.

The class **P**, i.e., the problems solvable in polynomial time, is defined as

$$\mathbf{P} = \bigcup_{c \geq 1} \text{DTime}(n^c)$$

8 Cobham-Edmond Thesis

Theorem 2.2. *A Turing Machine can compute any function computable by any reasonable model of computation with polynomial time overhead.*