Algorithm Analysis and Design

Tutorial 1

August 13, 2025

1 Course Logistics

1.1 Grading

- Quiz 1 – 7.5%
- Quiz 2 – 7.5%
- Midsem – 25%
- Endsem – 35%
- Project – 20%
- Miscellaneous – 5%

2 Tutorial

Note: This may contain typos. If there are any mistakes, trust the sources that are sent separately.

2.1 Brief History

1. Hilbert – Introduced decision problems (is a problem solvable?)
2. Church – Introduced lambda calculus - a branch of mathematics without any numerals. It consists of algorithms written in terms of its parameters
3. Godel – Introduced a μ-*recursive* function to describe problems

Finally, Alan Truing gave the idea of the Turing Machine. The Turing Machine (TM) has the following features:

1. A head (in essence, a pointer to a particular index / cell)
2. Can read or overwrite the input at the head
3. Has the ability to move the head to the left or the right

A simple notion to understand is to relate in the following way:

1. Turing Machine to a code / algorithm
2. The language to a problem, which is either solvable or unsolvable.

Church-Turing Thesis: Any algorithm that can be simulated in lambda calculus can also be represented as a TM or represented as *μ-recursive* functions.

2.2 Types of Languages

We will cover the following two types of languages:

1. **Decidable / Recursive Languages:** Output `yes` if the TM (Turing Machine) accepts and `no` if the TM rejects. It only accepts or rejects. **The TM always halts**.
2. **Recognizable / Recursively Enumerable Languages:** Output `yes` if the TM accepts. **Note:** Here it is not necessary for the TM to reject. It can keep on going on forever (*never halt*). Although, for all accept cases, the TM has to accept.

For example, $A_{DFA} = \{L \mid M(L) \text{ accepts } L\}$, which, in simpler terms, means the set of all languages that is accepted by a particular Deterministic Finite Automata, is **decidable** (and hence, **recognizable**).

2.3 Undecidability

There are some problems that cannot be solved by any computer (or a TM), or any algorithm. Such a language is undecidable.

2.3.1 Halting Problem

The task here is to write an algorithm that tells whether a function f will halt on input x or not.

Solution: We prove by contradiction. Assume there exists a program HALTS(f, x) that returns true if the function f will halt on input x, and false otherwise.
We can construct another function $D(f)$ that calls HALTS(f, f). If the return value of HALTS(f, f) is true, the function loops forever. Else, it stops immediately. If we run $D(D)$, the following happens:

1. If HALTS(D, D) is true, then the function $D(D)$ loops forever
2. If HALTS(D, D) is false, then the function halts immediately

Hence, we arrive at a contradiction and therefore HALTS doesnt exist.

2.4 Unrecognizable languages

Let M be a Turing Machine and $\langle M \rangle$ be a string representing M. Let

$$X = \{\langle M \rangle \mid M \textbf{ does not recognize } \langle M \rangle\}$$

We want to prove that no TM can recognize X.

For this, we create a matrix Z of $M_1, M_2, \cdots$ as the columns and $\langle M_1 \rangle, \langle M_2 \rangle, \cdots$ as the rows. Here $\langle M_i \rangle$ represents the TM M_i. We will then fill in the matrix cell $Z_{i,j}$ with `T` or `F` based on whether M_i recognizes $\langle M_j \rangle$ or not.

Now, we can create a new language by taking the reverse of all the diagonal elements. Hence, the new language L_{diag}, on $\langle M_i \rangle$ does the **opposite** of the result of M_i. Since all diagonal elements are reversed, there is no index k such that M_k that matches L_{diag}, and hence, no TM can recognize L_{diag}, which also fits the description of X here.

2.5 Reducibility

If you can *reduce* an unknown problem to a known problem (i.e., a solvable problem), or vice versa, then both problems are solvable.
In other words:

- If we can reduce an *unknown problem* to a *known solvable problem*, this proves the unknown problem is also solvable.
- We can also work backwards: if we can reduce a *known solvable problem* to an *unknown problem*, this will also prove the unknown problem is solvable.

2.5.1 RE-completeness

The Halting Problem is **RE-complete**, which means:

1. Is recursively enumerable (RE), and
2. It is the **hardest** problem in the class of recursively enumerable problems, meaning that every RE problem can be reduced to it.

3 Homework

3.1 Decidability and Recognizability

Prove the following questions:

1. Prove that $A_{DFA} = \{L \mid M(L) \text{ accepts } L\}$ is decidable
2. Is $A_{TM} = \{\langle M, w\rangle \mid M \text{ accepts } w\}$ decidable? Is it recognizable?
3. Denote the complement of a language L as $\overline{L}$, where $\overline{L}$ rejects whatever L accepts and vice-versa. If both L and $\overline{L}$ are recognizable, prove that L is decidable.
4. Is $L = \{\langle M\rangle \mid M \text{ accepts the empty string } \epsilon\}$ decidable? Is it recognizable?
5. Prove that $L = \{\langle M\rangle \mid M \text{ accepts } \langle M\rangle\}$ is Turing-recognizable, but $\overline{L}$ is not Turing-recognizable.

3.2 Reduction

Prove the following questions:

1. Reduce A_{TM} to the language X $= \{\langle M\rangle \mid L(M) \text{ is finite}\}$ to show that X is undecidable.
2. Reduce A_{TM} to language PAL $= \{\langle M\rangle \mid M \text{ accepts some palindrome}\}$ to show PAL is undecidable.
3. Reduce A_{TM} to language $L_{1000} = \{\langle M\rangle \mid M \text{ accepts a string of length } 1000\}$ to show L_{1000} is undecidable.

The solutions to these 8 questions will be released in the next tutorial.