Theory of Computation Cheatsheet

Edoardo Riggio

June 5, 2022

Theory of Computation - S.P. 2022
Computer Science
Università della Svizzera Italiana, Lugano

Contents

1 Introduction

The Turing Machine was invented by **Alan Turing** in 1936. A Turing Machine is a much more accurate model of a general-purpose computer than a PDA or FA. This type of machine can do anything that a real computer can.

The following are some characteristics of a Turing Machine:

- The input tape of the TM is infinite
- The input head of the TM can both read from and write to the tape
- The input head of the TM can move both left and right
- The RM has both accepting and rejecting states. Once it reaches such states, it accepts/rejects immediately – i.e., it does not need to get to the end of the input.

1.1 Formal Definition

A Turing Machine is defined as follows:

$$TM : (Q, \Sigma, G, \delta, q_0, q_{accept}, q_{reject})$$

Where Q is a **set of states**, Σ is the **input alphabet**, G is the **tape alphabet** – where $\Sigma \subset G$, δ is the **transition function**, q_0 is the **initial state**, q_{accept} is the **set of accepting states**, and q_{reject} is the **set of rejecting states**.

1.2 Configurations

1.2.1 Starting Configuration

The starting configuration of M on ω is $q_0\omega$, which indicates that the machine is in the start stating state q_0. Moreover, the head of the TM is in the leftmost position.

1.2.2 Accepting and Rejecting Configurations

The accepting and rejecting configurations are the halting configurations. They do not yield any further configuration.

1.3 Acceptance

A Turing Machine is said to accept an input string ω if a sequence of configurations $C_1, C_2, \ldots C_k$ exists where:

- C_1 is the starting configuration
- C_i yields C_{i+1}
- C_k is the accepting configuration

1.4 Turing-Recognizable Language

A collection of strings is called the language of a TM if the TM **accepts** it. A TM is said to **recognize** a language if it accepts all and only those strings in the language.

A language is said to be **Turing-recognizable** if some TM recognizes it. This means that a TM can accept, reject or loop.

1.5 Turing-decidable Language

While the possible outcomes of a TM are *accept*, *reject* and *loop*, a TM **decides** a language if it accepts all strings in the language, and reject all strings not in the language – i.e., it never loops.

1.6 Turing Machine Variants

1.6.1 Multitape Turing Machines

This is a TM that has k number of tapes. The transition function for this Turing Machine allows for reading, writing, and moving the heads on some or all tapes simultaneously.

Every multitape Turing Machine has an equivalent single-tape Turing Machine.

1.6.2 Non-Deterministic Turing Machines

The machine may proceed according to several different choices at any point in the computation. The resulting computation is a tree whose branches correspond to the various possibilities of the machine.

A non-deterministic Turing Machine **accepts** if some branch of the computation leads to an accepting state.

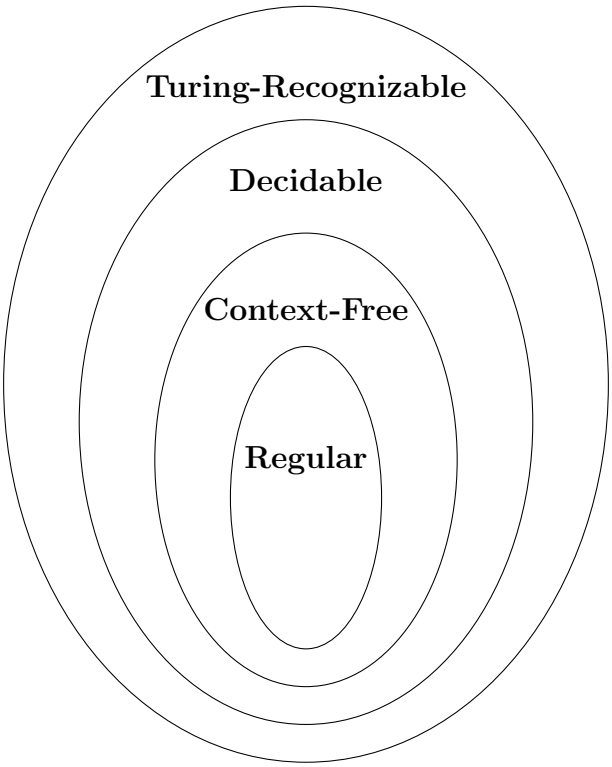
Every nondeterministic Turing Machine has an equivalent deterministic Turing Machine.

2 Computability Theory

We study unsolvability because knowing a problem cannot be solved helps us restate or simplify the problem.

2.1 Solvability

A problem is said to be solvable if we can find a Turing Machine that **decides** that problem.



2.2 Decidabile Problems

Showing that a language is decidable is the same as showing that the computational problem is decidable.

2.2.1 DFA Acceptance Problem

The language A_{DFA} is decidable. To prove it, we need to build a Turing Machine M that decides A_{DFA}.

M = On input $\langle B, \omega \rangle$, where B is a DFA and ω is a string:

1. Simulate B on ω
2. If the simulation ends in an accepting state **accept**, otherwise **reject**

2.2.2 NFA Acceptance Problem

The language A_{NFA} is decidable. To prove it, we need first to convert the NFA into a DFA and then apply the same procedure as the one in the A_{DFA} proof.

N = On input $\langle B, \omega \rangle$, where B is an NFA and ω is a string:

1. Convert B into an equivalent DFA C
2. Simulate C on ω
3. If the simulation ends in an accepting state **accept**, otherwise **reject**

2.2.3 DFA Emptiness Problem

The language E_{DFA} is decidable. We need to check if we can reach an accepting state by traveling the DFA to prove it.

S = On input $\langle A \rangle$, where A is a DFA:

1. Mark the starting state of the DFA
2. Continue marking the states of the DFA until no new state gets marked
3. If, once all states have been marked, no accept state was marked, **accept**, otherwise **reject**

2.2.4 DFA Equivalence Problem

The language EQ_DFA is decidable. We need to build an automaton that checks if both DFAs or neither accepts the string to prove it.

M = On input $\langle A, B \rangle$ where both A and B are DFAs:

1. Construct a DFA C with symmetric difference feature. If two automata recognize the same language, the newly constructed automaton accepts nothing when the languages are the same.
2. Feed the newly constructed DFA to M
3. Mark the starting state of C
4. Continue marking the states of the DFA until no new state gets marked
5. If, once all states have been marked, no accept state was marked, **accept**, otherwise **reject**

2.2.5 CFG Emptiness Problem

The language E_{CFG} is decidable. To prove it, we construct a Turing machine M.

M = On input $\langle G \rangle$, where G is a CFG:

1. Mark all terminal symbols in G
2. Repeat this operation until no new varibale gets marked
3. If, once all varibles have been marked, the start variable was not marked **accept**, otherwise **reject**