Theory of Computation Cheatsheet

Edoardo Riggio

June 7, 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.

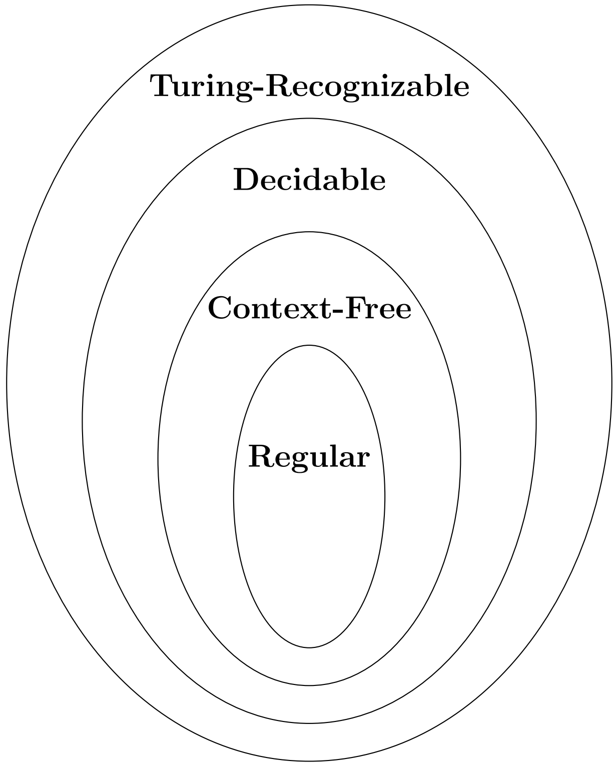
Every nondeterministic Turing Machine has an equivalent deterministic Turing Machine.

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 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**

2.2.5 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**

3 Undecidable Langauges

A language is said to be undecidable if no Turing Machine accepts the language and makes a decision for every input string.

3.1 Membership Problem

Given a Turing Machine M and a string ω, we need to determine whether M accepts ω.

Let L be a Turing-recognizable language, and let M_L be the Turing Machine that accepts L. We prove that L is also decidable by building a decider for L. This would mean that L is decidable. But, since L is chosen arbitrarily, this would mean that every Turing-recognizable language is decidable. Since we already know that decidable languages are a subset of Turing-recognizable languages, some languages are not decidable but still Turing-recognizable.

This will generate a contradiction, thus making the membership problem undecidable.

3.2 Correspondence

A function $f : A \to B$ is a **correspondence** if f is **one-to-one** and **onto**. A function is said to be **one-to-one**, if $\forall y \in B, \exists$ at most one $x \in A$, with $(x, y) \in R$. While a function is said to be **onto** if $\forall y \in B, \exists$ at least one $x \in A$, with $(x, y) \in R$.

Correspondences are essential because two sets are considered to have **the same number of elements** if there exists a one-to-one and onto correspondence between them.

3.3 Halting Problem

The halting problem occurs when a TM loops on its input while simulating ω on M. This will cause the TM not to halt, making it impossible to determine if ω is accepted or not.

To prove that this problem is unsolvable, we can use the **diagonalization method**.

Assuming that the halting problem is decidable, we construct a decider H' with another decider H inside it, which decides if a machine halts or not. If M halts on input ω, then make the decider H' loop forever, otherwise halt. If we pass H as an input to itself, then the machine will stop on input $\langle H \rangle$ if H terminates on $\langle H \rangle$ and loop forever; otherwise. we now have a contradiction; thus, the halting problem is **undecideble**.

4 Reductions

If a problem X is reduced to problem Y, this means that if we can solve Y, we can also solve X.

4.1 Computing Functions with Turing Machines

A function $f(w)$ has a **domain** and a **result region**. Given an element in the domain $w \in D$, by appying the function $f(w)$, we obtain the element in the resulting region $f(w) \in S$.

In order for this to work also on Turing Machines, we prefer to use **unary** representations of integers.

A function f is said to be **computable**, if there is a Turing Machine M such that on every input ω it halts with $f(\omega)$ on its tape.

4.2 Reduction

If a language A is **reduced** to language B ($A \leq_m B$), this means that there is a computable function f, such that for every ω the following double implication is proven:

$$\omega \in A \iff f(\omega) \in B$$

4.2.1 Reduction Theorem 1

If a language A is reduced to language B, then the language B is decidable. To prove this, we follow these steps:

1. Take the decider of B
2. Use the decider of B to build the decider for A
3. In the decider for A, first we compute the reduction $f(\omega)$, then we feed the result to the decider for B
4. The decider for B will either accept or reject
5. The decider for A now can accept or reject based on the outcome of B

4.2.2 Reduction Theorem 2

If a language A is reduced to B, and language A is undecidable, then language B is undecidable. To prove this, we follow these steps:

1. Take the decider for B
2. Use the decider for B to build the decider for A

3. In the decider for A, first we compute the reduction $f(\omega)$, then we feed the result to the decider for B
4. The decider for B will either accept or reject
5. The decider for A now can accept or reject based on the outcome of B

Since we know that A is undecidable, we have a contradiction. Thus, B also needs to be undecidable.

4.3 Examples

4.3.1 DFA Equality Problem

The language

$$EQUAL_{DFA} = \{\langle M_1, M_2 \rangle : M_1 \text{ and } M_2 \text{ accept the same languages}\}$$

can be reduced to $EMPTY_{DFA}$, which decides if a DFA has accepting states or not. $EMPTY_{DFA}$ is defined as follows:

$$EMPTY_{DFA} = \{\langle M \rangle : M \text{ accepts the empty language } \emptyset\}$$

Since $EMPTY_{DFA}$ gets only one DFA as input, we need to apply some transformation to $\langle M_1, M_2 \rangle$ in order to obtain M. This can be done as follows:

$$L(M) = (L_1 \cap \bar{L}_2) \cup (\bar{L}_1 \cap L_2)$$

This makes sure that the language of M is $\emptyset$. Thus

$$\langle M_1, M_2 \rangle \in EQUAL_{DFA} \iff \langle M \rangle \in EMPTY_{DFA}$$

4.3.2 State-Entry Problem

Given a Turing Machine M, a state q, and string w, we need to prove that the language $STATE_{TM}$ is undecidable. This language is represented as:

$$STATE_{TM} = \{\langle M, w, q \rangle : M \text{ enters state } q \text{ on input string } \omega\}$$

In order to prove that this language is undecidable, we need to reduce $HALT_TM$ to $STATE_TM$. The following steps are used:

1. Inside of the decider for $HALT_TM$, we compute the reduction to transform $\langle M, \omega \rangle$ into $\langle \hat{M}, q, \omega \rangle$
2. $\langle \hat{M}, q, w \rangle$ is passed to the decider for $STATE_{TM}$
3. If $STATE_{TM}$ accepts, then $HALT_{TM}$ **accepts**. Otherwise, it **rejects**

This will give us that if $STATE_{TM}$ is decidable, then $HALT_{TM}$ is decidable. But this is a contradiction since $HALT_{TM}$ is **undecidable**. Thus $STATE_{TM}$ is undecidable.

4.3.3 Blank-Tape Halting Problem

Given a Turing Machine M, we need to prove that the language $BLANK_{TM}$ is undecidable. This language is defined as:

$$BLANK_{TM} = \{\langle M \rangle : \ M \text{ halts when started on a blank tape}\}$$

In order to prove that the language above is undecidable, we reduce $HALT_{TM}$ to $BLANK_{TM}$. The steps are the following:

1. Inside of the decider for $HALT_{TM}$, we compute the reduction in order to transform $\langle M, w \rangle$ into $\langle \hat{M} \rangle$
2. $\langle \hat{M} \rangle$ is fed to the decider for $BLANK_{TM}$
3. If $BLANK_{TM}$ accepts, then $HALT_{TM}$ **accepts**. Otherwise, it **rejects**