Turing Machine and Decidability

Hsiang-Hsuan (Alison) Liu

1 Formal language framework

An *alphabet* is a nonempty finite set. The members of the alphabet are called *symbols*. We usually use Σ or Γ to denote an alphabet. The following are some examples of alphabets and their symbols.

- $\Sigma_1 = \{\mathtt{0}, \mathtt{1}\}$
- $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \cdots, \mathtt{z}\}$
- $\Gamma_1 = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$
- $\Gamma_2 = \{\mathtt{0}, \mathtt{1}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$

A *string* over an alphabet is a finite sequence of symbols from that alphabet. For example:

- $\mathtt{121}$ is a string over Γ_1
- $\mathtt{alien}$ is a string over Σ_2

We also denote the *empty string* (that is, a string with length 0) by ϵ.

A *language* is a set of strings. For example:

- $\{\mathtt{1}, \mathtt{01}, \mathtt{001}, \mathtt{0001}, \cdots, \mathtt{0^*1}\}$
- $\{\mathtt{0}, \mathtt{1}\}^*$
- $\{\mathtt{a}^k, \mathtt{b}^k | k \geq 0\}$
- $\{w \sharp w | w \in \{\mathtt{0}, \mathtt{1}\}^*\}$
- $\{\mathtt{2}, \mathtt{3}, \mathtt{5}, \mathtt{7}, \mathtt{11}, \cdots\} = \{\text{prime number}\}$
- $\{\langle G \rangle | G \text{ is a connected graph}\}$
- $\{\langle p \rangle | p \text{ is a polynomial with an integral root}\}$

Here we use the brackets $\langle\rangle$ to represent the *encoding* of the object in the brackets. For example, $\langle G \rangle$ is an encoding of the graph G.

Problem and language. The relation between languages and strings can be interpreted as problems and problem instances. When we are asked if a string w is in the language L, it is equivalent to asking if a problem instance w satisfied the problem definition L. For example, consider the language $L = \{\langle G \rangle | G \text{ is a connected graph}\}$ and a string $w = \langle G \rangle$. Asking if $w \in L$ is equivalent to asking if the instance graph G is connected.

2 Turing machine

The Turing machine is an abstract model of computation. It can compute everything that a real computer can compute.

A Turing machine is equipped with an unlimited *tape* as its memory, which initially contains only the input string and is blank everywhere else. There is a *read-write head* that can move around on the tape and read or write symbols. The read-write head is controlled by a finite-state *control*. Within the finite states of the control, there are two special states, *accept* and *reject*. Once the control state turns to accept (or reject), the Turing machine is in an *accepting configuration* (or *rejecting configuration*). Accepting and rejecting configurations are *halting configurations* and do not yield further configurations.

At any time, the *configuration* of the Turing machine is defined by the state of the control and what the read-write head reads. According to the current configuration, the Turing machine can decide what to do next. That is, it writes something in its current position on the tape, moves the read-write head to the left or to the right, and changes the state of the control.

Running on an input, a Turing machine may *accept*, *reject*, or enter an infinite loop.

Turing machines and algorithms. When we are talking about Turing machines, they can be seen as algorithms. Hence, a Turing machine M run on a string w is equivalent to running an algorithm M on the input instance w.

3 Turing-recognizable languages and Turing-decidable languages

Recall that running on an input, a Turing machine may *accept*, *reject*, or enter infinite loop.

Definition 1. *A language L is* Turing-recognizable *if there exists a Turing machine M that accepts every string $w \in L$ and does not accept any string $w \notin L$.*

Notice that for an input string $w \notin L$, where L is a recognizable language, the Turing machine may reject it or enter an infinite loop.

To show that a language L is recognizable, we have to:

- Design a Turing machine M
- Show that M accepts every string $w \in L$ and does not accept any string $w \notin L$.

Definition 2. *A language L is* Turing-decidable *if there exists a Turing machine M that accepts every string $w \in L$ and rejects every string $w \notin L$.*

Notice that the Turing machine M here halts on every input string. We call this kind of Turing machine a *decider*.

To show that a language L is decidable, we have to:

- Design a Turing machine M
- Show that M halts on every string correctly. That is, M accepts every string $w \in L$ and rejects every string $w \notin L$.

4 Undecidable languages: languages that are not decidable

Definition 3. *A language L is* undecidable *if for all Turing machines M, there exists some string $w \in L$ that M does not accept or there exists some string $w \notin L$ that M does not reject. That is, for any Turing machine, it may enter infinite loops on some input $w \in L$ or $w \notin L$.*

Similarly, we can define the languages that are unrecognizable:

Definition 4. *A language L is* unrecognizable *if, for all Turing machines M, there exists some string $w \in L$ that M does not accept.*

Surprisingly, the undecidable languages and unrecognizable languages are not only hypothetical but do exist. Consider the problem $\mathrm{A_{TM}}$ defined as the following:

$$\mathrm{A_{TM}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ accepts the input string } w\}$$

The problem $\mathrm{A_{TM}}$ can be interpreted as testing how an algorithm (M) reacts on input w. Surprisingly, it is undecidable.

Theorem 1. *A_{TM} is undecidable.*

Proof. Assume, on the contrary, that $\mathrm{A_{TM}}$ is decidable. There exists a Turing machine H that decides $\mathrm{A_{TM}}$ on any input $\langle M, w\rangle$. That is, H accepts if M accepts on input w and H rejects if M rejects or enters a loop on input w.

Now we design another Turing machine P using H as a subroutine and output the opposite of what H outputs.

$P =$ "On input $\langle M\rangle$:

1. Run H on input $\langle M, \langle M\rangle\rangle$.

2. *Accept* if H rejects $\langle M, \langle M\rangle\rangle$. *Reject* if H accepts $\langle M, \langle M\rangle\rangle$."

In other words, P accepts $\langle M\rangle$ if M rejects $\langle M\rangle$ and P rejects $\langle M\rangle$ if M accepts $\langle M\rangle$.

Now, what happens if we run P on input $\langle P\rangle$? By the same arguments, P accepts $\langle P\rangle$ if P rejects $\langle P\rangle$ and P rejects $\langle P\rangle$ if P accepts $\langle P\rangle$. That is a contradiction. Hence, neither Turing machine P nor Turing machine H can exist. That completes the proof that $\mathrm{A_{TM}}$ is undecidable. □

Notice that although $\mathrm{A_{TM}}$ is undecidable, it is recognizable. The fact that $\mathrm{A_{TM}}$ is undecidable means that it is impossible to write a program to detect how a program behaves. It may work for some special (small) programs but does not work for *any* cases.

Using the fact that $\mathrm{A_{TM}}$ is undecidable, we can show that the *halting problem* is undecidable. That is, it is impossible to write a program to detect if any program will stop.

Theorem 2. *$HALT_{TM}$ is undecidable.*

Proof. See document 083 *Exercise Turing Decidable Languages and Undecidable Languages*, question 2. □

5 Unrecognizable languages

According to the definition, for an undecidable language $w \in L$, a Turing machine may enter a loop on an input string $w \in L$ or $w \notin L$. If there is a Turing machine that only enters a loop for input strings $w \notin L$, the language L is recognizable (although it is undecidable). Hence, unrecognizable languages are all undecidable.

Is there any language that is unrecognizable? The answer is yes. By definition, for an *unrecognizable* language L, any Turing machine enters a loop for some input string $w \in L$. The undecidable languages do not only exist conceptually. There is a solid example of undecidable language:

$$\overline{\mathrm{A_{TM}}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ does not accept the input string } w\}.$$

The language $\overline{\mathrm{A_{TM}}}$ is the complement of language $\mathrm{A_{TM}}$.

Theorem 3. *$\overline{A_{TM}}$ is unrecognizable.*

Proof. We prove by contradiction. Assume that $\overline{\mathrm{A_{TM}}}$ is recognizable. There exists a Turing machine R that recognizes $\overline{\mathrm{A_{TM}}}$. If R exists, we can use R to construct another Turing machine H that decides $\mathrm{A_{TM}}$. That eventually contradicts to the fact that $\mathrm{A_{TM}}$ is undecidable.

First, recall that $\mathrm{A_{TM}}$ is recognizable. There exists a Turing machine Q that accepts $\langle M, w\rangle$ if M accepts w. Now, we construct Turing machine H:

$H =$ "On input $\langle M, w\rangle$:

1. Run Q and R on input w in parallel.

2. If Q accepts $\langle M, w\rangle$, *accept*. If R accepts $\langle M, w\rangle$, *reject*.

Here, running two Turing machines in parallel means that H has two tapes, one for simulating Q and the other one for simulating R until one of Q or$R$ accepts.

Now we show that H decides $\mathrm{A_{TM}}$. By the definition of $\overline{\mathrm{A_{TM}}}$, every string $\langle M, w\rangle$ is either in $\mathrm{A_{TM}}$ or $\overline{\mathrm{A_{TM}}}$. Hence, either R or Q must accept $\langle M, w\rangle$. Therefore, H always halts on any input $\langle M, w\rangle$. Furthermore, H accepts all strings in $\mathrm{A_{TM}}$ and rejects all strings in $\overline{\mathrm{A_{TM}}}$ (that is, strings not in $\mathrm{A_{TM}}$). So H is a decider for $\mathrm{A_{TM}}$. It contradicts the fact that $\mathrm{A_{TM}}$ is undecidable. Therefore, R does not exist, and it completes the proof that $\overline{\mathrm{A_{TM}}}$ is unrecognizable. □