Theory of Computation Notes: Turing Reductions and Undecidability

Arjun Chandrasekhar

We used diagonalization to prove that the halting problem was undecidable. We showed that if HALT were decidable, we could create a TM that literally contradicted itself. Now, we will see how we can use this result to identify other undecidable problems. We will do this using the powerful technique of *reductions*.

1 Reducibility

First we will give an informal explanation of the concept of reducibility. Let's say I want to earn some money. If I can get a job, I can get some money. Therefore the problem of earning money can be reduced to the problem of getting a job.

Now how do I get a job? Let's pretend that if I can just get to Los Angeles, I will easily find a job. Then the problem of getting a job can be reduced to the problem of getting to Los Angeles. In fact, the problem of earning money can be reduced to the problem of getting to Los Angeles.

Now let's say that if I have a map to Los Angeles it is straightforward to get to LA. Then the problem of getting to Los Angeles can be reduced to finding a map to Los Angeles. Also, the problems of earning money and getting a job can be reduced to the problem of finding a map to Los Angeles.

1.1 Reducibility and Impossibility

If you have read the Harry Potter series, you may remember that initially Harry couldn't get to Hogsmead; however once he found the Marauder's Map he was able to get to Hogsmead. We thus say that the problem of getting to Hogsmead can be reduced to the problem of obtaining the Marauder's Map.

Note that when we make this statement, we are <u>not</u> saying that either obtaining the Marauder's Map or getting to Hogsmead is possible. We are simply stating that if one is possible, so is the other.

And what happens if one of those problems is impossible? Well, then we can actually say the other problem is also impossible!

Let's say I convince you that it is impossible to get to Hogsmead (because it's not a real place). This means that it is also impossible to obtain the Marauder's Map. Why? Well, if I could obtain the Marauder's map, I could get to Hogsmead, but we established that this is impossible. Thus, obtaining the Marauder's Map would lead to a logical contradiction, and we conclude that it can't be obtained.

2 Turing reducibility

Definition 2.1. Let A and B be formal languages. Suppose that if there were a machine M_B which could decide B, then we could construct a machine M_A which uses M_B to decide A. Then we would conclude that A is **Turing-reducible**, or **reducible** to B. De denote this by saying $A \leq_T B$.

Note that when we say $A \leq_T B$, we are not saying that A or B is decidable. We are simply stating that *if* B is decidable, then A will also be decidable.

Example 2.1. Consider the following languages:

$$A = \{n | n \in \mathbb{N}, n \text{ is even}\}$$
$$B = \{n | n \in \mathbb{N}, n \text{ is odd}\}$$

We will show that $A \leq_T B$. First, note that $n \in A \Leftrightarrow n+1 \in B$. Now, let's assume we have a machine M_B that decides B; given an integer n, M_B will tell us if it is odd. We don't know how M_B does the job, we only know that it exists.

We will now construct a machine M_A that decides A as follows:

1. M_A takes n as input
2. Make sure n is a valid integer
3. Run M_B on $n+1$
4. If M_B accepts $n+1$, accept n
   If M_B rejects $n+1$, reject n

M_A accepts $n \Leftrightarrow M_B$ accepts $n+1 \Leftrightarrow n+1$ is odd $\Leftrightarrow n$ is even

Example 2.2. Consider the following two languages

$$A = \{\langle D \rangle | D \text{ is a DFA, } L(D) = \Sigma^*\}$$
$$B = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

We will prove that $A \leq_T B$. Let's assume we have a machine M_B that decides B. This means M_B can take two DFA descriptions and decide whether the two DFAs are equivalent.

We design a machine M_A to decide A as follows:

1. M_A takes $\langle D \rangle$ as input
2. Create a DFA D_2 which recognizes Σ^*
3. Run M_B on $\langle D, D_2 \rangle$
4. If M_B accepts $\langle D, D_2 \rangle$, accept $\langle D \rangle$
   If M_B rejects $\langle D, D_2 \rangle$, reject $\langle D \rangle$

M_A accepts $\langle D \rangle \Leftrightarrow M_B$ accepts $\langle D, D_2 \rangle \Leftrightarrow L(D) = L(D_2) = \Sigma^*$

3 Using Reducibility to Prove Undecidability

3.1 A_{TM}

Definition 3.1. The following language represents one of the most famous problems in computability theory

$$A_{TM} = \{\langle M, w \rangle | w \in L(M)\}$$

For this language we receive two inputs: a description of a machine M, and a string w. We accept $\langle M, w \rangle$ if M halts and accepts n w; we reject $\langle M, w \rangle$ if M either rejects or loops on w.

Lemma 3.1. A_{TM} is Turing-recognizable.

Proof. The following machine A recognizes A_{TM}

1. A receives $\langle M, w \rangle$ as input

2. Run M on w

3. If M ever accepts w, accept $\langle M, w\rangle$
   If M ever rejects w, reject $\langle M, w\rangle$
   If M loops forever then A will loop forever

If $\langle M, w\rangle \in \mathrm{A_{TM}}$ then eventually M will halt and accept w, so A will eventually accept $\langle M, w\rangle$. If $\langle M, w\rangle \notin \mathrm{A_{TM}}$ then A will never accept $\langle M, w\rangle$. □

Remark. A does not *decide* $\mathrm{A_{TM}}$, because A might loop if M loops on w.

Theorem 3.1. $\mathrm{A_{TM}} \leq_T$ HALT

Proof Idea. If we could decide HALT, this would help us immensely. Before running M on w, we can check whether M will go into an infinte loop. If it will, we reject $\langle M, w\rangle$ immediately. If there is no risk of looping, we can safely run M on w. If M accepts w, we accept $\langle M, w\rangle$; if M rejects w we reject $\langle M, w\rangle$. There is no risk of looping, because we never run M unless it's safe.

Proof. Suppose we have a machine M_H which decides HALT. Given $\langle M, w\rangle$, M_H will tell us (with 100% accuracy) whether M halts or loops on w. We will then construct a machine M_A to decide $\mathrm{A_{TM}}$ as follows:

1. M_A takes $\langle M, w\rangle$ as input

2. Run M_H on $\langle M, w\rangle$

3. If M_H rejects $\langle M, w\rangle$, immediately reject $\langle M, w\rangle$

4. If M_H accepts $\langle M, w\rangle$, run M on w. By this point we know that M will safely halt.

5. If M accepts w, accept $\langle M, w\rangle$
   If M rejects w, reject $\langle M, w\rangle$

If M loops on w then $w \notin L(M)$; our machine M_A will detect this and reject $\langle M, w\rangle$. Otherwise, M_A will run M on w without risk of looping, and accept if and only if M accepts w, which happens if and only if $w \in L(M)$. □

Remark. If we could decide HALT, we could decide $\mathrm{A_{TM}}$. But we can't decide HALT; so ultimately this doesn't actually prove that $\mathrm{A_{TM}}$ is decidable.

3.2 Decidability of $\mathrm{A_{TM}}$

So is $\mathrm{A_{TM}}$ decidable? Can we design a more clever machine to analyze the source code of M?

The answer is a resounding no! While we could prove this using diagonalization, we will see how reducibility is a powerful technique for establishing undecidability.

Theorem 3.2. HALT $\leq_T \mathrm{A_{TM}}$

Proof Idea. We show that if we had a machine to decide $\mathrm{A_{TM}}$ we could decide HALT. Given $\langle M, w\rangle$, we will modify M so that it accepts whenever it halts; it never rejects. This way, if we can figure out whether M accepts w, we will know whether it halts or not, because rejecting is not a possibility.

Proof. Suppose we have a machine M_A which decides $\mathrm{A_{TM}}$. We will construct a machine M_H that decides HALT as follows:

1. M_H takes $\langle M, w\rangle$ as input

2. Creates a machine P that does the following:

   (a) P takes a string s as input

(b) P simulates M on s
M is a hard-coded constant

(c) If M halts (by either accepting or rejecting s), P will accept s
If M loops forever on s then so will P

3. Run M_A on $\langle P, w\rangle$

4. If M_A accepts $\langle P, w\rangle$, accept
If M_A rejects $\langle P, w\rangle$, reject

If M halts on w. then P will accept w, because it accepts as long as M halts. If M loops on w then P will also loop and not accept. Thus $\langle M, w\rangle \in \text{HALT} \Leftrightarrow \langle P, w\rangle \in \text{A}_{\text{TM}}$. This means that because we can test whether $\langle P, w\rangle \in \text{A}_{\text{TM}}$, we can use this to test whether $\langle M, w\rangle \in \text{HALT}$. Thus, M_H decides HALT. □

Corollary 3.1. A_{TM} is undecidable.

Proof. AFSOC A_{TM} is decidable. Then under this assumption, theorem 3.2 is decidable. However, this is a contradiction. Thus, A_{TM} is cannot be decidable. □

4 Exercises

1. Prove that the language $\text{E}_{\text{TM}} = \{\langle M\rangle | M \text{ is a TM, } L(M) = \emptyset\}$ is undecidable

2. Prove that the language $\text{ALL}_{\text{TM}} = \{\langle M\rangle | M \text{ is a TM, } L(M) = \Sigma^*\}$ is undecidable

3. Prove that the language $\text{EQ}_{\text{TM}} = \{\langle M_1, M_2\rangle | L(M_1) = L(M_2)\}$ is undecidable. You may wish to reduce from one of the previous three languages.

4. Prove that the language $\text{SUB}_{\text{TM}} = \{\langle M_1, M_2\rangle | M_1, M_2 \text{ are TMs, } L(M_1) \subseteq L(M_2)\}$ is undecidable

- Prove this by reducing from E_{TM}
- Prove this by reducing from ALL_{TM}
- Prove this by reducing from EQ_{TM}