Theory of Computation Notes: The Halting Problem

Arjun Chandrasekhar

Using the concept of countability and diagonalization we have shown that there are strictly more languages than there are Turing machines. This means not every language can be recognized by a Turing machine. But can we actually describe such languages? What might an unrecognizable language look like?

1 HALT is Undecidable

To construct an unrecognizable language, we will first need to construct an undecidable one. We will prove that the following language is undecidable.

Definition 1.1. The infamous **halting problem** is the following language:

$$\text{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$

Every element of this language is a combination of a machine M and a string w. We have $\langle M, w\rangle \in \text{HALT}$ if M halts on w, and $\langle M, w\rangle \notin \text{HALT}$ if M loops on w.

Theorem 1.1. HALT is undecidable.

Proof. To show that it is undecidable, we will assume for sake of contradiction (AFSOC) that it is decidable. We will then show that under this assumption, it is possible to construct a strange machine that contradicts itself. This will lead us to conclude that our original assumption was incorrect.

During the course of this proof, we will often refer to descriptions of machines. Here, if M is a machine then $\langle M\rangle$ is the *description* of M. Note that $\langle M\rangle$ is a string. You can think of the difference between M and $\langle M\rangle$ as the difference between a binary executable and a source code file.

Now we return to the proof. Formally, AFSOC some machine H decides HALT. This means that on input $\langle M, w\rangle$

- If M halts on w then H accepts $\langle M, w\rangle$
- If M loops on w then H rejects $\langle M, w\rangle$

We will construct a machine S that does the following:

1. S takes a machine description $\langle M\rangle$ as input
2. S runs H on input $\langle M, \langle M\rangle\rangle$
3. S "does the opposite" of H More specifically:
   (a) If H accepts $\langle M, \langle M\rangle\rangle$ then S goes into an infinite loop.
   (b) If A rejects $\langle M, \langle M\rangle\rangle$ then S immediately halts.

Because we have assumed that H exists and is a legitimate Turing machine, then clearly our machine S is just as legitimate.

To see where things go off the rails, let's see what happens when S receives its own description as $\langle S\rangle$ as input. If you think this is strange, convince yourself that you could do the same thing in your favorite

programming language. Suppose you have a program that takes a string as a command line argument; there is nothing stopping you from creating a long string out of your program's source code, and then passing that string as the command line argument to the program. Or if that makes you uncomfortable, you can verify that in any programming language you can have your program read its own source code file and make a sting out of its own source code.

So, what happens when S receives $\langle S \rangle$ as its input?

1. S runs H on $\langle S, \langle S \rangle \rangle$
2. S "does the opposite" of H
   (a) If H accepts $\langle S, \langle S \rangle \rangle$ then S goes into an infinite loop. But this is a contradiction! When H accepts $\langle S, \langle S \rangle \rangle$, it is saying that S should halt on $\langle S \rangle$. But this causes S to loop on $\langle S \rangle$ - it is contradicting what it is supposed to do!
   (b) If H rejects $\langle S, \langle S \rangle \rangle$ then S halts on $\langle S \rangle$. But this is a contradiction! When H rejects $\langle S, \langle S \rangle \rangle$, it is saying that S should loop on $\langle S \rangle$. But this causes S to halt on $\langle S \rangle$ - it is contradicting what it is supposed to do!

Either way, our machine S contradicts what it is supposed to do. This is obviously not logically possible, so we conclude that our machine H is faulty, and it is not actually deciding A_{TM}.

We can interpret this proof as a form of diagonalization (Figure 1).

□

1.1 HALT is Recognizable

We have shown that HALT is undecidable, meaning no machine will accept every positive instance and reject every negative instance. But what if we only need to accept the positive instances and not worry about the negative instances?

Lemma 1.1. HALT is Turing-recognizable.

Proof. We design a machine H that recognizes HALT as follows:

1. H takes $\langle M, w \rangle$ as input
2. H runs M on w
   (a) If M halts on w then H accepts $\langle M, w \rangle$
   (b) If M loops forever then H loops forever.

We will give a painfully detailed proof of why I recognizes A_{TM}.

- Suppose $\langle M, w \rangle \in$ HALT. This means M halts on w. In this case, when H runs M on w, eventually it will halt, so H will accept $\langle M, w \rangle$.
- Suppose $\langle M, w \rangle \notin$ HALT. This means M loops on w. When H runs M on w, M will loop forever, so H will not accept $\langle M, w \rangle$.

□

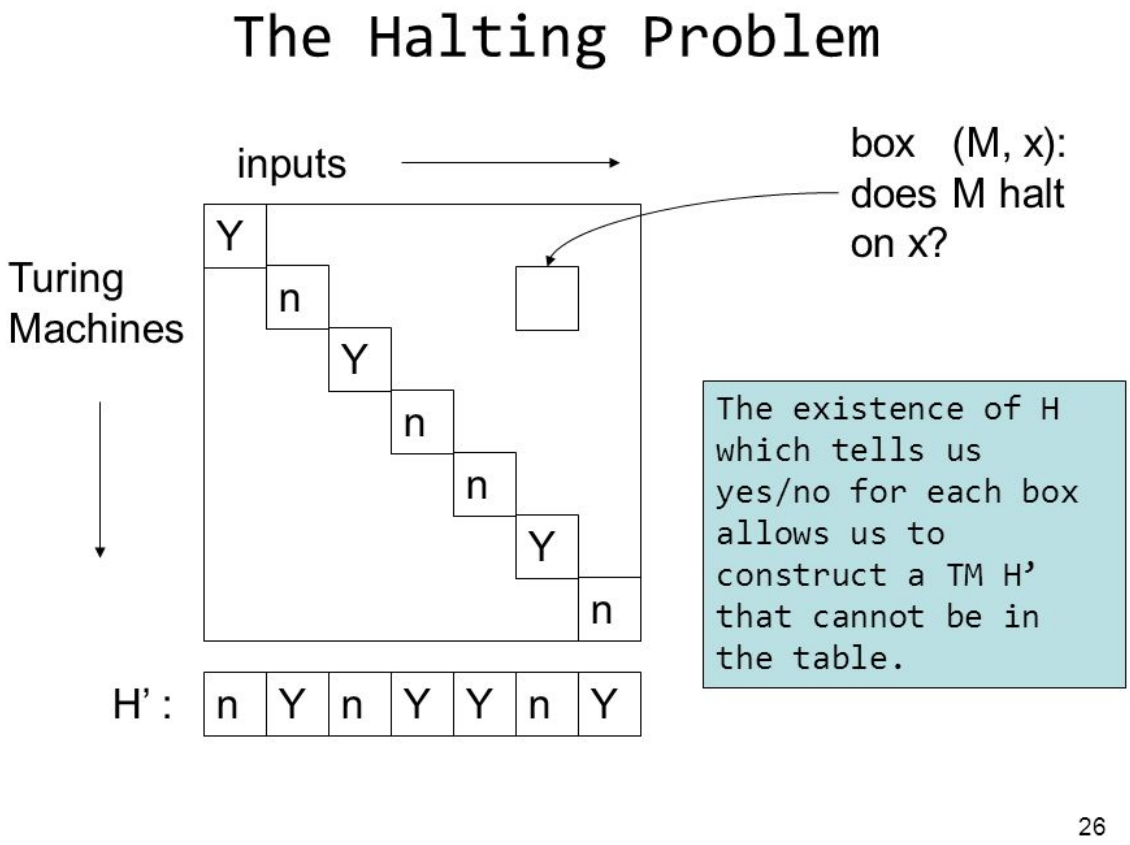

Figure 1: **Diagonalizing the** HALT**:** We assume we can determine how each machine behaves on each input. This allows us to construct a machine that contradicts every machine, including itself.

2 Co-recognizable Languages

Definition 2.1. We say a language L is **co-Turing recognizable** if its complement $\overline{L}$ is recognizable. Essentially we can check if a string w is not in L, but we cannot check if a string $w \in L$.

If L is co-Turing recognizable, then $\underline{L}$ is recognizable, so there is a machine $\underline{M}$ that does the following:

- M accepts $w \Leftrightarrow w \in \overline{L} \Leftrightarrow w \notin L$
- M rejects or loops on $w \Leftrightarrow w \notin \overline{L} \Leftrightarrow w \in L$

Remark.

- We often simply say L is **co-recognizable**
- We proved that a language is recognizable if and only it is recursively enumerable (RE); as a result, we used recognizable and RE interchangeably. Similarly, we will often use co-recognizable and co-RE interchangeably.
- In the past we have used L^c to denote the complement of L. In the rest of this document, and in future notes, we will use $\overline{L}$ to denote the complement of L.

Theorem 2.1. A language L is decidable if and only if L is both recognizable and co-recognizable.

We will prove this theorem in two parts.

Lemma 2.1. ($\Rightarrow$) If L is decidable then L is both recognizable and co-recognizable

Proof. Suppose some machine D decides L. Then we will show that Both L and $\overline{L}$ are recognizable.

- **L is recognizable:** We simply note that D actually recognizes L! If $w \in L$ then D halts and accepts w; if $w \notin L$ then D does not accept w - in fact, D will halt and reject w.
- **$\overline{L}$ is recognizable:** To recognize $\overline{L}$ we construct a machine $\overline{D}$ that runs D and then does the opposite, i.e. if D accepts then $\overline{D}$ rejects and if D rejects then $\overline{D}$ accepts. We will show that $\overline{D}$ recognizes $\overline{L}$.

  If $w \in \overline{L}$ then $w \notin L$. This means D halts and rejects w, so $\overline{D}$ halts and accepts w. If $w \notin \overline{L}$ then $w \in L$. This means D halts and accepts w, so $\overline{D}$ will halt and reject.

□

Lemma 2.2. ($\Leftarrow$) If L is both recognizable and co-recognizable then it is decidable.

Proof. Assume we have a machine M which recognizes L, and we have a machine $\overline{M}$ which recognizes $\overline{L}$. We will construct a machine D that decides L.

On input w, D does the following:

1. Run M and $\overline{M}$ in parallel
   (a) If M ever accepts w, then D accepts w
   (b) If $\overline{M}$ ever accepts w, then D rejects w

Now we will argue that our machine is correct. We must show that M always halts, and it accepts all strings in L, and it rejects all strings not in L.

- Suppose $w \in L$. Then eventually M halts and accepts w, so eventually D will halt and accept w. Note that $\overline{M}$ may loop forever on w but that is OK because we run the two machines in parallel.
- Suppose $w \notin L$. Then eventually $\overline{M}$ halts and accepts w, so eventually D will halt and reject w. Note that M may loop forever on w but that is OK because we run the two machines in parallel.

□

3 $\overline{\text{HALT}}$ is Unrecognizable

We are now ready to define a language that is not Turing-recognizable.

Definition 3.1.

$$\overline{\text{HALT}} = \{\langle M, w\rangle | M \text{ loops on } w\}$$

This is the complement of HALT. Note that because HALT is recognizable, by definition $\overline{\text{HALT}}$ is co-recognizable.

Theorem 3.1. $\overline{\text{HALT}}$ is not Turing-recognizable.

Proof. AFSOC $\overline{\text{HALT}}$ is Turing-recognizable. This means that HALT is co-recognizable. By lemma 1.1, we also know that HALT is recognizable. Thus, by theorem 2.1, HALT is a decidable language.

But this is a contradiction! Theorem 1.1 established that HALT is undecidable.

Thus, we conclude that $\overline{\text{HALT}}$ is not Turing-recognizable. □