1 Undecidability (4.2)

We will now talk about several computationally unsolvable problems, along with proving that such problems are actually unsolvable.

1.1 Computational Problems

We begin by going over some examples of computational problems.

1.1.1 Simple Problems: DFAs

Recall the following examples from the previous notes.

- Check whether a string is accepted by a DFA.

$$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA over } \Sigma, w \in \Sigma^*, w \in L(B)\}$$

- Check whether the language of a DFA is empty.

$$E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

- Check whether the languages of two DFAs are equal.

$$EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

These are **all** decidable; we did this by constructing algorithms that could solve them.

1.1.2 More Complex Problems: Pushdown Automaton

Consider the following examples.

- Check whether a string is accepted by a PDA.

$$A_{\text{PDA}} = \{\langle B, w\rangle \mid B \text{ is a PDA over } \Sigma, w \in \Sigma^*, w \in L(B)\}$$

- Check whether the language of a PDA is empty.

$$E_{\text{PDA}} = \{\langle A\rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$$

- Check whether the languages of two PDAs are equal.

$$EQ_{\text{PDA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$$

Some of these are decidable, and some are not. We note that, because a PDA has a stack which can be infinite in size, we cannot just simulate a PDA with the infinite because there can be infinitely many options.

1.1.3 Complex Problems: Turing Machine

Consider the following examples.

- Check whether a string is accepted by a TM.

$$A_{\text{TM}} = \{\langle B, w\rangle \mid B \text{ is a TM over } \Sigma, w \in \Sigma^*, w \in L(B)\}$$

- Check whether the language of a TM is empty.

$$E_{\text{TM}} = \{\langle A\rangle \mid A \text{ is a TM and } L(A) = \emptyset\}$$

- Check whether the languages of two TMs are equal.

$$EQ_{\text{TM}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are TMs and } L(A) = L(B)\}$$

These are all undecidable. There are no algorithms which can solve A_{TM}, for example.

1.2 A_{TM}: An Algorithm

We will now consider an algorithm for solving A_{TM}. For an input $\langle M, w\rangle$, define the Turing machine N as follows.

1. Simulate M on input w.
2. If M ever enters its accept state, then accept; if M ever enters its reject state, then reject.

Then, we can define

$$L(N) = A_{\text{TM}}$$

and say that N is a **recognizer** for A_{TM}. However, N is not a *decider* for A_{TM} because it's possible that the Turing machine M can *loop* (i.e. not accept or reject); if M loops, then so will N.

1.3 Diagonalization Method

We now talk about the diagonalization method as a way to show that A_{TM} is undecidable.

1.3.1 Proof

Proof. Assume towards a contradiction that A_{TM} is decidable. Then, call $M_{A_{\text{TM}}}$ the decider for A_{TM}. Then, for every Turing machine M and every string w:

- The computation of $M_{A_{\text{TM}}}$ on $\langle M, w\rangle$ halts and accepts if w is in $L(M)$.
- The computation of $M_{A_{\text{TM}}}$ on $\langle M, w\rangle$ halts and rejects if w is not in $L(M)$.

For some input $\langle M\rangle$, we can define the Turing machine D as follows.

- Run $M_{A_{\text{TM}}}$ on $\langle M, \langle M\rangle\rangle$.
- If $M_{A_{\text{TM}}}$ accepts, then D rejects. Likewise, if $M_{A_{\text{TM}}}$ rejects, then D accepts.

Now, suppose we run D on input $\langle D\rangle$. Because D is a decider, then either the computation halts and accepts, or the computation halts and rejects. Now:

- If $D(\langle D\rangle)$ accepts, then $M_{A_{\text{TM}}}$ with $\langle D, \langle D\rangle\rangle$ rejects. So, $\langle D, \langle D\rangle\rangle \notin A_{\text{TM}}$ and so $D(\langle D\rangle)$ rejects.
- If $D(\langle D\rangle)$ rejects, then $M_{A_{\text{TM}}}$ with $\langle D, \langle D\rangle\rangle$ accepts. So, $\langle D, \langle D\rangle\rangle \in A_{\text{TM}}$ and so $D(\langle D\rangle)$ accepts.

This is a contradiction since D cannot accept its own input and cannot reject its own input and cannot halt. □

1.3.2 Another Way of Seeing It

Recall that we can list all of the Turing machines since they are countable:

$$M_1, M_2, M_3, \ldots$$

Suppose we also have a list of *distinct* inputs, call them:

$$w_1, w_2, w_3, \ldots$$

We can create this two-dimensional table where $A_{i,j}$ is defined to be the cell $[i, j]$, which is the result of running M_i on input w_j; that is, the cell will either be 1 if $w_J \in L(M_i)$ and 0 if not.

	w_1	w_2	w_3	$\ldots$
M_1	1	0	1	
M_2	0	0	1	
M_3				
$\vdots$				

Note that we made up these entries. Suppose we take the opposite of the diagonals:

	w_1	w_2	w_3	...
M_1	**0**	0	1	
M_2	0	**1**	1	
M_3				
$\vdots$				

We claim that there is no Turing machine M such that $M(w_j)$ accepts if and only if $A_{j,j} = 0$ for all j. So, if M_1 accepted w_1, then this Turing machine will reject w_1; if M_2 rejected w_2, then this Turing machine will accept w_2.

Proof. We know that $M \neq M_i$ because $M(w_i) \neq M_i(w_i)$. □

The proof in the previous section is simply what we have here when $w_i = \langle M_i \rangle$.

1.4 Summary of this Algorithm

So, we have shown that this language is not decidable. Note that this language is also recognizable. As a general fact, a language is decidable if and only if it and its complement are both recognizable. Additionally, the complement of A_{TM} is unrecognizable.

1.5 The Halting Problem

Consider the Halting problem, defined by

$$\text{HALT} = \{(\langle M \rangle, w) \mid M(w) \text{ hals}\}$$

We can actually make use of the fact that A_{TM} is undecidable to show that HALT_{TM} is undecidable.

Consider the code:

```
f(n):
    while n >= 1:
        if n is even:
            n = n / 2
        else:
            n = 3n + 1
```

Is $f(n)$ halt for all inputs? We don't know.