Chapter 1

Decidability

Chapter 2

Reduction

Reduction is the primary method by which we prove that problems are computationally unsolvable. It converts a problem to a simpler problem in such a way that the simpler problem can be used to solve the original one.

As we've shown in the previous chapter, A_{TM} is an undecidable language. We will reduce problems that seem complex to the simpler A_{TM} problem.

Technique 2.0.1 ▸ Reduction

To prove that a problem P is undecidable by reduction:

1. Find a problem Q known to be undecidable.
2. Assume P is decidable by a TM, say M_P.
3. Use M_P to construct a TM M_Q that solves Q: encode every instance q of problem Q as an instance q_P of problem P.

Chapter 3

Complexity

Complexity in terms of Turing Machines deals with the number of steps the machine takes to finish computation. In general, the running time of an algorithm is a function of the length of the string that represents the input.

3.1 Asymptotic Analysis

Definition 3.1.1 ▸ Big-O, Little-O

Intuitively, a function f is (in) ***Big O*** of g (written $f \in \mathcal{O}(g)$) if f grows no faster than g. A function f is (in) ***Little O*** of g (written $f \in o(g)$) if f grows strictly slower than g.

Formally, let $g : \mathbb{N} \to \mathbb{R}$ be a function. We define ***Big-O*** of g as:

$$\mathcal{O}(g) \coloneqq \{f : \mathbb{N} \to \mathbb{R} \mid \exists(k > 0)\exists(a \in \mathbb{N})(x > a \implies 0 \leq f(x) \leq g(x))\}$$

We define ***Little-O*** of g as:

$$o(g) \coloneqq \{f : \mathbb{N} \to \mathbb{R} \mid \forall(k > 0)\exists(a \in \mathbb{N})(x > a \implies 0 \leq f(x) \leq g(x))\}$$

3.2 P, NP, NP-Hard, NP-Complete

In complexity theory, we have four distinct sets of problems:

- P (polynomial time): the set of decision problems that can be solved in polynomial time

- *NP* (non-deterministic polynomial time): the set of decision problems that can be verified in polynomial time
- *NP*-Hard: the set of decision problems that are at least as hard as the hardest problems in *NP*. More formally, a problem is *NP*-hard if every *NP* problem can be reduced to the *NP*-hard problem in polynomial time. So if we could solve an *NP*-hard problem in polynomial time, then we can solve any *NP* problem in polynomial time.
- *NP*-complete: the set of decision problems that are *NP* and *NP*-hard.

3.3 Dynamic Programming

If we wanted to prove that every context-free language is a member of P, we can deploy a dynamic programming algorithm to determine whether each variable in a context-free grammar G generates each substring of a given string $w := w_1 w_2 \cdots w_n$ in the language generated by G. The algorithm enters the solution to each sub-problem into an $n \times n$ table. For $i \leq j$, the (i, j) entry of the table contains the collection of variables that generate the substring $w_i w_{i+1} \cdots w_j$. For $i > j$, the table entries are unused.