CS301: Theory of Computation
Sections 5.3 & 7.1 - Mapping Reducibility and Complexity

Daniel White

Gettysburg College
Computer Science

Spring 2024

We previously defined **reducibility** as the act of converting a problem of language decidability (or recognizability) into a similar problem for another language that we already know things about.

e.g. We proved that E_{TM} is undecidable using the fact that A_{TM} is undecidable, and then proved Q_{TM} is undecidible using E_{TM}.

The notion of reducing one problem to another can be defined more formally in a variety of ways. In this section, we will explore **mapping reducibility**.

Definition 5.17 (pg. 234)

A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Definition 5.20 (pg. 235)

Language A is **mapping reducible** to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$ where

$$w \in A \iff f(w) \in B.$$

The function f is called a **reduction** from A to B.

To test whether $w \in A$, we use the reduction f to map w to $f(w)$ and then test whether $f(w) \in B$.

homework: 5.6

Theorem 5.22 (pg. 236)

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof. Let M be the decider for B and f be the reduction from A to B. We construct a decider N for A. On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.

Corollary 5.23 (pg. 236)

If $A \leq_m B$ and A is undecidable, then B is undecidable.

exercise: Why does the corollary above follow from the theorem?

Theorem 5.1 (pgs. 216 & 236)

$H_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$ is undecidable.

Proof. We proceed by mapping reducibility with A_{TM}, which is known to be undecidable. We must produce a computable function f such that $\langle M, w\rangle \in A_{\text{TM}}$ if and only if $f(\langle M, w\rangle) \in H_{\text{TM}}$.

Consider the machine F which, on input $\langle M, w\rangle$:

1. Construct the machine M', which on input x:
   1. Run M on x.
   2. If M accepts, *accept*; otherwise, *enter a loop*.
2. Output $\langle M', w\rangle$.

The TM F defines such a computable function f, which establishes that $A_{\text{TM}} \leq_m H_{\text{TM}}$.

Theorem 5.2 (pgs. 217 & 237)

$E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \varnothing\}$ is undecidable.

Proof. We proceed via mapping reduction from A_{TM} to $\overline{E_{\text{TM}}}$. Define the TM F (and hence the reduction f) on input $\langle M, w \rangle$:

1. Define TM M_1 on input x:
   1. If $x \neq w$, *reject*.
   2. If $x = w$, run M on input w and *accept* if M does.
2. Halt with $\langle M_1 \rangle$ on tape.

Our mapping reduction establishes that $A_{\text{TM}} \leq_m \overline{E_{\text{TM}}}$. Since A_{TM} is undecidable, so are both $\overline{E_{\text{TM}}}$ and E_{TM}.

exercise: Verify that f is a mapping reduction.
exercise: Argue why undecidibility implies co-undecidibility.

Theorem 5.28 (pg. 237)

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof. Essentially the same as the proof for decidability.

Corollary 5.29 (pg. 237)

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

This corollary can often be profitably applied in the following way: Suppose we wish to prove B is not Turing-recognizable, and recall that $\overline{A_{\text{TM}}}$ is not Turing-recognizable. Prove that $A_{\text{TM}} \leq_m \overline{B}$.

exercise: $C \leq_m D \iff \overline{C} \leq_m \overline{D}$ homework: 5.5 & Thm 5.30

Definition 7.1 (pg. 276)

Let TM M be a decider. The **time complexity** of M is $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

The definition above is used in **worst-case analysis**, but we often consider **average-case analysis** as well.

Average time complexity is defined as the function $f : \mathbb{N} \to \mathbb{R}^+$ which measures the average number of steps that M uses on any input of length n.

Definition 7.2 (pg. 277)

Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say $f(n) = O(g(n))$ if $f(n) \leq c \cdot g(n)$ for some $c > 0$ and all $n \geq n_0$ for some n_0.

Definition 7.5 (pg. 278)

Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say $f(n) = o(g(n))$ if $\lim_{n\to\infty} f(n)/g(n) \to 0$.

exercise: Let $f(x) = x^2 + 2$ and $g(x) = 3x^2 - x$. Are there any big-O or little-o relationships between f and g?

exercise: Let $a, b > 0$. Prove that $ax^i + bx^j = O(x^{\max(i,j)})$.

exercise: Prove that $\ln x = o(x^\varepsilon)$ for all $\varepsilon > 0$.

Definition 7.7 (pg. 279)

Let $t : \mathbb{N} \to \mathbb{R}^+$. The **time complexity class**, denoted $\text{TIME}(t(n))$, is the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

exercise: Find a time complexity class for $A = \{0^k 1^k \mid k \geq 0\}$. Is this the *only* time complexity class for A?

exercise: Try to find a "better" time complexity class for A by scanning left-right over the tape and crossing off every other 0 and every other 1. The parity of remaining 0s is the next bit of k, and similarly for 1. From here: be clever! *(pg. 280)*

Theorem 7.8 (pg. 282)

Let $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t(n)^2)$ time single-tape Turing machine.

Proof. Let M be a k-tape Turing machine that runs in $t(n)$ time. We will create a simulation machine S that runs in $O(t(n)^2)$ time. At each of M's computational steps, we will simulate M with S by

1. Scanning left-right over the simulation tape to determine M's next move. Each of M's tapes has length at most $t(n)$. This simulation scan takes $O(t(n))$ steps.
2. Simulate each of M's k-tape updates, requiring at most k right shifts (to make room on simulation tape). This requires $O(t(n))$ steps, due to the length of the tapes.

exercise: Why are M's tapes bounded at length $t(n)$?

Definition 7.9 (pg. 283)

Let N be a nondeterministic decider. The **time complexity** of N is the function f which gives the maximum number of steps N uses on any branch of its computation on any input of length n.

Theorem 7.11 (pg. 284)

Let $t(n) \geq n$. Then every $t(n)$ time nondeterministic Turing machine has an equivalent $2^{O(t(n))}$ time deterministic Turing machine.

homework: Thm 7.11