Problem 1: Factoring, NP, and coNP

Consider the factoring decision problem:

$$\text{FACTORIZATION} := \{\langle N, L, U \rangle : \exists \text{ prime } p \text{ s.t. } p \mid N, L \leq p \leq U\}.$$ 

That is, the factoring decision problem is to decide whether $N$ has a prime factor between $L$ and $U$.

The factoring search problem is: Given $N \geq 2$, find primes $p_1, \ldots, p_n$ (not necessarily distinct) such that $p_1 \cdot \ldots \cdot p_n = N$.

(a) Assume you have a polynomial-time Turing machine $M$ that solves the factoring search problem.\footnote{That is, for any positive $N$, we have $M(N) = \langle p_1, \ldots, p_n \rangle$ such that all $p_i$ are prime and $p_1 \cdot \ldots \cdot p_n = N$.} Construct a polynomial-time Turing machine $M'$ that solves $\text{FACTORIZATION}$\footnote{That is, $M'(N, L, U) = 1$ iff $\langle N, L, U \rangle \in \text{FACTORIZATION}$.}

**Note:** You do not need to describe the Turing machine in detail (giving the list of states, symbols, transition function). It is sufficient to describe the algorithm in pseudocode. You do not need to prove that the resulting Turing machine is polynomial-time (but it should be polynomial-time). You may assume without proof that there is a polynomial-time algorithm that decides whether a number is a prime.

(b) Construct an polynomial-time oracle Turing machine $M$ such that $M^{\text{FACTORIZATION}}$ solves the factoring search problem.\footnote{If you were not in the practice session: the definition of an oracle Turing machine is given in Arora-Barak, Sec. 3.4.}

**Note:** Same as the note in (a).

**Hint:** Do a binary search for the smallest prime factor first. And then use recursion.

(c) Show that $\text{FACTORIZATION} \in \text{NP}$.\footnote{That is, $M'\langle N, L, U \rangle = 1$ iff $\langle N, L, U \rangle \in \text{FACTORIZATION}$.}

**Note:** It is sufficient to say what the certificate $u$ is and what the Turing machine $M(\langle N, L, U \rangle, u)$ computes. You may assume without proof that there is a polynomial-time algorithm that decides whether a number is a prime.
(d) Show that $\text{FACTORING} \in \text{coNP}$.

**Note:** Same as the note in (c).

**Hint:** Prime factors.

(e) (Bonus points) Show that if $\text{FACTORING}$ is $\text{NP}$-complete, then $\text{NP} = \text{coNP}$.

**Note:** For this reason, it is commonly assumed that factoring is not $\text{NP}$-complete. (Since it is believed that $\text{NP} \neq \text{coNP}$.)

**Hint:** For an $L \in \text{NP}$, show that $L \leq_p \text{FACTORING}$. Then show $L \in \text{NP}$ and $L \in \text{coNP}$. Then you will have shown $\text{NP} \subseteq \text{coNP}$. For the other direction, proceed similarly. Recall that $\overline{L}$ denotes the complement of $L$.

**Problem 2: The Turing Hierarchy**

(a) Show that for any language $L$, the Halting problem

$$\text{HALT}^L := \{ \langle x, \alpha \rangle : M^L_\alpha(x) \text{ halts} \}$$

is undecidable given oracle access to $L$. (Here $M_\alpha$ is the oracle Turing machine with description $\alpha$.) That is, for no oracle Turing machine $M$, we have that $M^L(x, \alpha) = 1 \iff \langle x, \alpha \rangle \in \text{HALT}^L$.

**Hint:** The proof is almost the same as the proof that $\text{HALT}$ is undecidable by a normal Turing machine (without oracle access to $L$). A proof at the level of detail as done in the lecture is sufficient.

**Note:** What you are essentially asked to do here is to show that the proof of the undecidability of the Halting problem relativizes.

(b) Show that there is an infinite sequence of languages $L_1, L_2, \ldots$, such that:

- $L_i \leq_p L_{i+1}$. (That is, $L_{i+1}$ is at least as hard as $L_i$.)
- Given oracle access to $L_i$, no Turing machine can decide $L_{i+1}$. (Not even an unlimited Turing machine. This in particular implies that $L_{i+1} \not\leq_p L_i$.)

**Note:** If you don’t manage both properties, the second one is more important.

**Hint:** Assume you have constructed $L_1, \ldots, L_n$, then construct $L_{n+1}$. Use (a). Also the following construction may turn out to be useful: If $L, M$ are languages, then $L + M := \{ 0\|x : x \in L \} \cup \{ 1\|x : x \in M \}$ encodes a language that contains both $L$ and $M$. 


Problem 3: Polynomial identity testing and SAT (bonus problem)

In this problem I will present a (wrong) proof that there is a polynomial-time algorithm for deciding SAT. (This would imply that $\textbf{NP} \subseteq \textbf{BPP}$.)

(i) If we interpret Boolean operations as functions on $0, 1$, then we can represent $A \land B$ as $A \cdot B$, and $A \lor B$ as $1 - (1 - A) \cdot (1 - B)$, and $\neg A$ as $1 - A$.

(ii) Thus we can translate a Boolean formula $\varphi$ (in particular a CNF formula) into a formula $p$ containing only $\cdot, +, -$. We then have that for all $x_1, \ldots, x_n \in \{0, 1\}$, $\varphi(x_1, \ldots, x_n) = p(x_1, \ldots, x_n)$.

(iii) Deciding whether $\varphi$ is satisfiable is equivalent to deciding whether $\varphi \neq 0$ for some input.

(iv) To decide whether $\varphi \neq 0$ for some input, we just check whether $p \neq 0$ for some input.

(v) Whether $p \neq 0$ for some input can be tested probabilistically using the algorithm for polynomial identity testing from the practice.

(vi) Concluding, we have shown that we can decide whether $\varphi$ is satisfiable in polynomial-time (up to a small error probability $\varepsilon$).

In fact, no probabilistic algorithm for deciding SAT in polynomial time is known. Where is the mistake in the above proof? Why is it wrong?

For bonus points: Give a formula $\varphi$ on which the algorithm will give the wrong answer.

---

$^4$\textbf{BPP} is the class of problems that can be decided in probabilistic polynomial-time. We have not defined it yet. You do not need to understand the comment about $\textbf{NP} \subseteq \textbf{BPP}$ for solving this problem.

$^5$Reminder: Given a polynomial $p$, that algorithm decides in polynomial-time with small error $\varepsilon$ whether $p = 0$. The polynomial is encoded as an algebraic circuit which in particular allows us to encode formulas containing only $\cdot, +, -$. 

3