Problem 1: Factoring, NP, and coNP

Consider the factoring decision problem:

FACTORIZATION := \{⟨N, L, U⟩ : \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.\(^1\) Construct a polynomial-time Turing machine \(M'\) that solves FACTORIZATION.\(^2\)

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.

Solution. \(M'\) executes the following algorithm:

```
Input: integers \(N, L, U\).
⟨\(p_1, \ldots, p_n\)⟩ ← \(M(N)\)
for \(i = 1\) to \(n\) do
    if \(L \leq p_i \leq U\) then
        return 1
    return 0
```

(b) Construct a polynomial-time oracle Turing machine \(M\) such that \(M^{\text{FACTORIZATION}}\) solves the factoring search problem.\(^3\)

Note: Same as the note in (a).

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

---

\(^1\)That is, for any positive \(N\), we have \(M(N) = ⟨p_1, \ldots, p_n⟩\) such that all \(p_i\) are prime and \(p_1 \cdot \ldots \cdot p_n = N\).

\(^2\)That is, \(M'(N, L, U) = 1\) iff \(⟨N, L, U⟩\) \(\in\) FACTORIZATION.

\(^3\)If you were not in the practice session: the definition of an oracle Turing machine is given in Arora-Barak, Sec. 3.4.
Solution. $M^{\text{FACTORED}}$ executes the following recursive algorithm:

\begin{verbatim}
Input: integer $N$
if $N$ is prime then
  return $N$
$L := 2, U := N$
// Doing binary search for $N$'s smallest prime factor
while $L < U$ do // Invariant: $N$'s smallest prime factor is in $[L,U]$
  if $\langle N, L, \lfloor (U + L)/2 \rfloor \rangle \in \text{FACTORED}$ then
    $U := \lfloor (U + L)/2 \rfloor$
  else
    $L := \lfloor (U + L)/2 \rfloor + 1$
// Now $U = L$ is $N$'s smallest prime factor
$p_1 := L$
// Get the remaining prime factors by recursive call
$\langle p_2, \ldots, p_n \rangle := M^{\text{FACTORED}}(N/p_1)$
return $\langle p_1, \ldots, p_n \rangle$
\end{verbatim}

In this algorithm, the condition $\langle N, L, \lfloor (U + L)/2 \rfloor \rangle \in \text{FACTORED}$ in the if-statement is evaluated by calling the oracle $\text{FACTORED}$. 

(c) Show that $\text{FACTORED} \in \mathbf{NP}$.

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.

Solution. A certificate is a prime factor $p$ of $N$ with $L \leq p \leq U$. $M(\langle N, L, U \rangle, u)$ checks whether $u$ is prime and whether $u | N$ and whether $L \leq p \leq U$. In that case $M$ outputs 1.

(d) Show that $\text{FACTORED} \in \mathbf{coNP}$.

Note: Same as the note in (c).

Hint: Prime factors.

Solution. A certificate is a prime factorization $u = \langle p_1, \ldots, p_n \rangle$ of $N$. $M(\langle N, L, U \rangle, u)$ with $u = \langle p_1, \ldots, p_n \rangle$ checks whether all $p_i$ are prime, and whether $N = p_1 \cdot \ldots \cdot p_n$, and whether $L \leq p_i \leq U$ for no $i$. In that case $M$ outputs 1.

If $N, L, U$ are not integers, $M$ outputs 1.

We show that this TM is indeed doing what it should (i.e., $\langle N, L, U \rangle \notin \text{FACTORED} \iff \exists u. M(\langle N, L, U \rangle, u) = 1$). If $\langle N, L, U \rangle \notin \text{FACTORED}$, then with $u$ as described, $M$ will output 1 because no prime factor is between $L, U$. If
\[ (N, L, U) \in \text{FACTORING}, \text{ then if } u \text{ is the prime factorization (which is unique up to reordering), } M \text{ will output } 0 \text{ because there is a } p_i \text{ with } L \leq p_i \leq U. \text{ And if } u \text{ is not the prime factorization, } M \text{ will output } 0. \]

Thus \( M \) does what it should, hence \( \text{FACTORING} \in \text{coNP} \).

Alternative proof: you could also have shown that \( \text{FACTORING} \in \text{NP} \) (with a very similar TM \( M \)) and then concluded that \( \text{FACTORING} \in \text{coNP} \).

(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 \( \overline{L} \leq_p \text{FACTORING} \). Then show \( \overline{L} \in \text{NP} \) 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 \( L \).

Solution. Assume that \( \text{FACTORING} \) is \( \text{NP} \)-complete.

We first show \( \text{NP} \subseteq \text{coNP} \). Fix \( L \in \text{NP} \), we have to show that \( L \in \text{coNP} \).

Since \( L \in \text{NP} \) and \( \text{FACTORING} \) is \( \text{NP} \)-complete, \( L \leq_p \text{FACTORING} \). The same reduction \( f \) that shows \( L \leq_p \text{FACTORING} \) also shows \( \overline{L} \leq_p \text{FACTORING} \). (Because “\( x \in L \iff f(x) \in \text{FACTORING} \)” and \( x \in \overline{L} \iff f(x) \in \text{FACTORING} \) are logically equivalent.) Since \( \text{FACTORING} \in \text{coNP} \) by (d), we have \( \text{FACTORING} \in \text{NP} \). With \( \overline{L} \leq_p \text{FACTORING} \), it follows that \( \overline{L} \in \text{NP} \). By definition of \( \text{coNP} \), this means \( L \in \text{coNP} \).

We have shown \( \text{NP} \subseteq \text{coNP} \).

We now show \( \text{coNP} \subseteq \text{NP} \). Fix \( L \in \text{coNP} \), we have to show that \( L \in \text{NP} \). Since \( L \in \text{coNP} \), we have \( \overline{L} \in \text{NP} \). Since \( \text{FACTORING} \) is \( \text{NP} \)-complete, it follows that \( \overline{L} \leq_p \text{FACTORING} \). Thus \( L \leq_p \text{FACTORING} \). Since \( \text{FACTORING} \in \text{NP} \) (see above), it follows that \( L \in \text{NP} \).

We have shown \( \text{coNP} \subseteq \text{NP} \).

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 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.

Solution. Assume $\text{HALT}^L$ can be decided by some oracle Turing machine $M^L$ with access to $L$. Then we can construct an OTM $\hat{M}^L$ that does:

$$\hat{M}^L(\alpha) = \begin{cases} \text{halts} & \text{if } M^L_\alpha(\alpha) \text{ does not halt} \\ \text{does not halt} & \text{if } M^L_\alpha(\alpha) \text{ halts.} \end{cases}$$

$M^L(\alpha)$ can be easily constructed: it just calls $M^L(\alpha,\alpha)$, and if $M^L$ returns $1$, $\hat{M}^L$ enters an infinite loop, else $\hat{M}^L$ halts.

Since every OTM has a description, there is a $\beta$ such that $\hat{M}^L$ is $M^L_\beta$. Thus for all $\alpha$, $M^L_\beta(\alpha)$ halts iff $M^L_\alpha(\alpha)$ does not halt. With $\alpha := \beta$ this is a contradiction. Thus our assumption that $M^L$ decides $\text{HALT}$ was wrong. Hence $\text{HALT}^L$ is not decidable by OTMs with access to $L$.

(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 $L_{i+1} \not<_p L_i$)

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 $M$.

Solution. Let $L_1 := \emptyset$. Let $L_{i+1} := \text{HALT}^{L_i} + L_i$.

We have $L_i \leq_p L_{i+1}$ very easily: $f(x) := 1\|x$ is polynomial-time computable and

$$x \in L_i \iff f(x) = 1\|x \in \{0\|x : x \in \text{HALT}^{L_i}\} \cup \{1\|x : x \in L_i\}$$

$$= \text{HALT}^{L_i} + L_i = L_{i+1}.$$ 

Thus $f$ is a Karp reduction from $L_i$ to $L_{i+1}$.

We show that $L_{i+1}$ cannot be decided with oracle access to $L_i$. Assume that $M^{L_i}$ decides $L_{i+1}$. Let $\hat{M}^{L_i}(x)$ return $M^{L_i}(0\|x)$. Since $M^{L_i}$ decides $L_{i+1} = \text{HALT}^{L_i} + L_i$, $M^{L_i}$ decides $\text{HALT}^{L_i}$. But by (a), no Turing machine with $L_i$-oracle can decide $\text{HALT}^{L_i}$. We have a contradiction. Thus $L_{i+1}$ cannot be decided with oracle access to $L_i$.  

\footnote{nocitation}
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 $\mathbf{NP} \subseteq \mathbf{BPP}$.)

(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.

Solution. Step [iv] is wrong. The following would be true: “To decide whether $\varphi(x_1, \ldots, x_n) \neq 0$ for some input $x_1, \ldots, x_n \in \{0, 1\}$, we just check whether $p(x_1, \ldots, x_n) \neq 0$ for some input $x_1, \ldots, x_n \in \{0, 1\}$.”

However, $p$ takes arbitrary integers as inputs. So if we say “$p \neq 0$ for some input” that means “$p(x_1, \ldots, x_n) \neq 0$ for some input $x_1, \ldots, x_n \in \mathbb{Z}$.”

It could be that $\varphi = 0$ for all inputs, and thus $p = 0$ for all inputs $x_1, \ldots, x_n \in \{0, 1\}$, but still $p \neq 0$ for some $x_1, \ldots, x_n \notin \{0, 1\}$. Then the polynomial identity testing algorithm will return $p \neq 0$, and our algorithm would incorrectly claim that $p$ is satisfiable.

Actually, the problem occurs already with extremely simple formulas. E.g., $\varphi := x \land \neg x$. $\varphi$ is not satisfiable. But if we use the translation from above we get $p = x \cdot (1 - x)$. We have that $p = x - x^2 \neq 0$. (But $p(0) = p(1) = 0$.)

\[ \text{Solution} \]

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

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

---

5