Problem 1: 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 2: Polynomial identity testing and SAT

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

(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 $1 - A$. 

\[ \text{HALT}^L := \{ \langle x, \alpha \rangle : M^L_\alpha(x) \text{ halts} \} \]
(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.$^2$

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

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.

---

$^2$Reminder: Given a polynomial $p$, that algorithm decides in polynomial-time with small error $\epsilon$ whether $p = 0$. The polynomial is encoded as an algebraic circuit which in particular allows us to encode formulas containing only $\cdot, +, -$.