Homework 4

Ameyassh Nagarajan

May 2024

1 Problem 1

Step 1: Setup and Definitions

- Let $M_1, M_2, M_3, \ldots$ be an enumeration of all oracle Turing machines.
- M_i denotes the i-th machine in the enumeration.
- We will build the oracle A in stages, ensuring that at each stage, certain properties hold.

Step 2: Building the Oracle A

We will construct A iteratively. At each stage i, we ensure that M_i does not correctly decide a certain language related to A.

Step 3: Simulating NP Machines

For each i, consider the oracle Turing machine M_i. Define

$$L_i = \{x \mid \exists y \in A \text{ such that } |x| = |y|\}.$$

L_i is in NP^A.

Step 4: Ensuring Incorrect Decisions

For each i, perform the following steps:

- Use the part of A constructed so far.
- Simulate M_i on an input 0^m, where m is chosen to be sufficiently large (greater than lengths considered in previous steps).

Case 1: M_i accepts 0^m

Do not modify A. This ensures that M_i might accept some input it should reject.

Case 2: M_i rejects 0^m

We need to add a string to A to ensure M_i doesn't answer correctly.

Since M_i is an oracle Turing machine, it runs in polynomial time. It might query all strings of length m. To ensure it doesn't answer correctly, we need to consider the nature of coNP machines:

- A machine M_i in coNP accepts if and only if all computation paths accept.
- Since M_i rejects 0^m, there is at least one computation path that rejects.
- We only need to keep the answers to the queries in this rejecting path the same.

- The number of queries in this path is polynomial, so there are exponentially many strings of length m that this path doesn't query.
- Add one such string to A that this rejecting path does not query.

Step 5: Completeness of the Construction

By following the above steps, we ensure that for each i, M_i does not correctly decide L_i related to A. This iterative process constructs A such that there exists a language $L \in \mathrm{NP}^A$ that no machine in coNP^A can decide correctly. Consequently, $\mathrm{NP}^A \neq \mathrm{coNP}^A$.

Algorithm 1 Constructing an oracle A such that $\mathrm{NP}^A \neq \mathrm{coNP}^A$

Initialize: $A \leftarrow \emptyset$
for each $i \in \mathbb{N}$ **do**
  Let M_i be the i-th oracle Turing machine in the enumeration
  Choose a sufficiently large m such that m is greater than any length considered in previous steps
  Simulate M_i on input 0^m using the current A
  if M_i accepts 0^m **then**
    Do not modify A
  else
    M_i rejects 0^m. Identify a rejecting computation path P
    The path P queries a polynomial number of strings of length m
    Select a string x of length m that is not queried by P
    Add the string x to A: $A \leftarrow A \cup \{x\}$
  end if
end for

Proof Explanation

We construct an oracle A iteratively to ensure that $\mathrm{NP}^A \neq \mathrm{coNP}^A$.

Step-by-Step Construction

1. We start with A initialized as an empty set.
2. For each i, corresponding to the i-th oracle Turing machine M_i, we choose a sufficiently large input length m.
3. We simulate M_i on the input 0^m using the current contents of A.
4. If M_i accepts 0^m, we do not modify A to allow the possibility of an incorrect decision by M_i in the future.
5. If M_i rejects 0^m, there must be a rejecting computation path P. Since P queries only a polynomial number of strings of length m, there are exponentially many strings of length m that P does not query.
6. We select one such unqueried string x of length m and add it to A.

Conclusion

Through this construction, we have defined an oracle A such that $\mathrm{NP}^A \neq \mathrm{coNP}^A$, proving the separation by diagonalizing against all potential coNP machines.

$$\boxed{\text{Therefore, there exists an oracle } A \text{ such that } \mathrm{NP}^A \neq \mathrm{coNP}^A.}$$

2 Problem 2

To demonstrate that the problem $\{(G,k) \mid G \text{ is a graph with exactly one independent set of cardinality } k\}$ is in P^{NP}, we show that it can be decided by a polynomial-time deterministic Turing machine with access to an NP oracle.

Let M be a deterministic Turing machine that solves this problem with access to a combined NP oracle C. The oracle C queries two NP oracles A and B, and returns the appropriate response based on their outputs.

Algorithm Using Turing Machine M with Combined Oracle C

Algorithm 2 Determine if a graph has exactly one independent set of size k

Require: Graph G, integer k
Ensure: True if G has exactly one independent set of size k, False otherwise

Query NP oracle C to check if G has exactly one independent set of size k
if $C(G,k) = \text{no}$ **then**
  return False
end if
return True

Definition of Combined Oracle C

Define the combined NP oracle C as follows:

$$C(G,k) = \begin{cases} \text{no} & \text{if } A(G,k) = \text{False} \\ \text{no} & \text{if } B(G,k) = \text{True} \\ \text{yes} & \text{if } A(G,k) = \text{True and } B(G,k) = \text{False} \end{cases}$$

where:

- $A(G,k)$ checks if there is at least one independent set of size k in G.
- $B(G,k)$ checks if there is more than one independent set of size k in G.

Proof

To show that the problem of determining whether a graph G has exactly one independent set of size k is in P^{NP}, we proceed as follows:

1. Define the language L to be the set of pairs (G,k) where G is a graph and G has exactly one independent set of size k.
2. Construct an algorithm that uses the combined NP oracle C to decide L.
3. The algorithm operates as follows:
   (a) Query the combined NP oracle $C(G,k)$:
      - If $C(G,k) = \text{no}$, then G does not have exactly one independent set of size k and the algorithm returns False.
      - If $C(G,k) = \text{yes}$, then G has exactly one independent set of size k and the algorithm returns True.

Proof of Polynomial Time Execution

The combined NP oracle C internally queries oracles A and B, which are both NP oracles. Each query to A and B runs in polynomial time($O(|V| + |E|)$). The overall algorithm runs in polynomial time with a constant number of calls to the combined NP oracle C. Thus, the problem is in P^{NP}.

$$\boxed{\text{Thus, the problem is in } \mathrm{P}^{\mathrm{NP}}.}$$

3 Problem 3

Theorem

If PH has a complete problem (with respect to usual Karp reductions), then PH collapses.

Proof

Understanding the Polynomial Hierarchy (PH)

The Polynomial Hierarchy (PH) is a multi-level classification of complexity classes:

- Base level: $\Sigma_0^P = \Pi_0^P = P$.
- Higher levels: $\Sigma_{k+1}^P = \mathrm{NP}^{\Sigma_k^P}$ and $\Pi_{k+1}^P = \mathrm{coNP}^{\Sigma_k^P}$.

PH-Complete Problems

A problem is PH-complete if it represents the upper bound of complexity within the entire Polynomial Hierarchy.

Implications of Solving a PH-Complete Problem level k

If the PH complete problem exists at level k then the Polynomial Hierarchy collapses to that level. That is all the problems in PH can be reduced to this problem using karp-reductions and the polynomial hierarchy will not go beyond level k.

Proof

1. Assume PH has a complete problem L at level Σ_k^P:
   - Let L be a Σ_k^P-complete problem.
   - By definition of completeness, every problem in Σ_j^P (such that $j \geq k$) can be reduced to L in polynomial time.
   - Since $L \in \Sigma_k^P$, there exists a polynomial-time reduction from any problem in Σ_j^P to L, which implies that $\Sigma_j^P \subseteq \Sigma_k^P$.
2. Implications of $\Sigma_j^P \subseteq \Sigma_k^P$:
   - If $\Sigma_j^P \subseteq \Sigma_k^P$ and $k < j$, it means that Σ_j^P problems can be solved with the resources available at level k.
   - Therefore, $\Sigma_k^P = \Sigma_j^P$.
3. Resulting Collapse of the Polynomial Hierarchy:
   - Since $\Sigma_k^P = \Sigma_j^P$, for any $j \geq k$, we have $\Sigma_k^P = \Sigma_j^P$, leading to a collapse of the hierarchy to level k.
4. Special Case $k = 1$:
   - If $k = 1$ and Σ_1^P has a complete problem that can be solved in P (level 0), it implies $\Sigma_1^P = P$, and thus $NP = P$.
   - This would lead to $\Sigma_i^P = P$ for all $i \geq 1$, collapsing PH to P.

Conclusion

If PH has a complete problem at any level Σ_k^P, then PH collapses to that level. Therefore, the existence of a complete problem in PH implies that the polynomial hierarchy collapses to a finite level.

$$\boxed{\text{Thus, if PH has a complete problem, PH collapses.}}$$