CS 6810: Theory of Computing **Spring 2026**

Lecture 3: Hierarchy Theorems

Lecturer: Eshan Chattopadhyay *Scribe: Matthew Cowan*

1 Overview

The hierarchy theorems formalize the intuition that:

> *"Given more computational resources, we can solve more problems."*

The main technique used to prove these results is *diagonalization*, and in some cases *lazy diagonalization*. In later lectures, we will show the failure of diagonalization techniques to solve questions like $\mathbf{P} = \mathbf{NP}$, introducing the idea of *barrier results* which show what kinds of proof techniques are strong enough (or not strong enough) to separate complexity classes. Future lectures will also extend some of these ideas to Oracle Turing Machines. In all of the following proofs and definitions, we consider Turing Machines that compute functions of the form $f\colon \{0,1\}^* \to \{0,1\}$.

2 Deterministic Time Hierarchy Theorem

Theorem 2.1 (Deterministic Time Hierarchy)**.** *Let $f, g\colon \mathbb{N} \to \mathbb{N}$ be time-constructible functions such that*

$$f(n)\log f(n) = o(g(n)).$$

Then,

$$\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n)).$$

Why the $\log f(n)$ Overhead? (and other questions)

The logarithmic overhead arises because we will need to simulate machines on a universal Turing machine, incurring a logarithmic overhead (which is more efficient than the quadratic overhead needed we saw in lecture 1). Moreover, if you're wondering what *time-constructible* means, its definition will follow but it is a necessary assumption for a TM to be able to run exactly n steps of simulation when it's simulating another TM. Lastly, a reminder of "little-oh" notation. When we say $f(n)\log(n) = o(g(n))$, we mean that for any ε, there exists an N such that for all $n \geq N$

$$\frac{f(n)\log f(n)}{g(n)} < \varepsilon,$$

or in other words,

$$\lim_{n\to\infty} \frac{f(n)\log f(n)}{g(n)} = 0.$$

Note that the first definition of little-oh captures its asymptotic nature which we will have to account for in the proof.

Proof

We define a diagonalizing machine D.

D on input x:

1. Let $n = |x|$.
2. Simulate the machine M_x (the machine encoded by x) on input x for $g(n)$ steps using a universal TM.
3. If M_x halts during this simulation, output the opposite of its answer.
4. Otherwise, reject.

This defines a language $L = \mathcal{L}(D)$. By construction, $L \in \text{DTIME}(g(n))$.

Claim 2.2. $L \notin \text{DTIME}(f(n))$.

Proof. Suppose there exists a machine M' running in time $O(f(n))$ such that $\mathcal{L}(M') = \mathcal{L}(D) = L$. Let y be the bitstring describing M', i.e. $\langle M' \rangle = y$. Let $k := |y|$.

Consider running D on input y:

- D simulates $M'_y(y)$ for $g(k)$ steps.
- We claim that M' halts during this simulation, due to the assumption $f \log f = o(g)$, but this requires a slight refinement in how we construct M'. Here, k is a fixed integer, but it is possible that $k < N$, where N is the same as in the little-oh definition above. Therefore, instead of letting M' be any TM deciding L, let it be one with more states such that its encoding, y, is long enough such that $g(n)$ grows strictly faster than $f(n) \log f(n)$. We can think of this as adding lots and lots of comments to a C program until its executable has increased significantly in size. With this in mind, it follows that the time to run f on the UTM satisfies $C \cdot f(k) \cdot \log(k) < g(k)$, and therefore the simulation halts.

Thus $D(y)$ outputs the opposite of $M'(y)$, implying

$$D(y) \neq M'(y),$$

a contradiction. □

3 Time-Constructible Functions

Definition 3.1. *A function $f(n)$ is* time-constructible *if:*

1. *On input 1^n, a Turing machine outputs $1^{f(n)}$ in $O(f(n))$ time.*
2. *$f(n)$ is non-decreasing.*
3. *$f(n) \geq n$.*

Time constructibility ensures we can allocate exactly $f(n)$ steps in a simulation, which is required in the diagonalization proof. Moreover, most "reasonable" functions we encounter when constructing algorithms are time constructible.

4 Nondeterministic Time Hierarchy Theorem

Theorem 4.1 (Nondeterministic Time Hierarchy)**.** *Let $f(n)$ and $g(n)$ be time-constructible functions such that*

$$f(n) = o(g(n)).$$

Then,

$$\text{NTIME}(f(n)) \subsetneq \text{NTIME}(g(n)).$$

Issue with Naive Diagonalization(and other remarks)

If we attempt to negate the output of a nondeterministic TM directly, as we did in our first diagonalization proof, the method fails because nondeterministic machines accept if *any* branch accepts. Negating a single branch does not invert the overall computation. Another thing worth noting is that in theorem statement there is nothing requiring logarithmic overhead as in the deterministic case. In this proof, like the first one, we will have to simulate a NTM on a universal Machine, but it is a well known result (which will not be proved here) that if you have a NTM running in time T(n), then there exists a Universal Non Deterministic TM that can simulate it in time O(T(n)).

Lazy Diagonalization Approach

The following proof of the non deterministic time hierarchy is a more recent one attributed to Fortnow and Santhanam. To see an alternative proof, please consult the Arora and Barak textbook on page 67.

We construct a machine D that takes two inputs (x, y), with $n := |x| + |y|$.

$D(x, y)$:

1. If $|y| < g(|x|)$:
   - Nondeterministically simulate M_x on two inputs: $(x, y0)$ and $(x, y1)$ for $g(n)$ steps.
   - Accept iff M_x accepts both branches. (NOTE: This can be easily checked by guessing a certificate for both inputs, check if it's satisfied for each of them, and accept if it is.)
2. If $|y| \geq g(|x|)$:
   - Simulate M_x on the empty string using y as the nondeterministic choice.
   - Stop after $g(n)$ steps.
   - If M_x halts, output the opposite answer.
   - Otherwise, reject.

This construction ensures that

$$L(D) \in \text{NTIME}(g(n)).$$

Contradiction Argument

Suppose there exists an NTM N' running in $C \cdot f(n)$ time such that

$$\mathcal{L}(N') = \mathcal{L}(D).$$

Let x' be a sufficiently large bitstring encoding of N'. Let $L := \mathcal{L}(D)$ and let ε denote the empty string. Then,

$$(x', \varepsilon) \in L \iff N' \text{ accepts } (x', 0), (x', 1).$$

This is because in Case 1 of D, $(x', \varepsilon) \in L$ if and only if N' accepts $(x', 0)$ and $(x', 1)$. Since N' decides L, this means $(x', 0), (x', 1)$ are also in L. Moreover, since it is always the case that $|\varepsilon| < g(|x'|)$, we fall into Case 1 of D, and are valid in applying this logic.

Iterating this reasoning,

$$\begin{aligned}
(x', \varepsilon) \in L &\iff (x', 0), (x', 1) \in L \\
&\iff (x', 00), (x', 01), (x', 10), (x', 11) \in L \\
&\vdots \\
&\iff (x', y) \in L \quad \text{where } |y| = g(|x'|).
\end{aligned}$$

Thus, we fall into the second case of D, occurring when $|y| \geq g(|x'|)$, and by the same diagonal argument as in the proof for deterministic machines, D returns the opposite output, a contradiction.

Therefore,

$$\text{NTIME}(f(n)) \subsetneq \text{NTIME}(g(n)).$$

5 Remarks

- The deterministic proof requires a $\log f(n)$ overhead due to universal simulation.
- The nondeterministic proof uses lazy diagonalization to handle branching behavior.
- The same proof ideas extend to space complexity, where the log overhead is not needed.