# Lecture 7: Diagonalization

Arijit Bishnu

16.03.2010

# Outline

# Outline

# Warm Up

## Diagonalization and its Uses

- We want to separate interesting complexity classes. How do we do it?

# Warm Up

## Diagonalization and its Uses

- We want to separate interesting complexity classes. How do we do it?
- To separate two complexity classes, we need to describe a machine in one class that gives a different answer on some input from every machine in the other class.

# Warm Up

## Diagonalization and its Uses

- We want to separate interesting complexity classes. How do we do it?
- To separate two complexity classes, we need to describe a machine in one class that gives a different answer on some input from every machine in the other class.
- Diagonalization is the only general technique known for constructing such a machine.

# Warm Up

## Diagonalization and its Uses

- We want to separate interesting complexity classes. How do we do it?
- To separate two complexity classes, we need to describe a machine in one class that gives a different answer on some input from every machine in the other class.
- Diagonalization is the only general technique known for constructing such a machine.
- In this lecture, we prove some hierarchy theorems and a consequence if P $\neq$ NP is proved using diagonalization.

# Time-Constructible Functions

## Time-Constructible Functions

We say that a function $f : \mathbb{N} \to \mathbb{N}$ is a time constructible function if $f(n) \geq n$ and there is a TM $M$ that, given the input $1^n$, writes down $1^{f(n)}$ on its tape in $O(f(n))$ time.

# Time-Constructible Functions

## Time-Constructible Functions

We say that a function $f : \mathbb{N} \to \mathbb{N}$ is a time constructible function if $f(n) \geq n$ and there is a TM $M$ that, given the input $1^n$, writes down $1^{f(n)}$ on its tape in $O(f(n))$ time.

## Examples

$n$, $n \log n$, $n^5$, $2^n$

# Time-Constructible Functions

## Time-Constructible Functions

We say that a function $f : \mathbb{N} \to \mathbb{N}$ is a time constructible function if $f(n) \geq n$ and there is a TM $M$ that, given the input $1^n$, writes down $1^{f(n)}$ on its tape in $O(f(n))$ time.

## Examples

$n$, $n \log n$, $n^5$, $2^n$

## Remark

Functions that are much larger than exponential in $n$ are non-time constructible. As an example, $T : \mathbb{N} \to \mathbb{N}$ such that every function computable in time $T(n)$ can also be computed in the much shorter time $\log T(n)$.

# Universal Turing Machine (UTM)

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.

# Universal Turing Machine (UTM)

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, $\delta$.

# Universal Turing Machine (UTM)

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, $\delta$.
- So, list all inputs and outputs of $\delta$ and encode it as a string over $\{0, 1\}^*$.

# Universal Turing Machine (UTM)

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, $\delta$.
- So, list all inputs and outputs of $\delta$ and encode it as a string over $\{0, 1\}^*$.
- Our representation scheme satisfies the following
  - Every string in $\{0, 1\}^*$ represents some TM.
  - Every TM is represented by infinitely many strings.

# Universal Turing Machine (UTM)

- We can write the description of any TM on paper. Hence, we can encode it using strings over $\{0, 1\}$.
- The action of a TM is determined by its transition function, $\delta$.
- So, list all inputs and outputs of $\delta$ and encode it as a string over $\{0, 1\}^*$.
- Our representation scheme satisfies the following
  - Every string in $\{0, 1\}^*$ represents some TM.
  - Every TM is represented by infinitely many strings.
- Some notations:
  - For a TM $M$, we use $M_b$ to denote the binary string representation of $M$.
  - For a string $\alpha$, $M_\alpha$ denotes the TM represented by $\alpha$.

# Universal Turing Machine

- a UTM can simulate the execution of every other TM $M$ given $M$'s description as an input.

# Universal Turing Machine

- a UTM can simulate the execution of every other TM $M$ given $M$'s description as an input.
- The parameters of a UTM are fixed - alphabet size, number of states, and the number of tapes.

# Universal Turing Machine

- a UTM can simulate the execution of every other TM $M$ given $M$'s description as an input.
- The parameters of a UTM are fixed - alphabet size, number of states, and the number of tapes.
- The UTM encodes any other TM $M$'s states and transition table on its tapes, and then follows along the computation step by step.

# Universal Turing Machine

- a UTM can simulate the execution of every other TM $M$ given $M$'s description as an input.
- The parameters of a UTM are fixed - alphabet size, number of states, and the number of tapes.
- The UTM encodes any other TM $M$'s states and transition table on its tapes, and then follows along the computation step by step.
- For $i \in \mathbb{N}$, we will also use the notation $M_i$ for the machine represented by the string that is the binary expansion of the number $i$.

# Theorem on Efficient UTM

**Theorem: Efficient UTM**

There exists a TM $\mathcal{U}$ such that for every $x, \alpha \in \{0,1\}^*$, $\mathcal{U}(x, \alpha) = M_\alpha(x)$, where $M_\alpha$ denotes the TM represented by $\alpha$. Furthermore, if $M_\alpha$ halts on input $x$ within $T$ steps, then $\mathcal{U}(x, \alpha)$ halts within $CT \log T$ steps, where $C$ is a number independent of $|x|$ and depends only on $M_\alpha$'s alphabet size, number of tapes, and number of states.

# Recall Class DTIME

# Recall Class DTIME

**Definition: The Class DTIME**

Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We let DTIME($T(n)$) be the set of all boolean functions that are computable in $d \cdot T(n)$-time for some constant $d > 0$.

# Recall Class DTIME

**Definition: The Class DTIME**

Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We let DTIME($T(n)$) be the set of all boolean functions that are computable in $d \cdot T(n)$-time for some constant $d > 0$.

**Definition: The Class P**

$\mathsf{P} = \bigcup_{c \geq 1} \mathsf{DTIME}(n^c)$.

# Outline

# Time Hierarchy Theorem

**Theorem**

If $f$, $g$ are time-constructible functions (TCF) satisfying $f(n)\log f(n) = o(g(n))$, then $\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$

# Time Hierarchy Theorem

## Theorem

If $f$, $g$ are time-constructible functions (TCF) satisfying $f(n)\log f(n) = o(g(n))$, then $\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$

## Essence of the Theorem

- For any TCF $f$, $\exists$ a language $L$ that is decidable in time $O(f(n))$ but not in time $o(\frac{f(n)}{\log f(n)})$.
- For every deterministic time-bounded complexity class, there is a strictly larger time-bounded complexity class, and so the time-bounded hierarchy of complexity classes does not completely collapse.

# Time Hierarchy Theorem

## Theorem

If $f$, $g$ are time-constructible functions (TCF) satisfying $f(n)\log f(n) = o(g(n))$, then $\text{DTIME}(f(n)) \subsetneq \text{DTIME}(g(n))$

## Essence of the Theorem

- For any TCF $f$, $\exists$ a language $L$ that is decidable in time $O(f(n))$ but not in time $o(\frac{f(n)}{\log f(n)})$.
- For every deterministic time-bounded complexity class, there is a strictly larger time-bounded complexity class, and so the time-bounded hierarchy of complexity classes does not completely collapse.
- Even more generally, it can be shown that if $f(n)$ is time-constructible, then $\text{DTIME}(o(\frac{f(n)}{\log f(n)}))$ is properly contained in $\text{DTIME}(f(n))$..

# Proof of Time Hierarchy Theorem

## Proof of a Simpler Version

$\text{DTIME}(n) \subsetneq \text{DTIME}(n^{1.5})$

## The Proof

- Let $M_x$ be the machine represented by string $x$.

# Proof of Time Hierarchy Theorem

## Proof of a Simpler Version

$\text{DTIME}(n) \subsetneq \text{DTIME}(n^{1.5})$

## The Proof

- Let $M_x$ be the machine represented by string $x$.
- Define a new TM $D$ as follows: On input $x$, run for $x^{1.4}$ steps the UTM $\mathcal{U}$ to simulate the execution of $M_x$ on $x$. If $M_x$ outputs an answer in this time, namely, $M_x(x) \in \{0, 1\}$ then output the opposite answer; else output REJECT.

# Proof of Time Hierarchy Theorem

## Proof of a Simpler Version

$\text{DTIME}(n) \subsetneq \text{DTIME}(n^{1.5})$

## The Proof

- Let $M_x$ be the machine represented by string $x$.
- Define a new TM $D$ as follows: On input $x$, run for $x^{1.4}$ steps the UTM $\mathcal{U}$ to simulate the execution of $M_x$ on $x$. If $M_x$ outputs an answer in this time, namely, $M_x(x) \in \{0, 1\}$ then output the opposite answer; else output REJECT.
- $D$ halts within $n^{1.4}$ steps and hence, $L \in \text{DTIME}(n^{1.5})$ ($L$ was decided by $D$ in the said time).

# The Proof of $L \notin \mathrm{DTIME}(n)$

For a contradiction, assume that some TM $M$ decides $L$ but runs in time $cn$ on inputs of size $n$. Then for every $x \in \{0,1\}^*$, $M(x) = D(x)$.

- By the earlier theorem on UTM, the time to simulate $M$ by the UTM $\mathcal{U}$ on every input $x$ is at most $c'c\,|x|\log|x|$; here $c'$ depends on the parameters of the TM $M$ but not on $|x|$.

# The Proof of $L \notin \text{DTIME}(n)$

For a contradiction, assume that some TM $M$ decides $L$ but runs in time $cn$ on inputs of size $n$. Then for every $x \in \{0,1\}^*$, $M(x) = D(x)$.

- By the earlier theorem on UTM, the time to simulate $M$ by the UTM $\mathcal{U}$ on every input $x$ is at most $c'c\,|x|\log|x|$; here $c'$ depends on the parameters of the TM $M$ but not on $|x|$.
- $\exists$ a number $n_0$ such that $\forall\ n \geq n_0$, $n^{1.4} > c'cn\log n$.

# The Proof of $L \notin \text{DTIME}(n)$

For a contradiction, assume that some TM $M$ decides $L$ but runs in time $cn$ on inputs of size $n$. Then for every $x \in \{0,1\}^*$, $M(x) = D(x)$.

- By the earlier theorem on UTM, the time to simulate $M$ by the UTM $\mathcal{U}$ on every input $x$ is at most $c'c\,|x|\log|x|$; here $c'$ depends on the parameters of the TM $M$ but not on $|x|$.
- $\exists$ a number $n_0$ such that $\forall\ n \geq n_0$, $n^{1.4} > c'cn\log n$.
- Let $x \in \{0,1\}^*$ represent the TM $M$ of length at least $n_0$.

# The Proof of $L \notin \text{DTIME}(n)$

For a contradiction, assume that some TM $M$ decides $L$ but runs in time $cn$ on inputs of size $n$. Then for every $x \in \{0,1\}^*$, $M(x) = D(x)$.

- By the earlier theorem on UTM, the time to simulate $M$ by the UTM $\mathcal{U}$ on every input $x$ is at most $c'c\,|x|\log|x|$; here $c'$ depends on the parameters of the TM $M$ but not on $|x|$.
- $\exists$ a number $n_0$ such that $\forall\ n \geq n_0$, $n^{1.4} > c'cn\log n$.
- Let $x \in \{0,1\}^*$ represent the TM $M$ of length at least $n_0$.
- Then, $D(x)$ will obtain the output $M(x)$ within $x^{1.4}$ steps, but by definition of $D$, we have $D(x) = 1 - M(x) \neq M(x)$ which is a contradiction.

# Space Hierarchy Theorem

## Space-Constructible Functions

A function $f : \mathbb{N} \to \mathbb{N}$, where $f(n) \geq \log n$ is called space constructible if there is a TM $M$ that maps $1^n$ to the binary representation of $f(n)$ on its tape in $O(f(n))$ space.

# Space Hierarchy Theorem

## Space-Constructible Functions

A function $f : \mathbb{N} \to \mathbb{N}$, where $f(n) \geq \log n$ is called space constructible if there is a TM $M$ that maps $1^n$ to the binary representation of $f(n)$ on its tape in $O(f(n))$ space.

## Theorem

If $f$, $g$ are space-constructible functions satisfying $f(n) = o(g(n))$, then $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(g(n))$

# Space Hierarchy Theorem

## Space-Constructible Functions

A function $f : \mathbb{N} \to \mathbb{N}$, where $f(n) \geq \log n$ is called space constructible if there is a TM $M$ that maps $1^n$ to the binary representation of $f(n)$ on its tape in $O(f(n))$ space.

## Theorem

If $f$, $g$ are space-constructible functions satisfying $f(n) = o(g(n))$, then $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(g(n))$

## Proof

The proof is same as the earlier one. Only note the absence of the $\log f(n)$ factor. This happens because the UTM does not need the logarithmic factor.

# Nondeterministic Time Hierarchy Theorem

**Theorem**

If $f, g$ are time-constructible functions satisfying $f(n+1) = o(g(n))$, then $\mathrm{NTIME}(f(n)) \subsetneq \mathrm{NTIME}(g(n))$

# Outline

# Existence of NP-intermediate problems

- Many problems that were in NP turned out to be NP-complete.

# Existence of NP-intermediate problems

- Many problems that were in NP turned out to be NP-complete.
- This gave rise to a conjecture that the class NP is the union of two disjoint classes P and NP-complete.

# Existence of NP-intermediate problems

- Many problems that were in NP turned out to be NP-complete.
- This gave rise to a conjecture that the class NP is the union of two disjoint classes P and NP-complete.
- In effect, if P turns out to be equal to NP, then the conjecture is true.

# Existence of NP-intermediate problems

- Many problems that were in NP turned out to be NP-complete.
- This gave rise to a conjecture that the class NP is the union of two disjoint classes P and NP-complete.
- In effect, if P turns out to be equal to NP, then the conjecture is true.
- But, it turns out that the conjecture is false if $P \neq NP$, i.e. if $P \neq NP$, then $\exists$ a language $L \in NP \setminus P$ that is not NP-complete.

# Ladner's Theorem

## Theorem

Suppose that $P \neq NP$. Then, $\exists$ a language $L \in NP \setminus P$ that is not NP-complete.

## Proof Idea

- If $P \neq NP$, then $\exists$ at least a language $SAT \in NP \setminus P$.

# Ladner's Theorem

## Theorem

Suppose that P $\neq$ NP. Then, $\exists$ a language $L \in \text{NP} \setminus \text{P}$ that is not NP-complete.

## Proof Idea

- If P $\neq$ NP, then $\exists$ at least a language SAT $\in$ NP $\setminus$ P.
- Consider $\text{SAT}_H = \{< \varphi 1^{n^{H(n)}} > \quad | \quad \varphi \in \text{SAT} \textit{and} \, |\varphi| = n\}$ where $H : \mathbb{N} \rightarrow \mathbb{N}$ is a poly-time computable function.

# Ladner's Theorem

## Theorem

Suppose that P $\neq$ NP. Then, $\exists$ a language $L \in \text{NP} \setminus \text{P}$ that is not NP-complete.

## Proof Idea

- If P $\neq$ NP, then $\exists$ at least a language SAT $\in$ NP $\setminus$ P.
- Consider $\text{SAT}_H = \{< \varphi 1^{n^{H(n)}} > \quad | \quad \varphi \in \text{SAT} and \, |\varphi| = n\}$ where $H : \mathbb{N} \rightarrow \mathbb{N}$ is a poly-time computable function.
- Consider two cases:

# Ladner's Theorem

## Theorem

Suppose that P $\neq$ NP. Then, $\exists$ a language $L \in \text{NP} \setminus \text{P}$ that is not NP-complete.

## Proof Idea

- If P $\neq$ NP, then $\exists$ at least a language SAT $\in$ NP $\setminus$ P.
- Consider $\text{SAT}_H = \{< \varphi 1^{n^{H(n)}} > \mid \varphi \in \text{SAT} and\ |\varphi| = n\}$ where $H : \mathbb{N} \rightarrow \mathbb{N}$ is a poly-time computable function.
- Consider two cases:
  - $H(n) = c$, a constant $\forall n$. So, $\text{SAT}_H$ is simply SAT with a polynomial amount of padding. Thus, $\text{SAT}_H$ is also NP-complete and is not in P assuming P $\neq$ NP.

# Ladner's Theorem

## Theorem

Suppose that P $\neq$ NP. Then, $\exists$ a language $L \in \text{NP} \setminus \text{P}$ that is not NP-complete.

## Proof Idea

- If P $\neq$ NP, then $\exists$ at least a language SAT $\in$ NP $\setminus$ P.
- Consider $\text{SAT}_H = \{< \varphi 1^{n^{H(n)}} > \mid \varphi \in \text{SAT} and\ |\varphi| = n\}$ where $H : \mathbb{N} \rightarrow \mathbb{N}$ is a poly-time computable function.
- Consider two cases:
  - $H(n) = c$, a constant $\forall n$. So, $\text{SAT}_H$ is simply SAT with a polynomial amount of padding. Thus, $\text{SAT}_H$ is also NP-complete and is not in P assuming P $\neq$ NP.
  - $H(n)$ tends to $\infty$ with $n$, thus the padding is of superpolynomial size. Our claim is that $\text{SAT}_H$ is not NP-complete. Why? Let's try contradiction.

# Proof Idea Continued...

## Proof Idea

- For a contradiction, assume that $SAT_H$ is NP-complete. So, $SAT \leq_P SAT_H$. So, instances of $SAT_H$ should be of length $O(n^i)$.

# Proof Idea Continued...

## Proof Idea

- For a contradiction, assume that $SAT_H$ is NP-complete. So, $SAT \leq_P SAT_H$. So, instances of $SAT_H$ should be of length $O(n^i)$.
- So, $|\varphi| + |\varphi|^{H(|\varphi|)} = O(n^i)$. So, $|\varphi| = o(n)$.

# Proof Idea Continued...

## Proof Idea

- For a contradiction, assume that $\text{SAT}_H$ is NP-complete. So, $\text{SAT} \leq_P \text{SAT}_H$. So, instances of $\text{SAT}_H$ should be of length $O(n^i)$.
- So, $|\varphi| + |\varphi|^{H(|\varphi|)} = O(n^i)$. So, $|\varphi| = o(n)$.
- The above implies a poly-time reduction from a SAT instance of length $O(n)$ to a SAT instance of length $o(n)$, which in turn implies SAT can be solved in poly-time. This contradicts $\text{P} \neq \text{NP}$.

# Proof Idea Continued...

## Proof Idea

- For a contradiction, assume that $\text{SAT}_H$ is NP-complete. So, $\text{SAT} \leq_P \text{SAT}_H$. So, instances of $\text{SAT}_H$ should be of length $O(n^i)$.
- So, $|\varphi| + |\varphi|^{H(|\varphi|)} = O(n^i)$. So, $|\varphi| = o(n)$.
- The above implies a poly-time reduction from a SAT instance of length $O(n)$ to a SAT instance of length $o(n)$, which in turn implies SAT can be solved in poly-time. This contradicts $\text{P} \neq \text{NP}$.
- So, $H$ has to be designed properly so that it grows slowly enough in tending towards infinity.

# Definition of the function $H(n)$

## $H(n)$

$H(n)$ is the smallest number $i < \log\log n$ such that for every $x \in \{0,1\}^*$ with $|x| \leq \log n$,

$$M_i \text{ halts on } x \text{ within } i\,|x|^i \text{ steps and } M_i \text{ outputs 1 iff } x \in \text{SAT}_H$$

where $M_i$ is the machine represented by the binary expansion of $i$ according to the representation scheme of UTM $\mathcal{U}$. If there is no such $i$, then let $H(n) = \log\log n$.

# Computation of $H(n)$

## $H(n)$ is well defined

$H(n)$ is well defined. Show that yourself.

## Computation of $H(n)$

To compute $H(n)$, we need to

# Computation of $H(n)$

## $H(n)$ is well defined

$H(n)$ is well defined. Show that yourself.

## Computation of $H(n)$

To compute $H(n)$, we need to

- compute $H(k)$ for every $k \leq \log n$.

# Computation of $H(n)$

## $H(n)$ is well defined

$H(n)$ is well defined. Show that yourself.

## Computation of $H(n)$

To compute $H(n)$, we need to

- compute $H(k)$ for every $k \leq \log n$.
- simulate at most $\log \log n$ machines for every input of length at most $\log n$ for $\log \log n (\log n)^{\log \log n} = o(n)$ steps.

# Computation of $H(n)$

## $H(n)$ is well defined

$H(n)$ is well defined. Show that yourself.

## Computation of $H(n)$

To compute $H(n)$, we need to

- compute $H(k)$ for every $k \leq \log n$.
- simulate at most $\log \log n$ machines for every input of length at most $\log n$ for $\log \log n (\log n)^{\log \log n} = o(n)$ steps.
- compute SAT on all inputs of length at most $\log n$.

# Computation of $H(n)$

## $H(n)$ is well defined

$H(n)$ is well defined. Show that yourself.

## Computation of $H(n)$

To compute $H(n)$, we need to

- compute $H(k)$ for every $k \leq \log n$.
- simulate at most $\log \log n$ machines for every input of length at most $\log n$ for $\log \log n (\log n)^{\log \log n} = o(n)$ steps.
- compute SAT on all inputs of length at most $\log n$.
- So, $H(n)$ can be computed in polynomial time ($O(n^3)$).

# Claim: $SAT_H \notin P$

## $SAT_H \notin P$

- For a contradiction, assume that there is a machine $M$ solving $SAT_H$ in at most $cn^c$ steps.

# Claim: $SAT_H \notin P$

## $SAT_H \notin P$

- For a contradiction, assume that there is a machine $M$ solving $SAT_H$ in at most $cn^c$ steps.
- Since, $M$ is represented by infinitely many strings, there is a number $i > c$ such that $M = M_i$.

# Claim: $\mathrm{SAT}_H \notin \mathrm{P}$

## $\mathrm{SAT}_H \notin \mathrm{P}$

- For a contradiction, assume that there is a machine $M$ solving $\mathrm{SAT}_H$ in at most $cn^c$ steps.
- Since, $M$ is represented by infinitely many strings, there is a number $i > c$ such that $M = M_i$.
- By the definition of $H(n)$, this implies that for $n > 2^{2^i}$, $H(n) < i$.

# Claim: $\text{SAT}_H \notin \text{P}$

## $\text{SAT}_H \notin \text{P}$

- For a contradiction, assume that there is a machine $M$ solving $\text{SAT}_H$ in at most $cn^c$ steps.
- Since, $M$ is represented by infinitely many strings, there is a number $i > c$ such that $M = M_i$.
- By the definition of $H(n)$, this implies that for $n > 2^{2^i}$, $H(n) < i$.
- So, for sufficiently large input lengths, $\text{SAT}_H$ is simply the language padded with $n^i$ (polynomial) number of 1s, and hence, $\text{SAT}_H \notin \text{P}$ unless $\text{NP} \neq \text{P}$.

# Claim: $SAT_H \notin$ NP-complete

## $SAT_H \notin$ NP-complete

The idea is to show that $H(n)$ tends to $\infty$ with $n$ and then the earlier proof idea follows. We prove the equivalent statement: for every integer $i$, there are only finitely many $n$'s such that $H(n) = i$.

- Since, $SAT_H \notin P$, for each $i$, we know that there is an input $x$ such that given $i\,|x|^i$ time, $M_i$ gives the incorrect answer to whether or not $x \in SAT_H$.

# Claim: $SAT_H \notin$ NP-complete

## $SAT_H \notin$ NP-complete

The idea is to show that $H(n)$ tends to $\infty$ with $n$ and then the earlier proof idea follows. We prove the equivalent statement: for every integer $i$, there are only finitely many $n$'s such that $H(n) = i$.

- Since, $SAT_H \notin P$, for each $i$, we know that there is an input $x$ such that given $i\,|x|^i$ time, $M_i$ gives the incorrect answer to whether or not $x \in SAT_H$.
- The definition of $H$ ensures that for every $n > 2^{|x|}$, $H(x) \neq i$.