Computational Complexity Theory
Fall 2025

Time complexity and Hierarchy theorems: Part II
September 4, 2025

Lijie Chen

University of California, Berkeley
✉ lijiechen@berkeley.edu

Some Logistics

- **Office Hours:** 2:00 - 3:00 PM, SODA 627, Tuesday

Some Logistics

- **Office Hours:** 2:00 - 3:00 PM, SODA 627, Tuesday
- **Additional Office Hours:** 11:30 AM - 12:30 PM, SODA 627, Thursday

Some Logistics

- **Office Hours:** 2:00 - 3:00 PM, SODA 627, Tuesday
- **Additional Office Hours:** 11:30 AM - 12:30 PM, SODA 627, Thursday
- **First HW out: Sept 5**

Some Logistics

- **Office Hours:** 2:00 - 3:00 PM, SODA 627, Tuesday
- **Additional Office Hours:** 11:30 AM - 12:30 PM, SODA 627, Thursday
- **First HW out: Sept 5**
- **Some suggested projects (mostly survey) already out on the course website**

Some Logistics

- **Office Hours:** 2:00 - 3:00 PM, SODA 627, Tuesday
- **Additional Office Hours:** 11:30 AM - 12:30 PM, SODA 627, Thursday
- **First HW out: Sept 5**
- **Some suggested projects (mostly survey) already out on the course website**
- **Course website:** https://chen-lijie.github.io/cs278-complexity.html

Some Logistics

- **Office Hours:** 2:00 - 3:00 PM, SODA 627, Tuesday
- **Additional Office Hours:** 11:30 AM - 12:30 PM, SODA 627, Thursday
- **First HW out: Sept 5**
- **Some suggested projects (mostly survey) already out on the course website**
- **Course website:** https://chen-lijie.github.io/cs278-complexity.html
- **Course discord:** https://discord.gg/U3965mgE2p

Suggested projects: Survey

One possible type of course project is a survey of a complexity-related topic we haven't covered in class. (More details about the open direction projects coming next week.)

- You goal is very simple: pick a frontier complexity paper and understand it.

Suggested projects: Survey

One possible type of course project is a survey of a complexity-related topic we haven't covered in class. (More details about the open direction projects coming next week.)

- You goal is very simple: pick a frontier complexity paper and understand it.
- This may require reading some prior works.

Suggested projects: Survey

One possible type of course project is a survey of a complexity-related topic we haven't covered in class. (More details about the open direction projects coming next week.)

- You goal is very simple: pick a frontier complexity paper and understand it.
- This may require reading some prior works.
- Your survey should set up right context for this paper, explains the motivations, and give an overview of the main proof ideas.

Suggested projects: Survey

One possible type of course project is a survey of a complexity-related topic we haven't covered in class. (More details about the open direction projects coming next week.)

- You goal is very simple: pick a frontier complexity paper and understand it.
- This may require reading some prior works.
- Your survey should set up right context for this paper, explains the motivations, and give an overview of the main proof ideas.
- Some of the harder papers may require you to collaborate with others.

Recap: Time hierarchy theorem for deterministic time and non-deterministic time

- In the prior lecture, we shown the determnistic time and non-deterministic time hierarchy theorems.

Recap: Time hierarchy theorem for deterministic time and non-deterministic time

- In the prior lecture, we shown the determnistic time and non-deterministic time hierarchy theorems.
- For each of the proof, the idea is to construct a hard language H such that, letting $M_1, M_2, \ldots, M_k, \ldots$ be an enumeration of all Turing machines (for say DTIME$[T(n)]$), for every M_i there exists an input x_i such that $M_i(x_i) \neq H(x_i)$.

Recap: Time hierarchy theorem for deterministic time and non-deterministic time

- In the prior lecture, we shown the determnistic time and non-deterministic time hierarchy theorems.
- For each of the proof, the idea is to construct a hard language H such that, letting $M_1, M_2, \ldots, M_k, \ldots$ be an enumeration of all Turing machines (for say $\mathrm{DTIME}[T(n)]$), for every M_i there exists an input x_i such that $M_i(x_i) \neq H(x_i)$.
- DTIME hierachy theorem: $H(\langle M_i \rangle) \neq M_i(\langle M_i \rangle)$, i.e., the input x_i is the encoding of M_i itself.

Recap: Time hierarchy theorem for deterministic time and non-deterministic time

- In the prior lecture, we shown the determnistic time and non-deterministic time hierarchy theorems.
- For each of the proof, the idea is to construct a hard language H such that, letting $M_1, M_2, \ldots, M_k, \ldots$ be an enumeration of all Turing machines (for say $\mathrm{DTIME}[T(n)]$), for every M_i there exists an input x_i such that $M_i(x_i) \neq H(x_i)$.
- DTIME hierachy theorem: $H(\langle M_i \rangle) \neq M_i(\langle M_i \rangle)$, i.e., the input x_i is the encoding of M_i itself.
- NTIME hierachy theorem: $H(1^t) \neq M_i(1^t)$ for some $t \in [n_i, 2^{f(n_i)^2}]$.

infinitey often and almost everywhere separation

- You can always adding dummy states to a Turing machine (think of “adding comments” to Python code), every machine M appears in the enumeration infinitely many times.

infinitey often and almost everywhere separation

- You can always adding dummy states to a Turing machine (think of "adding comments" to Python code), every machine M appears in the enumeration infinitely many times.

- **Infinite often separation (default)**: The language L is not in $\mathrm{NTIME}[T(n)]$ if and only if for **all** $L' \in \mathrm{NTIME}[T(n)]$, for **infinitely many** input lengths n, $L_n \neq L'_n$. (here, L_n is the language L on input length n.)

infinitey often and almost everywhere separation

- You can always adding dummy states to a Turing machine (think of "adding comments" to Python code), every machine M appears in the enumeration infinitely many times.
- **Infinite often separation (default)**: The language L is not in $\mathrm{NTIME}[T(n)]$ if and only if for **all** $L' \in \mathrm{NTIME}[T(n)]$, for **infinitely many** input lengths n, $L_n \neq L'_n$. (here, L_n is the language L on input length n.)
- **Almost everywhere separation**: For **all** $L' \in \mathrm{NTIME}[T(n)]$, for **all except finitely many** n, $L_n \neq L'_n$.

infinitey often and almost everywhere separation

- You can always adding dummy states to a Turing machine (think of "adding comments" to Python code), every machine M appears in the enumeration infinitely many times.

- **Infinite often separation (default)**: The language L is not in $\mathrm{NTIME}[T(n)]$ if and only if for **all** $L' \in \mathrm{NTIME}[T(n)]$, for **infinitely many** input lengths n, $L_n \neq L'_n$. (here, L_n is the language L on input length n.)

- **Almost everywhere separation**: For **all** $L' \in \mathrm{NTIME}[T(n)]$, for **all except finitely many** n, $L_n \neq L'_n$.

- **Big open question**: prove an almost everywhere separation between $\mathrm{NTIME}[n^2]$ and $\mathrm{NTIME}[2^n]$.

Why is infinite often separation not enough?

- For the hard language H, suppose there is a NTIME$[T(n)]$ machine M such that $H(x) = M(x)$ for all x of length n, except when n is of the form $2^{2^{2^{2^k}}}$ for some $k \in \mathbb{N}$.

Why is infinite often separation not enough?

- For the hard language H, suppose there is a $\mathsf{NTIME}[T(n)]$ machine M such that $H(x) = M(x)$ for all x of length n, except when n is of the form $2^{2^{2^{2^k}}}$ for some $k \in \mathbb{N}$.
- This is allowed for infinite often separation (H can still be hard for $\mathsf{NTIME}[T(n)]$). But "practically", H is easy for $\mathsf{NTIME}[T(n)]$.

An almost-everywhere deterministic time hierarchy theorem

Theorem

- *There is a language* $L \in TIME[n^2]$ *such that for every* $L' \in TIME[n]$, *for all except finitely many* n, $L_n \neq L'_n$.

An almost-everywhere deterministic time hierarchy theorem

Theorem

- *There is a language* $L \in TIME[n^2]$ *such that for every* $L' \in TIME[n]$, *for all except finitely many* n, $L_n \neq L'_n$.
- *Same holds for* $NTIME[T(n) \cdot \log^2 T(n)]$ *and* $NTIME[T(n)]$.

An almost-everywhere deterministic time hierarchy theorem

Proof

- **paddable encoding**: For simplicity, we assume that if a TM M is encoded as a binary string $\langle M \rangle$, then $\langle M \rangle 0^t$ represents the same machine M, for any $t \in \mathbb{N}$. (i.e., we can pad the encoding with any number of 0.) Let $\langle M \rangle_n = \langle M \rangle 0^{n-|\langle M \rangle|}$.

□

An almost-everywhere deterministic time hierarchy theorem

Proof

- **paddable encoding**: For simplicity, we assume that if a TM M is encoded as a binary string $\langle M \rangle$, then $\langle M \rangle 0^t$ represents the same machine M, for any $t \in \mathbb{N}$. (i.e., we can pad the encoding with any number of 0.) Let $\langle M \rangle_n = \langle M \rangle 0^{n-|\langle M \rangle|}$.
- Let

$$H = \{\langle M \rangle_n \mid M(\langle M \rangle_n) \text{ rejects in } |\langle M \rangle_n|^{1.5} \text{ steps}, n \geqslant |\langle M \rangle|\}$$

□

An almost-everywhere deterministic time hierarchy theorem

Proof

- **paddable encoding**: For simplicity, we assume that if a TM M is encoded as a binary string $\langle M \rangle$, then $\langle M \rangle 0^t$ represents the same machine M, for any $t \in \mathbb{N}$. (i.e., we can pad the encoding with any number of 0.) Let $\langle M \rangle_n = \langle M \rangle 0^{n-|\langle M \rangle|}$.
- Let

$$H = \{\langle M \rangle_n \mid M(\langle M \rangle_n) \text{ rejects in } |\langle M \rangle_n|^{1.5} \text{ steps}, n \geqslant |\langle M \rangle|\}$$

- Can show $H \in \mathrm{TIME}[n^2]$ and, and it is almost-everywhere separated from $\mathrm{TIME}[n]$.

□

A weaker a.e. ntime hierarchy theorem

Definition (Non-deterministic time with bounded guess)

- For a function $T, G : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{NTIMEGUESS}[T(n), G(n)]$ to be the class of languages that can be decided by a non-deterministic multi-tape Turing machine in time $O(T(n))$ with making at most $G(n)$ non-deterministic guesses.

A weaker a.e. ntime hierarchy theorem

Definition (Non-deterministic time with bounded guess)

- For a function $T, G : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{NTIMEGUESS}[T(n), G(n)]$ to be the class of languages that can be decided by a non-deterministic multi-tape Turing machine in time $O(T(n))$ with making at most $G(n)$ non-deterministic guesses.

- That is, $L \in \mathrm{NTIMEGUESS}[T(n), G(n)]$ if there exists a non-deterministic multi-tape Turing machine M and a constant c such that:

A weaker a.e. ntime hierarchy theorem

Definition (Non-deterministic time with bounded guess)

- For a function $T, G : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{NTIMEGUESS}[T(n), G(n)]$ to be the class of languages that can be decided by a non-deterministic multi-tape Turing machine in time $O(T(n))$ with making at most $G(n)$ non-deterministic guesses.

- That is, $L \in \mathrm{NTIMEGUESS}[T(n), G(n)]$ if there exists a non-deterministic multi-tape Turing machine M and a constant c such that:
  - M decides L (i.e., M accepts x if and only if $x \in L$)

A weaker a.e. ntime hierarchy theorem

Definition (Non-deterministic time with bounded guess)

- For a function $T, G : \mathbb{N} \to \mathbb{N}$, we define $\text{NTIMEGUESS}[T(n), G(n)]$ to be the class of languages that can be decided by a non-deterministic multi-tape Turing machine in time $O(T(n))$ with making at most $G(n)$ non-deterministic guesses.

- That is, $L \in \text{NTIMEGUESS}[T(n), G(n)]$ if there exists a non-deterministic multi-tape Turing machine M and a constant c such that:
  - M decides L (i.e., M accepts x if and only if $x \in L$)
  - For all inputs x of length n, M halts within $c \cdot T(n)$ steps and makes at most $G(n)$ non-deterministic guesses.

A weaker a.e. ntime hierarchy theorem

Theorem (Non-deterministic time hierarchy theorem with bounded guess)

- *Let* $T, G, W\colon \mathbb{N} \to \mathbb{N}$ *be time-constructible functions such that* $G(n) = o(T(n))$ *and* $W(n) = o(n)$. *Then there is a language* $L \in NTIME[T(n)]$ *but* L *is almost-everywhere separated from* $NTIMEGUESS[G(n), W(n)]$.

A weaker a.e. ntime hierarchy theorem

Theorem (Non-deterministic time hierarchy theorem with bounded guess)

- *Let* $T, G, W\colon \mathbb{N} \to \mathbb{N}$ *be time-constructible functions such that* $G(n) = o(T(n))$ *and* $W(n) = o(n)$*. Then there is a language* $L \in NTIME[T(n)]$ *but* L *is almost-everywhere separated from* $NTIMEGUESS[G(n), W(n)]$.

- **Proof:** see the white board!

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O} : \{0, 1\}^* \to \{0, 1\}$ be a function denoted as the **oracle**.

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O} : \{0, 1\}^* \to \{0, 1\}$ be a function denoted as the **oracle**.
- An $\mathcal{O}$-oracle multi-tape Turing machine is a multi-tape Turing machine with an additional **oracle tape** and two special states q_{query} and q_{answer}.

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O} : \{0,1\}^* \to \{0,1\}$ be a function denoted as the **oracle**.
- An $\mathcal{O}$-oracle multi-tape Turing machine is a multi-tape Turing machine with an additional **oracle tape** and two special states q_{query} and q_{answer}.
- The machine can make **oracle queries** as follows:

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O} : \{0,1\}^* \to \{0,1\}$ be a function denoted as the **oracle**.

- An $\mathcal{O}$-oracle multi-tape Turing machine is a multi-tape Turing machine with an additional **oracle tape** and two special states q_{query} and q_{answer}.

- The machine can make **oracle queries** as follows:
  - Write a string y on the oracle tape and enter state q_{query}

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ be a function denoted as the **oracle**.
- An $\mathcal{O}$-oracle multi-tape Turing machine is a multi-tape Turing machine with an additional **oracle tape** and two special states q_{query} and q_{answer}.
- The machine can make **oracle queries** as follows:
  - Write a string y on the oracle tape and enter state q_{query}
  - In the next step, the machine automatically transitions to state q_{answer} and the oracle tape contains $\mathcal{O}(y)$

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ be a function denoted as the **oracle**.

- An $\mathcal{O}$-oracle multi-tape Turing machine is a multi-tape Turing machine with an additional **oracle tape** and two special states q_{query} and q_{answer}.

- The machine can make **oracle queries** as follows:
  - Write a string y on the oracle tape and enter state q_{query}
  - In the next step, the machine automatically transitions to state q_{answer} and the oracle tape contains $\mathcal{O}(y)$
  - This counts as a single computation step

Oracles and relativization

Definition (Oracle Turing Machine)

- Let $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ be a function denoted as the **oracle**.
- An $\mathcal{O}$-oracle multi-tape Turing machine is a multi-tape Turing machine with an additional **oracle tape** and two special states q_{query} and q_{answer}.
- The machine can make **oracle queries** as follows:
  - Write a string y on the oracle tape and enter state q_{query}
  - In the next step, the machine automatically transitions to state q_{answer} and the oracle tape contains $\mathcal{O}(y)$
  - This counts as a single computation step
- The running time includes all computation steps, including oracle queries.

Oracle complexity classes

Definition (Oracle complexity classes)

- For an oracle $\mathcal{O}$ and a time bound $T : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{DTIME}^{\mathcal{O}}[T(n)]$ to be the class of languages that can be decided by a deterministic $\mathcal{O}$-oracle multi-tape Turing machine in time $O(T(n))$.

Oracle complexity classes

Definition (Oracle complexity classes)

- For an oracle $\mathcal{O}$ and a time bound $T : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{DTIME}^{\mathcal{O}}[T(n)]$ to be the class of languages that can be decided by a deterministic $\mathcal{O}$-oracle multi-tape Turing machine in time $O(T(n))$.

- That is, $L \in \mathrm{DTIME}^{\mathcal{O}}[T(n)]$ if there exists a deterministic $\mathcal{O}$-oracle multi-tape Turing machine M and a constant c such that:

Oracle complexity classes

Definition (Oracle complexity classes)

- For an oracle $\mathcal{O}$ and a time bound $T : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{DTIME}^{\mathcal{O}}[T(n)]$ to be the class of languages that can be decided by a deterministic $\mathcal{O}$-oracle multi-tape Turing machine in time $O(T(n))$.

- That is, $L \in \mathrm{DTIME}^{\mathcal{O}}[T(n)]$ if there exists a deterministic $\mathcal{O}$-oracle multi-tape Turing machine M and a constant c such that:
  - $M^{\mathcal{O}}$ decides L (i.e., $M^{\mathcal{O}}$ accepts x if and only if $x \in L$)

Oracle complexity classes

Definition (Oracle complexity classes)

- For an oracle $\mathcal{O}$ and a time bound $T : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{DTIME}^{\mathcal{O}}[T(n)]$ to be the class of languages that can be decided by a deterministic $\mathcal{O}$-oracle multi-tape Turing machine in time $O(T(n))$.

- That is, $L \in \mathrm{DTIME}^{\mathcal{O}}[T(n)]$ if there exists a deterministic $\mathcal{O}$-oracle multi-tape Turing machine M and a constant c such that:
  - $M^{\mathcal{O}}$ decides L (i.e., $M^{\mathcal{O}}$ accepts x if and only if $x \in L$)
  - For all inputs x of length n, $M^{\mathcal{O}}$ halts within $c \cdot T(n)$ steps

Oracle complexity classes

Definition (Oracle complexity classes)

- For an oracle $\mathcal{O}$ and a time bound $T : \mathbb{N} \to \mathbb{N}$, we define $\mathrm{DTIME}^{\mathcal{O}}[T(n)]$ to be the class of languages that can be decided by a deterministic $\mathcal{O}$-oracle multi-tape Turing machine in time $O(T(n))$.

- That is, $L \in \mathrm{DTIME}^{\mathcal{O}}[T(n)]$ if there exists a deterministic $\mathcal{O}$-oracle multi-tape Turing machine M and a constant c such that:
  - $M^{\mathcal{O}}$ decides L (i.e., $M^{\mathcal{O}}$ accepts x if and only if $x \in L$)
  - For all inputs x of length n, $M^{\mathcal{O}}$ halts within $c \cdot T(n)$ steps

- Similarly, we can define $\mathrm{NTIME}^{\mathcal{O}}[T(n)]$ for non-deterministic $\mathcal{O}$-oracle Turing machines.

Oracles and relativization

All previous results holds for all $\mathcal{O}$-oracle Turing machines.

Theorem (Time hierarchy theorem for oracle Turing machines)

- *For any oracle $\mathcal{O}$, and any time-constructible functions t_1, t_2 with $t_1(n) \log t_1(n) = o(t_2(n))$, we have:*

$$DTIME^{\mathcal{O}}[t_1(n)] \subsetneq DTIME^{\mathcal{O}}[t_2(n)]$$

To prove this, we only need to show the existence of a universal Turing machine for $\mathcal{O}$-oracle Turing machines.

Oracles and relativization

All previous results holds for all $\mathcal{O}$-oracle Turing machines.

Theorem (Time hierarchy theorem for oracle Turing machines)

- *For any oracle $\mathcal{O}$, and any time-constructible functions t_1, t_2 with $t_1(n) \log t_1(n) = o(t_2(n))$, we have:*

$$DTIME^{\mathcal{O}}[t_1(n)] \subsetneq DTIME^{\mathcal{O}}[t_2(n)]$$

- *Similarly, for any oracle $\mathcal{O}$, and any time-constructible functions f, g with $f(n+1) = o(g(n))$, we have:*

$$NTIME^{\mathcal{O}}[f(n)] \subsetneq NTIME^{\mathcal{O}}[g(n)]$$

To prove this, we only need to show the existence of a universal Turing machine for $\mathcal{O}$-oracle Turing machines.

Time hierarchy theorem for oracle Turing machines

Theorem (Universal Oracle Turing Machine)

- *For any oracle $\mathcal{O}$, there exists a universal $\mathcal{O}$-oracle Turing machine $U^{\mathcal{O}}$ such that:*

Time hierarchy theorem for oracle Turing machines

Theorem (Universal Oracle Turing Machine)

- *For any oracle $\mathcal{O}$, there exists a universal $\mathcal{O}$-oracle Turing machine $U^{\mathcal{O}}$ such that:*
  - *$U^{\mathcal{O}}$ takes as input $\langle M, x\rangle$ where M is the description of an $\mathcal{O}$-oracle Turing machine and x is an input string*

Time hierarchy theorem for oracle Turing machines

Theorem (Universal Oracle Turing Machine)

- *For any oracle $\mathcal{O}$, there exists a universal $\mathcal{O}$-oracle Turing machine $U^{\mathcal{O}}$ such that:*
  - $U^{\mathcal{O}}$ *takes as input $\langle M, x\rangle$ where M is the description of an $\mathcal{O}$-oracle Turing machine and x is an input string*
  - $U^{\mathcal{O}}(\langle M, x\rangle) = M^{\mathcal{O}}(x)$ *(same output)*

Time hierarchy theorem for oracle Turing machines

Theorem (Universal Oracle Turing Machine)

- *For any oracle $\mathcal{O}$, there exists a universal $\mathcal{O}$-oracle Turing machine $U^{\mathcal{O}}$ such that:*
  - *$U^{\mathcal{O}}$ takes as input $\langle M, x\rangle$ where M is the description of an $\mathcal{O}$-oracle Turing machine and x is an input string*
  - *$U^{\mathcal{O}}(\langle M, x\rangle) = M^{\mathcal{O}}(x)$ (same output)*
  - *If $M^{\mathcal{O}}$ runs in time $t(n)$ on input x where $|x| = n$, then $U^{\mathcal{O}}$ runs in time $O(t(n)\log t(n))$ on input $\langle M, x\rangle$*

Time hierarchy theorem for oracle Turing machines

Theorem (Universal Oracle Turing Machine)

- *For any oracle $\mathcal{O}$, there exists a universal $\mathcal{O}$-oracle Turing machine $U^{\mathcal{O}}$ such that:*
  - *$U^{\mathcal{O}}$ takes as input $\langle M, x\rangle$ where M is the description of an $\mathcal{O}$-oracle Turing machine and x is an input string*
  - *$U^{\mathcal{O}}(\langle M, x\rangle) = M^{\mathcal{O}}(x)$ (same output)*
  - *If $M^{\mathcal{O}}$ runs in time $t(n)$ on input x where $|x| = n$, then $U^{\mathcal{O}}$ runs in time $O(t(n)\log t(n))$ on input $\langle M, x\rangle$*

- *The simulation overhead is the same as in the non-oracle case, and oracle queries are handled transparently.*

Relativization

- A **relativizing proof technique** is one that applies equally well to all oracle Turing machines, regardless of the oracle.

Relativization

- A **relativizing proof technique** is one that applies equally well to all oracle Turing machines, regardless of the oracle.
- If a theorem T can be proven using only relativizing techniques, then $T^{\mathcal{O}}$ (the relativized version of T with oracle $\mathcal{O}$) holds for *all* oracles $\mathcal{O}$.

Relativization

- A **relativizing proof technique** is one that applies equally well to all oracle Turing machines, regardless of the oracle.
- If a theorem T can be proven using only relativizing techniques, then $T^{\mathcal{O}}$ (the relativized version of T with oracle $\mathcal{O}$) holds for *all* oracles $\mathcal{O}$.
- Conversely, if there exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that $T^{\mathcal{O}_1}$ is true but $T^{\mathcal{O}_2}$ is false, then T cannot be proven using relativizing techniques alone.

Relativization

- A **relativizing proof technique** is one that applies equally well to all oracle Turing machines, regardless of the oracle.
- If a theorem T can be proven using only relativizing techniques, then $T^{\mathcal{O}}$ (the relativized version of T with oracle $\mathcal{O}$) holds for *all* oracles $\mathcal{O}$.
- Conversely, if there exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that $T^{\mathcal{O}_1}$ is true but $T^{\mathcal{O}_2}$ is false, then T cannot be proven using relativizing techniques alone.
- **Example:** The time hierarchy theorems (deterministic and non-deterministic) relativize because their proofs work for any oracle $\mathcal{O}$.

Relativization

- If the proof relativizes, it means the proof just does not “open up” the computation enough.

Relativization

- If the proof relativizes, it means the proof just does not "open up" the computation enough.
- The time hierarchy theorems relativize because their proofs is "just" some simulation-based proof, which does not really try to "analyze" the computation.

Relativization

- If the proof relativizes, it means the proof just does not "open up" the computation enough.
- The time hierarchy theorems relativize because their proofs is "just" some simulation-based proof, which does not really try to "analyze" the computation.
- A lot of early results in complexity theory are relativizing.

The Relativization Barrier for P vs NP

- The P vs NP question asks whether P = NP.

The Relativization Barrier for P vs NP

- The P vs NP question asks whether P $=$ NP.
- **Baker-Gill-Solovay Theorem (1975):** There exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that:

The Relativization Barrier for P vs NP

- The P vs NP question asks whether $P = NP$.
- **Baker-Gill-Solovay Theorem (1975):** There exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that:
  - $P^{\mathcal{O}_1} = NP^{\mathcal{O}_1}$ (P equals NP relative to oracle $\mathcal{O}_1$)

The Relativization Barrier for P vs NP

- The P vs NP question asks whether $\mathrm{P} = \mathrm{NP}$.
- **Baker-Gill-Solovay Theorem (1975):** There exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that:
  - $\mathrm{P}^{\mathcal{O}_1} = \mathrm{NP}^{\mathcal{O}_1}$ (P equals NP relative to oracle $\mathcal{O}_1$)
  - $\mathrm{P}^{\mathcal{O}_2} \neq \mathrm{NP}^{\mathcal{O}_2}$ (P does not equal NP relative to oracle $\mathcal{O}_2$)

The Relativization Barrier for P vs NP

- The P vs NP question asks whether P $=$ NP.
- **Baker-Gill-Solovay Theorem (1975):** There exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that:
  - $P^{\mathcal{O}_1} = NP^{\mathcal{O}_1}$ (P equals NP relative to oracle $\mathcal{O}_1$)
  - $P^{\mathcal{O}_2} \neq NP^{\mathcal{O}_2}$ (P does not equal NP relative to oracle $\mathcal{O}_2$)
- **Consequence:** The P vs NP question cannot be resolved using relativizing proof techniques alone.

The Relativization Barrier for P vs NP

- The P vs NP question asks whether P = NP.
- **Baker-Gill-Solovay Theorem (1975):** There exist oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ such that:
  - $P^{\mathcal{O}_1} = NP^{\mathcal{O}_1}$ (P equals NP relative to oracle $\mathcal{O}_1$)
  - $P^{\mathcal{O}_2} \neq NP^{\mathcal{O}_2}$ (P does not equal NP relative to oracle $\mathcal{O}_2$)
- **Consequence:** The P vs NP question cannot be resolved using relativizing proof techniques alone.
- This creates a **relativization barrier** — any proof technique that works equally well for all oracles cannot settle P vs NP.

Constructing $\mathcal{O}$ such that $\mathbf{P}^{\mathcal{O}} = \mathbf{NP}^{\mathcal{O}}$

- Let

$$\mathcal{O} = \{\langle M, x, t\rangle \mid M \text{ accepts input } x \text{ in } t \text{ steps}\}$$

Constructing $\mathcal{O}$ such that $\mathrm{P}^{\mathcal{O}} = \mathrm{NP}^{\mathcal{O}}$

- Let
$$\mathcal{O} = \{\langle M, x, t\rangle \mid M \text{ accepts input } x \text{ in } t \text{ steps}\}$$
- $\mathrm{P}^{\mathcal{O}}$ and $\mathrm{NP}^{\mathcal{O}}$ both equal to exponential time.

Constructing $\mathcal{O}$ such that $\mathrm{P}^{\mathcal{O}} \neq \mathrm{NP}^{\mathcal{O}}$

Definition (the OR-language $L^{\mathcal{O}}$)

For an oracle $\mathcal{O}$, let

$$L^{\mathcal{O}} := \{ 1^n \mid \exists x \in \{0,1\}^n \text{ with } \mathcal{O}(x) = 1 \}.$$

- $L^{\mathcal{O}} \in \mathrm{NP}^{\mathcal{O}}$: on input 1^n, **guess** $x \in \{0,1\}^n$ and **query** $\mathcal{O}(x)$; accept iff the answer is 1.

Constructing $\mathcal{O}$ such that $\mathrm{P}^{\mathcal{O}} \neq \mathrm{NP}^{\mathcal{O}}$

Definition (the OR-language $L^{\mathcal{O}}$)

For an oracle $\mathcal{O}$, let

$$L^{\mathcal{O}} := \{ 1^n \mid \exists x \in \{0,1\}^n \text{ with } \mathcal{O}(x) = 1 \}.$$

- $L^{\mathcal{O}} \in \mathrm{NP}^{\mathcal{O}}$: on input 1^n, **guess** $x \in \{0,1\}^n$ and **query** $\mathcal{O}(x)$; accept iff the answer is 1.
- Intuition: any $\mathrm{P}^{\mathcal{O}}$ machine on input 1^n can ask only **polynomially many** length-n oracle questions, not enough to find a hidden x s.t. $\mathcal{O}(x) = 1$.

Constructing $\mathcal{O}$ such that $\mathrm{P}^{\mathcal{O}} \neq \mathrm{NP}^{\mathcal{O}}$

Definition (the OR-language $L^{\mathcal{O}}$)

For an oracle $\mathcal{O}$, let

$$L^{\mathcal{O}} := \{ 1^n \mid \exists x \in \{0,1\}^n \text{ with } \mathcal{O}(x) = 1 \}.$$

- $L^{\mathcal{O}} \in \mathrm{NP}^{\mathcal{O}}$: on input 1^n, **guess** $x \in \{0,1\}^n$ and **query** $\mathcal{O}(x)$; accept iff the answer is 1.
- Intuition: any $\mathrm{P}^{\mathcal{O}}$ machine on input 1^n can ask only **polynomially many** length-n oracle questions, not enough to find a hidden x s.t. $\mathcal{O}(x) = 1$.
- Proof: see the white board!

Relativization Barrier and query complexity

- The point of the proof above is that any $P^{\mathcal{O}}$ machine on input 1^n can ask only **polynomially query** length-n oracle questions, this is far from enough for solving OR of 2^n bits.
- **Query complexity**: Given $N = 2^n$ bits and a function $f: \{0,1\}^N \to \{0,1\}$, the **query complexity** of f is the minimum number of queries to f that a deterministic algorithm needs to compute f on all inputs.
- What's the query complexity of OR? How about AND? MAJ?
- Query complexity lower bound implies the Relativization Barrier.

Summary for Time hierarchy theorems and Relativization Barrier

- We have seen several hierarchy theorems, they are all proved using "simulation" argument, such argument does not really "open up" the computation enough, in the sense that the proof relativizes.

Summary for Time hierarchy theorems and Relativization Barrier

- We have seen several hierarchy theorems, they are all proved using "simulation" argument, such argument does not really "open up" the computation enough, in the sense that the proof relativizes.
- Relativization Barrier: any proof technique that works equally well for all oracles cannot settle P vs NP, and many other important questions in complexity theory:

Summary for Time hierarchy theorems and Relativization Barrier

- We have seen several hierarchy theorems, they are all proved using “simulation” argument, such argument does not really “open up” the computation enough, in the sense that the proof relativizes.
- Relativization Barrier: any proof technique that works equally well for all oracles cannot settle P vs NP, and many other important questions in complexity theory:
  - NP vs coNP

Summary for Time hierarchy theorems and Relativization Barrier

- We have seen several hierarchy theorems, they are all proved using "simulation" argument, such argument does not really "open up" the computation enough, in the sense that the proof relativizes.
- Relativization Barrier: any proof technique that works equally well for all oracles cannot settle P vs NP, and many other important questions in complexity theory:
  - NP vs coNP
  - P vs PSPACE

Summary for Time hierarchy theorems and Relativization Barrier

- We have seen several hierarchy theorems, they are all proved using "simulation" argument, such argument does not really "open up" the computation enough, in the sense that the proof relativizes.
- Relativization Barrier: any proof technique that works equally well for all oracles cannot settle P vs NP, and many other important questions in complexity theory:
    - NP vs coNP
    - P vs PSPACE
    - P vs BPP

Summary for Time hierarchy theorems and Relativization Barrier

- We have seen several hierarchy theorems, they are all proved using "simulation" argument, such argument does not really "open up" the computation enough, in the sense that the proof relativizes.
- Relativization Barrier: any proof technique that works equally well for all oracles cannot settle P vs NP, and many other important questions in complexity theory:
  - NP vs coNP
  - P vs PSPACE
  - P vs BPP
  - NEXP vs BPP