Computational Complexity Theory

Lecture 4: NTM, Class co-NP and EXP; Diagonalization

Department of Computer Science,
Indian Institute of Science

NTM: An alternate characterization of NP

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.
- At every step of computation, the machine applies one of two functions δ_0 and δ_1 *arbitrarily*.

also called *nondeterministically*

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.
- At every step of computation, the machine applies one of two functions δ_0 and δ_1 *arbitrarily*.

this is different from *randomly*

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* is like a deterministic Turing machines but with two transition functions.
- It is formally defined by a tuple $(\Gamma, Q, \delta_0, \delta_1)$. It has a special state q_{accept} in addition to q_{start} and q_{halt}.
- At every step of computation, the machine applies one of two functions δ_0 and δ_1 *arbitrarily*.
- Unlike DTMs, NTMs are **not intended to be physically realizable** (because of the arbitrary nature of application of the transition functions).

Nondeterministic Turing Machines

- Definition. An NTM M *<u>accepts</u>* a string $x \in \{0,1\}^*$ iff on input x there ***<u>exists</u>*** a sequence of applications of the transition functions δ_0 and δ_1 (beginning from the start configuration) that makes M reach q_{accept}.

- Defintion. An NTM M *<u>decides</u>* a language $L \subseteq \{0,1\}^*$ if
  - M accepts $x \iff x \in L$
  - On every sequence of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt}.

Nondeterministic Turing Machines

- Definition. An NTM M *accepts* a string $x \in \{0,1\}^*$ iff on input x there ***exists*** a sequence of applications of the transition functions δ_0 and δ_1 (beginning from the start configuration) that makes M reach q_{accept}.

- Defintion. An NTM M *decides* a language $L \subseteq \{0,1\}^*$ if
  - M accepts $x \Longleftrightarrow x \in L$
  - On every sequence of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt}.

remember in this course we'll always be dealing with TMs that halt on every input.

Nondeterministic Turing Machines

- Definition. An NTM M *accepts* a string $x \in \{0,1\}^*$ iff on input x there ***exists*** a sequence of applications of the transition functions δ_0 and δ_1 (beginning from the start configuration) that makes M reach q_{accept}.

- Defintion. An NTM M *decides* L in $T(|x|)$ time if
  - M accepts x $\Longleftrightarrow$ $x \in L$
  - On <u>every sequence</u> of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt} within $T(|x|)$ steps of computation.

Class NTIME

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\ T(n)$ time on inputs of length n, where c is a constant.

Alternate characterization of NP

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c>0} NTIME\,(n^c)$.

  Proof sketch: Let L be a language in NP. Then, there's a poly-time verifier M s.t,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

Alternate characterization of NP

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c>0} NTIME(n^c)$.

Proof sketch: Let L be a language in NP. Then, there's a poly-time verifier M s.t,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

Think of an NTM M' that on input x, at first *guesses* a $u \in \{0,1\}^{p(|x|)}$ by applying δ_0 and δ_1 nondeterministically

Alternate characterization of NP

- Definition. A language L is in $\mathrm{NTIME}(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $\mathrm{NP} = \bigcup_{c>0} \mathrm{NTIME}(n^c)$.

  Proof sketch: Let L be a language in NP. Then, there's a poly-time verifier M s.t,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

.... and then simulates M on (x, u) to <u>verify</u> $M(x,u) = 1$.

Alternate characterization of NP

- **Definition.** A language L is in NTIME(T(n)) if there's an NTM M that decides L in c. T(n) time on inputs of length n, where c is a constant.

- **Theorem.** $NP = \bigcup_{c>0} NTIME(n^c)$.

  Proof sketch: Let L be in NTIME (n^c). Then, there's an NTM M' that decides L in $p(n) = O(n^c)$ time. $(|x| = n)$

Alternate characterization of NP

- Definition. A language L is in $NTIME(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $NP = \bigcup_{c>0} NTIME(n^c)$.

  Proof sketch: Let L be in $NTIME(n^c)$. Then, there's an NTM M' that decides L in $p(n) = O(n^c)$ time. $(|x| = n)$

Think of a verifier M that takes x and $u \in \{0,1\}^{p(n)}$ as input,

Alternate characterization of NP

- Definition. A language L is in $\mathrm{NTIME}(T(n))$ if there's an NTM M that decides L in $c.\,T(n)$ time on inputs of length n, where c is a constant.

- Theorem. $\mathrm{NP} = \bigcup_{c>0} \mathrm{NTIME}(n^c)$.

  Proof sketch: Let L be in $\mathrm{NTIME}(n^c)$. Then, there's an NTM M' that decides L in $p(n) = O(n^c)$ time. $(|x| = n)$

Think of a verifier M that takes x and $u \in \{0,1\}^{p(n)}$ as input, and simulates M' on x with u as the sequence of choices for applying δ_0 and δ_1 .

Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$.

  A language L is in co-NP if $\overline{L}$ is in NP.

- Example. $\overline{\mathrm{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.

Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$.
  A language L is in co-NP if $\overline{L}$ is in NP.

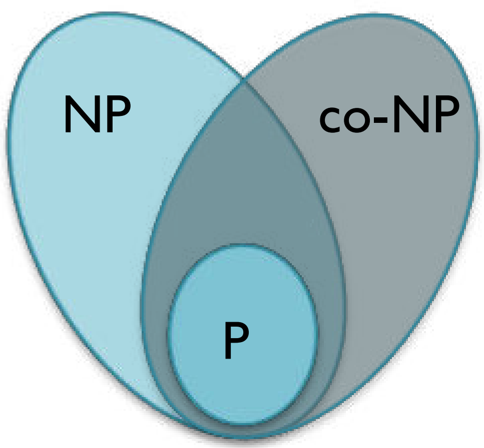
- Example. $\overline{\text{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.

- Note: co-NP is ***not*** complement of NP. Every language in P is in both NP and co-NP.

Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$.
  A language L is in co-NP if $\overline{L}$ is in NP.

- Example. $\overline{\text{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.



Class co-NP

- Definition. For every $L \subseteq \{0,1\}^*$ let $\overline{L} = \{0,1\}^* \setminus L$. A language L is in co-NP if $\overline{L}$ is in NP.

- Example. $\overline{\text{SAT}} = \{\phi : \phi$ is *not* satisfiable$\}$.

- Note: $\overline{\text{SAT}}$ is Cook reducible to SAT. But, there's a fundamental difference between the two problems that is captured by the fact that $\overline{\text{SAT}}$ is not known to be Karp reducible to SAT. In other words, there's no known poly-time verification process for $\overline{\text{SAT}}$.

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

$$x \in \bar{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 0$$

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \overline{M}(x,u) = 1$$

$\overline{M}$ outputs the *opposite* of M

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0$$

$$x \in \overline{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \overline{M}(x,u) = 1$$

$\overline{M}$ is a poly-time TM

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

$$x \in \bar{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0$$

$$x \in \bar{L} \iff \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \bar{M}(x,u) = 1$$

is in co-NP

Class co-NP : Alternate definition

- Recall, a language $L \subseteq \{0,1\}^*$ is in NP if there's a *poly-time verifier* M such that

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$
$$x \in \bar{L} \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0$$
$$x \in \bar{L} \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } \bar{M}(x,u) = 1$$

- Definition. A language $L \subseteq \{0,1\}^*$ is in co-NP if there's a *poly-time TM* M such that

$$x \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

for NP this was $\exists$

co-NP-completeness

- Definition. A language L' $\subseteq \{0,1\}^*$ is *co-NP-complete* if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{\text{SAT}}$ is co-NP-complete.

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is *co-NP-complete* if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{\text{SAT}}$ is co-NP-complete.

  Proof. Let $L \in$ co-NP. Then

  $\overline{L} \in$ NP

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is *co-NP-complete* if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{SAT}$ is co-NP-complete.

Proof. Let $L \in$ co-NP. Then

$$\overline{L} \in NP$$

$$\Longrightarrow \overline{L} \leq_p SAT$$

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is *co-NP-complete* if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. $\overline{SAT}$ is co-NP-complete.

Proof. Let $L \in$ co-NP. Then

$$\overline{L} \in NP$$

$$\Rightarrow \overline{L} \leq_p SAT$$

$$\Rightarrow L \leq_p \overline{SAT}$$

co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is *co-NP-complete* if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. Let

$$\text{TAUTOLOGY} = \{\phi : \text{every assignment satisfies } \phi\}.$$

TAUTOLOGY is co-NP complete.

Proof. Similar (homework)

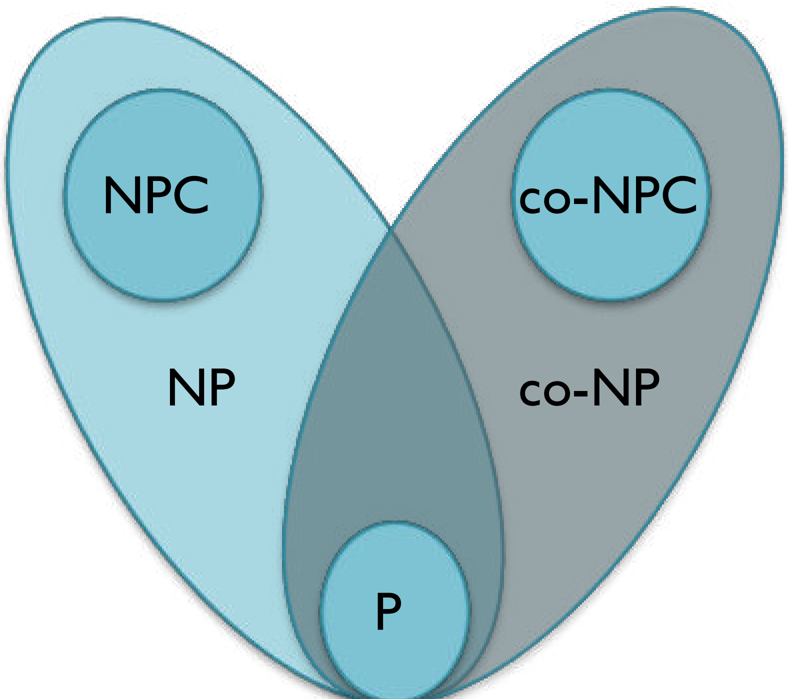
co-NP-completeness

- Definition. A language $L' \subseteq \{0,1\}^*$ is *co-NP-complete* if
  - L' is in co-NP
  - Every language L in co-NP is polynomial-time (Karp) reducible to L'.

- Theorem. If L in NP-complete then $\overline{L}$ is co-NP-complete

  Proof. Similar (homework)

The diagram again



If a co-NP complete language belongs to NP then

$$\text{co-NP} \subseteq \text{NP}$$

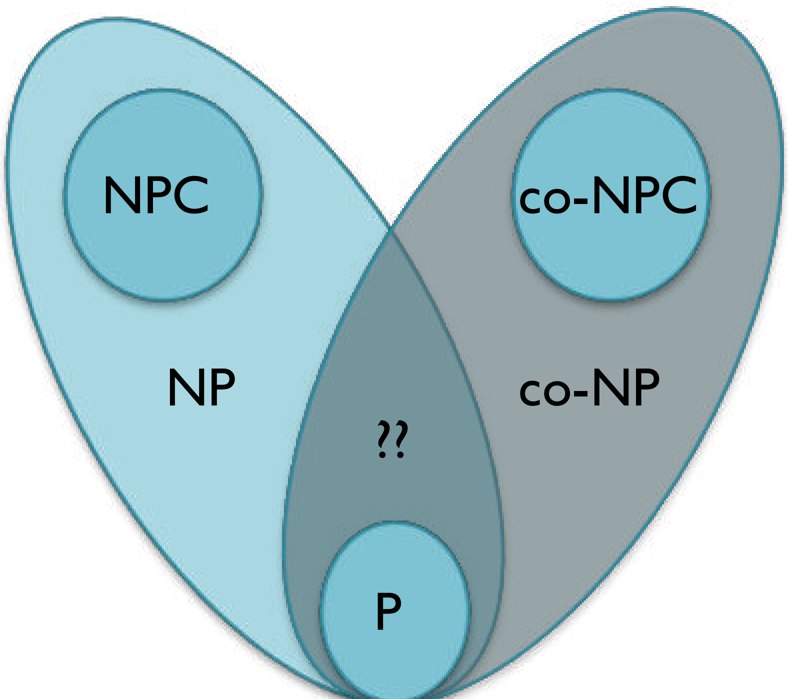
$$\Rightarrow \text{co-NP} = \text{NP}$$

Let C_1 and C_2 be two complexity classes.

If $C_1 \subseteq C_2$, then $\text{co-}C_1 \subseteq \text{co-}C_2$.

Obs. $\text{co-}(\text{co-}C) = C$.

The diagram again



If a co-NP complete language belongs to NP then

$$\text{co-NP} \subseteq \text{NP}$$
$$\Rightarrow \text{co-NP} = \text{NP}$$

Let C_1 and C_2 be two complexity classes.

If $C_1 \subseteq C_2$, then $\text{co-}C_1 \subseteq \text{co-}C_2$.

Obs. $\text{co-}(\text{co-}C) = C$.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP

- So, FACT is NP-complete implies NP = co-NP.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP
- Proof. FACT ∈ NP : Give p as a certificate. The verifier checks if p is prime (AKS test), $1 \leq p \leq U$ and p divides N.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP
- Proof. FACT ∈ NP : Give complete prime factorization of N as a certificate. The verifier checks the correctness of the factorization, and then checks if none of the prime factors is in [U].

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Claim. FACT ∈ NP ∩ co-NP
- Proof. FACT ∈ NP : Give complete prime factorization of N as a certificate. The verifier checks the correctness of the factorization, and then checks if none of the prime factors is in [U].

- Homework: If FACT ∈ P, then there's a algorithm to find the prime factorization a given n-bit integers in poly(n) time.

Integer factoring in NP ∩ co-NP

- Integer factoring.

  FACT = {(N, U): there's a prime in [U] dividing N}

- Factoring algorithm. Dixon's randomized algorithm factors an n-bit number in $\exp(O(\sqrt{n} \log n))$ time.

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$\mathrm{EXP} = \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c}\right)$$

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$\text{EXP} = \bigcup_{c \geq 1} \text{DTIME}\left(2^{n^c}\right)$$

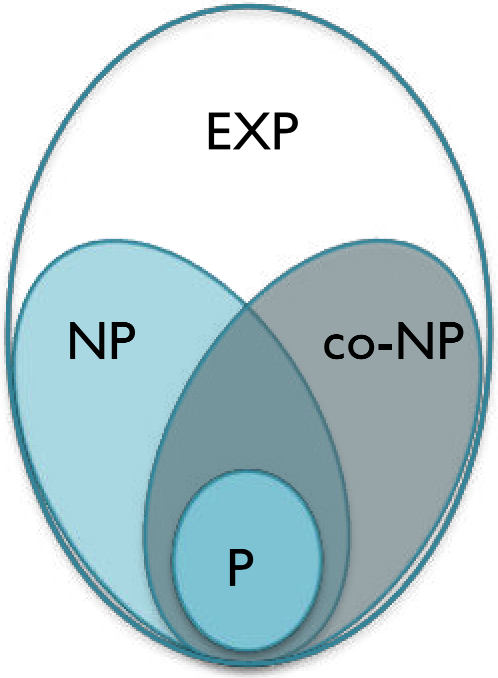
- Observation. P $\subseteq$ NP $\subseteq$ EXP

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$EXP = \bigcup_{c \geq 1} DTIME(2^{n^c})$$

- Observation. $P \subseteq NP \subseteq EXP$



Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$\mathrm{EXP} = \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c}\right)$$

- Observation. $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$

- Exponential Time Hypothesis. *(Impagliazzo & Paturi 1999)* Any algorithm for 3-SAT takes $\geq 2^{\delta . n}$ time, where $\delta > 0$ is some fixed constant and n is the no. of variables.

In other words, δ cannot be made arbitrarily close to 0.

Class EXP

- Definition. Class EXP is the exponential time analogue of class P.

$$EXP = \bigcup_{c \geq 1} DTIME\left(2^{n^c}\right)$$

- Observation. $P \subseteq NP \subseteq EXP$

- Exponential Time Hypothesis. *(Impagliazzo & Paturi 1999)* Any algorithm for 3-SAT takes $\geq 2^{\delta . n}$ time, where $\delta > 0$ is some fixed constant and n is the no. of variables.

ETH $\Rightarrow$ $P \neq NP$

Diagonalization

Diagonalization

- *Diagonalization* refers to a class of techniques used in complexity theory to separate complexity classes.

Diagonalization

- *Diagonalization* refers to a class of techniques used in complexity theory to separate complexity classes.
- These techniques are characterized by <u>two</u> main features:

Diagonalization

- *Diagonalization* refers to a class of techniques used in complexity theory to separate complexity classes.
- These techniques are characterized by <u>two</u> main features:
    1. There's a universal TM U that when given strings α and x, simulates M_α on x with only a <u>*small*</u> overhead.

Diagonalization

- *Diagonalization* refers to a class of techniques used in complexity theory to separate complexity classes.
- These techniques are characterized by <u>two</u> main features:

  1. There's a universal TM U that when given strings α and x, simulates M_α on x with only a *<u>small</u>* overhead.

     If M_α takes T time on x then U takes $O(T \log T)$ time to simulate M_α on x.

Diagonalization

- *Diagonalization* refers to a class of techniques used in complexity theory to separate complexity classes.
- These techniques are characterized by <u>two</u> main features:

    1. There's a universal TM U that when given strings α and x, simulates M_α on x with only a <u>*small*</u> overhead.

    2. Every string represents some TM, and every TM can be represented by <u>*infinitely many*</u> strings.

Time Hierarchy Theorem

- An application of Diagonalization

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be <u>time-constructible</u> functions s.t.,

  $f(n)\ .\log f(n) = o(g(n))$. e.g. $f(n) = n,\ g(n) = n^2$

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  Task: Show that there's a language L decided by a TM D with time complexity $O(n^2)$ s.t., any TM M with runtime $O(n)$ cannot decide L.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  TM D :

  1. On input x, compute $|x|^2$.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  TM D :

  1. On input x, compute $|x|^2$.
  2. Simulate M_x on x for $|x|^2$ steps.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  TM D :

  1. On input x, compute $|x|^2$.
  2. Simulate M_x on x for $|x|^2$ steps.

D's time steps not M_x's time steps.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

- Theorem. $\text{DTIME}(f(n)) \subsetneq \text{DTIME}(g(n))$

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  TM D :

  1. On input x, compute $|x|^2$.
  2. Simulate M_x on x for $|x|^2$ steps.
     a. If M_x stops and outputs b then output $1\text{-}b$

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  TM D :

  1. On input x, compute $|x|^2$.
  2. Simulate M_x on x for $|x|^2$ steps.
     a. If M_x stops and outputs b then output 1-b.
     b. Otherwise, output 1.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  D runs in $O(n^2)$ time as n^2 is <u>time-constructible</u>.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  Claim. There's no TM M with running time $O(n)$ that decides L (the language accepted by D).

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.

    c is a constant

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a <u>sufficiently large</u> x such that $M = M_x$.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a <u>sufficiently large</u> x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a <u>sufficiently large</u> x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.
  - D on input x, simulates M_x on x for $|x|^2$ steps.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a <u>sufficiently large</u> x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.
  - D on input x, simulates M_x on x for $|x|^2$ steps. Since M_x stops within $c.|x|$ steps, D's simulation also stops within $c'.c.\, |x|.\log |x|$ steps.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

- Theorem. $DTIME(f(n)) \subsetneq DTIME(g(n))$

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a <u>sufficiently large</u> x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.
  - D on input x, simulates M_x on x for $|x|^2$ steps. Since M_x stops within $c.|x|$ steps, D's simulation also stops within $c'.c.\, |x|.\, \log |x|$ steps.

c' is a constant

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a <u>sufficiently large</u> x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.
  - D on input x, simulates M_x on x for $|x|^2$ steps. Since M_x stops within $c.|x|$ steps, D's simulation also stops within $c'.c.\, |x|.\log |x|$ steps. (as $c'.c.\, |x|.\log |x| < |x|^2$ for <u>sufficiently large</u> x)

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

- For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
- Think of a sufficiently large x such that $M = M_x$.
- Suppose $M(x) = M_x(x) = b$.
- D on input x, simulates M_x on x for $|x|^2$ steps. Since M_x stops within $c.|x|$ steps, D's simulation also stops within $c'.c.\,|x|.\log|x|$ steps. And D outputs the opposite of what M_x outputs!

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

- Theorem. $DTIME(f(n)) \subsetneq DTIME(g(n))$

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a sufficiently large x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.
  - Hence, $D(x) = 1\text{-}b$.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

  Proof. We'll prove with $f(n) = n$ and $g(n) = n^2$.

  - For contradiction, suppose M decides L and runs for at most $c.n$ steps on inputs of length n.
  - Think of a sufficiently large x such that $M = M_x$.
  - Suppose $M(x) = M_x(x) = b$.
  - Hence, $D(x) = 1\text{-}b$.

Contradiction! M does not decide L.

Time Hierarchy Theorem

- Let $f(n)$ and $g(n)$ be time-constructible functions s.t.,

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

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

- Theorem. $\mathrm{P} \subsetneq \mathrm{EXP}$

  Proof. Similar (homework)

Time Hierarchy Theorem

- Are there natural problems that take $\Omega(n^2)$ time?

Time Hierarchy Theorem

- Are there natural problems that take $\Omega(n^2)$ time?

- 3SUM: Given a list of n numbers, check if there exists 3 numbers in the list that sum to zero.

Time Hierarchy Theorem

- Are there natural problems that take $\Omega(n^2)$ time?

- 3SUM: Given a list of n numbers, check if there exists 3 numbers in the list that sum to zero.
- Conjecture. ***No*** algorithm solves 3SUM in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$.

Time Hierarchy Theorem

- Are there natural problems that take $\Omega(n^2)$ time?

- 3SUM: Given a list of n numbers, check if there exists 3 numbers in the list that sum to zero.
- Conjecture. ***No*** algorithm solves 3SUM in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$.
- kSUM: Given a list of n numbers, check if there exists k numbers in the list that sum to zero.

Time Hierarchy Theorem

- Are there natural problems that take $\Omega(n^2)$ time?

- 3SUM: Given a list of n numbers, check if there exists 3 numbers in the list that sum to zero.
- Conjecture. ***No*** algorithm solves 3SUM in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$.
- kSUM: Given a list of n numbers, check if there exists k numbers in the list that sum to zero.
- Theorem *(Patrascu & Williams 2010)*. ETH implies kSUM requires $n^{\Omega(k)}$ time.