Computational Complexity Theory

Lecture 5: Class co-NP and EXP; Diagonalization

Indian Institute of Science

Recap: Alternate definition of NP

- 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}.

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

Recap: Search versus Decision for NP

- Theorem. Let $L \subseteq \{0,1\}^*$ be NP-complete. Then, the search version of L can be solved in poly-time if and only if the decision version can be solved in poly-time.

- Theorem. *(Bellare-Goldwasser)* If EE $\neq$ NEE then there's a language in NP for which search does not reduce to decision.

- Sometimes, the decision version of a problem can be trivial but the search version is possibly hard. E.g. Computing Nash Equilibrium (see class PPAD).

Two types of poly-time reductions

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is <u>*polynomial time (Karp / many-one) reducible*</u> to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial time computable function f s.t.

$$x \in L_1 \iff f(x) \in L_2$$

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is <u>*polynomial time (Cook / Turing) reducible*</u> to a language $L_2 \subseteq \{0,1\}^*$ if there's a TM that decides L_1 using poly many calls to a

Two types of poly-time reductions

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is <u>*polynomial time (Karp / many-one) reducible*</u> to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial time computable function f s.t.

$$x \in L_1 \iff f(x) \in L_2$$

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is <u>*polynomial time (Cook / Turing) reducible*</u> to a language $L_2 \subseteq \{0,1\}^*$ if there's a TM that decides L_1 using poly many calls to a

Will be called an Oracle later

Two types of poly-time reductions

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is *polynomial time (Karp / many-one) reducible* to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial time computable function f s.t.

$$x \in L_1 \qquad f(x) \in L_2$$

- Definition. A language $L_1 \subseteq \{0,1\}^*$ is *polynomial time (Cook / Turing) reducible* to a language $L_2 \subseteq \{0,1\}^*$ if there's a TM that decides L_1 using poly many calls to a

Karp reducible implies Cook reducible

Class co-NP

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

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

- Example. $\overline{SAT} = \{\phi : \phi$ 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.** $\overline{SAT} = \{\phi : \phi$ is <u>*not*</u> satisfiable$\}$.

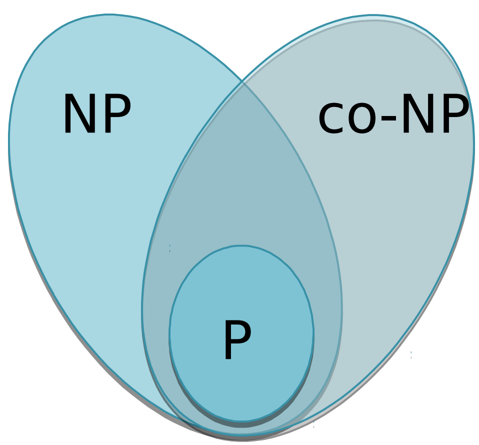
- **Note:** co-NP is <u>***not***</u> complement of NP. Every language in P is in both NP and co-NP.

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. $\overline{SAT} = \{\phi : \phi$ 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. $\overline{\text{SAT}}$ = {ϕ : ϕ 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

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

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

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

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

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

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

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

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

$\overline{M}$ 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

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

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

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

$\overline{M}$ 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

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

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

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

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

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

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

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

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

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

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. 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. SAT is co-NP-complete.

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

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

  Proof. Let $\overline{L} \in$ co-NP. Then

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

  $L \leq_p SAT$

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

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

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

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

  $L \le_p \overline{\text{SAT}}$

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)

Class EXP

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

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

Class EXP

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

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

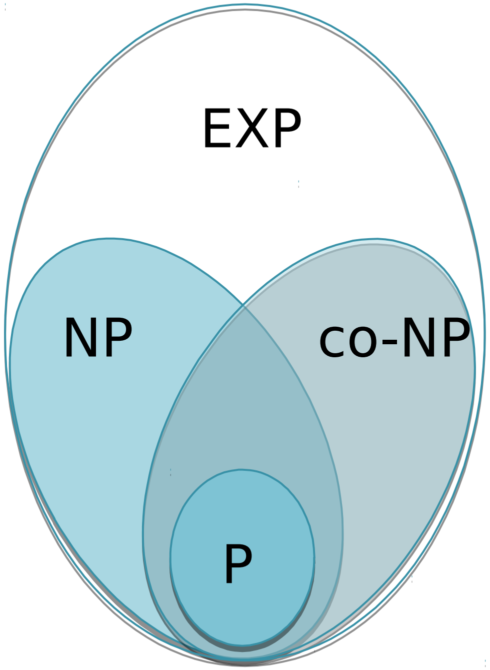
- Observation. P ⊆ NP ⊆ EXP

Class EXP

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

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

- 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

- Exponential Time Hypothesis. *(Impagliazzo & Paturi)* Any algorithm for 3-SAT takes time $\Omega(2^{\delta.n})$, where δ is a constant and n is the input size.

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 two main features:
  1. There's a universal TM U that when given strings α and x, simulates M_α on x with only a *small* 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 two main features:
    1. There's a universal TM U that when given strings α and x, simulates M_α on x with only a *small* overhead.
    2. Every string represents some TM, and every TM can be represented by infinitely many 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.,

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

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

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

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 $g(n) = n^2$.

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

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 $g(n) = n^2$.

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

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.
   a. If M_x stops and outputs b then output $1-b$

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 $1\text{-}b$

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 $g(n) = n^2$.

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

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)) ⊊ 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 $M_x(x) = b$

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 $M_x(x) = b$
- 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 $M_x(x) = b$
- 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. $\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$

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 $M_x(x) = b$
  - 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 $M_x(x) = b$
  - 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 <u>sufficiently large</u> x such that $M = M_x$
  - Suppose $M_x(x) = b$
  - 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 <u>sufficiently large</u> x such that $M = M_x$
- Suppose $M_x(x) = b$
- 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 <u>sufficiently large</u> x such that $M = M_x$
- Suppose $M_x(x) = b$
- 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.,

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

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

- Theorem. $P \subsetneq EXP$

  Proof. Similar (homework)

Ladner’s Theorem

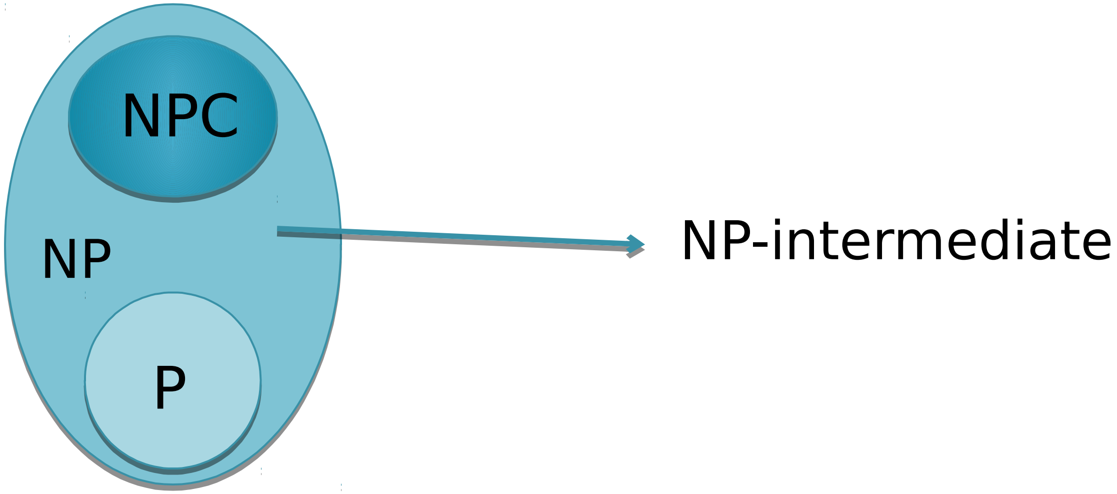
- Another application of Diagonalization

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

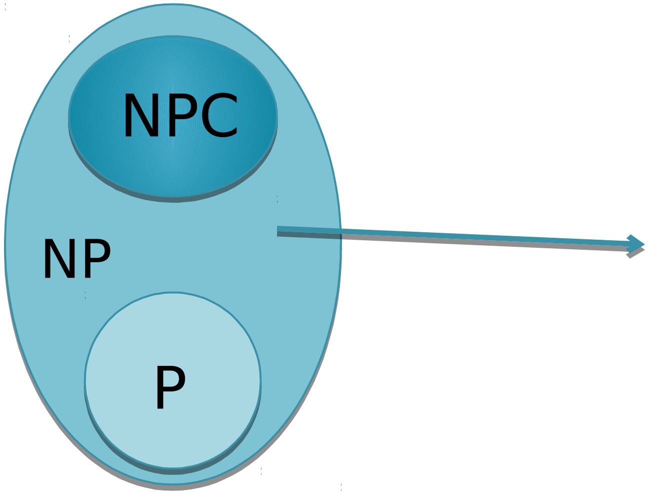
NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.



NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.



NP-intermediate (integer factoring, graph isomorphism ??)

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

… the notion makes sense only if P ≠ NP

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

- Theorem. *(Ladner)* If $P \neq NP$ then there is an NP-intermediate language.

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

- Theorem. *(Ladner)* If P $\neq$ NP then there is an NP-intermediate language.

  Proof. A delicate argument using diagonalization.

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

- Theorem. *(Ladner)* If $P \neq NP$ then there is an NP-intermediate language.

Proof. Let $H: \mathbb{N} \rightarrow \mathbb{N}$ be a function.

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

- Theorem. *(Ladner)* If $P \neq NP$ then there is an NP-intermediate language.

Proof. Let $H: N \to N$ be a function.

$m^{H(m)}$

Let $SAT_H = \{\Psi 0\, 1 \quad : \Psi \in SAT$ and $|\Psi| = m\}$

NP-intermediate problems

- Definition. A language L in NP is *NP-intermediate* if L is neither in P nor NP-complete.

- Theorem. *(Ladner)* If $P \neq NP$ then there is an NP-intermediate language.

Proof. Let $H: \mathbb{N} \rightarrow \mathbb{N}$ be a function.

Let $SAT_H = \{\Psi 0 1^{m^{H(m)}} : \Psi \in SAT \text{ and } |\Psi| = m\}$

H would be defined in such a way that SAT_H is NP-intermediate

(assuming $P \neq NP$)

Ladner's theorem: Constructing H

- Theorem. There's a function $H: N \rightarrow N$ such that

  1. $H(m)$ is computable from m in $O(m^3)$ time

Ladner's theorem: Constructing H

- Theorem. There's a function $H: \mathbb{N} \rightarrow \mathbb{N}$ such that

1. $H(m)$ is computable from m in $O(m^3)$ time
2. $SAT_H \in P \iff H(m) \leq C$ (a constant) for every m

Ladner's theorem: Constructing H

- Theorem. There's a function $H: \mathbb{N} \to \mathbb{N}$ such that

1. $H(m)$ is computable from m in $O(m^3)$ time
2. $SAT_H \in P \iff H(m) \leq C$ (a constant)
3. If $SAT_H \notin P$ then $H(m) \to \infty$ with m

Ladner's theorem: Constructing H

- Theorem. There's a function $H: N \rightarrow N$ such that

1. $H(m)$ is computable from m in $O(m^3)$ time

2. $SAT_H \in P \iff H(m) \leq C$ (a constant)

3. If $SAT_H \notin P$ then $H(m) \rightarrow \infty$ with m

Proof: Later (uses diagonalization).

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$. Then $H(m) \leq C$.

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$. Then $H(m) \leq C$.
- This implies a poly-time algorithm for SAT as follows:

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$. Then $H(m) \leq C$.
- This implies a poly-time algorithm for SAT as follows:
  - On input ϕ , find $m = |\phi|$.

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$. Then $H(m) \leq C$.
- This implies a poly-time algorithm for SAT as follows:
  - On input ϕ, find $m = |\phi|$.

    $m^{H(m)}$

  - Compute $H(m)$, and construct the string $\phi\ 0\ 1$

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$.   Then $H(m) \leq C$.
- This implies a poly-time algorithm for SAT as follows:
  - On input ϕ , find $m = |\phi|$.

    $m^{H(m)}$
  - Compute $H(m)$, $m^{H(m)}$ and construct the string $\phi\ 0\ 1$
  - Check if $\phi\ 0\ 1$ belongs to SAT_H

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$. Then $H(m) \leq C$.
- This implies a poly-time algorithm for SAT as follows:
  - On input ϕ , find $m = |\phi|$.
  - Compute $H(m)$, and construct the string $\phi\ 0\ 1^{m^{H(m)}}$
  - Check if $\underbrace{\phi\ 0\ 1^{m^{H(m)}}}_{\text{length at most } m + 1 + m^c}$ belongs to SAT_H

Ladner's theorem: Proof

$P \neq NP$

- Suppose $SAT_H \in P$. Then $H(m) \leq C$.
- This implies a poly-time algorithm for SAT as follows:
  - On input ϕ, find $m = |\phi|$.
  - Compute $H(m)$, and construct the string $\phi\, 0\, 1^{m^{H(m)}}$
  - Check if $\phi\, 0\, 1^{m^{H(m)}}$ belongs to SAT_H
- As $P \neq NP$, it must be that $SAT_H \notin P$.

Ladner’s theorem: Proof

$$P \neq NP$$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$ $\qquad \phi \stackrel{f}{\longmapsto} \psi\, 0\, 1^k$

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$

$$\underbrace{\phi}_{|\phi| = n} \xmapsto{f} \underbrace{\Psi\ 0\ 1^k}_{|\Psi\ 0\ 1^k| = n^c}$$

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$

$\phi \xrightarrow{f} \Psi\, 0\, 1^k$

> On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT: $SAT \leq_p SAT_H$ $\qquad \phi \overset{f}{\longmapsto} \Psi\ 0\ 1^k$

  - On input ϕ, compute $f(\phi) = \Psi\ 0\ 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.

Ladner's theorem: Proof

P $\neq$ NP

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$ $\qquad \phi \overset{f}{\longmapsto} \Psi\, 0\, 1^k$

➢ On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.

➢ Compute $H(m)$ and check if $k = m^{H(m)}$.

Either m is small (in which case the task reduces to checking if a small Ψ is satisfiable),

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$

$\phi \overset{f}{\longmapsto} \Psi\, 0\, 1^k$

- On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
- Compute $H(m)$ and check if $k = m^{H(m)}$.

Either m is small (in which case the task reduces to checking if a small Ψ is satisfiable), or $H(m) > 2c$ (as $H(m)$ tends to infinity with m).

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$ $\qquad \phi \xmapsto{f} \Psi\, 0\, 1^k$

  - On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, w.l.o.g. $|f(\phi)| \geq m^{2c}$

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$ $\qquad \phi \xrightarrow{f} \Psi\, 0\, 1^k$

  - On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, w.l.o.g. $n^c = |f(\phi)| \geq m^{2c}$

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$$SAT \leq_p SAT_H \qquad \phi \xmapsto{f} \Psi\, 0\, 1^k$$

  - On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, $\sqrt{n} \geq m$

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$ $\qquad \phi \xmapsto{f} \Psi\, 0\, 1^k$

- On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
- Compute $H(m)$ and check if $k = m^{H(m)}$.
- Hence, $\sqrt{n} \geq m$. Also $\phi \in SAT$ iff $\Psi \in SAT$

Ladner’s theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$

$$\phi \xmapsto{f} \Psi\, 0\, 1^k$$

  - On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, $\sqrt{n} \geq m$. Also $\phi \in SAT$ iff $\Psi \in SAT$

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$

$\phi \xrightarrow{f} \Psi\ 0\ 1^k$

  - On input ϕ, compute $f(\phi) = \Psi\ 0\ 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, $\sqrt{n} \geq m$. Also $\phi \in SAT$ iff $\Psi \in SAT$

Thus, checking if an n-size formula ϕ is satisfiable reduces to checking if a $\sqrt{n}$-size formula Ψ is satisfiable.

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

$SAT \leq_p SAT_H$ $\qquad \phi \xmapsto{f} \Psi\, 0\, 1^k$

  - On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, $\sqrt{n} \geq m$. Also $\phi \in SAT$ iff $\Psi \in SAT$

Do this recursively! Only $O(\log \log n)$ recursive steps required.

Ladner's theorem: Proof

$P \neq NP$

- Suppose SAT_H is NP-complete. Then $H(m) \to \infty$ with m.
- This also implies a poly-time algorithm for SAT:

  $SAT \leq_p SAT_H$ $\qquad \phi \xmapsto{f} \Psi\ 0\ 1^k$

  - On input ϕ, compute $f(\phi) = \Psi\ 0\ 1^k$. Let $m = |\Psi|$.
  - Compute $H(m)$ and check if $k = m^{H(m)}$.
  - Hence, $\sqrt{n} \geq m$. Also $\phi \in SAT$ iff $\Psi \in SAT$

- Hence SAT_H is not NP-complete, as $P \neq NP$.