Computational Complexity Theory

Lecture 7: Time Hierarchy Theorem; Ladner's Theorem

Department of Computer Science,
Indian Institute of Science

Recap: 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).

Recap: 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 ⟷ $x \in L$
  - On every sequence of applications of the transition functions on input x, M either reaches q_{accept} or q_{halt} within $T(|x|)$ steps of computation.

Recap: 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.

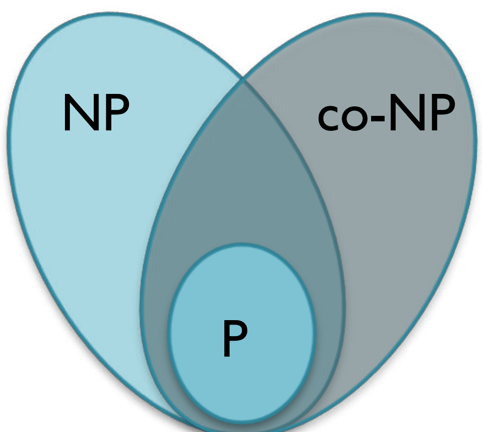
- **Theorem.** $NP = \bigcup_{c > 0} NTIME(n^c)$.

Recap: 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$\}$.



Recap: Alternate definition of co-NP

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

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

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

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

- **Definition.** A language $L \subseteq \{0,1\}^*$ is in co-NP if there's a *poly function* p and 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 $\exists$

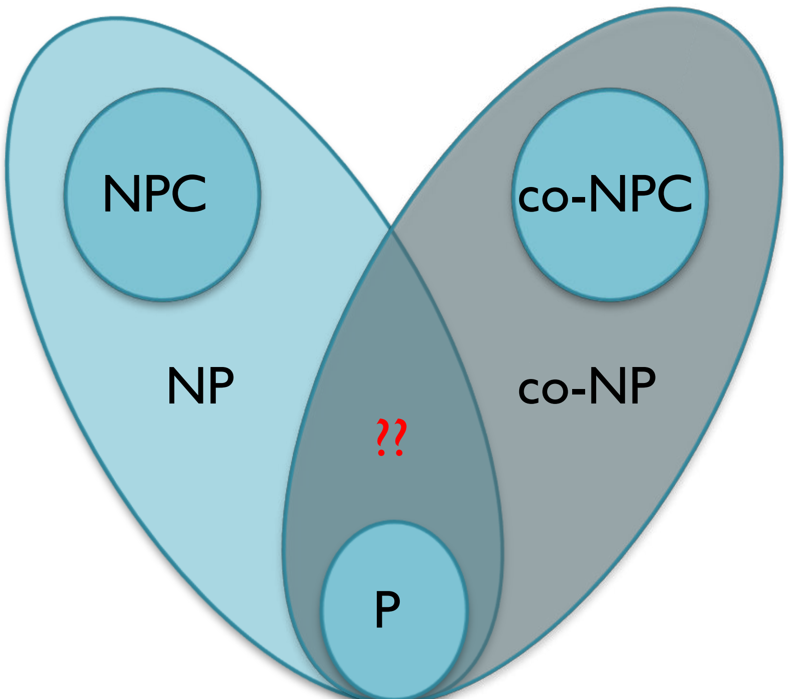
Recap: co-NP-completeness

- 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. $\overline{\text{SAT}}$ and TAUTOLOGY are co-NP-complete.

- Theorem. If L in NP-complete then L is co-NP-complete

Recap: co-NPC not in NP



If a co-NP-complete language belongs to NP then

co-NP $\subseteq$ NP

→ co-NP = NP

Let C_1 and C_2 be two complexity classes.

If $C_1 \subseteq C_2$, then co-$C_1 \subseteq$ co-C_2.

Obs. co-(co-C) = C .

Recap: 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.

- FACT not known to be in P.

Recap: 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 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 ➡ $P \neq NP$

Recap: Class EXP

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

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

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

Is $P \subsetneq EXP$?

- *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 ⇒ $P \neq NP$

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 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.,

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

- Theorem. *(Hartmanis & Stearns 1965)*

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

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

- This type of results are called **<u>lower bounds</u>**.

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.,

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

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

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

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

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

  TM D :

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

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 $g(n) = n^2$.

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

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

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.
    a. If M_x stops and outputs b then output $1\text{-}b$.
    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 $g(n) = n^2$.

  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.

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

  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.,

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

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

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

  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. (i.e., $M(x) = D(x)$ for all x)

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

  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. (i.e., $M(x) = D(x)$ for all x)

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. (i.e., $M(x) = D(x)$ for all x)
  - 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. $DTIME(f(n)) \subsetneq DTIME(g(n))$

  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. (i.e., $M(x) = D(x)$ for all x)
  - 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. $DTIME(f(n)) \subsetneq DTIME(g(n))$

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. (i.e., $M(x) = D(x)$ for all x)
  - 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. $DTIME(f(n)) \subsetneq DTIME(g(n))$

  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. (i.e., $M(x) = D(x)$ for all x)
  - 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.

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

  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. (i.e., $M(x) = D(x)$ for all x)
  - 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. (i.e., $M(x) = D(x)$ for all x)
  - 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. 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. (i.e., $M(x) = D(x)$ for all x)
  - Think of a <u>sufficiently large</u> 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. (i.e., $M(x) = D(x)$ for all x)
  - 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.,

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

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

- Theorem. $P \subsetneq EXP$

  Proof. Similar (homework)

Time Hierarchy Theorem

- Is there a natural problem that takes close to n^2 time?

Time Hierarchy Theorem

- Is there a natural problem that takes close to 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

- Is there a natural problem that takes close to 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 constant $\varepsilon > 0$.

Time Hierarchy Theorem

- Is there a natural problem that takes close to 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 constant $\varepsilon > 0$.

- However, there's a $\sim O(n^2 / (\log n)^2)$ time algorithm for 3SUM. ("~" suppressing a poly(log log n) factor.)

Time Hierarchy Theorem

- Is there a natural problem that takes close to n^2 time?

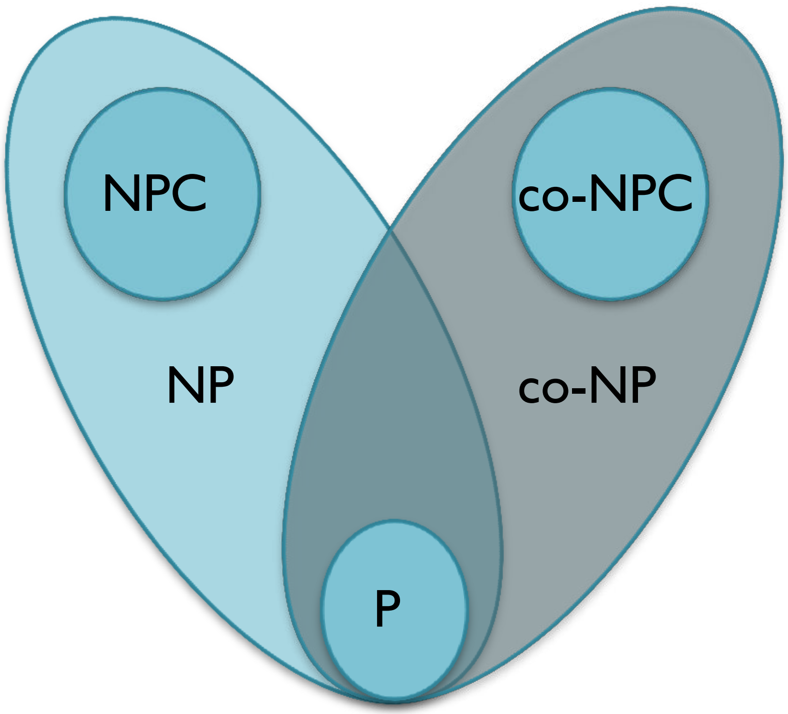
- 3SUM: Given a list of n numbers, check if there exists 3 numbers in the list that sum to zero.
- Conjecture. ***No*** algorithm solves 3SUM in $O(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$.
- kSUM: Given a list of n numbers, check if there exists k numbers in the list that sum to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

Time Hierarchy Theorem

- Is there a natural problem that takes close to 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 constant $\varepsilon > 0$.
- 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.

Revisiting NP∩co-NP



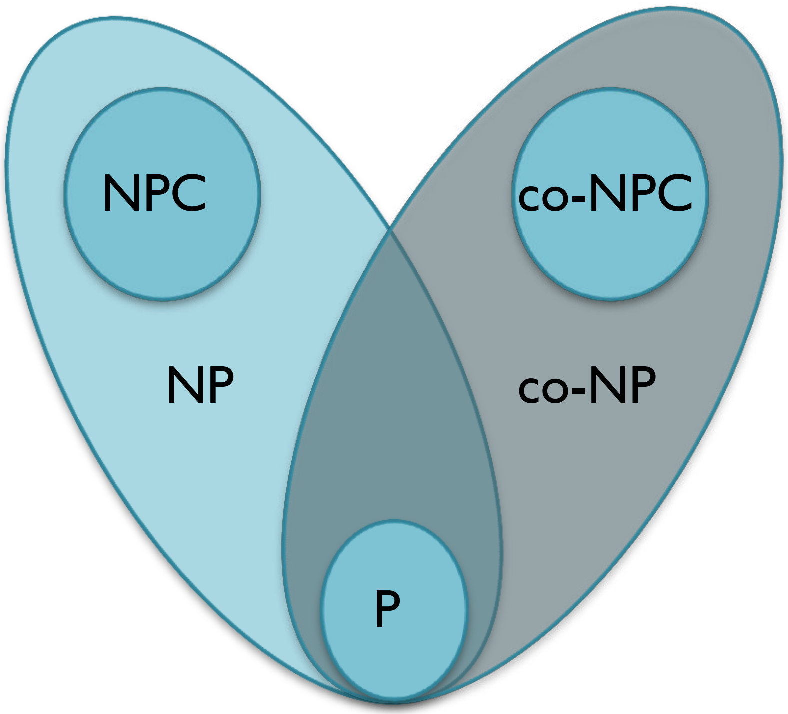
Conjecture: NP ≠ co-NP

⇓

P ≠ NP

General belief: P ≠ NP ∩ co-NP

Revisiting NP∩co-NP



Conjecture: NP ≠ co-NP

⬇

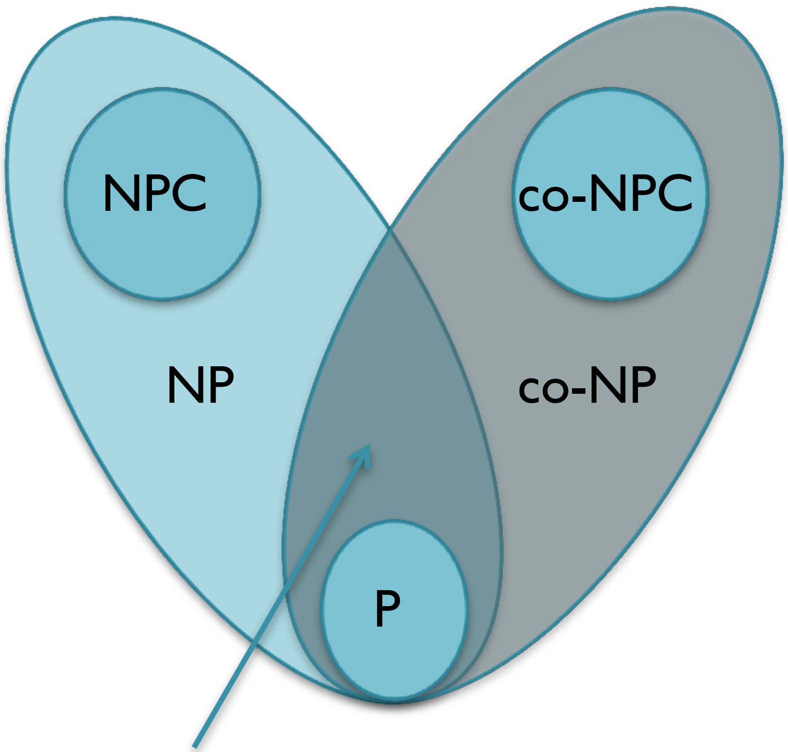
P ≠ NP

General belief: P ≠ NP ∩ co-NP

Edmonds (1966)

...conjectured P = NP ∩ co-NP

Revisiting NP∩co-NP



Conjecture: NP ≠ co-NP

⇓

P ≠ NP

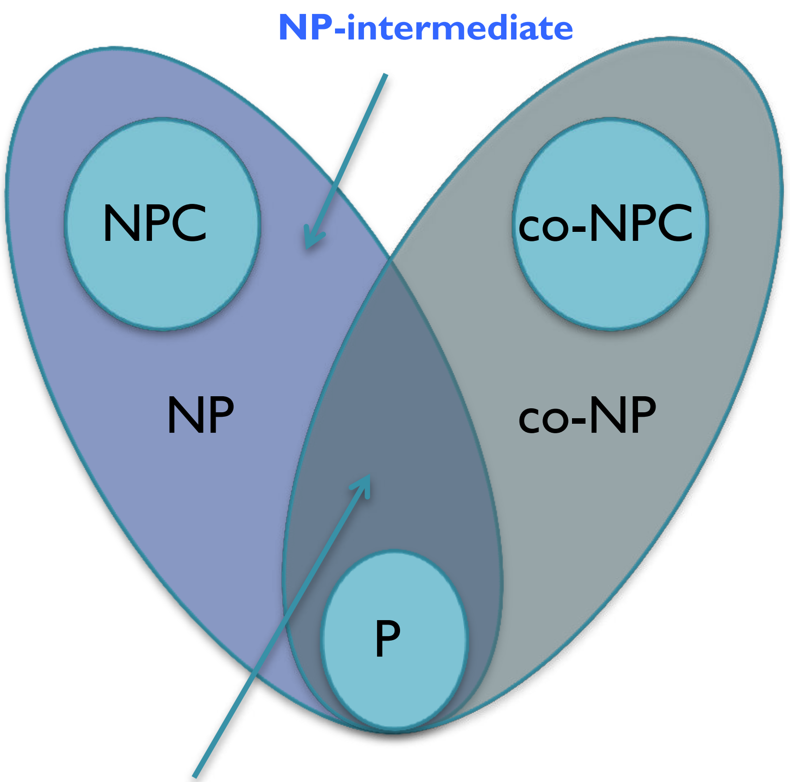
General belief: P ≠ NP ∩ co-NP

Check:
https://cstheory.stackexchange.com/questions/20021/reasons-to-believe-p-ne-np-cap-conp-or-not

- Integer factoring (FACT)
- Approximate shortest vector in a lattice

Ref: "*Lattice problems in NP∩co-NP*" by Aharonov & Regev (2005)

NP-intermediate problems



Conjecture: NP ≠ co-NP

⇓

P ≠ NP

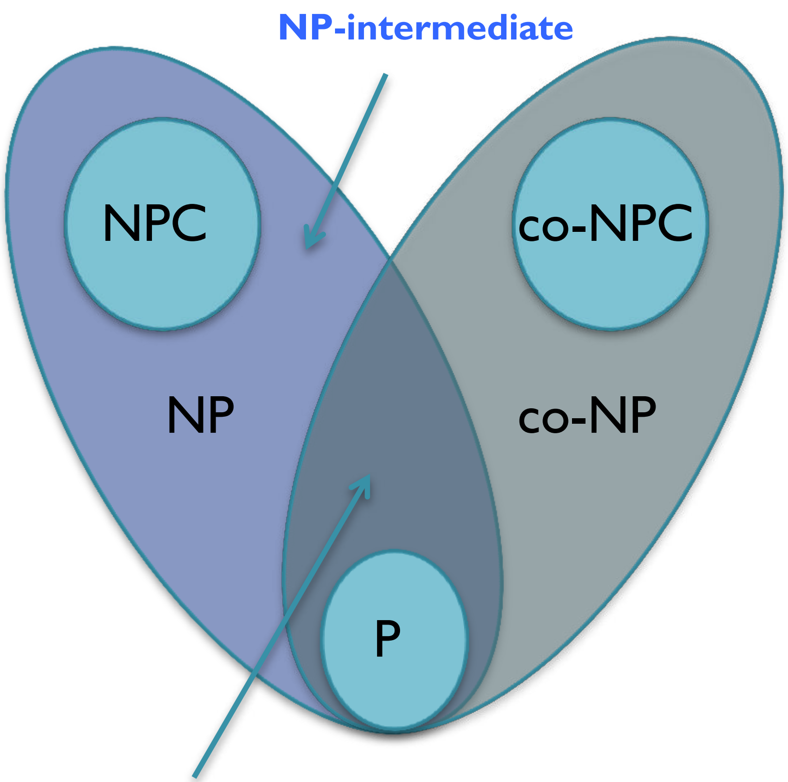
General belief: P ≠ NP ∩ co-NP

Obs: If NP ≠ co-NP and FACT ∉ P then FACT is NP-intermediate.

- Integer factoring (FACT)
- Approximate shortest vector in a lattice

Ref: "*Lattice problems in NP∩co-NP*" by Aharonov & Regev (2005)

NP-intermediate problems



- Integer factoring (FACT)
- Approximate shortest vector in a lattice

Ref: "*Lattice problems in NP∩co-NP*" by Aharonov & Regev (2005)

Conjecture: NP ≠ co-NP

⇓

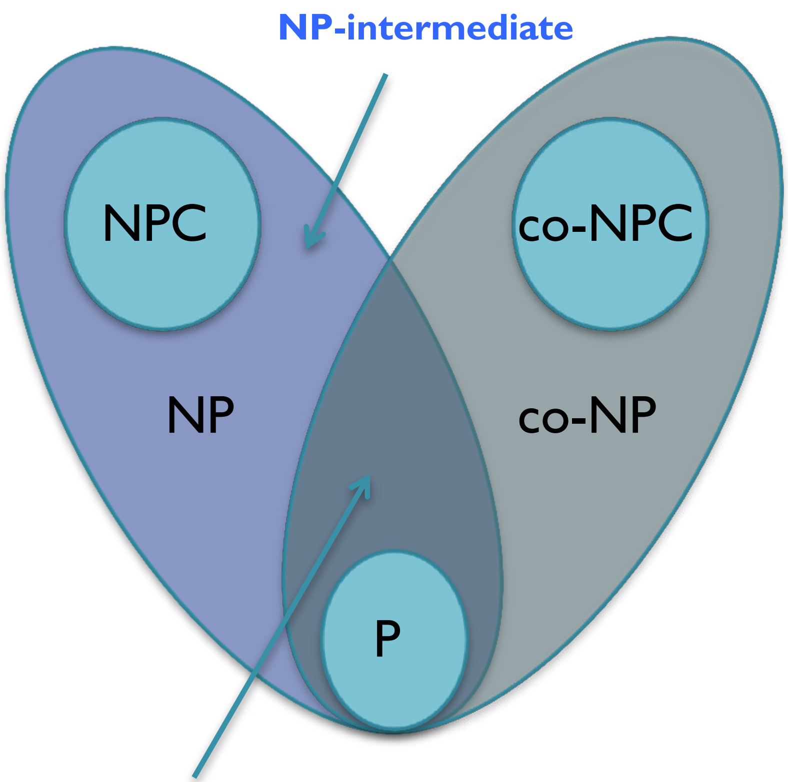
P ≠ NP

General belief: P ≠ NP ∩ co-NP

Obs: If NP ≠ co-NP and FACT ∉ P then FACT is NP-intermediate.

Ladner's theorem: P ≠ NP implies existence of a NP-intermediate language.

NP-intermediate problems



Conjecture: NP ≠ co-NP

⇓

P ≠ NP

General belief: P ≠ NP ∩ co-NP

Obs: If NP ≠ co-NP and FACT ∉ P then FACT is NP-intermediate.

Ladner's theorem: P ≠ NP implies existence of a NP-intermediate language.

(proved using ***diagonalization***)

- Integer factoring (FACT)
- Approximate shortest vector in a lattice

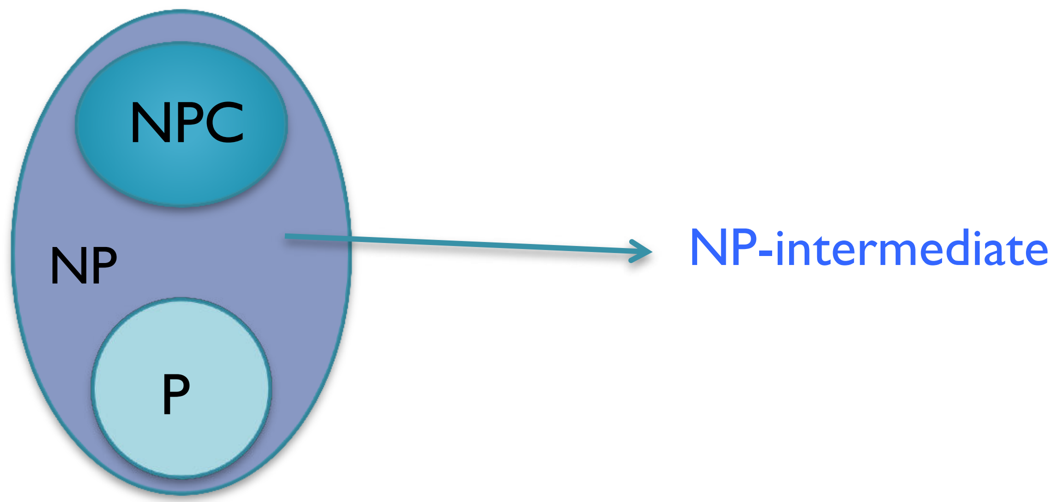
Ref: "*Lattice problems in NP∩co-NP*" by Aharonov & Regev (2005)

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.

- Theorem. *(Ladner 1975)* If $P \neq NP$ then there is a 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 1975)* If P $\neq$ NP then there is a 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 1975)* If P $\neq$ NP then there is a 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 1975)* If $P \neq NP$ then there is a NP-intermediate language.

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

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

NP-intermediate problems

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

- Theorem. *(Ladner 1975)* If $P \neq NP$ then there is a 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 $H: \mathbb{N} \rightarrow \mathbb{N}$ 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. If $SAT_H \in P$ then $H(m) \leq C$ (a constant).

     for every 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. If $SAT_H \in P$ then $H(m) \leq C$ (a constant).
3. If $SAT_H \notin P$ then $H(m) \rightarrow \infty$ with m.

Ladner's theorem: Constructing H

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

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

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

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

Proof: Later (uses diagonalization).

Let's see the proof of Ladner's theorem assuming the existence of such a "special" H.

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 $m = |\phi|$.
  - Compute $H(m)$, and construct the string $\phi\, 0\, 1^{m^{H(m)}}$

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.

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 $H(m) \rightarrow \infty$ 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:

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

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:

$$SAT \leq_p SAT_H$$

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

$$|\phi| = n \qquad |\Psi\, 0\, 1^k| = n^c$$

Ladner's theorem: Proof

$$P \neq NP$$

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

$$SAT \leq_p SAT_H$$

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

$|\phi| = n$ $\qquad$ $|\Psi\, 0\, 1^k| = n^c$

Let m_0 be the largest s.t. $H(m_0) \leq 2c$.

Ladner's theorem: Proof

$P \neq NP$

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

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

Let m_0 be the largest s.t. $H(m_0) \leq 2c$.

  - 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 $H(m) \rightarrow \infty$ with m.
- This also implies a poly-time algorithm for SAT:

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

Let m_0 be the largest s.t. $H(m_0) \leq 2c$.

- ➢ 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 $H(m) \rightarrow \infty$ with m.
- This also implies a poly-time algorithm for SAT:

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

Let m_0 be the largest s.t. $H(m_0) \leq 2c$.

- ➢ On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
- ➢ Compute $H(m)$ and check if $k = m^{H(m)}$. (Homework: Verify that this can be done in poly(n) time.)

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:

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

Let m_0 be the largest s.t. $H(m_0) \leq 2c$.

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

Either $m \leq m_0$ (in which case the task reduces to checking if a constant-size Ψ is satisfiable),

Ladner's theorem: Proof

$$P \neq NP$$

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

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

Let m_0 be the largest s.t. $H(m_0) \leq 2c$.

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

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 $H(m) \rightarrow \infty$ with m.
- This also implies a poly-time algorithm for SAT:

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

- 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. $\quad |f(\phi)| \geq k > m^{2c}$

Ladner's theorem: Proof

$$P \neq NP$$

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

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

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

Ladner's theorem: Proof

$$P \neq NP$$

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

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

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

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

- On input ϕ, compute $f(\phi) = \Psi\, 0\, 1^k$. Let $m = |\Psi|$.
- Compute $H(m)$ and check if $k = m^{H(m)}$.
- 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 $H(m) \rightarrow \infty$ with m.
- This also implies a poly-time algorithm for SAT:

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

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

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

- On input ϕ, compute $f(\phi) = \Psi\ 0\ 1^k$. Let $m = |\Psi|$.
- Compute $H(m)$ and check if $k = m^{H(m)}$.
- 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 $H(m) \rightarrow \infty$ with m.
- This also implies a poly-time algorithm for SAT:

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

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

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

Natural NP-intermediate problems ??

- Integer factoring
- Approximate shortest vector in a lattice
- Minimum Circuit Size Problem

  ("*Multi-output MCSP is NP-hard*", Ilango, Loff & Oliveira 2020)

- Graph isomorphism

  ("*GI in QuasiP time*", Babai 2015)