Computational Complexity Theory

Lecture 7: Ladner's theorem (contd.); Relativization

Department of Computer Science,
Indian Institute of Science

Recap: 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 $P \neq NP$)

Recap: 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.

Recap: Proof of Ladner's theorem

$P \neq NP$

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

Recap: Proof of Ladner's theorem

$$P \neq NP$$

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

$$SAT \leq_p SAT_H$$

$$\varphi \overset{f}{\longmapsto} \psi\ 0\ 1^k$$

$$|\varphi| = n \qquad |\psi\ 0\ 1^k| = n^c$$

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

Recap: Proof of Ladner's theorem

$P \neq NP$

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

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

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

- On input φ, compute $f(\varphi) = \Psi\ 0\ 1^k$. 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),

Recap: Proof of Ladner's theorem

$P \neq NP$

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

$SAT \leq_p SAT_H$ $\varphi \xmapsto{f} \psi\, 0\, 1^k$

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

- On input φ, compute $f(\varphi) = \psi\, 0\, 1^k$. 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).

Recap: Proof of Ladner's theorem

$P \neq NP$

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

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

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

Recap: Proof of Ladner's theorem

$P \neq NP$

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

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

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

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

Recap: Proof of Ladner's theorem

$P \neq NP$

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

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

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

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

Ladner's theorem: Properties of 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. 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.

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

Construction of H

- Observation. The value of $H(m)$ determines membership in SAT_H of strings whose length is $\geq m$.

- Therefore, it is OK to define $H(m)$ based on strings in SAT_H whose lengths are $< m$ (say, $\log m$).

Construction of H

- Observation. The value of $H(m)$ determines membership in SAT_H of strings whose length is $\geq m$.

- Therefore, it is OK to define $H(m)$ based on strings in SAT_H whose lengths are $< m$ (say, $\log m$).

- Think of computing $H(m)$ sequentially: Compute $H(1)$, $H(2),\ldots,H(m-1)$. Just before computing $H(m)$, find $SAT_H \cap \{0,1\}^{\log m}$.

Construction of H

- Observation. The value of $H(m)$ determines membership in SAT_H of strings whose length is $\geq m$.

- Therefore, it is OK to define $H(m)$ based on strings in SAT_H whose lengths are $< m$ (say, $\log m$).

- Construction. $H(m)$ is the smallest $k < \log \log m$ s.t.
  1. M_k decides membership of **all** length up to $\log m$ strings x in SAT_H within $k.|x|^k$ time.
  2. If no such k exists then $H(m) = \log \log m$.

Construction of H

- Observation. The value of $H(m)$ determines membership in SAT_H of strings whose length is $\geq m$.

- Therefore, it is OK to define $H(m)$ based on strings in SAT_H whose lengths are $< m$ (say, $\log m$).

- Homework. Prove that $H(m)$ is computable from m in $O(m^3)$ time.

Construction of H

- Claim. If $SAT_H \in P$ then $H(m) \leq C$ (a constant).
- Proof. There is a poly-time M that decides membership of every x in SAT_H within $c.|x|^c$ time.

Construction of H

- Claim. If $SAT_H \in P$ then $H(m) \leq C$ (a constant).
- Proof. There is a poly-time M that decides membership of every x in SAT_H within $c.|x|^c$ time.

- As M can be represented by infinitely many strings, there's an $\alpha \geq c$ s.t. $M = M_\alpha$ decides membership of every x in SAT_H within $\alpha.|x|^\alpha$ time.

- So, for every m satisfying $\alpha < \log \log m$, $H(m) \leq \alpha$.

Construction of H

- Claim. If $H(m) \leq C$ (a constant) for infinitely many m, then $SAT_H \in P$.
- Proof. There's a $k \leq C$ s.t. $H(m) = k$ for infinitely many m.

Construction of H

- Claim. If $H(m) \leq C$ (a constant) for infinitely many m, then $SAT_H \in P$.
- Proof. There's a $k \leq C$ s.t. $H(m) = k$ for infinitely many m.

- Pick any $x \in \{0,1\}^*$. Think of a large enough m s.t. $|x| \leq \log m$ and $H(m) = k$.

Construction of H

- Claim. If $H(m) \leq C$ (a constant) for infinitely many m, then $SAT_H \in P$.
- Proof. There's a $k \leq C$ s.t. $H(m) = k$ for infinitely many m.

- Pick any $x \in \{0,1\}^*$. Think of a large enough m s.t. $|x| \leq \log m$ and $H(m) = k$.

- This means x is correctly decided by M_k in $k.|x|^k$ time. So, M_k is a poly-time machine deciding SAT_H.

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;
  "NP-hardness of learning programs and partial MCSP", Hirahara 2022)
- Graph isomorphism

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

Natural NP-intermediate problems ??

- Discrete logarithm
- Isomorphism problems (for groups, rings, polynomials)
- Unique games
- Check this link for more candidate problems:

  https://cstheory.stackexchange.com/questions/79/problems-between-p-and-npc

Limits of diagonalization

- Like in the proof of P $\neq$ EXP, can we use diagonalization to show P $\neq$ NP ?

Limits of diagonalization

- Like in the proof of P $\neq$ EXP, can we use diagonalization to show P $\neq$ NP ?
- The answer is No, if one insists on using only the two features of diagonalization.
- The proof of this fact uses diagonalization and the notion of *oracle Turing machines*!

Oracle Turing Machines

- Definition: Let $L \subseteq \{0,1\}^*$ be a language. An *oracle TM* M^L is a TM with a special query tape and three special states q_{query}, q_{yes} and q_{no} such that whenever the machine enters the q_{query} state, it immediately transits to q_{yes} or q_{no} depending on whether the string in the query tape belongs to L. (M^L has *oracle access* to L)

Oracle Turing Machines

- Definition: Let $L \subseteq \{0,1\}^*$ be a language. An <u>*oracle TM*</u> M^L is a TM with a special query tape and three special states q_{query}, q_{yes} and q_{no} such that whenever the machine enters the q_{query} state, it immediately transits to q_{yes} or q_{no} depending on whether the string in the query tape belongs to L. (M^L has *oracle access* to L)

- Think of physical realization of M^L as a device with access to a subroutine that decides L. We don't count the time taken by the subroutine.

Oracle Turing Machines

- We can define a <u>nondeterministic</u> Oracle TM similarly.

- "Important note": Oracle TMs (deterministic or nondeterministic) have the same two features used in diagonalization: For any **fixed** $L \subseteq \{0,1\}^*$,

  1. There's an <u>*efficient universal TM*</u> with oracle access to L,
  2. Every M^L has <u>*infinitely many representations*</u>.

Complexity classes using oracles

- Definition: Let $L \subseteq \{0,1\}^*$ be a language. Complexity classes P^L, NP^L and EXP^L are defined just as P, NP and EXP respectively, but with TMs replaced by oracle TMs with oracle access to L in the definitions of P, NP and EXP respectively. For e.g., $\overline{SAT} \in P^{SAT}$.

Complexity classes using oracles

- Definition: Let $L \subseteq \{0,1\}^*$ be a language. Complexity classes P^L, NP^L and EXP^L are defined just as P, NP and EXP respectively, but with TMs replaced by oracle TMs with oracle access to L in the definitions of P, NP and EXP respectively. For e.g., $\overline{SAT} \in P^{SAT}$.

- Such complexity classes help us identify a class of complexity theoretic proofs called *relativizing proofs*.

Relativization

Relativizing results

- Observation: Let $L \subseteq \{0,1\}^*$ be an arbitrarily fixed language. Owing to the "Important note", the proof of $P \neq EXP$ can be easily adapted to prove $P^L \neq EXP^L$ by working with TMs with oracle access to L.
- We say that the $P \neq EXP$ result/proof ***relativizes***.

Relativizing results

- Observation: Let $L \subseteq \{0,1\}^*$ be an arbitrarily fixed language. Owing to the "Important note", the proof of $P \neq EXP$ can be easily adapted to prove $P^L \neq EXP^L$ by working with TMs with oracle access to L.
- We say that the $P \neq EXP$ result/proof ***relativizes***.

- **Observation:** Let $L \subseteq \{0,1\}^*$ be an arbitrarily fixed language. Owing to the 'Important note', any proof/result that uses only the two features of diagonalization ***relativizes***.

Relativizing results

- If there is a resolution of the P vs. NP problem using **only** the two features of diagonalization, then such a proof must relativize.
- Is it true that

    - either $P^L = NP^L$ for every $L \subseteq \{0,1\}^*$,

    - or $P^L \neq NP^L$ for every $L \subseteq \{0,1\}^*$?

Relativizing results

- If there is a resolution of the P vs. NP problem <u>using **only** the two features</u> of diagonalization, then such a proof must relativize.
- Is it true that

  - either $P^L = NP^L$ for every $L \subseteq \{0,1\}^*$,

  - or $P^L \neq NP^L$ for every $L \subseteq \{0,1\}^*$?

Theorem *(Baker, Gill & Solovay 1975)*: The answer is No. Any proof of $P = NP$ or $P \neq NP$ must <u>**not**</u> relativize.

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: Using diagonalization!

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: Let $A = \{(M, x, 1^m)$: M accepts x in 2^m steps$\}$.
- A is an EXP-complete language under poly-time Karp reduction. *(simple exercise)*

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: Let $A = \{(M, x, 1^m)$: M accepts x in 2^m steps$\}$.
- A is an EXP-complete language under poly-time Karp reduction.

- Then, $P^A = EXP$.
- Also, $NP^A = EXP$. Hence $P^A = NP^A$.

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: Let $A = \{(M, x, 1^m)$: M accepts x in 2^m steps$\}$.
- A is an EXP-complete language under poly-time Karp reduction.

- Then, $P^A = EXP$.
- Also, $NP^A = EXP$. Hence $P^A = NP^A$.

Why isn't $EXP^A = EXP$?

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: The construction of B uses diagonalization.

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: For any language B let

$$L_B = \{1^n : \text{there's a string of length } n \text{ in } B\}.$$

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: For any language B let

$$L_B = \{1^n : \text{there's a string of length } n \text{ in } B\}.$$

- Observe, $L_B \in NP^B$ for any B. (Guess the string, check if it has length n, and ask oracle B to verify membership.)

Baker-Gill-Solovay theorem

- Theorem: There exist languages A and B such that $P^A = NP^A$ but $P^B \neq NP^B$.
- Proof: For any language B let

$$L_B = \{1^n : \text{there's a string of length } n \text{ in } B\}.$$

- Observe, $L_B \in NP^B$ for any B.
- We'll construct B (using diagonalization) in such a way that $L_B \notin P^B$, implying $P^B \neq NP^B$.

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps. Moreover, n will grow monotonically with stages.

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps. Moreover, n will grow monotonically with stages.

whether or not a string belongs to B

The machine with oracle access to B that is represented by i

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps. Moreover, n will grow monotonically with stages.

- Clearly, a B satisfying the above implies $L_B \notin P^B$. Why?

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps. Moreover, n will grow monotonically with stages.

- Clearly, a B satisfying the above implies $L_B \notin P^B$. Why?
- …because M_i^B has infinitely many representations, and for sufficiently large n, $2^n/10 \gg n^{O(1)}$.

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps. Moreover, n will grow monotonically with stages.

- Stage i: Choose n larger than the length of any string whose status has already been decided. Simulate M_i^B on input 1^n for $2^n/10$ steps.

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps.

- Stage i: If M_i^B queries oracle B with a string whose status has already been decided, answer consistently.

  If M_i^B queries oracle B with a string whose status has not been decided yet, answer 'No'.

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps.

- Stage i: If M_i^B outputs 1 within $2^n/10$ steps then don't put any string of length n in B.

  If M_i^B outputs 0 or doesn't halt, put a string of length n in B.

(This is possible as the status of at most $2^n/10$ many length n strings have been decided during the simulation)

Constructing B

- We'll construct B in stages, starting from Stage 1.
- Each stage determines the status of finitely many strings.
- In Stage i, we'll ensure that the oracle TM M_i^B doesn't decide 1^n correctly (for some n) within $2^n/10$ steps.

- Homework: In fact, we can assume that B ∈ EXP.