Computational Complexity Theory

Lecture 6: Ladner's theorem (contd.); Relativization

Indian Institute of Science

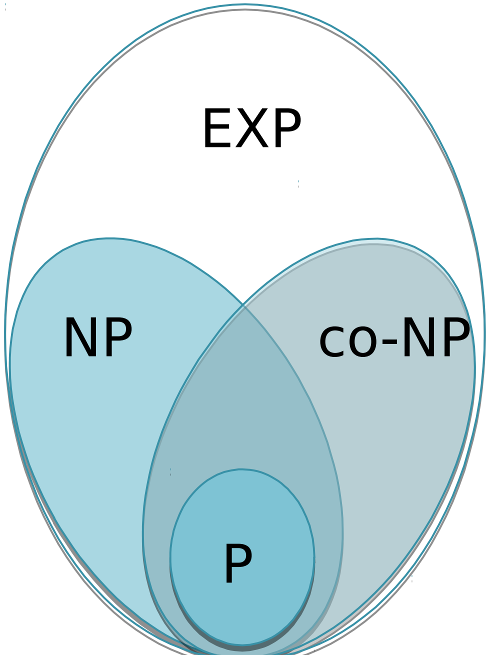
Recap: Class co-NP and EXP

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

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

- **Definition.**

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



Recap: 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.

Recap: 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. $\mathrm{P} \subsetneq \mathrm{EXP}$

Recap: Ladner's theorem

- 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 $H: \mathbb{N} \longrightarrow \mathbb{N}$ be a function.

$m \mapsto H(m)$

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: 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. $SAT_H \in P \iff H(m) \le C$ (a constant)
3. If $SAT_H \notin P$ then $H(m) \to \infty$ with m

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 $m = |\phi|$.
  - 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

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:

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

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

$\sqrt{n} \geq m$

Recap: Proof of Ladner's theorem

$P \neq NP$

- 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 $k = m^{H(m)}$.
  - W.l.o.g. $n^c = |f(\phi)| \geq m^{2c}$

    $\sqrt{n} \geq m$

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

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 length is $< m$ (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 length is $< m$ (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 length is $< m$ (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 $H(m) \le C$ (a constant) then $SAT_H \in P$.
- Proof. There's a $k \le C$ s.t. $H(m) = k$ for infinitely many m.

Construction of H

- Claim. If $H(m) \leq C$ (a constant) 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) 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 problem?

- Integer factoring.

FACT = {(N, U): there's a prime $\leq$ U dividing N}

- Claim. FACT $\in$ NP $\cap$ co-NP

- So, FACT is NP-complete if and only if NP = co-NP.

Natural NP-intermediate problem?

- Integer factoring.

FACT = {(N, U): there's a prime ≤ 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), p ≤ U and p divides N.

Natural NP-intermediate problem?

- Integer factoring.

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

- Claim. FACT ∈ NP ∩ co-NP
- Proof. FACT ∈ NP : Give complete prime factorization of N as a certificate. The verifier checks if none of the prime factors is ≤ U.

Natural NP-intermediate problem?

- Integer factoring.

  FACT = {(N, U): there's a prime $\leq$ U dividing N}

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

Power & 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 <u>using only the two features of diagonalization</u>.

Oracle Turing Machines

- 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 <u>using only the two features of diagonalization</u>.

- 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

Oracle Turing Machines

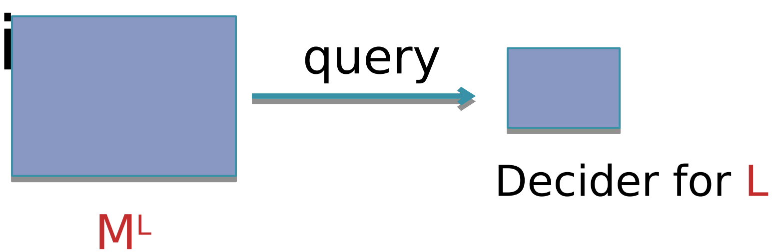
- Like in the proof of $P \neq EXP$, can we use diagonalization to show $P \neq NP$?
- The answer is No, if one insists on using only the two features of diagonalization.

- 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

- Like in the proof of P ≠ EXP, can we use diagonalization to show P ≠ NP ?
- The answer is No, if one insists on using only the two features of diagonalization.

- 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 subrouti



Oracle Turing Machines

- 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 <u>using only the two features of diagonalization</u>.

- 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.
- The transition table of M^L doesn't have any rule of the kind $(q_{query}, b) \rightarrow (q, c, L/R)$.

Oracle Turing Machines

- 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.

- 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.

- We can define a nondeterministic Oracle TM similarly.

Oracle Turing Machines

- 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 <u>using only the two features of diagonalization</u>.

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

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

Relativization

Complexity classes using oracles

- 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.

- 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.

Complexity classes using oracles

- 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.

- 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. $SAT \in P^{SAT}$

Relativizing results

- 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 <u>using only the two features of diagonalization</u>.

- 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 <u>*relativizes*</u>.

Relativizing results

- 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 <u>using only the two features of diagonalization</u>.

- 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

- 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.

- Is is 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): The answer is No. Any proof of $P = NP$ or $P \neq NP$ must not relativize.

Relativizing results

- 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 <u>using only the two features of diagonalization</u>.

- Is is 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): 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.

- 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: For any language B let

$L_B = \{1^n$: there's a string of length 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

$L_B = \{1^n$: there's a string of length 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 $P^B \neq NP^B$.