Computational Complexity Theory

Lecture 10: Relativization (contd.); Space complexity classes

Department of Computer Science,
Indian Institute of Science

Recap: 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*!

Recap: 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)

- 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.

Recap: 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>.

Recap: 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

Recap: 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', <u>any proof/result that uses only the two features of diagonalization ***relativizes***</u>.

Recap: 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 <u>grow monotonically</u> 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 $B \in EXP$.

Space bounded computation

Space bounded computation

- Here, we are interested to find out how much of *work space* is required to solve a problem.
- For convenience, think of TMs with a separate read-only input tape and one or more work tapes. Work space is the number of cells in the work tapes of a TM M visited by M's heads during a computation.

Space bounded computation

- Here, we are interested to find out how much of *work space* is required to solve a problem.
- For convenience, think of TMs with a separate read-only input tape and one or more work tapes. Work space is the number of cells in the work tapes of a TM M visited by M's heads during a computation.

- Definition. Let $S: \mathbb{N} \rightarrow \mathbb{N}$ be a function. A language L is in DSPACE(S(n)) if there's a TM M that decides L using O(S(n)) work space on inputs of length n.

Space bounded computation

- Here, we are interested to find out how much of *work space* is required to solve a problem.
- For convenience, think of TMs with a separate read-only input tape and one or more work tapes. Work space is the number of cells in the work tapes of a TM M visited by M's heads during a computation.

- Definition. Let $S: \mathbb{N} \rightarrow \mathbb{N}$ be a function. A language L is in NSPACE(S(n)) if there's a NTM M that decides L using O(S(n)) work space on inputs of length n, regardless of M's nondeterministic choices.

Space bounded computation

- We'll refer to 'work space' as 'space'. For convenience, assume there's a single work tape.
- If the output has many bits, then we will assume that the TM has a separate write-only output tape.

Space bounded computation

- We'll refer to 'work space' as 'space'. For convenience, assume there's a single work tape.
- If the output has many bits, then we will assume that the TM has a separate write-only output tape.

- Definition. Let $S: \mathbb{N} \longrightarrow \mathbb{N}$ be a function. S is *space constructible* if $S(n) \geq \log n$ and there's a TM that computes $S(|x|)$ from x using $O(S(|x|))$ space.

Relation between time and space

- Obs. DTIME(S(n)) $\subsetneq$ DSPACE(S(n)) $\subseteq$ NSPACE(S(n)).

Hopcroft, Paul & Valiant 1977

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.
- Proof. Uses the notion of *configuration graph* of a TM. We'll see this shortly.

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Definition.

$$\mathrm{L} = \mathrm{DSPACE}(\log n)$$

$$\mathrm{NL} = \mathrm{NSPACE}(\log n)$$

$$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{DSPACE}(n^c)$$

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Definition.

$$L = \mathrm{DSPACE}(\log n)$$

$$NL = \mathrm{NSPACE}(\log n)$$

$$\mathrm{PSPACE} = \bigcup_{c > 0} \mathrm{DSPACE}(n^c)$$

Giving space at least $\log n$ gives a TM at least the power to remember the index of a cell.

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Caution. The *Hopcroft-Paul-Valiant* theorem does not imply $\mathrm{P} \subsetneq \mathrm{PSPACE}$.

- Open. Is $\mathrm{P} \neq \mathrm{PSPACE}$?

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Theorem. $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$

Follows from the above theorem

Relation between time and space

- Obs. $DTIME(S(n)) \subsetneq DSPACE(S(n)) \subseteq NSPACE(S(n))$.

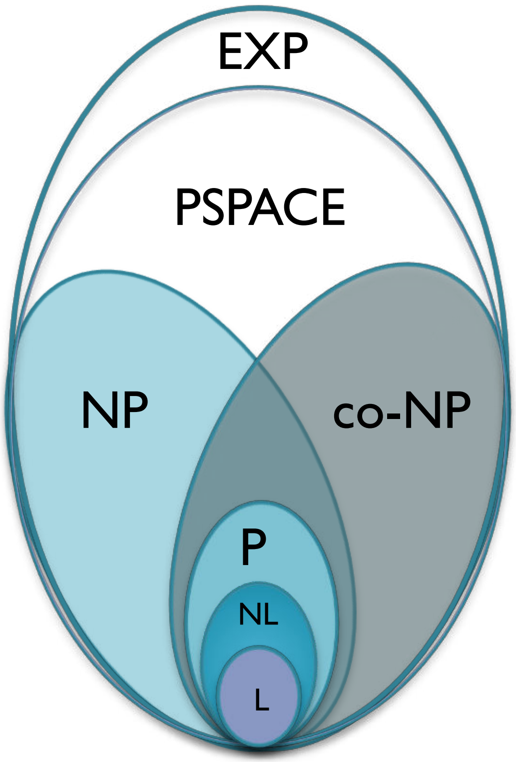
- Theorem. $NSPACE(S(n)) \subseteq DTIME(2^{O(S(n))})$, if S is space constructible.

- Theorem. $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP$

Run through all possible choices of certificates of the verifier and **reuse** space.

Relation between time and space

- Obs. DTIME(S(n)) $\subsetneq$ DSPACE(S(n)) $\subseteq$ NSPACE(S(n)).
- Theorem. NSPACE(S(n)) $\subseteq$ DTIME($2^{O(S(n))}$), if S is space constructible.



Relation between time and space

- Obs. DTIME(S(n)) $\subsetneq$ DSPACE(S(n)) $\subseteq$ NSPACE(S(n)).

- Theorem. NSPACE(S(n)) $\subseteq$ DTIME($2^{O(S(n))}$), if S is space constructible.

Homework: Integer addition and multiplication are in (functional) L.

Integer division is also in (functional) L. *(Chiu, Davida & Litow 2001)*