



Computational Complexity Theory

Lecture 12: PSPACE-completeness; Log-space reductions

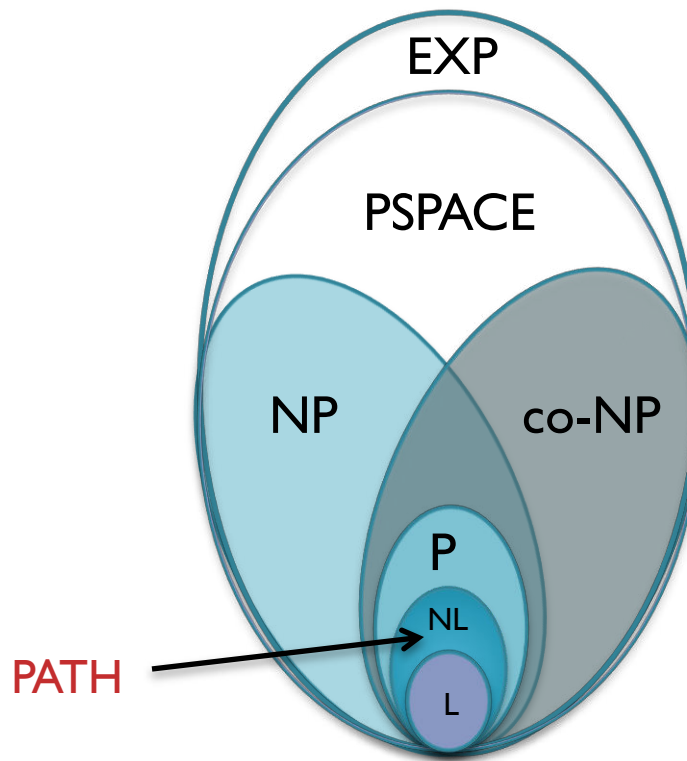
Department of Computer Science,
Indian Institute of Science

Recap: Natural problems?

- Definition.
 $L = \text{DSPACE}(\log n)$
 $NL = \text{NSPACE}(\log n)$
 $\text{PSPACE} = \bigcup_{c > 0} \text{DSPACE}(n^c)$
- Theorem. $L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSPACE} \subseteq \text{EXP}$.
- Are there natural problems in L , NL and PSPACE ?

PATH: A canonical problem in NL

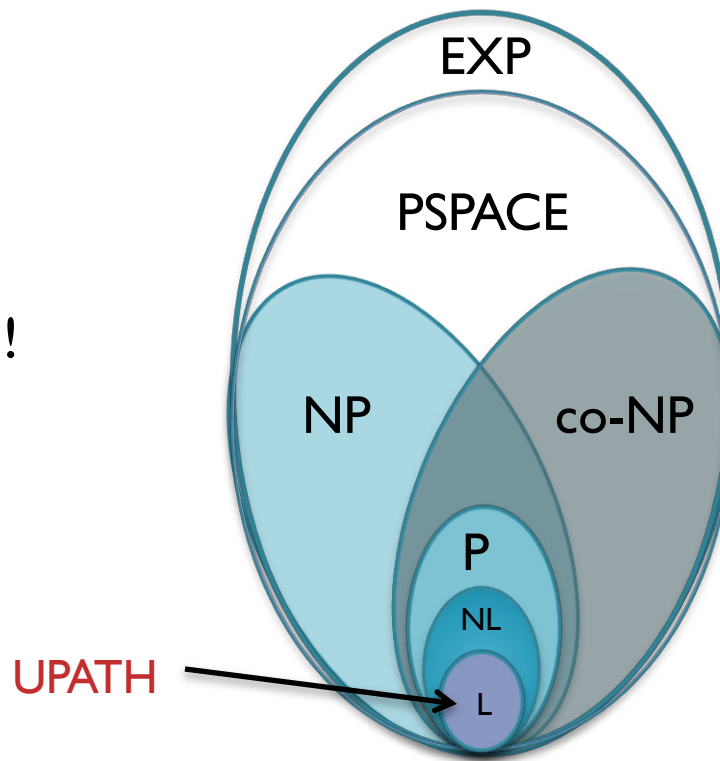
- **PATH** = $\{(G,s,t) : G \text{ is a directed graph having a path from } s \text{ to } t\}$.
- Obs. **PATH** is in **NL**.



UPATH: A problem in L

- **UPATH** = $\{(G,s,t) : G \text{ is an undirected graph having a path from } s \text{ to } t\}$.
- **Theorem** (Reingold 2005). **UPATH** is in **L**.

Is **PATH** in **L** ?
If yes, then **L** = **NL** !
(will prove later)



Recap: Space Hierarchy Theorem

- **Theorem.** (*Stearns, Hartmanis & Lewis 1965*) If f and g are space-constructible functions and $f(n) = o(g(n))$, then $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(g(n))$.
- **Proof.** Homework.
- **Theorem.** $L \subsetneq \text{PSPACE}$.

Recap: Savitch's theorem

- **Theorem.** $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, $\text{PSPACE} = \text{NPSPACE}$)
- **Proof.**
- $\text{REACH}(C_1, C_2, i) \{$
 - If $i = 0$ check if C_1 and C_2 are adjacent.
 - Else, for every configurations C ,
 - $a_1 = \text{REACH}(C_1, C, i-1)$
 - $a_2 = \text{REACH}(C, C_2, i-1)$
 - if $a_1 = 1$ & $a_2 = 1$, return 1. Else return 0.
- }

Recap: Savitch's theorem

- **Theorem.** $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, $\text{PSPACE} = \text{NPSPACE}$)

- **Proof.**

$$\text{Space}(i) = \text{Space}(i-1) + O(S(n))$$

- Space complexity: $O(S(n)^2)$

$$\text{Time}(i) = 2^m \cdot \text{Time}(i-1) + O(S(n))$$

- Time complexity: $2^{O(S(n)^2)}$ 

Recall, $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$.
There's an algorithm with time complexity $2^{O(S(n))}$, but higher space requirement.

PSPACE-completeness

PSPACE-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a reduction.
- What kind of reductions will be suitable is guided by a complexity question, like a comparison between the complexity class under consideration & another class.
- Is $P = PSPACE$?

PSPACE-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a reduction.
- What kind of reductions will be suitable is guided by a complexity question, like a comparison between the complexity class under consideration & another class.
- Is $P = PSPACE$? ...use poly-time Karp reduction!
- **Definition.** A language L' is *PSPACE-hard* if for every L in $PSPACE$, $L \leq_p L'$. Further, if L' is in $PSPACE$ then L' is *PSPACE-complete*.

A PSPACE-complete problem

- Recall, to define completeness of a complexity class, we need an appropriate notion of a reduction.
- What kind of reductions will be suitable is guided by a complexity question, like a comparison between the complexity class under consideration & another class.
- Is $P = PSPACE$? ...use poly-time Karp reduction!
- **Example.** $L' = \{(M, w, l^m) : M \text{ accepts } w \text{ using } m \text{ space}\}$

Natural PSPACE-complete problem

- **Definition.** A *quantified Boolean formula (QBF)* is a formula of the form

$$Q_1x_1 \ Q_2x_2 \ \dots \ Q_nx_n \ \underbrace{\phi(x_1, x_2, \dots, x_n)}_{\text{Just a formula on Boolean variables}}$$

Quantifiers \exists or \forall

The diagram illustrates the structure of a Quantified Boolean Formula (QBF). It shows a sequence of quantifiers and variables, $Q_1x_1 \ Q_2x_2 \ \dots \ Q_nx_n$, followed by a Boolean formula $\phi(x_1, x_2, \dots, x_n)$. Blue arrows point from the text 'Quantifiers \exists or \forall ' to each of the Q_i terms. A blue bracket underneath the ϕ term is labeled 'Just a formula on Boolean variables'.

- A QBF is either true or false as all variables are quantified. This is unlike a formula we've seen before where variables were unquantified/free.

Natural PSPACE-complete problem

- **Example.** $\exists x_1 \exists x_2 \dots \exists x_n \phi(x_1, x_2, \dots, x_n)$
- The above QBF is true iff ϕ is satisfiable.
- We could have defined **SAT** as
$$\text{SAT} = \{\exists \mathbf{x} \phi(\mathbf{x}) : \phi \text{ is a CNF and } \exists \mathbf{x} \phi(\mathbf{x}) \text{ is true}\}$$
instead of
$$\text{SAT} = \{\phi(\mathbf{x}) : \phi \text{ is a CNF and } \phi \text{ is satisfiable}\}$$

Natural PSPACE-complete problem

- **Definition.** A *quantified Boolean formula (QBF)* is a formula of the form

$$Q_1x_1 \ Q_2x_2 \ \dots \ Q_nx_n \ \underbrace{\phi(x_1, x_2, \dots, x_n)}_{\text{Just a formula on Boolean variables}}$$

Quantifiers \exists or \forall

The diagram illustrates the structure of a Quantified Boolean Formula (QBF). It shows a sequence of quantifiers and variables, $Q_1x_1 \ Q_2x_2 \ \dots \ Q_nx_n$, followed by a Boolean formula $\phi(x_1, x_2, \dots, x_n)$. Three blue arrows originate from a single point below the first quantifier and point to each of the three quantifiers (Q_1 , Q_2 , and Q_n). A blue bracket is placed under the formula $\phi(x_1, x_2, \dots, x_n)$. Below the bracket, the text 'Just a formula on Boolean variables' is written. To the left of the quantifiers, the text 'Quantifiers \exists or \forall ' is written.

- **Homework:** By using auxiliary variables (as in the proof of Cook-Levin) and introducing some more \exists quantifiers at the end, we can assume w.l.o.g. that ϕ is a 3CNF.

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** Easy to see that **TQBF** is in **PSPACE** – just think of a suitable recursive procedure. We'll now show that every $L \in \text{PSPACE}$ reduces to **TQBF** via poly-time Karp reduction...

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** (contd.) Let **M** be a TM deciding **L** using **$S(n) = \text{poly}(n)$** space. We intend to come up with a poly-time reduction **f** s.t.

$$x \in L \quad \xleftrightarrow{f} \quad \psi_x \text{ is a true QBF}$$

Size of ψ_x must be bounded by **$\text{poly}(n)$** , where $|x| = n$

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** (contd.) Let **M** be a TM deciding **L** using **$S(n) = \text{poly}(n)$** space. We intend to come up with a poly-time reduction **f** s.t.

$$x \in L \quad \xleftrightarrow{f} \quad \psi_x \text{ is a true QBF}$$

Idea: Form ψ_x in such a way that ψ_x is true iff there's a path from C_{start} to C_{accept} in $G_{M,x}$.

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** (contd.) **f** computes **$S(n)$** from **n** (recall, any poly function **$S(n)$** is time constructible). It also computes **$m = O(S(n))$** , the no. of bits required to represent a configuration in **$G_{M,x}$** .

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** (contd.) **f** computes **$S(n)$** from **n** (recall, any poly function **$S(n)$** is time constructible). It also computes **$m = O(S(n))$** , the no. of bits required to represent a configuration in **$G_{M,x}$** . Then, it forms a semi-QBF **$\Delta_i(C_1, C_2)$** , such that **$\Delta_i(C_1, C_2)$** is true iff there's a path from **C_1** to **C_2** of length at most **2^i** in **$G_{M,x}$** .

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** (contd.) **f** computes **$S(n)$** from **n** (recall, any poly function **$S(n)$** is time constructible). It also computes **$m = O(S(n))$** , the no. of bits required to represent a configuration in **$G_{M,x}$** . Then, it forms a semi-QBF **$\Delta_i(C_1, C_2)$** , such that **$\Delta_i(C_1, C_2)$** is true iff there's a path from **C_1** to **C_2** of length at most **2^i** in **$G_{M,x}$** .

The variables corresponding to the bits of **C_1** and **C_2** are unquantified/free variables of **Δ_i**

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.
- **Proof:** (contd.) QBF $\Delta_i(C_1, C_2)$ is formed, recursively, as follows:

(first attempt)

$$\Delta_i(C_1, C_2) = \exists C \left(\Delta_{i-1}(C_1, C) \wedge \Delta_{i-1}(C, C_2) \right)$$

Issue: Size of Δ_i is twice the size of Δ_{i-1} !!

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.
- **Proof:** (contd.) QBF $\Delta_i(C_1, C_2)$ is formed, recursively, as follows:

(careful attempt)

$$\Delta_i(C_1, C_2) = \exists C \forall D_1 \forall D_2$$

$$\left(\left((D_1 = C_1 \wedge D_2 = C) \vee (D_1 = C \wedge D_2 = C_2) \right) \Rightarrow \Delta_{i-1}(D_1, D_2) \right)$$

Natural PSPACE-complete problem

- **Definition.** **TQBF** is the set of true quantified Boolean formulas.
- **Theorem.** **TQBF** is **PSPACE-complete**.
- **Proof:** (contd.) QBF $\Delta_i(C_1, C_2)$ is formed, recursively, as follows:

(careful attempt)

$$\Delta_i(C_1, C_2) = \exists C \forall D_1 \forall D_2$$

$$\left(\neg \left((D_1 = C_1 \wedge D_2 = C) \vee (D_1 = C \wedge D_2 = C_2) \right) \vee \Delta_{i-1}(D_1, D_2) \right)$$

Note: Size of $\Delta_i = O(S(n)) + \text{Size of } \Delta_{i-1}$

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.
- **Proof:** (contd.) Finally,

$$\psi_x = \Delta_m(C_{\text{start}}, C_{\text{accept}})$$

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.
- **Proof:** (contd.) Finally,

$$\psi_x = \Delta_m(C_{\text{start}}, C_{\text{accept}})$$

- But, we need to specify how to form $\Delta_0(C_1, C_2)$.
- Size of $\psi_x = O(S(n)^2) + \text{Size of } \Delta_0$

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.
- **Proof:** (contd.) Finally,

$$\Psi_x = \Delta_m(C_{\text{start}}, C_{\text{accept}})$$

- But, we need to specify how to form $\Delta_0(C_1, C_2)$.
- Size of $\Psi_x = O(S(n)^2) + \text{Size of } \Delta_0$

Remark: We can easily bring all the quantifiers at the beginning in Ψ_x (as in a *prenex normal form*).

Natural PSPACE-complete problem

- **Definition.** TQBF is the set of true quantified Boolean formulas.
- **Theorem.** TQBF is PSPACE-complete.
- **Proof:** (contd.) Finally,

$$\psi_x = \Delta_m(C_{\text{start}}, C_{\text{accept}})$$

- But, we need to specify how to form $\Delta_0(C_1, C_2)$.
- Size of $\psi_x = O(S(n)^2) + \text{Size of } \Delta_0 \rightarrow ??$

Adjacent configurations

- **Claim.** There's an $O(S(n)^2)$ -size circuit $\phi_{M,x}$ on $O(S(n))$ inputs such that for every inputs C_1 and C_2 , $\phi_{M,x}(C_1, C_2) = 1$ iff C_1 and C_2 encode two neighboring configurations in $G_{M,x}$.
- **Proof.** Think of a linear time algorithm that has the knowledge of M and x , and on input C_1 and C_2 it checks if C_2 is a neighbor of C_1 in $G_{M,x}$.

Adjacent configurations

- **Claim.** There's an $O(S(n)^2)$ -size circuit $\phi_{M,x}$ on $O(S(n))$ inputs such that for every inputs C_1 and C_2 , $\phi_{M,x}(C_1, C_2) = 1$ iff C_1 and C_2 encode two neighboring configurations in $G_{M,x}$.
- **Proof.** Think of a linear time algorithm that has the knowledge of M and x , and on input C_1 and C_2 it checks if C_2 is a neighbor of C_1 in $G_{M,x}$. Applying ideas from the proof of Cook-Levin theorem, we get our desired $\phi_{M,x}$ of size $O(S(n)^2)$.

Size of Δ_0

- **Obs.** We can convert the circuit $\phi_{M,x}(C_1, C_2)$ to a quantified CNF $\Delta_0(C_1, C_2)$ by introducing auxiliary variables (as in the proof of Cook-Levin theorem).
- Hence, size of $\Delta_0(C_1, C_2)$ is $O(S(n)^2)$.
- Therefore, size of $\psi_x = O(S(n)^2)$.

Other PSPACE complete problems

- Checking if a player has a winning strategy in certain two-player games, like (generalized) Hex, Reversi, Geography etc.
- Integer circuit evaluation (*Yang 2000*).
- Implicit graph reachability.
- Check the wiki page:
https://en.wikipedia.org/wiki/List_of_PSPACE-complete_problems

Log-space reductions

NL-completeness

- Recall again, to define completeness of a complexity class, we need an appropriate notion of a reduction.
- What kind of reductions will be suitable is guided by a complexity question, like a comparison between the complexity class under consideration & another class.
- Is $L = NL$?

NL-completeness

- Recall again, to define completeness of a complexity class, we need an appropriate notion of a reduction.
- What kind of reductions will be suitable is guided by a complexity question, like a comparison between the complexity class under consideration & another class.
- Is $L = NL$? ...poly-time (Karp) reductions are much too powerful for L .
- We need to define a suitable 'log-space' reduction.

Log-space reductions



- **Issue:** A log-space TM may not have enough space to write down the whole output $f(x)$ in one shot.

...unless we restrict $|f(x)| = O(\log |x|)$, in which case we're severely restricting the power of the reduction.

Log-space reductions

$$(x, i) \xrightarrow{\text{Log-space TM}} f(x)_i$$

- **Issue:** A log-space TM may not have enough space to write down the whole output $f(x)$ in one shot.
- **Solution:** Have the log-space TM output a bit of $f(x)$.

Log-space reductions

$$(x, i) \xrightarrow{\text{Log-space TM}} f(x)_i$$

- **Issue:** A log-space TM may not have enough space to write down the whole output $f(x)$ in one shot.
- **Solution:** Have the log-space TM output a bit of $f(x)$.
- **Definition:** A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is implicitly log-space computable if
 1. $|f(x)| \leq |x|^c$ for some constant c ,
 2. The following two languages are in L :
$$L_f = \{(x, i) : f(x)_i = 1\} \quad \text{and} \quad L'_f = \{(x, i) : i \leq |f(x)|\}$$