Computational Complexity

Lecture 6: Space complexity

Ronald de Haan
me@ronalddehaan.eu

University of Amsterdam

February 19, 2026

Recap

What we saw last time..

- Limits of diagonalization, relativizing results
- Oracles
- There exist $A, B \subseteq \{0,1\}^*$ such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.

What will we do today?

- Space-bounded computation
- Limits on memory space
- L, NL, PSPACE, NPSPACE
- Logspace reductions
- NL-completeness

Space-bounded computation

- Instead of measuring the number $T(n)$ of steps, we will measure the number $S(n)$ of tape cells used
- For time bounds, $T(n) < n$ typically makes no sense
  - In less than n steps, the machine cannot even read the input
- However, for space bounds, $S(n) < n$ does make sense in some situations
- For space-bounded computation:
  - The input tape is read-only
  - We count how many tape cells on the 'work tapes' are used

SPACE and NSPACE

Definition (SPACE)

Let $S : \mathbb{N} \to \mathbb{N}$ be a function. A decision problem $L \subseteq \Sigma^*$ is in SPACE($S(n)$) if there exists a Turing machine that decides L and that on inputs of length n its tape heads (excluding on the input tape) visit at most $c \cdot S(n)$ tape cells.

Definition (NSPACE)

Let $S : \mathbb{N} \to \mathbb{N}$ be a function. A decision problem $L \subseteq \Sigma^*$ is in NSPACE($S(n)$) if there exists a *nondeterministic* Turing machine that decides L and that on inputs of length n its tape heads (excluding on the input tape) visit at most $c \cdot S(n)$ tape cells.

Some first relations between time and space

Theorem

If $S : \mathbb{N} \to \mathbb{N}$ *is a space-constructible function, then:*

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))}).$$

- Assumption of space-constructibility rules out 'weird' functions.
  - S is *space-constructible* if there exists a TM that computes the function $x \mapsto S(|x|)$ in space $O(S(|x|))$, for each $x \in \{0,1\}^*$

Some space classes

Definition

$$\mathsf{PSPACE} = \bigcup_{c \geq 1} \mathsf{SPACE}(n^c) \qquad \mathsf{L} = \mathsf{SPACE}(\log n)$$

$$\mathsf{NPSPACE} = \bigcup_{c \geq 1} \mathsf{NSPACE}(n^c) \qquad \mathsf{NL} = \mathsf{NSPACE}(\log n)$$

- By the previous theorem, then $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P}$ and $\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}$.
- What is an example of a problem in PSPACE? SAT
- What is an example of a problem in NL? Reachability in graphs

Space Hierarchy Theorem

Theorem

If $f, g : \mathbb{N} \to \mathbb{N}$ are space-constructible functions such that $f(n)$ is $o(g(n))$, then:

$$\mathrm{SPACE}(f(n)) \subsetneq \mathrm{SPACE}(g(n)) \quad \textit{and} \quad \mathrm{NSPACE}(f(n)) \subsetneq \mathrm{NSPACE}(g(n)).$$

- As a result: $\mathrm{L} \subsetneq \mathrm{PSPACE}$ and $\mathrm{NL} \subsetneq \mathrm{NPSPACE}$.

Quantified Boolean Formulas

Definition (QBFs)

A *quantified Boolean formula (QBF)* (in prenex form) is of the form $Q_1 x_1 Q_2 x_2 \cdots Q_m x_m \ \varphi(x_1, \dots, x_m)$, where each Q_i is one of the two quantifiers $\exists$ or $\forall$, where the variables $x_1, \dots, x_m$ range over $\{0, 1\}$, and where φ is a propositional formula (without quantifiers).

Truth of QBFs is defined recursively, based on the typical semantics of $\exists$ and $\forall$.

- For example, $\exists x_1 \forall x_2 \ (x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$ is a QBF

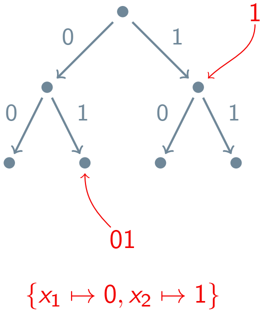
Definition (TQBF)

The language TQBF consists of all QBFs that are true.

Theorem

TQBF *is* PSPACE*-complete (under polynomial-time reductions).*

- Why is TQBF in PSPACE?
  - Use a recursive algorithm.
    For $\varphi = \exists x_i\ \psi$, recurse on $\psi[x_i \mapsto 0]$ and $\psi[x_i \mapsto 1]$, and return 1 if and only if at least one of the recursive calls returns 1. Similarly for $\varphi = \forall x_i\ \psi$.
  - This takes exponential time, but polynomial space:
    - The recursion depth is linear in $|\varphi|$.
    - Space can be reused.
    - With polynomial space, we keep track of the position in the recursion tree, and if we're going up or down.



Theorem

TQBF *is* PSPACE*-complete (under polynomial-time reductions).*

- Why is TQBF PSPACE-hard?
  - Reduce arbitrary polynomial-space computation of TM $\mathbb{M}$ on input x to TQBF; (computation that uses $p(n)$ space takes time at most $2^{q(n)}$)
  - Main idea: construct a QBF $\varphi_{c_1,c_2,t}$ that expresses that the computation leads from configuration c_1 to configuration c_2 within t steps, and return $\varphi_{c_0,c_{\text{accept}},2^{q(n)}}$
  - $\varphi_{c_1,c_2,t}$ has propositional variables that correspond to the configurations c_1, c_2
  - For $t = 1$, this can be done analogously to the proof of the Cook-Levin Theorem
  - For $t > 1$: $\varphi_{c_1,c_2,t}$ expresses $\exists m \, (\varphi_{c_1,m,\lceil t/2 \rceil} \wedge \varphi_{m,c_2,\lceil t/2 \rceil})$ – m is a sequence of vars
  - To avoid exponential blowup, write $\varphi_{c_1,c_2,t}$ in the following way:

$$\exists m \forall c_3 \forall c_4 \; ((\text{“}c_3 = c_1\text{”} \wedge \text{“}c_4 = m\text{”}) \vee (\text{“}c_3 = m\text{”} \wedge \text{“}c_4 = c_2\text{”})) \rightarrow \varphi_{c_3,c_4,\lceil t/2 \rceil}$$

Savitch's Theorem

Theorem (Savitch 1970)

For every space-constructible $S : \mathbb{N} \to \mathbb{N}$ *with* $S(n) \geq \log n$*:*

$$\mathrm{NSPACE}(S(n)) \subseteq \mathrm{SPACE}(S(n)^2).$$

- So, in particular, PSPACE = NPSPACE.
- Proof strategy (for PSPACE = NPSPACE):
  - Show that TQBF is NPSPACE-complete and in PSPACE.

Logspace reductions

- To investigate $L \stackrel{?}{=} NL$, we need reductions that are weak enough.
- Since $L \subseteq NL \subseteq P$, every problem in $L \cup NL$ is reducible to each other using polynomial-time reductions.
  - You can solve any problem in $L \cup NL$ in polynomial time.
  - Reduction: solve the problem, and output a trivial yes-input or a trivial no-input.

Logspace reductions (ct'd)

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is *implicitly logspace computable* if:

- f is *polynomially bounded*, i.e., there exists some c such that $|f(x)| \leq |x|^c$ for every $x \in \{0,1\}^*$, and
- the languages $L_f = \{ (x,i) \mid f(x)_i = 1 \}$ and $L'_f = \{ (x,i) \mid i \leq |f(x)| \}$ are in the complexity class L, where $f(x)_i$ denotes the ith bit of $f(x)$.

Definition

A language B is *logspace-reducible* to a language C (also written $B \leq_\ell C$) if there is a function $f : \{0,1\}^* \to \{0,1\}^*$ that is implicitly logspace computable and for each $x \in \{0,1\}^*$ it holds that $x \in B$ if and only if $f(x) \in C$.

NL-completeness

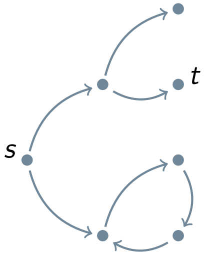
- A language B is NL-*complete* if $B \in$ NL and $C \leq_\ell B$ for every $C \in$ NL.
- Logspace reductions are transitive: if $B \leq_\ell C$ and $C \leq_\ell D$, then $B \leq_\ell D$.
- If $B \leq_\ell C$ and $C \in$ L, then $B \in$ L.

- So, if any NL-complete language is in L, then L $=$ NL.

An NL-complete problem

- Consider graph reachability in directed graphs:

$$\text{PATH} = \{\ (G, s, t) \mid\ G = (V, E) \text{ is a directed graph, } s, t \in V, \text{ and } t \text{ is reachable from } s \text{ in } G\ \}$$



- PATH is NL-complete. Why is it in NL?
  - Keep the current and next node in memory (logspace).
  - Guess the next node, check if they are connected, and forget the previous node.
  - Start at s, accept if you reach t.
  - Keep the length of the path you already visited in memory (logspace), and stop when it is longer than $|V|$ (to avoid looping forever).

Immerman-Szelepcsényi Theorem

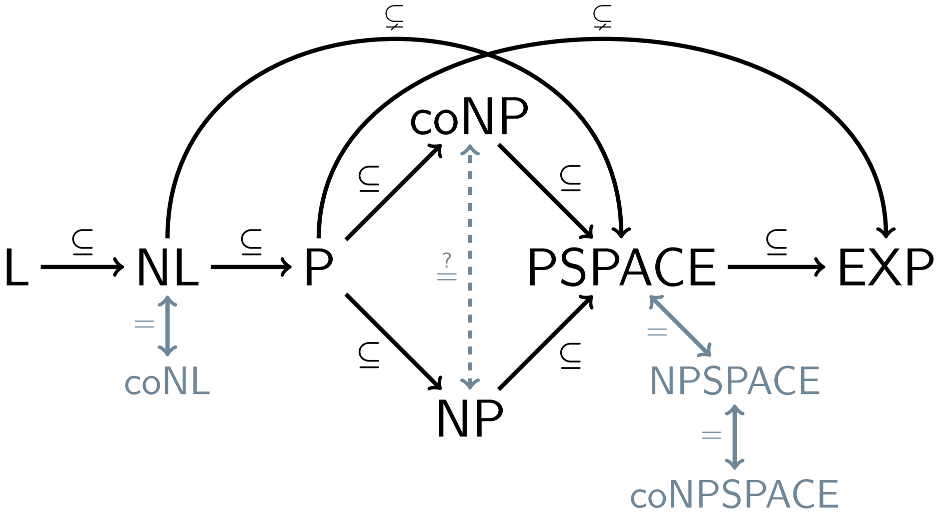
Theorem (Immerman 1988, Szelepcsényi 1987)

For every space-constructible $S : \mathbb{N} \to \mathbb{N}$ *with* $S(n) > \log n$*:*

$$\mathsf{NSPACE}(S(n)) = \mathsf{coNSPACE}(S(n)).$$

- In particular: $\mathsf{NL} = \mathsf{coNL}$.

An overview of complexity classes



Recap

- Space-bounded computation
- Limits on memory space
- L, NL, PSPACE = NPSPACE
- Logspace reductions
- NL-completeness

Next time

- Complexity classes between P and PSPACE
- The Polynomial Hierarchy
- Bounded quantifier alternation
- Alternating Turing machines