

Recap: Space complexity, and NL-completeness

Computational Complexity 2026/2027 — Massimo Lauria

We review space-bounded computation, the subtleties of sublinear space (read-only input, write-once output), the configuration graph as the organising idea of the whole chapter, Savitch and Immerman–Szelepcsényi, and the theory of NL-completeness.

READING. [AB, Sections 4.1, 4.2, 1, and 4.3]. The configuration-graph machinery will reappear when we will prove PSPACE-completeness of TQBF [AB, Thm. 4.13]

Summary of complexity classes

$$L \subseteq NL = \text{coNL} \subseteq P \begin{matrix} \subseteq \text{NP} \\ \subseteq \text{coNP} \end{matrix} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXP} \begin{matrix} \subseteq \text{NEXP} \\ \subseteq \text{coNEXP} \end{matrix}$$

Space-bounded computation

DEFINITION 1 (Space classes). $L \in \text{SPACE}(s(n))$ if some TM decides L visiting at most $O(s(n))$ cells of its *work* tapes on every input of length n ; $\text{NSPACE}(s(n))$ is the analogous nondeterministic class, where the bound must hold on *every* nondeterministic branch. Then

$$L = \text{SPACE}(\log n), \quad \text{NL} = \text{NSPACE}(\log n)$$

$$\text{PSPACE} = \bigcup_c \text{SPACE}(n^c), \quad \text{NPSPACE} = \bigcup_c \text{NSPACE}(n^c).$$

As for time, we only consider *space-constructible* s , and we always assume $s(n) \geq \log n$: the machine must at least be able to store a pointer into its own input.

OBSERVATION 2 (Exercise). Show that $\text{NTIME}(t(n)) \subseteq \text{SPACE}(t(n))$ and $\text{NP} \subseteq \text{PSPACE}$.

Cycle through all candidate certificates $u \in \{0, 1\}^{p(n)}$ in lexicographic order, reusing the same $p(n)$ cells, and run the polynomial-time verifier on each. Space is reusable, time is not.

THEOREM 3 (Space hierarchy, simplified). If f, g are space constructible and $f(n) = o(g(n))$ then $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(g(n))$.

COROLLARY 4. $L \subsetneq \text{PSPACE}$.

Take $f = \log n$, $g = n$. Note the payoff: this is an *unconditional* separation of two classes in the chain above, so at least one of the inclusions between L and PSPACE is strict — we just have no idea which.

[AB, Def. 4.1, Def. 4.5]

With $s(n) < \log n$ the theory degenerates and becomes model-dependent; not worth the trouble.

[AB, Ex. 4.6] gives the concrete instance $3_{\text{SAT}} \in \text{SPACE}(O(n))$.

[AB, Thm. 4.8], Stearns–Hartmanis–Lewis 1965. Proof is diagonalisation, verbatim as for the time hierarchy [AB, Thm. 3.1], only *easier*: the space overhead of a universal machine is a constant factor, not a log factor. Proof omitted, see [AB, Ex. 4.1].

The same trick with P vs. EXP was Lecture 10.

Two nuisances of sublinear space

Once $s(n) < n$ a naive definition would make the model trivial, and both fixes are in the definition.

Input. The input does not fit in the space budget. The machine should be allowed to read the input, but not to use the input cells to perform temporary computations. The solution is to have the input sitting in a special *read-only* tape whose cells are not charged; only work tapes count.

[AB, Fig. 4.1]. This is why $\text{SPACE}(\log n)$ is not vacuous.

Output. If we want to compute a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ where the output is long, we must allow the machine to write it, while forbidding the use of output tape as free scratch memory. There are two standard ways to fix avoid that.

Streaming output.

The output head may only write-and-move-right, or stay put; it never moves left and never re-reads. The output is produced in a streaming fashion. Output cells are not charged.

[AB, Ex. 4.8]

Implicit.

We define two machines: $M_b(x, i)$ decides whether $i \leq |f(x)|$, and $M_f(x, i)$ outputs the i -th bit of $f(x)$ when it exists. Both run in space s . Equivalently, the languages $\{\langle x, i \rangle : f(x)_i = 1\}$ and $\{\langle x, i \rangle : i \leq |f(x)|\}$ are in $\text{SPACE}(s)$.

[AB, Def. 4.16]

PROPOSITION 5 (Exercise). For $s(n) \geq \log n$ the two notions coincide. We may therefore speak of *logspace computable functions* without ambiguity.

[AB, Ex. 4.8]. The direction “implicit $\Rightarrow$ streaming” is a loop over i ; the converse is a re-simulation of the machine from scratch, discarding every output bit except the i -th.

LEMMA 6 (Composition, exercise). If f, g are logspace computable so is $g \circ f$.

[AB, Lem. 4.17] and [AB, Fig. 4.3]. This is exactly what makes relation $\leq_\ell$, defined later, transitive, and what makes NL-completeness a sensible notion.

The point of the proof: $M_{g \circ f}$ simulates M_g on a *virtual* input tape holding $f(x)$, keeping only the head position i (that is $\log |f(x)| = O(\log |x|)$ bits) and re-invoking $M_f(x, i)$ every time M_g reads a cell. Exponential time blow-up, no space blow-up. Note that you cannot do the obvious thing — write $f(x)$ down — which is exactly why the lemma needs a proof.

TAKE AWAY. In space-bounded computation you never store intermediate results; you *recompute* them on demand.

Space as a graph of configurations

Fix M using space $s(n) \geq \log n$ and an input x . A *configuration* is (state, work-tape contents, head positions), so there are $2^{O(s(n))}$ of them and each is describable by $O(s(n))$ bits. The configuration graph $G_{M,x}$ has these as vertices, with an arc $C \rightarrow C'$ when M can move from C to C' in one step. The out-degree is ≤ 1 if M is deterministic, ≤ 2 if nondeterministic. Without loss of generality we assume a

single accepting configuration C_{acc} . This can be ensured if before accepting, the machine erases the work tapes and moves the heads at the beginning of the tapes.

[AB, §4.1.1]

[AB, Claim 4.4]

CLAIM 7 (Locality of the configuration graph). Given x , the graph $G_{M,x}$ is computable in space $O(s(n))$:

- C_{start} and C_{acc} are recognisable in space $O(s(n))$;
- given C, C' , deciding whether (C, C') is an arc is doable in space $O(s(n))$ — indeed it only inspects $O(1)$ cells plus one input bit.

Moreover there is a CNF ϕ of size $O(s(n))$ that for every C, C' , $\phi(C, C')$ evaluates to 1 iff C' is a successor of C .

The CNF version is not needed today, but it is the engine the PSPACE-completeness of TQBF [AB, Thm. 4.13], so we record it now.

OBSERVATION 8. $M(x)$ accepts iff $\langle G_{M,x}, C_{\text{start}}, C_{\text{acc}} \rangle \in \text{PATH}$, where PATH is directed s - t reachability.

[AB, Thm. 4.2]

COROLLARY 9. $\text{NSPACE}(s(n)) \subseteq \text{DTIME}(2^{O(s(n))})$.

Build $G_{M,x}$ explicitly ($2^{O(s)}$ vertices) and run DFS/BFS. Up to logarithmic factors this is *the only* relation we about time-efficient simulation of space bounded computation; improving it would be a major result. Regarding space-efficient simulation of time bounded computation, very recently a surprising result was proved by ¹.

¹ R. Ryan Williams. Simulating time with square-root space. In *STOC 2025*, STOC '25, 2025. URL <https://doi.org/10.1145/3717823.3718225>

THEOREM 10. For every function $t(n) \geq n$ we get

$$\text{DTIME}(t(n)) \subseteq \text{SPACE}(\sqrt{t(n) \log n}).$$

Savitch theorem

THEOREM 11 (Savitch 1970). For space constructible $s(n) \geq \log n$, $\text{NSPACE}(s(n)) \subseteq \text{SPACE}(s(n)^2)$.

[AB, Thm. 4.14]. Proof: the recursion $\text{reach}(u, v, 2^k)$ tries all midpoints w and recurses twice with budget 2^{k-1} ; depth $O(s)$, each frame stores $O(s)$ bits.

COROLLARY 12. $\text{NL} \subseteq \text{SPACE}(\log^2 n)$ and $\text{PSPACE} = \text{NPSpace}$.

So nondeterminism is nearly free for space, in glaring contrast with the time world, where P vs. NP is open. The price is time: Savitch's algorithm runs in $2^{\Theta(s^2)}$ steps, worse than the $2^{O(s)}$ of the DFS above. Whether one can have both simultaneously is open. The proof that TQBF is PSPACE-complete will use a similar idea.

[AB, Ex. 4.12]

Immerman–Szelepcsényi

THEOREM 13 (Immerman 1988, Szelepcsényi 1987). For space constructible $s(n) \geq \log n$, $\text{NSPACE}(s(n)) = \text{coNSPACE}(s(n))$. In particular $\text{NL} = \text{coNL}$.

[AB, Thm. 4.20], [AB, Cor. 4.21]. Full proof on the blackboard.

It suffices to show $\overline{\text{PATH}} \in \text{NL}$, since PATH is NL-complete. The technique is *inductive counting*: let c_i be the number of vertices reachable from the starting vertex in $\leq i$ steps. Given c_i one can nondeterministically verify c_{i+1} in logspace, and given c_n one can certify that the

target vertex is *not* reachable by nondeterministically enumerating exactly c_n reachable vertices in increasing order, all different from the target one. Only c_i , a counter and two vertex names are stored at any time.

A theory of NL-completeness

DEFINITION 14. $B \leq_\ell C$ if there is a logspace computable f with $x \in B \iff f(x) \in C$. C is NL-complete if $C \in \text{NL}$ and $B \leq_\ell C$ for all $B \in \text{NL}$.

Logspace reductions — not polynomial-time ones — are forced on us here: $\text{NL} \subseteq \text{P}$, so under $\leq_p$ every non-trivial language would be NL-complete. By the composition lemma, $\leq_\ell$ is transitive and L is closed downward under it.

THEOREM 15. PATH is NL-complete.

Membership: guess the path one vertex at a time, keeping a counter. Hardness: the reduction $x \mapsto G_{M,x}$ is exactly Claim 7 — each entry of the adjacency matrix of $G_{M,x}$ is one local consistency check, computable in logspace.

COROLLARY 16. $\text{PATH} \in \text{L}$ iff $\text{L} = \text{NL}$.

Interestingly enough, for undirected graphs reachability *is* in L [Reingold, 2008], via zig-zag expanders; the directed case is open. One of the few places where a natural sub-problem of an open question got resolved.

TAKE AWAY. PATH is to NL what 3SAT is to NP , and for the same reason: the configuration graph, resp. the tableau, is a local object. What changes is the reduction budget, which forces us to recompute rather than store.

Certificates for NL computation

DEFINITION 17 (NL with a witness tape). $L \in \text{NL}$ iff there is a deterministic logspace verifier M with an extra *read-once* witness tape (the head stays or moves right, never left) and a polynomial p such that $x \in L \iff \exists u \in \{0,1\}^{p(|x|)} M(x,u) = 1$.

The read-once restriction is not cosmetic. A read-once witness is just a transcript of the nondeterministic choices, in the order in which they are taken; a two-way head would let the verifier *store* the witness for free by re-reading it, which is precisely the resource the whole theory denies.

EXERCISE 18. Suppose we define NP-completeness using logspace reductions instead of polynomial-time reductions. Show that SAT and 3-SAT continue to be NP-complete under this new definition. Show that most reduction between NP-complete problems from [AB, Chapter 2] are actually logspace reductions.

Compare with NP: nobody believes an analogous counting argument gives $\text{NP} = \text{coNP}$, because there the certificates cannot be recomputed for free. Again: space is reusable.

[AB, Def. 4.16]

[AB, Ex. 4.3]. Same phenomenon as using $\leq_p$ inside P .

[AB, Thm. 4.18]

See [AB, Chapters 7 and 21] for the expander machinery behind Reingold [2008].

[AB, Def. 4.19], [AB, Fig. 4.4]

[AB, Ex. 4.6]

EXERCISE 19. Show that if the verifier of Definition 4.19 may move its witness head back and forth, the class defined becomes NP.

[AB, Ex. 4.7]

EXERCISE 20. Can you check a satisfying assignment of a 3-CNF in logspace with one-directional access to the assignment? Say what goes wrong, and reconcile your answer with Exercise 19.

Hint: a variable occurs in many clauses, scattered along the formula. What would you have to store, and how does that compare with the witness itself?

Homework

- [AB] Exercises 4.1, 4.3, 4.5, 4.6, 4.7, 4.8, 4.11.
- Exercises 18, 19 and 20 above.
- [AB, Ex. 4.8] Prove the equivalence of write-once and implicit logspace computability. Then check that the composition lemma is *false* if “logspace” is replaced by “logspace with a read-write output tape not counted against the space bound”.
- [AB, Ex. 4.12] Is it possible to solve PATH in $\log^n$ space and polynomial time simultaneously?
- (Optional, looking ahead.) Read [RY, Chapter 10] and see how communication complexity gives *unconditional* space lower bounds for streaming algorithms and branching programs — the only route we have to lower bounds against restricted models of space.

References

- Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009. ISBN 978-0-521-42426-4. DOI: 10.1017/CBO9780511804090.
- Eli Ben-Sasson. *Expansion in Proof Complexity*. PhD thesis, Hebrew University of Jerusalem, Jerusalem, 2001. URL <http://www.math.ias.edu/~avi/STUDENTS/ebsthesis.ps>.
- Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-24507-7. DOI: 10.1007/978-3-642-24508-4.
- Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, Cambridge, 2020. ISBN 978-1-108-49798-5. DOI: 10.1017/9781108671644.
- O. Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.
- R. Ryan Williams. Simulating time with square-root space. In *STOC 2025*, STOC '25, 2025. URL <https://doi.org/10.1145/3717823.3718225>.