The Computational Complexity Column

by

Michal Koucký

Computer Science Institute, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
koucky@iuuk.mff.cuni.cz
https://iuuk.mff.cuni.cz/~koucky/

Using Hardness vs Randomness to Design Low-Space Algorithms

Edward Pyne*
MIT
epyne@mit.edu

Roei Tell†
University of Toronto
roei@cs.toronto.edu

Abstract

Can we use "hardness vs randomness" techniques to design low-space algorithms? This text surveys a sequence of recent works showing ways to do that. These works designed algorithms for certified derandomization and for catalytic computation (which work unconditionally), derandomization and isolation algorithms from remarkably mild assumptions, and "win-win" pairs of algorithms that, for every input, solve either derandomization or another important problem (e.g., *s*-*t* connectivity) on the input.

Underlying these constructions are new, specialized "hardness vs randomness" tools for the setting of low-space algorithms. We describe these technical tools, most notably constructions of pseudorandom generators whose reconstruction algorithm (i.e., the security reduction) is a deterministic low-space algorithm. We also explain a key part of obtaining deterministic reconstruction, which is deterministic transformations of distinguishers to bit-predictors.

We pose a host of open questions that explore new ways of using hardness vs randomness to design low-space algorithms. These questions address problems in derandomization, catalytic computation, explicit constructions, learning algorithms, and more.

*Supported by an NSF Graduate Research Fellowship.

†Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2024-04490.

Contents

1 Hardness vs Randomness when Memory is Scarce

One of the most influential techniques in theoretical computer science is the "hardness vs randomness" paradigm. Originating in cryptography [10, 87], this technique transforms lemons into lemonade: It takes hard problems, which no efficient algorithm can solve, and uses them to design algorithms for other problems. Perhaps the most well-known example is the Nisan-Wigderson generator [58], which (combined with additional tools by Impagliazzo and Wigderson [46]) yields the following textbook result:

Theorem 1.1 ([46]). *If* $\mathbf{E} = \mathbf{TIME}[2^{O(n)}]$ *has a problem hard for circuits of size* $2^{.01 \cdot n}$ *on all input lengths, then there is a deterministic polytime algorithm that, given a circuit* $D : \{0,1\}^n \to \{0,1\}$*, estimates* $\mathbb{E}[D]$ *up to an additive error of* $1/n$.

Since the problem of estimating the expectation of a given circuit is complete for **prBPP**, the conclusion in Theorem 1.1 implies that **prBPP** = **prP**.

Over the three decades since the underlying tools were developed, they have been used to *design other types of algorithms*. And most notably, many of the algorithms designed using these tools are *unconditional*, i.e. do not actually need to start from a hardness assumption! Among other things, hardness vs randomness tools have been used to unconditionally obtain constructions of randomness extractors, condensers, and expander graphs [45, 75, 76], learning algorithms [12], pseudodeterministic explicit constructions [15], circuit lower bounds [13], and worst-case to average-case reductions [43].

Can we design low-space algorithms using hardness vs randomness? Early on, Klivans and van Melkebeek [49] showed that hardness vs randomness tools can be adapted to yield low-space algorithms. In particular, under a stronger hardness assumption than in Theorem 1.1 – a hard problem in $\mathbf{SPACE}[O(n)]$ rather than only in $\mathbf{E} \supseteq \mathbf{SPACE}[O(n)]$ – the concluded algorithm can run in *logarithmic space*:

Theorem 1.2 ([49]). *If* $\mathbf{SPACE}[O(n)]$ *has a problem hard for circuits of size* $2^{.01 \cdot n}$ *on all input lengths, then there is a deterministic logspace algorithm that, given a circuit* $D : \{0,1\}^n \to \{0,1\}$ *that can be evaluated in space* $O(\log n)$*, estimates* $\mathbb{E}[D]$ *up to an additive error of* $1/n$.

Theorem 1.2, however, had less follow-ups compared to Theorem 1.1. In fact, we are not aware of *any* work prior to 2023 that used hardness vs randomness to design logspace algorithms. Why is this?

One explanation is that we didn't know how to exploit the fact that when designing low-space algorithms, we are usually trying to solve potentially easier problems, compared to the conclusion of Theorems 1.1 and 1.2. The untapped potential is that when tackling easier problems, one may hope to weaken the hardness assumption, in which case (using the many existing ideas) we could eventually obtain unconditional algorithms. For example, consider the problem of derandomizing probabilistic logspace machines. This problem reduces to estimating the expectation of a Read-Once Branching Program (ROBP),[1] a weak computational model for which exponential lower bounds are well-known (see, e.g., [9]). However, past applications of hardness vs randomness to this problem made little use of this fact (see [30, Section 1.1]).

[1] This is because the procedure $B_x(\cdot)$ obtained by fixing an input x to the machine (i.e., B_x takes the randomness r as its input) is an ROBP.

The good news. The main focus of this text is a sequence of recent works [28–30, 52, 60, 61] that used hardness vs randomness tools to design low-space algorithms for important problems – including derandomization, *s*-*t* connectivity, function composition, and catalytic computation – where some of these algorithms work unconditionally and others work under mild assumptions.

A key bottleneck to get through was resolving the issue above, i.e. figuring out that there *is* a critical feature of what we call "the logspace setting" (i.e., of various relaxed problems that we try to solve using logspace algorithms) that hardness to randomness tools can exploit.[2] Another facilitator of these works is recent developments in hardness vs randomness more generally (i.e., for algorithms that aren't necessarily low-space; see, e.g., the survey [24]), and in particular a sequence of new technical tools for the time-bounded setting, which the works above migrated and adapted to obtain new space-bounded tools.

What's in this text? The goal of this survey is to explain three things:

1. **How to exploit the logspace setting.** In Section 2 we will present the main new technical feature of hardness vs randomness tools that can be obtained in the logspace setting (and also possibly beyond that), which is called deterministic reconstruction.

2. **The new algorithms.** In Section 3, we present low-space algorithms from recent years that use hardness vs randomness, and in particular that exploit deterministic reconstruction. Specifically, we will mention certified derandomization, catalytic algorithms, win-win pairs of algorithms (for *s*-*t* connectivity and for efficient copmosition), and derandomization of BPL from remarkably weak assumptions.

3. **Underlying technical tools: New reconstructive generators.** Finally, in Section 4, we explain how to achieve deterministic reconstruction: first in the special case of trying to prove $\mathsf{BPL} = \mathsf{L}$, and then for several other problems mentioned above. We will introduce a concept called distinguish-to-predict transformations, and describe some new pseudorandom generators in more detail.

In Section 5 we suggest a host of interesting and potentially tractable open problems, which explore new ways of using hardness vs randomness to design low-space algorithms.

Finally, an extra goodie of these developments is elegant proofs for several classical results, which use the concepts introduced above; we present these proofs in Appendix A.

2 Exploiting the Logspace Setting: Deterministic Reconstruction

To explain the new feature of hardness vs randomness tools that can be achieved in the logspace setting, let us recall how these tools work. The "recipe" behind them looks roughly like the following.

A hardness vs randomness recipe. There is a string $f \in \{0,1\}^T$, which we call the hard string, that we hope (or assume) is a truth table that cannot be computed by circuits of size $s \ll T$.[3]

[2] For the special case of derandomizing probabilistic logspace machines (i.e., trying to prove $\mathsf{BPL} = \mathsf{L}$), the technical proof of this is simple and accessible – we encourage readers to check out Sections 4.1.2 and 4.2.1!

[3] By which we mean there is no size-s circuit C that, given $j \in [T]$, outputs f_j. If there is such a circuit C, we say its truth table is f, or that it computes f.

We give f to a deterministic algorithm GEN that has the following behavior. Given a circuit D, the algorithm $\mathsf{GEN}^f(D)$ outputs an estimate for $\mathbb{E}[D]$. If this estimate is good (say, within an additive factor of $1/6$ of the true expectation), then we solved an interesting problem (i.e., deterministically estimated D's acceptance probability). Otherwise, the proofs of correctness show that there exists a circuit C with the following properties:

- The size of C is less than s, and C makes queries to D.
- The truth-table of C^D is f.

Thus, if GEN^f fails to estimate $\mathbb{E}[D]$ well, and D is a small circuit (of size less than s), then there exists a small circuit for f, namely C^D. So assuming f has no size-s circuits, the derandomization must succeed.

The complexity of reconstruction. The recipe above shows that when GEN fails, f can be compressed, to the form of a small circuit. Being able to compress a given string f is potentially useful, but:

Where does this circuit C come from? Can we find it?

To formalize this question, we extend our recipe to a pair of algorithms $(\mathsf{GEN}, \mathsf{REC})$.

- The generator $\mathsf{GEN}^f(D)$ works as before, but if it does a bad job...
- ... the reconstruction algorithm $\mathsf{REC}^f(D)$ prints a small circuit C such that C^D computes f.[4]

In [46, 58], the reconstruction REC is non-explicit and inefficient. However, it was quickly realized [47] that in certain settings we can also design *efficient, probabilistic* reconstruction algorithms; for example, recent works showed that this is true when f is computable by uniform low-depth circuits (this line-of-work is often referred to as "uniform hardness vs randomness"; see, e.g., [21,22,47,77]).

The complexity of the reconstruction REC turns out to be more important than how it may initially seem. This is because the recipe above is used in non-obvious ways, some of which yield *unconditional* algorithms. Indeed, in these applications REC plays several roles in disguise; for example:

1. When the recipe is used for conditional results in the straightforward way (as in, say, [22, 46, 47, 49, 58]), the complexity of REC determines the hardness assumption.
2. The recipe can be instantiated with an *intentionally faulty* GEN (i.e., in a setting where GEN is guaranteed to fail), in which case REC is the actual algorithm, and it unconditionally works. In this case, the complexity of REC is the complexity of the final algorithm (see, e.g., [12, 43, 76]).
3. The recipe can also be instantiated with both GEN and REC trying to solve the same problem (!). In this case, the guarantee that at least one of them works means that we can unconditionally solve the problem, and the complexity of our algorithm is the worst of both worlds (i.e., the maximum among the complexity of GEN and of REC; see, e.g., [13, 15, 45]).

An important open problem is extending the class of functions for which we can get an efficient probabilistic REC (see, e.g., open problems in [22, 25]). We will now ask about another direction.

[4] In cryptography the reconstruction algorithm is usually called "the security reduction".

Can we hope for a generator with deterministic reconstruction? That is, can we hope to design $(\mathsf{GEN}, \mathsf{REC})$ such that for every f, whenever the generator $\mathsf{GEN}^f(D)$ fails, the reconstruction $\mathsf{REC}^f(D)$ efficiently and *deterministically* compresses f to a small circuit?

This is not a pipe dream: such a construction trivially exists if $\mathbf{prBPP} = \mathbf{prP}$ (because then there is GEN that simply ignores f and estimates $\mathbb{E}[D]$ correctly). However, for general D's, this might be too good to be proved at the moment, since it would imply that $\mathbf{prBPP} = \mathbf{prZPP}$. It is instructive to see why. Consider an algorithm that gets input D and wants to estimate $\mathbb{E}[D]$. Given $(\mathsf{GEN}, \mathsf{REC})$ with deterministic reconstruction, the algorithm can generate a random f, run $\mathsf{REC}^f(D)$, and if $\mathsf{REC}^f(D)$ fails to print a circuit for f, the algorithm uses $\mathsf{GEN}^f(D)$ to estimate $\mathbb{E}[D]$. Indeed, whenever REC fails, the algorithm can be certain that $\mathsf{GEN}^f(D)$ succeeds, and REC fails for most f's, since they are incompressible.[5]

Thus, deterministic reconstruction isn't just a way of transforming hypothesized hardness into randomness – it is also unconditional derandomization by itself (for a relevant class of D's).

> The main technical discovery opening the door for the recent line-of-works, from [60], is that for *restricted classes of D's* that arise when studying problems in the logspace setting, we can design $(\mathsf{GEN}, \mathsf{REC})$ whose reconstruction REC is deterministic. Moreover, both GEN and REC can be *logspace* algorithms, i.e. running in deterministic space $O(\log |f|)$.

In fact, there are by now several constructions of generators with deterministic reconstruction for restricted classes of D's. We survey some known constructions and explain which classes of D's they work for in Section 4. For now, we encourage the reader to think of the simplest case, which is when D is a Read-Once Branching Program (ROBP); as mentioned in Section 1, ROBPs arise when trying to prove $\mathbf{BPL} = \mathbf{L}$.

Remark 2.1. In the recipe above, we may also allow f to be a hard string that depends on D (and some generators presented in this survey will do so). In this setting it is more convenient to think of both $D = D_x$ and $f = f(x)$ as functions of an input x. That is, GEN_f gets input x tries to estimate $\mathbb{E}[D_x]$ by computing $f(x)$; and if it fails, a reconstruction $\mathsf{REC}_f(x)$ computes $f(x)$ too efficiently (even a small circuit whose truth-table is $f(x)$). This type of generator was introduced by Goldreich [34, 36], who called it a targeted generator (where x is the "target"). We elaborate on this in Section 4.3.

3 New Low-Space Algorithms

Now we have our deterministic reconstruction hammer, what nails can we hit with it? We first cover a straightforward application that we call certified derandomization. However, as is usually the case with hardness to randomness, there are many non-obvious applications which we will cover subsequently. As we go, we provide forward pointers to the technical tools (i.e., to the generators and reconstruction algorithms presented in Section 4) required for each application.

3.1 Certified Derandomization

Recall that Theorem 1.2 states that, assuming $\mathbf{SPACE}[O(n)]$ is hard for circuits of size $2^{.01 \cdot n}$, then $\mathbf{BPL} = \mathbf{L}$. But what if the assumption is false? In that case, the derandomized algorithm

[5] In fact, a slightly more general notion of "deterministic reconstruction" is equivalent to $\mathbf{prBPP} = \mathbf{prZPP}$; see [52].

for **BPL** will fail silently: it will still return a value, but this value has no correctness guarantee. With deterministic reconstruction, we can do better:

Theorem 3.1 ([60])**.** *Let $\mathcal{L}_{hard} \in$ **SPACE**$[O(n)]$. For every $\varepsilon > 0$ and $L \in$ **BPL** there is a deterministic logspace algorithm $\mathcal{A}$ that on input $x \in \{0,1\}^n$, either:*

1. *Prints L_x.*
2. *Outputs "I cannot produce an output because the hardness assumption is false", followed by a circuit C of size $2^{\varepsilon n'}$ for $\mathcal{L}_{hard}$ on inputs of size $n' = \Theta(\log n)$.*

We briefly sketch how this is a consequence of deterministic reconstruction. We try to use GEN^f to derandomize the **BPL** machine on x, where f is the truth-table of $\mathcal{L}_{\text{hard}}$ on n'-bit inputs. But before trusting the output of GEN^f, we first determine if REC^f outputs a small circuit for f, and if so we return that circuit; otherwise, we use the output of GEN^f to derandomize as usual. If the hardness assumption is true, no such small circuit exists, and hence the former situation will never occur, and hence our algorithm will always derandomize L in logspace. But if we are wrong and there is such a small circuit, we either get a certificate of this fact (i.e., the circuit that REC^f produces), or are guaranteed that GEN^f works nonetheless!

This result has something of a "win-win" favor (either we derandomize or we get a small circuit for f), which will come up again in subsequent results.

3.2 Catalytic algorithms

How valuable is a full memory? In the setting of catalytic algorithms, we augment a standard logspace algorithm with a much larger catalytic space. This space has an arbitrary initial configuration, which can be edited during the computation but must be restored to the initial configuration at the end of the computation. Introduced in 2014 by Buhrman et. al. [6], the model has played a central role in recent breakthrough results, such as the Tree Evaluation algorithm of Cook and Mertz [17, 18] and the simulation of time-T computation in space $\tilde{O}(\sqrt{T})$ by Williams [85].

A main technique to build catalytic algorithms (i.e., algorithms that utilize catalytic space) was dubbed "compress or random" by Mertz [54]. This technique starts with some dense property $\mathcal{P}$ of strings that is useful for designing an algorithm (e.g., being a hard truth-table). For the catalytic algorithm, if the initial content τ of its catalytic tape satisfies $\mathcal{P}$, the algorithm can just use τ. However, if $\tau \notin \mathcal{P}$, then the contents τ of the catalytic space lies in some small set of strings (i.e., the complement of $\mathcal{P}$), and hence the content τ is – at least information-theoretically – compressible! If we could algorithmically compress τ in this case, we would unconditionally design an algorithm by a win-win analysis: either the good property holds, or we free up a large amount of space and solve the problem by brute force.

But how do we implement this technique? The main challenge is that we need a deterministic, low-space algorithm that compresses τ when τ is bad. Indeed, now comes the punch-line – this is exactly what deterministic reconstruction provides! Hence, hardness vs randomness tools with deterministic reconstructions are useful for designing catalytic algorithms.

Two algorithms, and a possibility. First, a basic open problem about probabilistic logspace is search-to-decision reductions. Specifically, an algorithm for the *decision* problem of derandomization (i.e., decide whether $\mathbb{E}[D]$ is high or low) is currently not known to imply an algorithm of similar space complexity for the corresponding *search* problem (i.e., print a distribution that is pseudorandom for D).

For catalytic algorithms, however, this was recently proved to be true – indeed, using "compress or random" with a generator that has deterministic reconstruction. In particular, recall that **CL** denotes the class of problems solvable with logarithmic space and polynomial catalytic space; then:

Theorem 3.2 (Informal, see Theorem 5.4 [52]). *Let $\mathcal{D}$ be a class of circuits evaluable in* **CL**, *and suppose that there is a* **CL** *algorithm that estimates $\mathbb{E}[D]$ for $D \in \mathcal{D}$ up to additive error* $1/10n$. *Then there is a* **CL** *algorithm that, given D, prints a distribution that* $(1/3)$*-fools D.*

A second catalytic algorithm using this technique simulates **BPL** in **CL**. This result was already shown in the first paper about catalytic computing [6], relying on algebraic techniques and on the somewhat indirect route "**BPL** $\subseteq$ logspace-uniform-**TC**^{1} $\subseteq$ **CL**". A more direct algorithm simulating **BPL** in **CL** was recently shown, using "compress or random" and a generator with deterministic reconstruction.

Theorem 3.3 ([6], an alternative proof in [28]). *We have that* **BPL** $\subseteq$ **CL**.

Another alternative proof of Theorem 3.3 follows from the fact, proved recently in [14,48,62], that randomized catalytic space coincides with deterministic catalytic space (i.e., that **CBPL** = **CL**; the last of these works even showed that non-deterministic catalytic algorithms can be simulated by deterministic ones). The original algorithm in [14] was, indeed, a "compress-or-random" algorithm using a generator with deterministic reconstruction. However, in subsequent work [48] a simpler algorithm was shown, using a compression technique that does not involve hardness-vs-randomness.

We believe that deterministic reconstruction still has a major role to play in catalytic algorithms, because it gives a highly generic compression technique. We suggest this as an open problem in Section 5.

3.3 Win-win pairs of algorithms

As mentioned in Section 2, hardness vs randomness can turn lemons into lemonade, by turning the *non*-existence of efficient algorithms for one problem into an efficient algorithm for another problem.

Interestingly, recent works [29,61] go further than that, by showing an "instance-wise" win-win between derandomization and two fundamental problems in space complexity. Specifically, they showed that for every fixed input x, either we can derandomize **BPL** on x, or we can solve a fundamental problem in space complexity on x (i.e., much more efficiently than the best currently known algorithm for that problem).

3.3.1 Graph connectivity

First, consider the problem of s-t-connectivity, where we are given a directed graph $G = (V, E)$ on n vertices and two vertices s, t, and need to decide if there is a path from s to t.

This problem is complete for nondeterministic logspace (**NL**), and can be solved in space $O(\log^2 n)$ via the famous result of Savitch [66]. However, Savitch's algorithm runs in time $2^{O(\log^2 n)} \gg \text{poly}(n)$. The problem can also be solved in time $O(n^2)$ and space $O(n)$, via a simple Breadth First Search. But what about a time *and* space efficient solution? Whether such an algorithm exists is wide open!

A result of Barnes et. al. [5] solves s-t-connectivity in polynomial time and space $O(n/2^{\sqrt{\log n}}) = o(n)$, but even the following is up in the air:

Question 3.4. Is there a constant $\varepsilon > 0$ and an s-t-connectivity algorithm running in simultaneous time $n^{\log^{1-\varepsilon} n}$ and space $n^{1-\varepsilon}$?

In a restricted computational model that captures all known s-t-connectivity algorithms, an algorithm with these parameters matching Question 3.4 is known not to exist [20, 53, 59]. However, this restriced model does not capture general algorithms (cf., e.g., the difference between [19] and [18]).

A win-win pair of algorithms. If one is pessimistic, and believes that there isn't an algorithm as in Question 3.4, what does that imply? A natural hope is to use randomness vs randomness to build a useful algorithm from this hardness. Classical hardness vs randomness yields non-uniform circuits (or, in some settings, probabilistic algorithms), but with deterministic reconstruction we can do better.

Specifically, a recent result of [29] ties together s-t connectivity with the problem of estimating random walk probabilities (which is complete for **BPL**): For *every* given graph G, at least one of these problems can be solved on G much more efficiently than the best currently known algorithm.

Theorem 3.5 ([29, Theorem 1]). *There are algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for every graph G on n vertices, one of the following holds:*

- *$\mathcal{A}_1(G)$ solves s-t connectivity in G in polynomial time and polylogarithmic space.*
- *$\mathcal{A}_2(G)$ estimates length-n random walk probabilities in G in non-deterministic logspace.*[6]

Moreover, both algorithms report if they fail to compute the desired answer, and do not exceed their resource bounds in any case.

How is deterministic reconstruction used? The use of deterministic reconstruction in the proof of Theorem 3.5 is not straightforward, and proceeds roughly as follows. Consider the following algorithm for computing s-t-connectivity in small time and space. First, compute all 1-step connectivity information (i.e. for every vertex pair u, v, decide if there exists a 1-step path from u to v), and write this down in a matrix $M_1 \in \{0,1\}^{n \times n}$. But rather than storing M_1 on the worktape using n^2 bits, try to *compress* M_1 to a circuit C_1 of size polylog(n). If this compression step fails, abort. Otherwise, compute the 2-step connectivity information M_2 (which is easy to do given M_1), delete C_1, and compress M_2 into C_2. After n iterations, we obtain a compressed encoding of M_n, which holds the connectivity information of the graph.

But what compression algorithm can we use? The key idea of [29] is to use a deterministic reconstruction algorithm REC as our compression algorithm in each iteration i, with $f = M_i$ as the hard string. When the compression *fails* it must be the case GEN^{M_i} succeeds in estimating random walk probabilities. Since we can produce each matrix M_i in **NL**, either all the compression algorithms work (and we solve connectivity quickly and in low space) or we obtain a *nondeterministic* derandomization of walks on G.

In [29, 61] they showed that this approach can be instantiated in a more general way, yielding a new targeted generator for the logspace setting. We elaborate on this in Section 4.3.

[6] The estimation is up to an additive $1/\operatorname{poly}(n)$ error. The algorithm runs in logspace, makes non-deterministic guesses, and either declares *fail* if the guess sequence is bad, or outputs a single canonical matrix only depends on the graph G, or outputs a special symbol $\perp$ indicating that $\mathcal{A}_2$ does not succeed on G (in which case $\mathcal{A}_1$ succeeds on G).

A win-win for complexity classes. Theorem 3.5 gives a pair of algorithms such that, for every input G, at least one of these algorithms works. Working in a scaled-up regime and with different parameters (but with the same fundamental idea), in [29] they deduced a win-win for classical complexity classes.

Theorem 3.6 ([29, Theorem 2]). *For every constant $\varepsilon > 0$, at least one of the following holds:*

- $\mathsf{NSPACE}[n] \subseteq i.o.\mathsf{TISP}\left[2^{O(n^{2-\varepsilon})}, n^{O(1)}\right]$.
- $\mathsf{BPSPACE}[n] \subseteq \mathsf{SPACE}\left[O(n^{1+\varepsilon})\right]$.

Note that the second case in Theorem 3.6 does better than a simple scale-up of the second case in Theorem 3.5, because the simulation of **BPSPACE** (i.e., the derandomization) does not use non-determinism.

3.3.2 Composing low-space algorithms

Consider the following fundamental question: what is the complexity of *composing* low-space algorithms? (Jumping ahead, we will explain how this question was recently attacked using hardness vs randomness.)

Given a function $f : \{0,1\}^n \to \{0,1\}^n$ computable in time $T \geq n$ and $S = O(\log n)$, there are two basic ways to compute the function $x \to f(f(x))$. First, a naive strategy: compute $y = f(x)$, store it in the workspace, then compute $f(y)$. This method runs in time $O(T)$, but space at least $n \gg S$ due to needing to store y. This space overhead means that essentially every low-space algorithm in theoretical computer science makes use of a second option, coined "emulative composition" by Goldreich [33], or "composition of space-bounded algorithms". Here, we begin to compute $f(y)$, and each time the algorithm queries a bit of y, we recompute it on the fly. This reduces the space to $O(S)$, but quadratically blows up the time, i.e. to T^2 instead of T. One may also interpolate between the two methods using sub-quadratic time and super-logarithmic space, as long as the time-space product is at least $\tilde{O}(T^2 \cdot S)$ (see [61, Proposition 4.1]).

A seemingly natural question is whether we can get the best of both worlds. That is, can we compose algorithms in low space without paying a significant time overhead? It turns out that for linear time algorithms, i.e. $T(n) = O(n)$, the answer is *unconditionally* no:

Theorem 3.7 ([86], appearing in [61, Theorem 1.2]). *There is a length-preserving function $f : \{0,1\}^* \to \{0,1\}^*$ computable in simultaneous linear time and logspace, where any algorithm computing $x \to f(f(x))$ must have time-space product at least $n^{1.33}$.*

Thus, for linear time, composition incurs an unavoidable time-space overhead. An obvious next step is to ask if this lower bound holds also for large polynomial time bounds T. In [61] they showed that proving such a statement implies average-case derandomization of logspace:

Theorem 3.8 ([61, Theorem 1.3]). *Suppose that there is $\delta > 0$ such that for every polynomial $T(n)$ and constant $\varepsilon > 0$ the following holds. There is a function f computable in time T and space $O(\log n)$ such that any algorithm computing $f(f(x))$ successfully on an ε-fraction of inputs $x \in \{0,1\}^n$ requires time-space product $T^{1+\delta}$.*[7] *Then* $\mathsf{BPL} \subseteq \cap_{\varepsilon>0}\mathsf{avg}_\varepsilon\mathsf{L}$.

Theorem 3.8 was also extended in [61] to the setting of composing f for k times (i.e., asking whether low-space algorithms require time $T^{\Omega(k)}$), and to the scaled-up setting, in which we focus on worst-case composition and derandomization (rather than average-case).

[7] In this statement we refer to the runtime of both $f(x)$ and $f(f(x))$ as a function $T = T(n)$ of the length of the input $x \in \{0,1\}^n$ to the composition. We could alternatively describe f_1 as running in time T and outputting T bits, and describe f_2 as running in linear time (in T).

A word about the technique. The strategy for proving Theorem 3.8 is to again construct a win-win pair of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, as follows: For every x, either $\mathcal{A}_1(x)$ outputs $f(f(x))$ in small time and space, or $\mathcal{A}_2(x)$ simulates the (predetermined) **BPL** machine on input x.

The algorithms are designed in similar fashion to the win-win pair in Theorem 3.5, except that the generator now does not use the i-step connectivity matrices (for $i = 1, ..., n$) as a source of hardness, but instead uses the strings $f(x), f(f(x)), ...$ obtained by repeatedly computing f. (Indeed, in the version in Theorem 3.8 it uses the two strings $f(x)$ and $f(f(x))$, and when considering hardness of k-wise composition, it uses $f^{(i)}(x)$ for $i = 1, ..., k$). The main challenge comes in the reconstruction argument: whenever the generator fails, we want to compute the last layer $f^{(k)}(x)$ using near-linear time and polylogarithmic space. The technical tools described so far don't seem to suffice for this purpose, since the reconstruction runs in large poly(T) time. In [61] they showed a new $(\mathsf{GEN}, \mathsf{REC})$ pair (i.e., a generator and deterministic reconstruction) wherein REC runs in near-linear time and polylogarithmic space, and is also read-once over its input (i.e., reads its input in streaming fashion, bit-by-bit in order), which is another feature needed to avoid time blow-ups when implementing the idea above. This is described in detail in Section 4.3.2.

3.4 Derandomization and isolation: $\mathsf{BPL} = \mathsf{L}$ using hardness vs randomness?

The question of **BPL** = **L** has been studied for decades via combinatorial constructions of pseudorandom generators for ROBPs and related models (see, e.g., the surveys [44, 65]). Can we instead attack this question using hardness vs randomness? The suggestion in [28] is as follows:

> Reduce **BPL** = **L** (or relaxed versions of it) to lower bounds that are so weak that we can unconditionally prove them.

Their idea was to rely on deterministic reconstruction to reduce derandomization to lower bounds for *uniform, deterministic algorithms*, and in particular for classes of weak uniform deterministic circuits, in which case proving these lower bounds seems tractable. (For context, recall that versions of **BPP** = **P** are known to follow from certain hardness for uniform *probabilistic* algorithms [22, 47, 51, 77].)

3.4.1 Derandomization from remarkably weak lower bounds

We present two appealing instantiations of this approach, from [28] and [61] (improving on a prior result from [29]), which reduce a relaxed version of **BPL** = **L** to remarkably weak hardness assumptions. The conclusions below will be derandomization of linear space (rather than logspace), and the assumptions will be extensions of unconditionally known lower bounds.

Theorem 3.9 ([28, Theorem 2]). *Suppose that* $\mathbf{SPACE}[O(n)]$ *is hard for* $\mathbf{SPACE}[C \cdot n]$*-uniform* $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ *circuits of size* $2^{\varepsilon \cdot n}$,[8] *for some* $\varepsilon > 0$ *and all* $C > 1$. *Then,* $\mathbf{BPSPACE}[O(n)] \subseteq \mathbf{i.o.SPACE}[O(n)]$.

The assumption in Theorem 3.9 "almost" follows from a simple diagonalization argument, where the only problem is that the circuit in the lower bound is printed by an algorithm using

[8] The notation $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ refers to $\mathbf{TC}^0$ circuits with oracle access to ROBPs, where the size bound $2^{\varepsilon \cdot n}$ accounts for the total description size of the circuit and of the ROBP together.

space $C \cdot n$, whereas the upper bound uses space $c \cdot n$, where $c < C$. This obstacle is not insurmountable. In fact, a scaled-down version of the hypothesis in Theorem 3.9 (referring to logspace and to polynomial-sized circuits) is unconditionally known, using the techniques of Santhanam and Williams [73] (see [28, Proposition 1.2]). Thus, scaling up the known lower bound would suffice to prove that $\mathbf{BPSPACE}[O(n)] \subseteq \mathbf{i.o.SPACE}[O(n)]$.[9]

Theorem 3.10 ([61, Theorem 1.6])**.** *There is $c > 1$ such that the following holds. Suppose that the following is true:*

> *There is an algorithm that gets input 1^n, runs in space $O(\log n)$, and outputs a list of n-bit strings $f_{n,1}, ..., f_{n,m}$ such that every* deterministic uniform one-pass streaming algorithm *that runs in space* $(\log n)^c$ *and time n^c fails to compress $f_{n,i}$ to size* polylog(n)*, for some i.*[10]

Then, $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[O(n)]$.

The assumption in Theorem 3.10 is appealing because it refers to a very weak computational model, and in particular, because the reduction of derandomization to a lower bound does not incur overhead in the computational model. To see this, recall that deterministic uniform one-pass streaming algorithms are essentially equivalent to uniform deterministic ROBPs.[11] Thus, Theorem 3.10 reduces estimating the acceptance probability of ROBPs to a lower bound for ROBPs (and moreover, for uniform deterministic ROBPs). This is particularly striking, because exponential lower bounds for ROBPs, even non-uniform ones, have been well-known for decades (see, e.g., [9]). The lower bound needed in Theorem 3.10 differs from what is known because it refers to hardness of compressing an explicit string (by ROBPs of size quasipolynomial in the string's length), rather than to hardness of computing an explicit function (by ROBPs of size that is smaller than the function's truth-table).

Technically, at a high level the two results use hardness vs randomness in the straightforward way (i.e., the hardness assumption is that the reconstruction algorithm fails), and the main challenge is obtaining very efficient reconstruction algorithms. For example, in Theorem 3.10 the reconstruction algorithm is a small uniform deterministic ROBP (which tries to compress the $f_{n,i}$'s), rather than a general algorithm. The underlying technical tools are presented in Sections 4.1 and 4.2.2.

3.4.2 Derandomization with minimal memory footprint

The hardness to randomness paradigm allows us to deduce derandomization, but what is the precise cost of derandomization? Recall that classical conditional results [46, 58] transform any randomized linear time algorithm into a deterministic algorithm running in time that is polynomial but that may be very large (i.e., n^c for an unspecified constant $c \gg 1$).

Doron *et al.* [27] posed the question of whether it is possible to simulate all randomized algorithms while incurring minimal time overhead; ideally, we want to incur no time overhead

[9] A variation on Theorem 3.9 was shown in [61, Theorem 7.10], in which the circuits in the lower bound are only of polynomial size (rather than size $2^{\varepsilon \cdot n}$), but they are general circuits (rather than $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuits).

[10] The notion of "compressing" here means outputting a machine M of size polylog(n) that gets input $i \in [n]$, runs in space polylog(n) and time poly(n), and outputs $(f_n)_i$.

[11] The main difference between the two is that for uniform ROBPs it is more natural to specify their uniformity (e.g., the ROBP is logspace-uniform), whereas for streaming algorithms it is more natural to specify their computational complexity (e.g., the algorithm runs in polynomial time). The technical result in [61] refers to a model in which both restrictions simultaneously apply: as an ROBP it is logspace-uniform, and as a streaming algorithm its running time is at most n^c.

at all. Their work and follow-ups [7, 22, 23, 72] showed that, under strong hardness assumption, derandomization with very small time overhead is indeed possible.[12]

What about derandomizing with minimal space overhead? This question was posed in [30], which asked whether we can simulate probabilistic algorithms running in space S by deterministic algorithms running in space $c \cdot S$ for $c \geq 1$ that is as small as possible (ideally $c \approx 1$).

Loosely speaking, their original work deduced this with $c \approx 2$ under two assumptions: very efficient cryptographic PRGs, and the existence of a logspace algorithm printing strings that are hard to probabilistically compress.[13] Subsequently, in [29] they showed that these assumptions can be replaced by a single, non-cryptographic hardness assumption, for uniform deterministic machines.

Theorem 3.11 (Informal, see Theorem 5 [29])**.** *Assume for every C there exists a function f mapping n bits to n^2 bits that is computable in space $(C + 1) \cdot \log(n)$, but no deterministic, low-space algorithm R can achieve the following. On input x, R print a machine M of description size $O(n)$ where M runs in space $C \cdot \log(n)$ and prints $f(x)$. Then, for any $S(n) = \Omega(\log n)$ and constant $\varepsilon > 0$,*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{SPACE}[(2 + \varepsilon) \cdot S].$$

The high-level proof approach for Theorem 3.11 is similar to that underlying Theorems 3.9 and 3.10, but the technical details are considerably more complicated, since this result needs to get a generator that is computable with essentially no space overhead, while still maintaining a deterministic reconstruction. In [29] they showed this by composing a reconstructive generator with the Forbes-Kelley PRG [31], and building a specialized deterministic reconstruction for the latter; see Section 4.2.2 for details.

3.4.3 Isolation from remarkably weak lower bounds

Finally, consider the question of making nondeterministic algorithms *unambiguous*, i.e. having at most a single convincing witness. In the setting of time-bounded algorithms, the well-known result of Valiant and Varizani [82] gives a randomized reduction from **NP** to **prUP** (recall **UP** is the class of problems with a single valid witness for every YES instance, and **prUP** is the corresponding class of promise problems). However, assuming $\mathsf{NP} \not\subset \mathsf{P}/\text{poly}$, there is no *deterministic* algorithm that reduces **NP** to **prUP** by "isolating" witnesses, i.e. by mapping circuits with many satisfying assignments to circuits that reject all but exactly one of these assignments [26].

Perhaps surprisingly, in the setting of space-bounded algorithms there is evidence that we *can* deterministically turn all nondeterministic algorithms into unambiguous ones, given by Wigderson, Gál, Allender, Reinhardt, and Zhou:

Theorem 3.12 ([4, 40, 63, 83])**.** *Suppose that* $\mathsf{SPACE}[O(n)]$ *is hard for circuits of size* $2^{\varepsilon n}$ *on all input lengths, for some* $\varepsilon > 0$*. Then* $\mathsf{NL} = \mathsf{UL}$.

[12] Specifically, these works conditionally deduced worst-case derandomization with a multiplicative time overhead of n (i.e., simulating probabilistic time $T(n)$ in deterministic time $n \cdot T(n)$) and strong average-case derandomization with essentially no time overhead (i.e., simulating probabilistic time $T(n)$ in deterministic time $n^{\varepsilon} \cdot T(n)$ over all efficiently samplable distributions).

[13] The second assumption can be replaced by strong circuit lower bounds (see [30, Theorem 2]. Their work also deduced derandomization with $c \approx 1$ under even stronger assumptions, referring to catalytic computation (see [30, Section 1.4]).

Just as in space-bounded derandomization, there has been progress on resolving this question unconditionally, with the state of the art due to van-Melkebeek and Prakriya [81] establishing that $\mathsf{NL} \subseteq \mathsf{UL}^{3/2}$.

From the perspective of hardness-to-randomness results, we are again in the situation of assuming a large hammer to attack a possibly smaller nail (cf., Theorem 1.2). Can we do better? And again, using deterministic reconstruction, in [29, 61] they showed that the answer is yes:[14]

Theorem 3.13 ([29, 61])**.** *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that $\mathsf{USPACE}[n]$ is hard for $\mathsf{USPACE}[cn]$-uniform circuits of size n^c with oracle access to $\mathsf{USPACE}[\varepsilon cn]$. Then, $\mathsf{NSPACE}[n] \subseteq \mathsf{USPACE}[O_\varepsilon(n)]$.*

Technically, rather than a pair (GEN, REC) with deterministic reconstruction for a distinguisher related to fooling ROBPs, we have a pair (GEN, REC) with deterministic reconstruction for a distinguisher that checks if a certain condition related to the "path-isolation lemma" from [63] holds. In particular, the distinguisher checks if a weight function $W : E \to [n^c]$ on the edges of a graph G has the property that all shortest paths are unique. The works [29, 52, 61] constructed a deterministic reconstruction for this distinguisher; see explanation in Sections 4.1 and 4.2.2.

4 Technical Tools

At a high level, in this section we will present two technical tools, both consisting of a deterministic logspace generator and a deterministic logspace (or polylogspace) reconstruction algorithm. The results presented in Section 3 rely on these tools, or on variations on these tools that will be mentioned in the text below.

Hardness of compressing a string. In Section 4.1 we describe a pair (GEN, REC) of a generator and reconstruction algorithm, which follow the explanation in Section 2. Specifically, both algorithms get access to a string f, and whenever $\mathsf{GEN}^f(D)$ fails, $\mathsf{REC}^f(D)$ compresses f to a small circuit.

Then, in Section 4.2 we expand on a key technical part in this construction (and other similar constructions), which is Distinguish to Predict transformations for restricted classes of distinguishers.

Hardness of computing a function. In Section 4.3 we describe a targeted generator and a corresponding reconstruction algorithm, which follow the explanation in Remark 2.1. Specifically, the generator will be based on a function $f(\cdot)$, and for every input x such that $\mathsf{GEN}_f(x)$ fails, the reconstruction $\mathsf{REC}_f(x)$ computes $f(x)$ using less resources than the naive algorithm for f. The construction of these algorithms will use the (GEN, REC) pair presented before that in Section 4.1.

4.1 Deterministic reconstruction via deterministic D2P

The current state-of-the-art for deterministic logspace (GEN, REC) that rely on hardness of compressing a string is the following.

[14] The precise statement below does not appear in either [29] or [61], but it can be derived by combining [61, Theorem 5.1] with [29, Theorem 4.11].

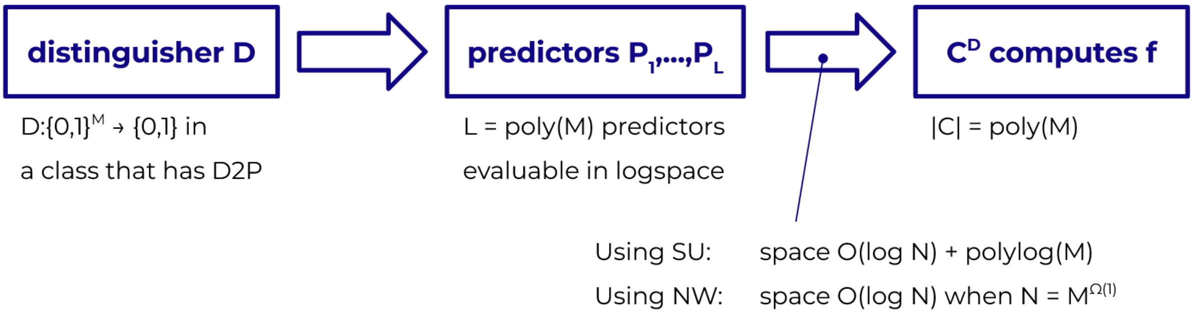


Figure 1: The high-level structure of most known reconstruction procedures, along with the parameters obtained when using the known derandomized reconstruction for the SU generator and for the NW generator.

Theorem 4.1 ([61, Theorem 5.1]; informal)**.** *Let $\mathcal{D}$ be a class of procedures such that there is an efficient distinguish-to-predict transform for $\mathcal{D}$.*[15] *There are two algorithms* (GEN, REC) *that for every $f \in \{0,1\}^N$ and every $M \leq N^{\Omega(1)}$ satisfy the following:*

- GEN(f) *runs in space $O(\log N)$ and outputs $\ell < \log(N)$ lists of M-bit strings.*
- *Fix any $D\colon \{0,1\}^M \to \{0,1\}$ in $\mathcal{D}$ that is a $(1/M)$-distinguisher for each of the ℓ lists that* GEN(f) *outputs. Then,* REC$^D(f)$ *runs in deterministic space $O(\log N) + \mathrm{polylog}(M)$ and prints a circuit C of size $\mathrm{poly}(M)$ such that the truth-table of C^D is f.*

The high-level observation opening the door to Theorem 4.1, from the first paper studying deterministic reconstruction [60], is that in almost all known generator constructions (e.g., in [58, 70, 80]), the reconstruction algorithm can be divided into two steps (see Figure 1):[16]

1. Transforming the distinguisher D into a predictor P (we will define this notion in Section 4.1.2).
2. Using a predictor P, constructing a circuit C such that the truth-table of C^P is f.[17]

The two steps above often work with a list of candidate predictors, in the following sense: The first step transforms D into a list of functions $P_1, ..., P_L$, one of which is a predictor; and the second step also works given access to such a list, in which case it outputs C^{P_i} for some P_i.

In classical constructions, both steps are probabilistic. Generally speaking, the first step follows a classical easy argument of Yao [87], and most of the technical work is devoted to the second step. For example, in the classical construction of [58, 69], the second step consists of hard-wiring partial truth-tables according to a combinatorial design, and then using a decoder of an underlying locally list-decodable error-correcting code (for a standard description, see e.g. [1, Chapter 19.6]).

[15] We will define distinguish-to-predict transforms in Section 4.1.2. For Theorem 4.1, we need is the transform to be computable in deterministic logspace, and for the predictors that it outputs to be evaluable in logspace (see Remark 4.7).

[16] An exception is generators with non-deterministic reconstruction, which have been useful for studying hardness vs randomness for **AM** and for superfast derandomization (see, e.g., [23, 27, 55, 68, 71]).

[17] In Theorem 4.1 we “promised” that REC outputs C^D whose truth-table is f, whereas this step yields C^P whose truth-table is f. In known arguments P is easily evaluable given D, and hence the difference between C^D and C^P is immaterial.

It turns out that for several known generators, we can derandomize the second step in a generic way, which doesn't rely on any properties of D; this takes technical work, but does not have much conceptual innovation. We will therefore start in Section 4.1.1 by describing this more generic second step, and then in Section 4.1.2 describe the first step, which is the main interesting bottleneck.

A key difference from the time-bounded setting: Derandomizing algorithms with two-sided error. Note that in Theorem 4.1 the generator outputs ℓ lists, and (assuming that the reconstruction fails) the guarantee is only that *one of the lists is pseudorandom*. A-priori, we don't know which of the ℓ lists is the pseudorandom one, so we might take the union of the lists to yield a hitting-set generator (and derandomize algorithms with one-sided error). However, using deterministic reconstruction, we can do better.

In the special case of trying to prove that $\mathsf{BPL} = \mathsf{L}$, one trick is to rely on the fact that fooling a BPL machine reduces to "hitting" the set of good seeds for Nisan's PRG; in other words, derandomizing logspace algorithms with two-sided error reduces to a "hitting" problem (see Section 4.2.2 for details).

A more general approach is to observe that the *derandomization algorithm can also run the reconstruction algorithm!* (Indeed, this is because the latter is deterministic.) That is, the derandomization algorithm can compute each of the ℓ output lists of the generator, and then run the reconstruction algorithm REC on each list to see if REC succeeds; whenever REC fails, the list must be pseudorandom. See, e.g., Section 4.3.2.

Known variants of Theorem 4.1. Parameter-wise, Theorem 4.1 is not optimal (see open problems in Section 5.3). However, we know how to do better in certain special cases. Let us mention two examples:

- **The output is long:** When the output length is $M = N^{\Omega(1)}$, we can get REC that runs in space $O(\log N)$ and outputs C that is a TC^0 circuit, rather than a general circuit. Moreover, in this case GEN also outputs a single list rather than ℓ lists. (See [28, Theorem 7.4], following [60].)

- **The distinguisher is an ROBP:** When $\mathcal{D}$ is the class of ROBPs of polynomial width, we can get REC that runs in space $O(\log N)$ and outputs C that is of size $\text{polylog}(M)$, rather than $\text{poly}(M)$. We will describe the proof idea in Section 4.2.2. (This result isn't explicitly stated in prior works, and it follows by combining [61, Theorem 5.1] with [29, Theorem 4.8].)

Another known alternative to Theorem 4.1 has REC that runs in near-linear time $N^{1+\varepsilon}$ and reads f in read-once fashion (i.e., bit-by-bit in order); the downside then is that this REC runs in slightly larger space $\text{polylog}(N)$, and it outputs a small low-space machine that computes $j \mapsto f_j$ in time $\text{poly}(|f|)$, rather than a small circuit (see [61, Theorem 2.4]). We will explain this and describe the proof idea in Section 4.3.2.

Remark 4.2. We do not know of any inherent reason to divide reconstruction arguments into the two steps above. In fact, there is evidence that the first step is an overkill,[18] and this step increases the runtime of REC at least quadratically (see [52, Appendix B]). The latter runtime overhead is an instance of what is often called "the hybrid argument barrier" (see. e.g., [32, 72]),

[18] This is since derandomizing a reconstruction argument in general is equivalent to $\mathsf{prBPP} = \mathsf{prZPP}$, whereas derandomizing the first step is equivalent to $\mathsf{prBPP} = \mathsf{prP}$; see Section 4.1.2.

and indeed the known reconstruction procedures that bypass this barrier do not involve the first step (see [23, 27], following [68]).

4.1.1 The second (generic) step

There are two known classical generators whose "second step" of reconstruction can be generically made deterministic and low-space:

- **The Shaltiel-Umans generator [70]** (with a modification from [15]): In [29, 61] they showed that for any output length $M \leq N^{\Omega(1)}$, the second reconstruction step can be implemented in deterministic space $O(\log N) + \mathrm{polylog}(M)$. This is used to prove of Theorem 4.1.

- **The Nisan-Wigderson generator [58]** (combined with a locally list-decodable code [69]): In [60] they showed that when the output length is $M = N^{\Omega(1)}$,[19] the second reconstruction step can be implemented in deterministic space $O(\log N)$. In [28] they showed that (using a suitable code) we can in addition get the second step to output a TC^0 circuit. This matches the special case of Theorem 4.1 with $M = N^{\Omega(1)}$ that was mentioned after the theorem's statement.

We briefly explain the proof ideas, starting with the simpler case of the NW reconstruction. The derandomization of the second step for this reconstruction replaces uniform random choices by pseudorandom choices output by an averaging sampler (i.e., a seeded randomness extractor; see [35]).

Example 4.3. One step in the reconstruction argument of the NW generator is local list-decoding of an underlying error-correcting code, which is usually chosen to be the Reed-Muller code concatenated with the Hadamard code. The decoding algorithm in [69] for the Reed-Muller code chooses a random degree-3 curve in $\mathbb{F}^m$, where the parameters are chosen such that $|\mathbb{F}|^m = \mathrm{poly}(N)$. Choosing a curve can be done using $O(\log N)$ coins, since we just need to choose three points in $\mathbb{F}^m$ to specify the curve.

However, the basic reconstruction only succeeds with probability $1/\mathrm{poly}(M)$, so this step is independently repeated $\mathrm{poly}(M)$ times, resulting in high overall randomness complexity. In [60], instead of using $\mathrm{poly}(M)$ independent choices, they use a logspace-computable sampler $\mathsf{Samp}\colon \{0,1\}^{O(\log N)} \to \left(\{0,1\}^{O(\log N)}\right)^{\mathrm{poly}(M)}$ with accuracy $1/\mathrm{poly}(M)$; that is, they choose coins $r \in \{0,1\}^{O(\log N)}$ for the sampler, and enumerate over the choices given by $\mathsf{Samp}(r)$ instead of over independent choices.

For the SU reconstruction, considerably more technical work went into derandomizing the second step. The idea above of replacing independent choices by samplers was used, in some cases with a special type of sampler called a "curve sampler" and a specific construction by Guo [38].

But unlike the case of the NW reconstruction, black-box replacement of independent choices by samplers is not enough here, and some parts of the reconstruction procedure needed to be

[19] This parameter setting is the typical one when using the NW generator, since this generator is not optimal when $M = N^{o(1)}$. Specifically, the NW generator has seed length $\ell = O((\log N)^2/\log(M))$, which in our terminology means that $\mathsf{GEN}^f(D)$ runs in time at least 2^ℓ. Generators with seed length $O(\log N)$ were constructed by Shaltiel and Umans [70] and by Umans [80].

refined (see [29, Section 2.1.3]). The reconstruction also does not a-priori run in low space,[20] and this required an additional idea (see [61, Section 2.2.3]). A self-contained proof appears in [61, Section 5].

4.1.2 The first step: Distinguish-to-Predict (D2P) transforms

In the previous section we established that (for some generators) the second step can be executed in deterministic logspace, and this didn't rely on any properties of D. Specifically, the procedures above just need black-box access to a predictor, so all we need now is a way to transform D into a predictor.

> The main technical bottleneck is the first step, i.e. transforming a distinguisher to a predictor. Known derandomizations of this step use the structure of D, and extending them to broader classes of D's is the key towards getting derandomized reconstruction in more settings.

We will define this precisely using a more general notion of a predictor than the standard one. Instead of allowing only "next-bit" predictors, we will also allow "previous-bit" predictors, and more generally:

Definition 4.4 (predictor)**.** For a distribution $\mathbf{w}$ over $\{0,1\}^M$, we say $P : \{0,1\}^m \to \{0,1\}$ is a δ-predictor for $\mathbf{w}$ if there is an interval $I \subseteq [M]$ where $|I| = m$ and a bit $j \in [M] \setminus I$ where

$$\mathop{\mathbb{E}}_{w \sim \mathbf{w}}[P(w_I) = w_j] \geq \frac{1}{2} + \delta.$$

We say P is a next-bit-predictor if $I = \{1, \ldots, m-1\}$, and a previous-bit-predictor if $I = \{m+1, \ldots, M\}$.

Definition 4.5 (D2P)**.** An ε-distinguish to δ-predict transform for a class $\mathcal{C}$ of circuits is an algorithm that takes as input an M-bit $C \in \mathcal{C}$ and outputs a set of procedures $P_1, ..., P_L$, where each P_j is a function $\{0,1\}^i \to \{0,1\}$ for some $i \leq M$, such that the following holds. For every distribution $\mathbf{w}$ such that C is an ε-distinguisher for $\mathbf{w}$, there is $j \in [L]$ such that P_j is a δ-predictor for $\mathbf{w}$.

The requirement in Definition 4.5 that a single set $P_1, ..., P_L$ works for *all* distributions $\mathbf{w}$ is not obvious. A relaxed requirement, wherein the algorithm may take $\mathbf{w}$ as input, also makes sense (see [52, Appendix C.2]). We focus on the stronger notion, because most of the D2Ps from recent years satisfy it.

Remark 4.6. Note that a useful property of a predictor P is that, given access to $\mathbf{w}$, we can efficiently *verify* how well it predicts the relevant bit. This stands in contrast to distinguishers, since there is no obvious way to check if D is a distinguisher for $\mathbf{w}$ (since a priori we do not know the value $\mathbb{E}_{r\in\{0,1\}^M}[D(r)]$).

Remark 4.7. When using a D2P transform to construct a logspace reconstruction algorithm, we also need that the predictors given by the D2P will be evaluable in logspace. That is, we need a logspace algorithm that gets $i \in [L]$ and input w and oracle access to D, and outputs $P_i(w)$. This is important because the second step of the reconstruction (from Section 4.1) makes queries to the candidate predictors.

[20] In [29] the reconstruction is presented as a probabilistic algorithm using $O(\log N)$ coins and space $O(\log N)$. The best known way to transform this algorithm into a deterministic algorithm, from [61], pays an an extra factor of polylog(M).

D2P for general circuits. The textbook lemma by Yao [87] yields a *probabilistic* algorithm that gets a general circuit D, and for every fixed $\mathbf{w}$, whp outputs a D2P transform that works for $\mathbf{w}$.

Proposition 4.8 ([87]). *Suppose $D\colon \{0,1\}^M \to \{0,1\}$ is a ε-distinguisher for $\mathbf{w}$. Then, with probability at least $1/\operatorname{poly}(M)$ over choice of $i \in [M]$ and $z \in \{0,1\}^{M-i}$ and $b \in \{0,1\}$, the algorithm $P : \{0,1\}^i \to \{0,1\}$ defined as $P(w) = D(w \circ z) \oplus b$ is an (ε/M)-next-bit-predictor for $\mathbf{w}$.*[21]

It is not a-priori clear that for all D's there even *exists*, non-explicitly, a single small set of predictors that works for every $\mathbf{w}$ (for which D is a distinguisher). This is because there can be $2^{\Omega(2^M)}$ distributions for which D is a distinguisher,[22] so naive probabilistic arguments yield exponentially many predictors.

Nevertheless, the claim is true. The proof from [52] does not use a standard probabilistic-method argument, and instead uses a specific (non-explicit) construction of predictors.

Theorem 4.9 ([52, Theorem B.1]). *For every $D : \{0,1\}^M \to \{0,1\}$ there are $L = O(M^2)$ candidate predictors $P_1, ..., P_L$ such that the following holds. For every distribution $\mathbf{w}$ for which D is a $(1/3)$-distinguisher there is $j \in [L]$ such that P_j^D is an $1/O(M)$-predictor for $\mathbf{w}$.*

In a gist, the predictors in Theorem 4.9 use a (non-explicit) CAPP algorithm to estimate the expectation of certain auxiliary circuits. This is similar to an argument of Goldreich [34, Appendix A].

In particular, assuming that there is an efficient CAPP algorithm, we obtain a deterministic D2P transform for general circuits. In fact, deterministic D2P turns out to be *equivalent* to general derandomization!

Theorem 4.10 ([52, Theorem 1.5]). *There exists a deterministic polynomial-time $(1/3)$-distinguish to $(1/\operatorname{poly}(M))$-predict transform for general circuits* if and only if $\mathbf{prBPP} = \mathbf{prP}$.

This result is not encouraging, since it indicates that constructing D2P transforms for general circuits may be too challenging at the moment. There is even worse news: For any typical circuit class C, a non-trivial D2P transform for C implies that **NEXP** is hard for C. Specifically, an ε-distinguish to $(2^{-o(M)})$-predict transform for C that runs in time $2^M/M^{\omega(1)}$ and outputs $2^{o(M)}$ predictors (where $\varepsilon > 0$ is a small universal constant) implies non-trivial derandomization for C (see [52, Appendix B.2]). Using Williams' algorithmic method [84], such derandomization yields a problem in **NEXP** that is hard for C-circuits.

4.2 Deterministic D2P for restricted distinguisher classes

To make progress, we turn our attention to D2P for restricted classes of distinguishers, for which lower bounds in **NEXP** are either known or seem feasible to prove. We also consider D2P transforms for specific useful function classes D that don't correspond to classical circuit classes.

[21] By repeating the algorithm in Proposition 4.8 for $L = \operatorname{poly}(M)$ times, we obtain a random set of $P_1, ..., P_L$ such that for any $\mathbf{w}$, with high probability over choice of P_j's one of them is a predictor for $\mathbf{w}$. The same argument establishes that with probability $1/\operatorname{poly}(M)$ over i, z it holds that $P(w) = D(z \circ w) \oplus b$ is an (ε/M)-previous-bit-predictor for $\mathbf{w}$.

[22] To see this, consider an unbiased D, and a collection of uniform distributions over subsets of $\{0,1\}^M$ on which the expected value of D differs from $1/2$.

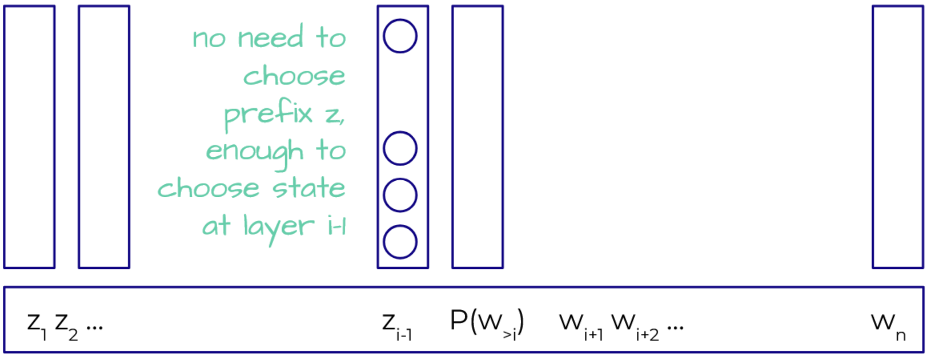

no need to choose prefix z, enough to choose state at layer i-1
z1 z2 ...
zi-1
P(w>i)
wi+1 wi+2 ...
wn

Figure 2: Visual depiction of the proof of Theorem 4.12.

4.2.1 Deterministic D2P for ROBPs

In the special case of ROBPs, there is a logspace deterministic D2P, and moreover, the latest known proof of this is simple! Let's recall the definition of ROBPs:

Definition 4.11 (ROBP)**.** An ROBP $B\colon \{0,1\}^M \to \{0,1\}$ consists of $M+1$ layers, where each layer $i \in [M]$ is labeled with a distinct input variable x_{j_i}. Each node in layer i has two outgoing edges to layer $i+1$, labeled with 0 and 1. The first layer has a "start" node s and each node in the last layer is labeled with an output value. The ROBP computes a function by starting from s and iterating through the layers $i = 1, \dots, M$, reading the value v of the corresponding input bit x_{j_i}, and following the edge labeled with v to the next layer. The maximal number of nodes in any layer is the main complexity parameter, called the width.

Theorem 4.12 ([28, 60], following [11, 37, 56])**.** *There is a deterministic logspace* $(1/M)$*-distinguish to* $(1/\operatorname{poly}(M))$*-predict transform for* ROBP*s of width* W*. The algorithm outputs* $\operatorname{poly}(M,W)$ *predictors, each of which is also an* ROBP.

Proof. Here we finally use a special property of the distinguisher class, and a remarkably simple one. Let $D\colon \{0,1\}^M \to \{0,1\}$ be the given ROBP, and consider some (unknown) distribution $\mathbf{w}$ over $\{0,1\}^M$.

By Proposition 4.8 and Footnote 21, for some $i \in [M]$ and $z \in \{0,1\}^{i-1}$ and $b \in \{0,1\}$, the function $P_{i,z,b}(w) = D(z \circ w) \oplus b$ is a previous-bit-predictor for $\mathbf{w}$. We can enumerate over $i \in [M]$ (as we are allowed to output a list of predictors), but enumerating over all $z \in \{0,1\}^{i-1}$ is infeasible.

The key observation is that for any z, the residual function $D_z(w) = D(z \circ w)$ is a sub-ROBP of D. Specifically, after fixing the first $i-1$ bits to z, the residual function $D_z(w)$ is the ROBP obtained by starting from a node $v = v_z$ in layer $i-1$ (i.e., the node v_z we reach by walking on D according to z) and then processing the input $w \in \{0,1\}^{M-(i-1)}$. Hence, instead of enumerating over all z's, it's enough to enumerate over all v's in layer $i-1$, of which there are only at most W. See Figure 2 for a visual depiction.

The algorithm outputs the set of $2M \cdot W$ predictors obtained by enumerating all choices of $i \in [M]$ and $v \in [W]$ and $b \in \{0,1\}$, and outputting $P_{i,v,b}(w) = D_v(w) \oplus b$. □

Remark 4.13. The only property of ROBPs that this result exploits is that fixing a prefix of the input to a given ROBP D yields one of at most W sub-ROBPs of D.

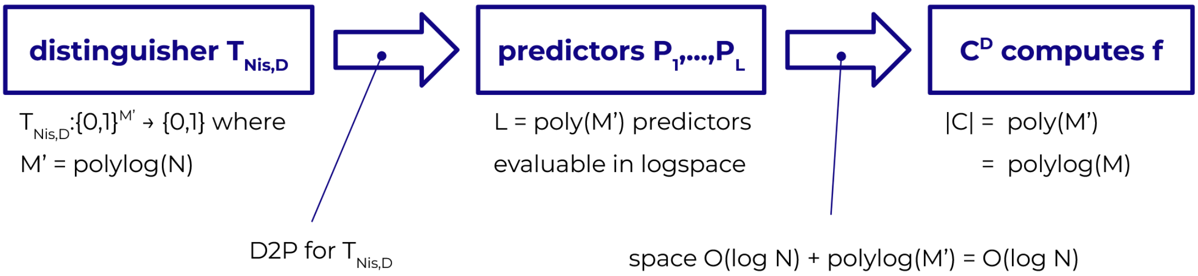

distinguisher $T_{\mathsf{Nis},D}$
$T_{\mathsf{Nis},D}$:$\{0,1\}^{M'} \to \{0,1\}$ where
M' = polylog(N)
D2P for $T_{\mathsf{Nis},D}$
predictors $P_1,\dots,P_L$
L = poly(M') predictors
evaluable in logspace
space O(log N) + polylog(M') = O(log N)
C^D computes f
|C| = poly(M')
= polylog(M)

Figure 3: The high-level structure of the reconstruction procedure when using the generator $\mathsf{Nis} \circ \mathsf{GEN}$.

Having a D2P for ROBPs is already useful, since $\mathbf{BPL} = \mathbf{L}$ reduces to estimating the expected value of a given ROBP (in logspace). Using Theorem 4.12 and the derandomized logspace algorithms mentioned in Section 4.1.1, we can get a simpler version of Theorem 4.1 in which C is of size $\mathrm{poly}(M, \log N)$ rather than $\mathrm{polylog}(MN)$. This was used in the proofs of Theorems 3.1 and 3.3.

4.2.2 Deterministic D2P beyond ROBPs

After the statement of Theorem 4.1, we mentioned that in the special case of ROBPs we can get an improved reconstruction algorithm, which runs in space $O(\log N)$ and outputs a circuit C of size $\mathrm{polylog}(M)$ (rather than $\mathrm{poly}(M)$). Let us explain the idea for doing so, which will naturally lead us to consider D2P transforms for distinguishers beyond ROBPs.

Composing the generator with Nisan's PRG. Recall that we want a generator that outputs M bits and fools ROBPs. We will instantiate the generator GEN with output length $M' = \mathrm{polylog}(M)$, and then compose it with an *unconditional PRG* that stretches M' bits to M bits and fools ROBPs. Specifically, for the "outer" PRG we will use the classical generator of Nisan [56]. That is, the final generator is $\mathsf{Nis} \circ \mathsf{GEN}$.

The point of doing this is that when this composed generator fails (which means that GEN failed, because Nis works unconditionally), the reconstruction REC outputs C of size $\mathrm{poly}(M') = \mathrm{polylog}(M)$, and runs in space $O(\log N) + \mathrm{polylog}(M') = O(\log N)$. The disadvantage, however, is that the distinguisher for the inner generator now isn't the ROBP D, but rather *Nisan's PRG composed with D*; that is, we need GEN to fool the distinguisher $D'(w) = \mathsf{Nis}(D(w))$. This is not an ROBP anymore, and hence the crucial bottleneck in the construction of $(\mathsf{GEN}, \mathsf{REC})$ – the D2P algorithm from Theorem 4.12 – does not work anymore.

Fortunately, this can be handled! The first observation is that, inspecting the construction of Nis, for derandomization it suffices to find a good key for a pairwise-independent hash function, which is of description length $M' = \mathrm{polylog}(N)$. Moreover, the function $T_{\mathsf{Nis},D}$ that gets input $k \in \{0,1\}^{M'}$ and decides whether or not k is a good key for D can be implemented in logspace (this is essentially the argument in [57]; for an explanation, see [29, Section 4.1]). Thus, the distinguisher for GEN is now $T_{\mathsf{Nis},D}$.

Theorem 4.14 ([29]). *There is a deterministic logspace* $(1/2)$*-distinguish to* $(1/\mathrm{poly}(M'))$*-predict transform for the class of* $T_{\mathsf{Nis},D}$*'s obtained from* ROBP*s* $D\colon \{0,1\}^M \to \{0,1\}$ *of width* $\mathrm{poly}(M)$*. The algorithm outputs* $\mathrm{poly}(M')$ *predictors, each of which is computable in logspace.*

Proof idea. Recall that Theorem 4.10 reduces D2P to $\mathbf{prBPP} = \mathbf{prP}$. Its proof shows something more specific: For any fixed $D\colon \{0,1\}^M \to \{0,1\}$, the proof gives a set of efficient predictors

$P_1, ..., P_L$ that work when given oracle access to a function that gets $z \in \{0,1\}^{i<M}$ and estimates $\mathbb{E}_{z \in \{0,1\}^{M-i}}[D(y \circ z)]$. In other words, to predict any distribution $\mathbf{w}$ (that D distinguishes from uniform), it suffices to estimate the acceptance probability of D when fixing a prefix of its input bits to a given y. For general circuits, this "prefix-CAPP" problem is equivalent to CAPP and hence complete for **prBPP**; however, in the current special we can use a trick to solve it.

Let us go back to $T_{\mathsf{Nis},D}$, which takes $k \in \{0,1\}^{O(\log N)^2}$ and decides if it is good for D. In Nisan's proof, k is partitioned into $\ell = \log(N)$ blocks of size $O(\log N)$, and k is good if and only if each block is good (for some notion of "good block" that we intentionally gloss over here).[23] Moreover, given a prefix of $i \leq \ell$ blocks in k, we can test in logspace whether all the i blocks are good.

Now let us look at the task we need to solve in order to construct a predictor. We are given a prefix y for a potential key k, and for simplicity let us assume that this is a prefix of blocks (i.e., the prefix not does end mid-block). We need to decide $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)]$. The key observation is that there are only two cases:

- If there is a block in the given prefix that is not good, then $k = y \circ z$ will not be good, no matter the suffix z. Hence $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)] = 0$.
- If all blocks are good, then the proof in [57] shows that a random suffix z yields a good $k = y \circ z$, with very high probability. Hence, $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)] \approx 1$.

Thus, to estimate $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)] = 0$ it suffices to check if all blocks in the given y are good, which we can indeed do in deterministic logspace. The D2P outputs the set of predictors from the proof of Theorem 4.10, with the required estimation task performed as above. □

To recap, the proof of Theorem 4.14 relied on a reduction from the proof of Theorem 4.10, and the main property used in the proof of Theorem 4.14 was a "polarization" effect: for any prefix y, either the acceptance probability over a random z is zero or it is very high. Moreover, we can efficiently distinguish between the two cases. Many known D2P transforms rely on similar "polarization" effects.

Fooling any-order branching programs by composing with the Forbes-Kelley PRG. For derandomization with minimal memory footprint (as in Theorem 3.11), we'd like both GEN and REC to incur almost no space overhead. In [30] they showed how to get GEN that adds essentially no space overhead, but this relies on one modification: instead of considering a distinguisher D that is an ROBP, we now need to consider D that is an Any Order Branching-Program (AOBP). For a definition see, e.g., [16, 31], though knowing the precise definition is not crucial when reading the current text. Unfortunately, the Nisan generator that we constructed a D2P transform for provably does *not* fool this model [79]!

To get a hyper-efficient GEN along with REC that deterministically compresses to size polylog(M), the idea in [28] was to replace Nisan's PRG with an unconditional PRG for AOBPs that can be evaluated in extremely low space, and such that there is a D2P for the composition of this PRG with an AOBP. It turns out that a modification of the Forbes-Kelley PRG [31] satisfies all these properties.

Theorem 4.15 (informal; see [29, Section 4.3]). *For any $\varepsilon > 0$, there is a deterministic algorithm that gets as input an* AOBP *$D\colon \{0,1\}^M \to \{0,1\}$ of size M and a distribution* $\mathbf{w}$ *over $\{0,1\}^{M^\varepsilon}$ for*

[23] To be more accurate, in the original proof this is not an "if and only if", since some k's happen to be good even if not all blocks are good. However, for our purposes we modify the definition of "a good k" to mean that all blocks are good.

which $D \circ FK$ *is a* $(1/10)$*-distinguisher,*[24] *runs in time* $\mathrm{poly}(M, |\mathbf{w}|)$ *and space* $O(M^{\varepsilon} + \log |\mathbf{w}|)$*, and outputs a* $(1/M^{O(\varepsilon)})$*-predictor* P *for* $\mathbf{w}$*. The predictor can be evaluated in space* $(1+O(\varepsilon)) \cdot \log(M)$ *with access to* D*.*

Observe that Theorem 4.15 achieves the more relaxed notion of D2P that was mentioned after Definition 4.5. At a high-level, the proof of Theorem 4.15 uses a "polarization" idea similar to the proof of Theorem 4.14, but significantly more technical work is needed to carry it through, along with modifications to the original FK generator. See [29, Sections 2.3.1 and 4.3] for a proof description. The resulting (GEN, REC) pair is implicit in the proof presented in [29, Section 6.4].

Path isolation by composing with the van Melkebeek and Prakriya PRG. There is also a deterministic D2P for a completely different type of distinguisher, which doesn't arise from ROBPs/AOBPs.

Recall that in Section 3.4.3 we mentioned a result from [29, 52, 61] deducing $\mathbf{NSPACE}[n] \subseteq \mathbf{USPACE}[O(n)]$ from lower bounds for $\mathbf{USPACE}[O(n)]$-uniform circuits. As mentioned there, that result is based on a generator that produces weight assignments for the edges a given graph, where the distinguisher $D(W)$ checks whether or not the weight assignment W induces unique shortest paths in the graph. Getting deterministic $\mathbf{USPACE}[O(n)]$-reconstruction for this generator calls for designing a deterministic D2P for D.

Reinhardt and Allender [63] showed that D can be implemented in unambiguous logspace (i.e., in $\mathbf{UL}$, which in the parameter above gives a $\mathbf{USPACE}[O(n)]$ algorithm), but their algorithm does not seem at all like an ROBP/AOBP. Fortunately, in [52] they still showed a D2P for it. Their idea was to mimic the proof of Theorem 4.14: They reduced D2P to solving a "prefix-CAPP" problem for D, observed that to solve the latter it suffices to check whether the given prefix violates a condition related to D, and relied on the algorithm of [63] for D to decide the latter. Details appear in [52, Section 2.2].

4.3 New reconstructive targeted generators for the logspace setting

We now design a generator and reconstruction based on hardness of *computing a function* $f(x)$, rather than of compressing a fixed string f. That is, we fix a uniformly computable function $x \mapsto f(x)$, and both the generator and the reconstruction will get input x. Intuitively, the generator tries uses the computation of $f(x)$ as a source of hardness (which it can convert to pseudorandomness).

The technical tool that we will rely on is a targeted PRG, as introduced by Goldreich [34, 36]. This is a generator that gets input x and tries to fool efficient uniform algorithms that also receive the same input x (i.e., rather than trying to fool all small non-uniform circuits, as in classical PRGs). The reconstruction algorithm also gets x, and if the generator fails, then the reconstruction manages to compute $x \mapsto f(x)$ "too efficiently". We stress that the analysis is now performed instance-wise, for every fixed input x. That is, if $x \mapsto f(x)$ is hard to compute on x specifically, then the generator with x produces a distribution that is pseudorandom for uniform algorithms that get access to this specific x.

We also note that the reconstruction in this setting does not get oracle access to a hard string anymore (let alone oracle access to f). The reconstruction simply gets x and tries to compute $f(x)$.

[24] The version of the Forbes-Kelley PRG from [29] stretches M^{ε} bits to M bits.

A concrete targeted generator. The generator below, from [61] (following [29]), is based on the supposed hardness of composing low-space algorithms "for free", i.e. without a significant overhead either in the running time or in the space complexity (see Section 3.3.2 for an exposition of this topic). Specifically, fix an arbitrary f_0 computable in logspace and large polynomial time T. The result below gives a pair of algorithms that, on every input, either compute the k-fold composition of f_0 in *near-linear* time $T^{1+\varepsilon}$ and polylogarithmic space, or derandomize **BPL** in space $O(k \cdot \log(n))$.[25]

Theorem 4.16 (win-win pair of algorithms for derandomization and composition; [61], following [29])**.** *Let $L \in$ **BPL**, let $\varepsilon > 0$, and let T be a sufficiently large polynomial. Fix a length-preserving f_0 computable in logspace and linear time, and $k = k(n)$. Then, there are two algorithms A, B such that for every $x \in \{0,1\}^n$, at least one of the following holds:*

- *$A(\bar{x}) = f_0^{(k)}(\bar{x})$ in time $T^{1+\varepsilon}$ and space* $\mathrm{polylog}(n)$*, where $\bar{x} = x0^{T-n}$.*
- *$B(x) = L(x)$ in space $O(k \cdot \log(n))$.*

In terms of hardness vs randomness, Theorem 4.16 can be parsed as follows: if computing k-wise composition is infeasible in near-linear time and low space, then we can derandomize **BPL**. We will explain below how the generator's construction can yield not only Theorem 3.8 but also Theorem 3.5.

For notational convenience, in the rest of the section we write $f(x) = f_0(x)$, and note that the hard function on which the generator is based is $x \mapsto f^{(k)}(x)$.

4.3.1 A bootstrapping system for low-space algorithms

The main technical challenge is that the reconstruction algorithm now gets input x and does not have query access to $f^{(k)}(x)$. In particular, it is not a-priori clear how to utilize classical generators, whose reconstruction algorithms need query access to $f^{(k)}(x)$. The construction underlying Theorem 4.16, which handles this challenge, is an adaptation of the generator by Chen and Tell [22] (using ideas from [47]).

The generator is given input $x \in \{0,1\}^n$, and for simplicity let us abuse the notation and use x to also denote the padded T-bit version $x0^{T-n}$. Consider the sequence of strings $f^{(1)}(x) = f(x), f^{(2)}(x) = f(f(x)), ...$ and so on until $f^{(k)}(x)$. Note that this sequence of strings is "downward self-reducible", in the sense that $f^{(i)}(x)$ is computable in logspace from $f^{(i-1)}(x)$. See Figure 4.

The targeted generator applies the generator GEN from Theorem 4.1 to each of the strings $f^{(i)}(x)$, and obtains a sequence of lists $L^{(i)} = \mathsf{GEN}(f^{(i)}(x))$.[26] The hope is that at least one of those lists will be pseudorandom for the **BPL** machine with the same input x. Indeed, assume that this is not the case. Then, the deterministic reconstruction algorithm REC from Theorem 4.1 runs in logspace and builds a small circuit C_1 (of size $\mathrm{polylog}(M) \le \mathrm{polylog}(T)$) whose truth-table is $f^{(1)}(x)$, as long as REC can make queries to the bits of $f^{(1)}(x)$. The key point is that we can compute answers to these queries in logspace, due to downward self-reducibility and since we have x; that is, when REC queries $f^{(1)}(x)$ at location i, we compute the answer by running the logspace algorithm that computes $f^{(1)}(x) = f(x)$ from x. Continuing this further, at each iteration i we have a circuit $C^{(i)}$ of size $\mathrm{polylog}(T)$ whose truth-table is $f^{(i)}(x)$, and REC builds a small

[25] To capture a more general setting, the result models f as a linear-time function mapping T bits to T bits, and pads the input x to be of length T. Other models work just as well, e.g. considering an initial function mapping n bits to T bits and then $k-1$ length-preserving functions.

[26] To get the statement of Theorem 4.16 specifically, we assume that T is a large enough polynomial so that the output length $M = T^{\Omega(1)}$ suffices to derandomize the **BPL** machine for L.

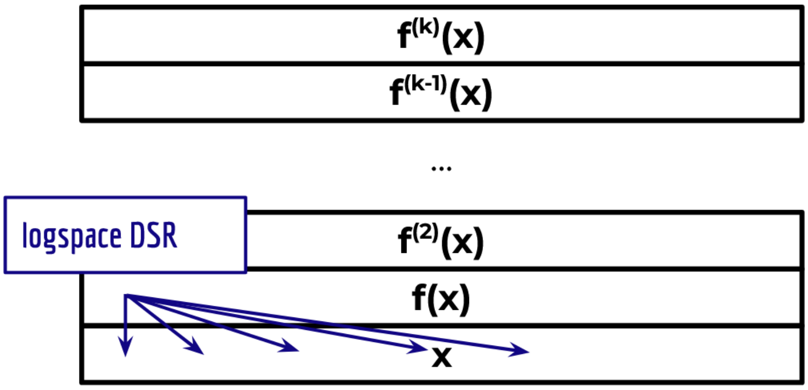


Figure 4: A boostrapping system based on repeated composition of a logspace-computable function.

circuit $C^{(i+1)}$ whose truth-table is $f^{(i+1)}(x)$, using $C^{(i)}$ and downward self-reducibility to answer queries to $f^{(i+1)}(x)$. (And after the iteration $C^{(i)}$ is discarded.) Note that we use the assumption that the **BPL** machine with x is a distinguisher for each list $L^{(i)}$ (since we want REC to succeed on each $f^{(i)}(x)$).

After k iterations the reconstruction algorithm obtained a small circuit $C^{(k)}$ whose truth-table contains the output $f^{(k)}(x)$, so it can evaluate this circuit to compute $f^{(k)}(x)$. This reconstruction runs in time poly(T), rather than $T^{1+\varepsilon}$ as promised in Theorem 4.16; in Section 4.3.2 we will explain how to improve the running time. But the space complexity of the reconstruction is polylogarithmic as we wanted, so as long as $f^{(k)}(x)$ is hard to compute in time poly(T) and space polylog(T), one of the output lists of the targeted generator fools the **BPL** machine with x.

Remark 4.17. The targeted generator above outputs k lists $L^{(1)}, ..., L^{(k)}$, hoping that at least one of them will be pseudorandom. In fact, inspecting things more closely, each $L^{(i)}$ is, in itself, a sequence of $\ell < \log(T)$ lists (because $L^{(i)} = \mathsf{GEN}(f^{(i)}(x))$, and GEN outputs ℓ lists), and if the reconstruction fails, then only one list inside one $L^{(i)}$ is guaranteed to be pseudorandom. In other words, the construction above is of a targeted "somewhere-PRG", rather than a targeted PRG. Fortunately, as explained in the beginning of Section 4.1, in the logspace setting we can often still use such an object to derandomize algorithms with two-sided error.

A win-win pair of algorithms for s-t connectivity. The construction above also suffices to prove Theorem 3.5, using one additional observation. Specifically, in this case the function f will be matrix squaring, and the generator uses $k = O(\log n)$ compositions to compute the transitive closure of the graph. The proof described above works as-is, and the only problem is that the generator's complexity is now $O(k\cdot\log(n)) = O(\log^2(n))$ (which is not useful for derandomization of **BPL**, as derandomization in space $O(\log^{3/2}(n))$ is already known unconditionally [74]).

The observation in [29] is that in this special case, we can compute each entry of $f^{(i)}(x)$ (for any $i \in [k]$) in **NL**, since computing such an entry reduces to a connectivity question in a graph. Hence, the generator is computable in non-deterministic logspace rather than only deterministic space $O(k\cdot\log(n))$, and the win-win pair of algorithms either solves s-t connectivity or derandomizes **BPL** in **NL**.

4.3.2 Faster reconstruction: The Tree-PRG

The construction described in Section 4.3.1 is essentially from [29], and the missing piece in the proof of Theorem 4.16 is to get reconstruction in time $T^{1+\varepsilon}$ rather than $\mathrm{poly}(T)$. A targeted generator with such reconstruction was shown in [61], using an additional construction on top of the one from Section 4.3.1.

Where is the problem, and what do we need? To explain the idea, observe that the $\mathrm{poly}(T)$ time overhead in reconstruction comes from two sources, both of them occurring when running the reconstruction algorithm REC from Theorem 4.1 (i.e., that compresses $f^{(i+1)}(x)$ at each layer $i+1$):

1. The running time of REC is a large polynomial $\mathrm{poly}(T)$.

2. Whenever REC tries to read a bit of its input $f^{(i+1)}(x)$, we run the downward self-reducibility algorithm DSR (answering its queries using $C^{(i)}$) and discard all but the relevant output bit. That is, we compute $\mathsf{REC}(\mathsf{DSR}(C^{(i)}))$ using emulative composition, incurring a quadratic time overhead (at least).

To handle the first problem, we need reconstruction that runs in near-linear time $T^{1+\varepsilon}$ instead of time $\mathrm{poly}(T)$. To handle the second problem, the observation in [61] is that if REC would have been read-once (i.e., if REC would read its input $f^{(i+1)}(x)$ bit-by-bit, in order, making only a single pass overall), then this problem would disappear. This is because we could run REC, and whenever it needs the next bit of $f^{(i+1)}(x) = \mathsf{DSR}(C^{(i)})$, we would run DSR from its current state until it produces the next bit, then "freeze" the execution of DSR and store its state until REC needs the subsequent bit.

A $(\mathsf{GEN}, \mathsf{REC})$ with reconstruction satisfying both conditions was shown in [61]. Specifically, they showed the following incomparable alternative to Theorem 4.1, which was mentioned after its statement:

Theorem 4.18 ([61, Theorem 2.4])**.** *There are algorithms $(\overline{\mathsf{GEN}}, \overline{\mathsf{REC}})$ such that for every $\varepsilon > 0$ and every $f \in \{0,1\}^N$ and $M \le N^{\Omega_\varepsilon(1)}$ satisfy the following:*

1. *$\overline{\mathsf{GEN}}(f)$ runs in space $O(\log N)$ and outputs $\ell < \log(N)$ lists of M-bit strings.*

2. *Fix any ROBP $D\colon \{0,1\}^M \to \{0,1\}$ that is a $(1/M)$-distinguisher for each of the ℓ lists that $\overline{\mathsf{GEN}}(f)$ outputs. Then, $\overline{\mathsf{REC}}^D(f)$ runs in time $N^{1+\varepsilon}$ and space $\mathrm{polylog}(N)$, reads f in read-once fashion, and prints a $\mathrm{polylog}(N)$-size oracle machine A such that A^D describes f. (Specifically, given $j \in [N]$, the machine $A^D(j)$ runs in space $\mathrm{polylog}(N)$ and time $\mathrm{poly}(N)$ and outputs f_j.)*

We will be using Theorem 4.18 with $N = T = |f^{(i)}(x)|$, and stated it with the notation N to facilitate a comparison with Theorem 4.1. Indeed, the two improvements over the latter are that $\overline{\mathsf{REC}}$ is faster and read-once, as we needed. The price we pay is that $\overline{\mathsf{REC}}$ uses space $\mathrm{polylog}(N)$, rather than $O(\log N)$; and that the compressed representation of f is not a small circuit, but rather a small oracle machine whose running time is larger than $|f|$.

Read-once reconstruction: A generic transformation. In [61] they showed a generic transformation of reconstructive generators into reconstructive generators whose reconstruction is read-once. Specifically, starting from $(\mathsf{GEN}, \mathsf{REC})$, consider the generator whose output lists consist of

$$\mathsf{GEN}(f_{\leq 1}),\ \ \mathsf{GEN}(f_{\leq 2}),\ \ \mathsf{GEN}(f_{\leq 3}),\ \ ..., \mathsf{GEN}(f_{\leq i}),\ \ ..., \mathsf{GEN}(f),$$

where $f_{\leq i}$ is the i-bit prefix of f (padded to length N). In words, the new generator applies GEN to each prefix of f, and its output collection of lists is the union of the lists output by GEN for all prefixes.

How does this yield a read-once reconstruction? The new reconstruction algorithm works iteratively. For $i = 1, ..., N$, it stores a small circuit whose truth-table is $f_{\leq i}$ (the base case $i = 1$ is trivial). Then it reads the next bit f_{i+1}, at which point it has all the information needed to answer queries to $f_{\leq i+1}$. It thus runs $\mathsf{REC}(f_{\leq i+1})$ to obtain a small circuit for $f_{\leq i+1}$, and continues to the next iteration. See [61, Section 6.1].

Remark 4.19. Since the new reconstruction needs to evaluate the stored circuit at each iteration, it now uses space $\mathrm{polylog}(M) \leq \mathrm{polylog}(N)$ rather than only $O(\log N)$. This seemingly matches the parameters in Theorem 4.18, but since later on we will construct a generator that needs to compute the output of this read-once reconstruction (and we want the generator to run in space $O(\log N)$), the current space complexity is not good enough. For simplicity, let us pretend that so far the reconstruction uses space $O(\log N)$.[27]

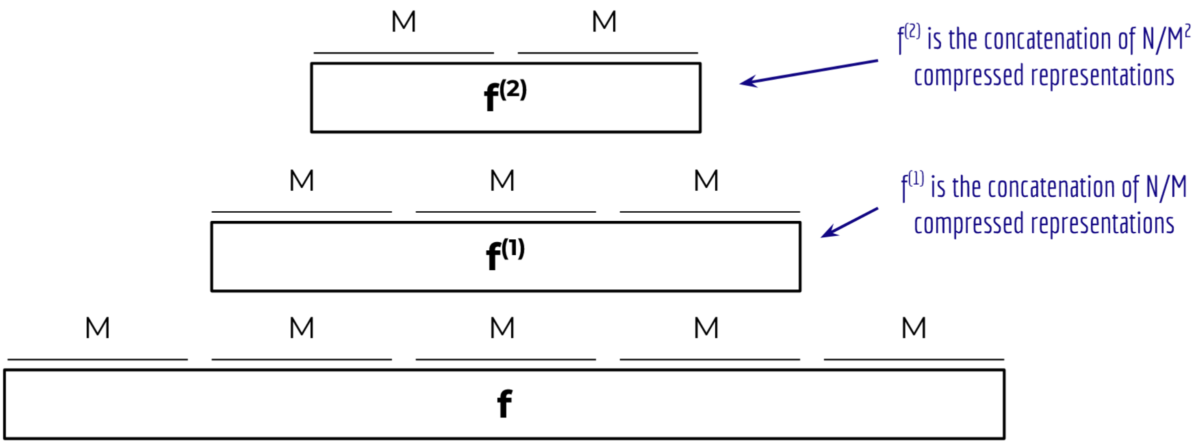
Near-linear time reconstruction: Another generic transformation. Let us assume from now on that $(\mathsf{GEN}, \mathsf{REC})$ are such that REC is read-once, but runs in large polynomial time. Recall that these $(\mathsf{GEN}, \mathsf{REC})$ originate from the Shaltiel-Umans PRG [71], so an obvious attempt would be to directly optimize the latter's reconstruction. However, this strikes us as challenging to solve directly, since this reconstruction algorithm is involved and repeatedly composes "heavy" pseudorandom primitives.

Instead, to handle this large runtime (i.e., the problem in Item 1 above), the idea in [61] is to never run GEN or REC on strings of length N – only on shorter strings, of length, say, M. In this case the runtime overhead of REC is $\mathrm{poly}(M) \leq N^{1+\varepsilon}$, where the inequality holds if N is a large enough polynomial in M.

The construction of $\overline{\mathsf{GEN}}$ is visually depicted in Figure 5. In words, the generator $\overline{\mathsf{GEN}}$ splits f into chunks of length M, and applies GEN to each chunk to obtain a sequence of output lists. It then *runs the original reconstruction* REC *on each chunk* $j \in [N/M]$, to obtain a circuit $C^{(j)}$ of size $\mathrm{polylog}(M)$ whose truth-table is, supposedly, the j^{th} chunk (indeed, REC may fail to output a valid circuit, but we can treat its output as a string of the relevant length nonetheless). Note that this crucially uses the fact that REC is a deterministic logspace algorithm (because we want $\overline{\mathsf{GEN}}$ to also be a deterministic low-space algorithm). Concatenating the outputs of REC on all N/M chunks, we obtain a new string $f^{(1)} \in [(N/M) \cdot \mathrm{polylog}(M)]$. Now $\overline{\mathsf{GEN}}$ recurses, splitting $f^{(1)}$ into chunks of length M, applying GEN to each chunk to obtain a sequence of output lists, and using REC to compress $f^{(1)}$ to $f^{(2)}$ of length $\tilde{O}(N/M^2)$, and so on.

After constantly many iterations (so that the final step is applied to one remaining M-bit chunk), the generator $\overline{\mathsf{GEN}}$ accumulated output lists of GEN from chunks in each layer, and it outputs the set of all these lists. Observe that the new generator can be computed in space

[27] In [61] they handle this problem by showing two reconstruction algorithms, one that is read-once and runs in space $\mathrm{polylog}(N)$ and time $\mathrm{poly}(N)$, and another that is not read-once and runs in space $O(\log N)$, such that both algorithms produce the same output. The generator will use the latter.



$f^{(2)}$ is the concatenation of N/M^2 compressed representations

$f^{(1)}$ is the concatenation of N/M compressed representations

Figure 5: Visual depiction of the tree-PRG.

$O(\log N)$, using emulative composition (i.e., the standard DFS-style simulation of circuits) to compute each chunk in each layer.[28]

Indeed, it is instructive to imagine this construction as a constant-depth tree, wherein each node is labeled by a polylog(M)-size circuit and has fan-in M. The reconstruction $\overline{\mathsf{REC}}$ will output a machine that has the label of the tree's root hard-wired, and that – when given index $i \in [N]$ – computes the labels of the nodes along path in the tree, and finally outputs the i^{th} bit of f. Computing the labels along the path is done by repeatedly evaluating each label; that is, each label is a polylog(M)-sized circuit whose truth-table is the next M-bit chunk, which contains the next label.

The only missing piece is explaining how $\overline{\mathsf{REC}}$ computes the label of the root (to hard-wire it into the machine above). At a high-level, it does so using emulative composition (i.e., the DFS-style algorithm, similarly to the generator), but to prevent a polynomial runtime overhead, it crucially uses the fact that REC at each layer is read-once. Specifically, $\overline{\mathsf{REC}}$ runs REC on the M-bit chunk that yields the root's label, providing REC virtual access to this chunk; the latter virtual access is provided by running REC and providing it virtual access to the chunks below it, and so on. The crucial point is that at each level, REC reads the relevant chunk bit-by-bit, in order; hence, at any given moment $\overline{\mathsf{REC}}$ only needs one path in the tree, and each path is only used once. Using the idea of computing the composition of a read-once f with a low-space g in near-linear time, by continuously storing the state of g (as described after Theorem 4.18), this procedure can be implemented in space polylog(N) and time $N \cdot \text{poly}(M) = N^{1+\varepsilon}$. See [61, Section 6.2] for details.

5 Open Problems

The big open problem that recent results suggest is to design more low-space algorithms using hardness vs randomness. In this section we suggest several interesting algorithmic problems that may be tackled using hardness vs randomness, as well as open problems related to improving the technical machinery underlying hardness vs randomness in the small-space setting.

[28] This statement assumes, for simplicity, that the reconstruction (which the generator needs to compute) runs in space $O(\log N)$. As stated in Remark 4.19, in [61] they show a second reconstruction algorithm which produces the same output as the one described above, and runs in space $O(\log N)$ (but is not read-once, which is a feature the generator does not need).

5.1 Algorithms using hardness vs randomness

Of course, the grandparent of the open algorithmic problems in this setting is to actually resolve that pesky 5-decades old question of derandomizing logspace. To make things (potentially) easier, we also raise the question of an average-case or scaled-up result:

Problem 1. Prove that $\mathsf{BPL} = \mathsf{L}$ (or $\mathsf{BPL} \subseteq \mathsf{avgL}$ or $\mathsf{BPSPACE}[O(n)] = \mathsf{SPACE}[O(n)]$) using hardness vs randomness; that is, reduce the problem to lower bounds that we can unconditionally prove.

As a more modest step along this direction:

Problem 2. Reduce $\mathsf{BPL} = \mathsf{L}$ (or $\mathsf{BPL} \subseteq \mathsf{avgL}$ or $\mathsf{BPSPACE}[O(n)] = \mathsf{SPACE}[O(n)]$) to lower bounds that are weaker than the ones in Theorems 3.9 and 3.10.

As explained in Section 3.2, a main technique for building catalytic algorithms is "compress or random", in which the catalytic tape is either useful or can be compressed. Since deterministic reconstruction provides a generic way to compress strings that are not good (i.e., that are not useful for fooling certain distinguishers), it is an obvious tool for generalizing "compress or random" to broader settings.

Problem 3. Design **catalytic algorithms** for interesting problems, in particular general problems (e.g., simulating classes of algorithms), using "compress or random" and deterministic reconstruction.

Hardness vs randomness tools were recently used by Chen *et al.* [15] to construct *pseudodeterministic* algorithms that (unconditionally) solve a broad class of explicit construction problems. The reason their algorithm is only pseudodeterminsitic, rather than deterministic, is that the reconstruction algorithm that they use is randomized. Given that in the logspace setting we have deterministic reconstruction algorithms, a natural direction is to try and use the same approach to design unconditional *deterministic* algorithms for explicit construction problems in the logspace setting.

Problem 4. Design new *deterministic* algorithms for interesting **explicit construction problems**, by leveraging hardness vs randomness with deterministic reconstruction.

Another unconditional algorithm using hardness vs randomness was shown by Carmosino *et al.* [12]. Loosely speaking, they instantiated a PRG that is intentionally faulty and can be broken, and deduced that the reconstruction unconditionally works. In their setting the reconstruction is a probabilistic learning algorithm, and indeed, they obtained a probabilistic learning algorithm for $\mathsf{AC}^0[\oplus]$. A natural direction is to try and use the same approach with deterministic reconstruction (which can be cast as a learning algorithm, cf. Theorem 4.1) to design unconditional deterministic learning algorithms for interesting classes.

Problem 5. Obtain **deterministic learning algorithms** for interesting function classes, by leveraging hardness vs randomness with deterministic reconstruction.

The assumption that composing low-space algorithms requires significant overhead in time or in space (i.e., the hypothesis in Theorem 3.8) *looks* natural, but should we believe that it is true? Remarkably, there is very little complexity-theoretic evidence that this assumption is true (as well as the corresponding assumption for k-wise composition, which was mentioned after Theorem 3.8). We believe that it is an important problem to establish complexity-theoretical foundations for this assumption.

Problem 6. Show that lower bounds on **composing low-space algorithms** (as in Theorem 3.8, or similar lower bounds) follow from natural and/or well-studied assumptions.

5.2 Distinguish-to-predict

As explained in Section 4.2, the key to extending deterministic reconstruction to broader settings is designing distinguish-to-predict transformations for more distinguishers.

Recent works constructed D2P transformations for ROBPs [28, 60], for ROBPs composed with certain PRGs for logspace [29], and for tests related to the isolation lemma [52]. As demonstrated by the D2P for the path-isolation lemma, it can be useful to consider specific types of distinguishers, which do not necessarily conform to classical complexity classes. We pose designing deterministic D2P for new types of distinguishers as an important and general open problem.

Problem 7. Construct a D2P transform for new types of distinguishers, and obtain (using the already available tools) generators with deterministic reconstruction for such distinguishers.

In the opposite direction, recall that for general circuits D, a D2P transform for D yields derandomization of D, i.e. an algorithm estimating $\mathbb{E}[D]$ (see Theorem 4.10). This result was not an obstacle for the works surveyed in this text, since constructing a D2P transform only yields a polynomial-time algorithm for estimating $\mathbb{E}[D]$ rather than a logspace algorithm, even if the D2P transform runs in logspace. The bottleneck in the proof is a construction of an unpredictable distribution by Goldreich and Wigderson [41], which uses large space (and is highly sequential).

Problem 8. Show a reduction of estimating $\mathbb{E}[D]$ for a given circuit D to constructing a D2P transform for D such that the reduction incurs as little complexity overhead as possible.

Another interesting open problem is to expand the reach of D2P by allowing more general notions of "predictor". Indeed, most predictors considered in the literature are next-bit-predictors, but list-predictors over large alphabets have been used extensively in the past (as one example see, e.g., [78]), and replacing next-bit-predictors with previous-bit-predictors has been useful for constructing D2P for ROBPs.

Problem 9. Is it easier to construct D2P transforms if we allow more general notions of predictors that can still be useful, such as list-predictors over non-binary alphabets (with a small list) or predict-one-out-of-k-symbols (for a small k)?

A specific interesting notion of a bit-predictor allows the predictor access to *all* bits except the one it tries to predict (cf., Definition 4.4). Interestingly, the polynomial-time algorithm of [41] that constructs an unpredictable distribution (for a given collection of predictors) does not work as-is with this more general definition of predictors (see Theorem A.4). There is a natural **ZPP** algorithm for this problem (guess a random distribution and verify all predictors do not achieve good advantage). Can we construct a *deterministic* polynomial-time algorithm, or would such an algorithm imply some breakthrough?

Problem 10. Give a polytime algorithm (or evidence that such an algorithm would imply new results) for the following problem: Given a collection of poly(n) predictors $P_i : \{0,1\}^{n-1} \to \{0,1\}$, each of which attempts to predict a bit j_i, output a distribution $\mathcal{S}$ that is $(1/n)$-unpredictable to all of them.

Lastly, perhaps the most important open problem in this context is to design reconstruction algorithms *without* D2P. As explained in Remark 4.2, relying on D2P in the reconstruction causes an inherent runtime overhead (which is an instance of the hybrid argument barrier), and there is concrete evidence that D2P is an overkill for reconstruction (since deterministic reconstruction is equivalent to $\mathsf{prZPP} = \mathsf{prBPP}$, whereas deterministic D2P is equivalent to the stronger $\mathsf{prBPP} = \mathsf{P}$). Remarkably, known reconstruction algorithms *all* fall into two camps: algorithms that use D2P, or algorithms that use non-determinism (e.g., as in [23,27,55,68,71]). Can we design a reconstruction algorithm that avoids both pitfalls?

Problem 11. Build a reconstructive pseudorandom generator whose reconstruction does not rely on nondeterminism *or* on a D2P transformation.

5.3 Generators for the logspace setting

In addition, we call for improving the main part of the hardness vs randomness framework in the logspace setting: designing better pseudorandom generators and reconstruction arguments. There are several directions that strike us as tractable and that are likely to have applications.

First, consider the targeted generator in Theorem 4.16, which is based on hardness of composing low-space algorithms. Can we design a targeted generator based on other types of hard functions, ideally more general function classes or more relaxed types of hardness?

Problem 12. Design a targeted generator that works in logspace and whose reconstruction works in deterministic logspace (or polylogspace), based on additional types of hard functions, beyond k-wise compositions of low-space algorithms for a small k.

A core technical component in the targeted generator constructions is the $(\overline{\mathsf{GEN}}, \overline{\mathsf{REC}})$ pair of Theorem 4.18, or alternatively the $(\mathsf{GEN}, \mathsf{REC})$ pair of Theorem 4.1, both of which are based on hardness of compressing a string. We believe that these two tools can be improved in a few ways.

First, the space complexity of the two reconstruction procedures is sub-optimal, i.e. $\mathrm{polylog}(N)$ and $O(\log N) + \mathrm{polylog}(M)$, respectively. This is because these two results rely on the SU PRG, and their reconstruction uses a derandomized version of the SU reconstruction with space complexity is $O(\log N) + \mathrm{polylog}(M)$.[29] Given that [28,60] showed a deterministic reconstruction for the NW PRG using space $O(\log N)$, it seems plausible that the reconstruction for the SU PRG can also be made to run in such space. (This is important because the NW PRG is suitable mostly when the output length is $M = N^{\Omega(1)}$, and the SU PRG is suitable more generally for any $M \leq N^{\Omega(1)}$; see Footnote 19.)

Problem 13. Improve the reconstruction algorithm for the SU generator so that it runs in space $O(\log N)$ when given access to a predictor. Alternatively, present another $(\mathsf{GEN}, \mathsf{REC})$ pair achieving the parameters as Theorem 4.1 but with reconstruction that runs in space $O(\log N)$ when given access to a predictor.

Secondly, the generator in Theorems 4.1 and 4.18 is a somewhere-PRG, rather than a PRG. As explained in Section 4.1, this disadvantage does not affect the specific results in this survey, since for the problems considered in this survey there are ways around it. However, it would be ideal to have a pseudorandom generator with deterministic reconstruction (and optimal parameters, unlike NW), rather than only a hitting-set generator. The natural candidate is the PRG by Umans [80], which is a refinement of the SU generator and thus its reconstruction may be amenable to similar derandomization techniques.

Problem 14. Derandomize the reconstruction procedure of Umans' [80] PRG (i.e., the reconstruction parts that follow the D2P transform).

[29] In Theorem 4.18 the reconstruction incurs an additional overhead due to the tree compression procedure described in Section 4.3.2.

Another possible improvement is to have a reconstruction algorithm that compresses the hard string to an oracle circuit *from a weak circuit class*. Indeed, classical non-uniform reconstruction arguments do yield weak circuits (e.g., TC^0 circuits, and this is true even in the logspace setting [30]), as does the deterministic reconstruction for the NW generator in [60]. However, the reconstruction arguments in Theorems 4.1 and 4.18 have no such guarantee. One concrete goal to shoot for is having a procedure from a class that is evaluable in logspace, either a circuit from a weak class or a Turing machine running in logspace.

Problem 15. Improve Theorems 4.1 and 4.18 such that the output of the reconstruction algorithm can be evaluated in space $O(\log N)$.

A related improvement, which is specific to Theorem 4.18, refers to the fact that the reconstruction in that result outputs an oracle machine A such that A^D computes f, but evaluating the machine with input j to produce f_j takes time $t \gg |f|$. This stands in contrast to standard reconstruction arguments (e.g., in [58, 71], or their deterministic counterparts in [28, 29, 60, 61]), which output a circuit C of size $|C| \ll |f|$ such that C^D computes f.

Problem 16. Improve Theorem 4.18 such that the reconstruction algorithm outputs a circuit C of size $\mathrm{poly}(M, \log(N))$.

A last open problem may have applications when considering derandomization with overhead that is simultaneously minimal in both time and space. In the results mentioned in Section 3.4.2, the generator has very low space consumption, but it may be slow, and ditto for the reconstruction. In Theorems 4.16 and 4.18 the reconstruction's running time is optimized to be essentially linear, but the generator may still be slow (i.e., GEN runs in time $\mathrm{poly}(|f|)$ for some large unspecified polynomial). Is this improvable?

Problem 17. Improve Theorem 4.18 such that GEN runs in time $|f|^{1+\varepsilon}$.

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, Cambridge, 2009.

[2] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. A new general derandomization method. *Journal of the ACM*, 45(1):179–213, 1998.

[3] Alexander E. Andreev, Andrea E. F. Clementi, José D. P. Rolim, and Luca Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM J. Comput.*, 28(6):2103–2116, 1999.

[4] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

[5] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed *s-t* connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998.

[6] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Proc. 46 Annual ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014.

[7] Marshall Ball, Lijie Chen, and Roei Tell. Towards free lunch derandomization from necessary assumptions (and owfs). In Srikanth Srinivasan, editor, *40th Computational Complexity Conference, CCC 2025, Toronto, Canada, August 5-8, 2025*, volume 339 of *LIPIcs*, pages 31:1–31:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[8] Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In *Proc. 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 100–109, 1999.

[9] László Babai, Péter Hajnal, Endre Szemerédi, and György Turán. A lower bound for read-once-only branching programs. *Journal of Computer and System Sciences*, 35(2):153–162, 1987.

[10] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[11] Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. *Theory Comput.*, 18:1–32, 2022.

[12] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, Tokyo, Japan, May 29 - June 1, 2016*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[13] Lijie Chen, Jiatu Li, and Jingxun Liang. Maximum circuit lower bounds for exponential-time Arthur Merlin. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1348–1358, [2025] ©2025.

[14] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. *Electron. Colloquium Comput. Complex.*, TR24-106, 2024.

[15] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. *arXiv preprint arXiv:2305.15140*, 2023.

[16] Lijie Chen, Xin Lyu, Avishay Tal, and Hongxun Wu. New PRGs for unbounded-width/adaptive-order read-once branching programs. In *Proc. 50 International Colloquium on Automata, Languages and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[17] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 752–760. ACM, 2020.

[18] James Cook and Ian Mertz. Tree evaluation is in space o(log n · log log n). *Electron. Colloquium Comput. Complex.*, TR23-174, 2023.

[19] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2):4:1–4:43, 2012.

[20] Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980.

[21] Lijie Chen, Ron D. Rothblum, and Roei Tell. Unstructured hardness to average-case randomness. In *Proc. 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 429–437, 2022.

[22] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2021.

[23] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. In *Proc. 53st Annual ACM Symposium on Theory of Computing (STOC)*, pages 283–291, 2021.

[24] Lijie Chen and Roei Tell. Guest column: New ways of studying the **BPP** = **P** conjecture. *ACM SIGACT News*, 54(2):44–69, 2023.

[25] Lijie Chen, Roei Tell, and Ryan Williams. Derandomization vs refutation: A unified framework for characterizing derandomization. In *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2023. To appear.

[26] Holger Dell, Valentine Kabanets, Dieter van Melkebeek, and Osamu Watanabe. Is Valiant-Vazirani's isolation probability improvable? *Computational Complexity*, 22(2):345–383, 2013.

[27] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *Journal of the ACM*, 69(6):1–55, 2022.

[28] Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for **BPL** = **L** that uses properties of **BPL**. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, 2024.

[29] Dean Doron, Edward Pyne, Roei Tell, and Ryan Williams. When connectivity is hard, random walks are easy with non-determinism. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, 2025.

[30] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[31] Michael A. Forbes and Zander Kelley. Pseudorandom generators for read-once branching programs, in any order. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 946–955, 2018.

[32] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013.

[33] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, New York, NY, USA, 2008.

[34] Oded Goldreich. In a world of **P** = **BPP**. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*, pages 191–232, 2011.

[35] Oded Goldreich. A sample of samplers: A computational perspective on sampling. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2011.

[36] Oded Goldreich. Two comments on targeted canonical derandomizers. *Electronic Colloquium on Computational Complexity: ECCC*, 2011.

[37] Uma Girish, Ran Raz, and Wei Zhan. Is untrusted randomness helpful? In *Proc. 14 Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 251 of *LIPIcs*, pages 56:1–56:18, 2023.

[38] Zeyu Guo. Randomness-efficient curve samplers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 575–590. Springer, 2013.

[39] Oded Goldreich, Salil Vadhan, and Avi Wigderson. Simplified derandomization of BPP using a hitting set generator. In *Studies in complexity and cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 59–67. Springer, Heidelberg, 2011.

[40] Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996.

[41] Oded Goldreich and Avi Wigderson. On pseudorandomness with respect to deterministic observers. *Electron. Colloquium Comput. Complex.*, TR00-056, 2000.

[42] Oded Goldreich and David Zuckerman. Another proof that $BPP \subseteq PH$ (and more). In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 2011.

[43] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within *NP*. *SIAM Journal on Computing*, 52(6):FOCS18–349–FOCS18–382, 2023.

[44] William M. Hoza. Recent progress on derandomizing space-bounded computation. *Bull. EATCS*, 138, 2022.

[45] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2000.

[46] Russell Impagliazzo and Avi Wigderson. $\mathsf{P} = \mathsf{BPP}$ if E requires exponential circuits: derandomizing the XOR lemma. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[47] Russell Impagliazzo and Avi Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 734–743, 1998.

[48] Michal Koucký, Ian Mertz, Edward Pyne, and Sasha Sami. Collapsing catalytic classes. *CoRR*, abs/2504.08444, 2025.

[49] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[50] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[51] Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of levin-kolmogorov complexity. In *Proc. 37 Annual IEEE Conference on Computational Complexity (CCC)*, 2022.

[52] Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness, 2024.

[53] Pinyan Lu, Jialin Zhang, Chung Keung Poon, and Jin-yi Cai. Simulating undirected *st*-connectivity algorithms on uniform JAGs and NNJAGs. In *Proceedings of 16th International Symposium on Algorithms and Computation (ISAAC)*, volume 3827 of *Lecture Notes in Computer Science*, pages 767–776. Springer, 2005.

[54] Ian Mertz. Reusing space: Techniques and open problems. *Bulletin of EATCS*, 141(3), 2023.

[55] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[56] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[57] Noam Nisan. $\mathsf{RL} \subseteq \mathsf{SC}$. *Computational Complexity*, 4:1–11, 1994.

[58] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[59] Chung Keung Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 218–227. IEEE Computer Society, 1993.

[60] Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, 2023.

[61] Edward Pyne and Roei Tell. Composing low-space algorithms. *Electronic Colloquium on Computational Complexity: ECCC*, 2025.

[62] Edward Pyne. Derandomizing logspace with a small shared hard drive. In *Proc. 39th Annual IEEE Conference on Computational Complexity (CCC)*, pages 4:1–4:20, 2024.

[63] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

[64] Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998.

[65] Michael E. Saks. Randomization and derandomization in space_bounded computation. In Steven Homer and Jin-Yi Cai, editors, *Proceedings of the Eleveth Annual IEEE Conference on Computational Complexity, Philadelphia, Pennsylvania, USA, May 24-27, 1996*, pages 128–149. IEEE Computer Society, 1996.

[66] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[67] Michael Sipser. A complexity theoretic approach to randomness. In *Proc. 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 330–335, 1983.

[68] Michael Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, 1988.

[69] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[70] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[71] Ronen Shaltiel and Christopher Umans. Low-end uniform hardness vs. randomness tradeoffs for AM. In *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 430–439, 2007.

[72] Ronen Shaltiel and Emanuele Viola. On hardness assumptions needed for “extreme high-end” PRGs and fast derandomization. In *Proc. 13 Conference on Innovations in Theoretical Computer Science (ITCS)*, 2022.

[73] Rahul Santhanam and R. Ryan Williams. On medium-uniformity and circuit lower bounds. In *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*, pages 15–23. IEEE, 2013.

[74] Michael E. Saks and Shiyu Zhou. $\mathbf{BP_HSPACE}[S] \subseteq \mathbf{DSPACE}[S^{3/2}]$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[75] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[76] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[77] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[78] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed-Muller codes. *Journal of Computer and System Sciences*, 72(5):786–812, 2006.

[79] Yoav Tzur. Notions of weak pseudorandomness and GF(2^n)-polynomials. Master's thesis, Weizmann Institute of Science, 2009.

[80] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[81] Dieter van Melkebeek and Gautam Prakriya. Derandomizing isolation in space-bounded settings. *SIAM J. Comput.*, 48(3):979–1021, 2019.

[82] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[83] Avi Wigderson. $nl/\mathrm{poly} \subseteq \oplus l/\mathrm{poly}$. In *Proc. 9th Conf. Structure in Complexity Theory*, 1994.

[84] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proc. 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 231–240, 2010.

[85] R. Ryan Williams. Simulating time with square-root space. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 13–23. ACM, 2025.

[86] Ryan Williams. Personal communication, 2025.

[87] Andrew C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

A Alternative proofs for classical theorems using D2P

The notion of D2P gives some nice proofs of previously known results. We cover two proofs from [52]: how can we estimate the expectation of a circuit in the polynomial hierarchy, and how can we solve derandomization of algorithms with two-sided error using only hitting sets?

Derandomizing with games. The strongest known simulation of **BPP** in the polynomial-time hierarchy, first due to Russell and Sundaram [64] (following [50, 67]), asserts that **BPP** $\subseteq$ $\mathbf{S_2P}$. The delightful class $\mathbf{S_2P}$ is a subclass of $\Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$, where the $\exists$ and $\forall$ players can ignore each other:

Definition A.1. A language $L \in \mathbf{S_2P}$ if there is a polynomial-time algorithm $V(x, g_0, g_1)$ (called the verifier) and a polynomial $p(n)$ such that:

- For $x \in L$, there exists $g_1 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the YES-player) such that for every $g_0 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the NO-player), $V(x, g_0, g_1) = 1$.

- For $x \notin L$, there exists $g_0 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the NO-player) such that for every $g_1 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the YES-player), $V(x, g_0, g_1) = 0$.

Theorem A.2. *There is a* $\mathbf{S_2P}$ *protocol to estimate* $\mathbb{E}[D] \pm (1/3)$ *for a circuit D.*

Proof Sketch. We want strategies for both players that ensure the other cannot convince the verifier that the expectation differs from the true value.

Lets assign Player 1 the job of giving a distribution S that approximates the expectation for *every* small circuit (which in particular includes the circuit D given to both players). But of course, Player 1 can lie and give a distribution that does not approximate $\mathbb{E}[D]$. How can Player 2 make sure that no distribution that misleads on the value of $\mathbb{E}[D]$ sneaks through? By giving a valid D2P transformation for D! If P1 gives a distribution $\mathcal{S}$ with no small predictors (which exists via a simple counting argument), no matter what predictors P2 writes down, $\mathcal{S}$ will be accepted and used to estimate $\mathbb{E}[D]$, and since a distribution with no small predictors has no small distinguishers either, that estimate will be accurate. On the other hand, if P2 gives a valid D2P transform for D, by the definition of D2P transformation any distribution $\mathcal{S}'$ that is not predicted must do a good job estimating $\mathbb{E}[D]$. Since a distribution $\mathcal{S}$ with no small predictors and a valid D2P transform for D always exist, both players can ignore each other. □

Hitting sets imply derandomization of algorithms with two-sided error. A similar construction gives a new proof that one-sided derandomization implies two-sided derandomization. Recall that in one-sided derandomization we are given a circuit $D : \{0,1\}^n \to \{0,1\}$ and asked to distinguish between $\mathbb{E}[D] = 0$ and $\mathbb{E}[D] \geq 1/2$. The analogue of a pseudorandom generator (resp., pseudorandom set) in this regime is known as a hitting set generator (reps., hitting set):

Definition A.3. For a class of circuits $\mathcal{D}$ on n bits, a hitting set $H \subseteq \{0,1\}^n$ for $\mathcal{D}$ has the property that for all $D \in \mathcal{D}$ with $\mathbb{E}[D] \geq 1/2$, there is $y \in H$ where $D(y) = 1$.

While hitting sets are not obviously capable of giving two-sided derandomization, classical results show that they do [2,3,8,39,42,50,67]. In [52] they showed that using D2P, we can get a new, relatively simple proof for this fact.

To do so, they use an algorithm of Goldreich and Widgerson [41] (slightly generalized in [52]) that gets a collection of next-bit predictors on $\{0,1\}^n$ (which we can think of as a D2P transform for D), runs in *deterministic* polynomial time, and outputs a distribution $\mathcal{S}$ that is unpredictable to all of them:

Theorem A.4 (Informal, see [41] and Lemma 4.2 [52]). *There is a deterministic polytime algorithm that given a collection of next-bit-predictors $P_1, \ldots, P_{\mathrm{poly}(n)}$ over $\{0,1\}^n$, outputs a distribution $\mathcal{S}$ of size at most n^5 such that no P_i is a $(1/3n)$-predictor for $\mathcal{S}$.*

We can then use Theorem A.4 to prove the result.

Theorem A.5. *Suppose there is an hitting set for circuits of size n on n input bits that can be constructed in time* $\mathrm{poly}(n)$. *Then there is a polynomial-time algorithm that, given a circuit* $D : \{0,1\}^n \to \{0,1\}$, *estimates* $\mathbb{E}[D]$ *to additive error* $1/3$.

Proof Sketch. We will show that using H, we can produce a D2P transform $\mathcal{P} = (P_1, \ldots, P_t)$ for D that is secure against all distributions of size at most n^5, in the sense that every small distribution that D distinguishes from random is predicted by some $P \in \mathcal{P}$. We then produce a distribution $\mathcal{S}$ that is unpredictable to all these predictors via Theorem A.4, and use that distribution to estimate $\mathbb{E}[D]$. Since the D2P transformation is secure, this estimate must be accurate. Details follow.

Let $\mathbf{w}$ be an arbitrary distribution of size n^5, and suppose D is a $(1/3)$-distinguisher for $\mathbf{w}$. By Proposition 4.8, a random string z, b, j of length $\widetilde{O}(n)$ has the property that the function

$$P_{(z,b,j)}(x_{<j}) = D(x_{<j} \circ z_{\geq j}) \oplus b$$

is a $(1/3n)$-predictor for $\mathbf{w}$, with probability $\rho = 1/\mathrm{poly}(n)$. Moreover, there is a circuit $\mathrm{TEST}_{D,\mathbf{w}}(y, b, j)$ of size $\widetilde{O}(n^6)$ that checks if the string (z, b, j) creates such a good predictor. We may assume by a standard error reduction that the hitting set H hits all size-$\widetilde{O}(n^6)$ circuits with expectation at least ρ, so if we consider the collection of predictors obtained by enumerating all elements of the hitting set

$$\mathcal{P} = \{P_{y=(z,b,j)} : y \in H\}$$

there must be some $P \in \mathcal{P}$ where P is a $(1/3n)$-predictor for $\mathbf{w}$. Thus, for *every* bad distribution of size at most n^5, there is $P \in \mathcal{P}$ that is a $(1/3n)$-predictor, and there are $|H| = \mathrm{poly}(n)$ total such predictors.

We invoke the algorithm of Theorem A.4 on this collection $\mathcal{P}$. The distribution $\mathcal{S}$ we receive as output is of size n^5 and $(1/3n)$-unpredictable to all $P \in \mathcal{P}$, so it must be the case that

$$\left| \mathop{\mathbb{E}}_{y \in \mathcal{S}}[D(y)] - \mathbb{E}[D] \right| \leq 1/3$$

and we can return $\mathbb{E}_{y \in \mathcal{S}}[D(y)]$ as our estimate. □

A related proof idea implies that *white-box* one-sided derandomization (i.e. an algorithm that given D returns if $\mathbb{E}[D] = 0$ or $\mathbb{E}[D] \geq 1/2$, with no correctness guarantees otherwise) likewise implies two-sided derandomization. Unfortunately, like all other proofs of this fact, if the non-black-box derandomization runs in time $T(n) \gg \mathrm{poly}(n)$, the corresponding two-sided derandomization runs in time $T(T(n)) \gg T$. For full results, see [52, Theorem A.2].