Certified Hardness vs. Randomness for Log-Space

Edward Pyne*
MIT
epyne@mit.edu

Ran Raz†
Princeton University
ranr@cs.princeton.edu

Wei Zhan‡
Princeton University
weizhan@cs.princeton.edu

January 7, 2026

Abstract

Let $\mathcal{L}$ be a language that can be decided in linear space and let $\epsilon > 0$ be any constant. Let $\mathcal{A}$ be the exponential hardness assumption that for every n, membership in $\mathcal{L}$ for inputs of length n cannot be decided by circuits of size smaller than $2^{\epsilon n}$. We prove that for every function $f : \{0,1\}^* \to \{0,1\}$, computable by a randomized logspace algorithm R, there exists a deterministic logspace algorithm D (attempting to compute f), such that on every input x of length n, the algorithm D outputs one of the following:

1. The correct value $f(x)$.
2. The string: "I am unable to compute $f(x)$ because the hardness assumption $\mathcal{A}$ is false", followed by a (provenly correct) circuit of size smaller than $2^{\epsilon n'}$ for membership in $\mathcal{L}$ for inputs of length n', for some $n' = \Theta(\log n)$; that is, a circuit that refutes $\mathcal{A}$.

Moreover, D is explicitly constructed, given R.

We note that previous works on the hardness-versus-randomness paradigm give derandomized algorithms that rely blindly on the hardness assumption. If the hardness assumption is false, the algorithms may output incorrect values, and thus a user cannot trust that an output given by the algorithm is correct. Instead, our algorithm D verifies the computation so that it never outputs an incorrect value. Thus, if D outputs a value for $f(x)$, that value is certified to be correct. Moreover, if D does not output a value for $f(x)$, it alerts that the hardness assumption was found to be false, and refutes the assumption.

Our next result is a universal derandomizer for **BPL** (the class of problems solvable by bounded-error randomized logspace algorithms)[1]: We give a deterministic algorithm U that takes as an input a randomized logspace algorithm R and an input x and simulates the computation of R on x, deteriministically. Under the widely believed assumption $\mathbf{BPL} = \mathbf{L}$, the space used by U is at most $C_R \cdot \log n$ (where C_R is a constant depending on R). Moreover, for every constant $c \geq 1$, if $\mathbf{BPL} \subseteq \mathbf{SPACE}[(\log(n))^c]$ then the space used by U is at most $C_R \cdot (\log(n))^c$.

Finally, we prove that if optimal hitting sets for ordered branching programs exist then there is a deterministic logspace algorithm that, given a black-box access to an ordered branching program B of size n, estimates the probability that B accepts on a uniformly random input. This extends the result of (Cheng and Hoza CCC 2020), who proved that an optimal hitting set implies a white-box two-sided derandomization.

Keywords: pseudorandomness, space-bounded computation

*Supported by an Akamai Presidential Fellowship. Part of this work was done while visiting the Simons Institute Program on Meta-Complexity.

†Supported by a Simons Investigator Award and by the National Science Foundation grant No. CCF-2007462.

‡Supported by a Simons Investigator Award and by the National Science Foundation grant No. CCF-2007462.

[1]Our result is stated and proved for **promise-BPL**, but we ignore this difference in the abstract.

1 Introduction

In a recent work, Girish, Raz and Zhan studied the power of untrusted randomness [GRZ23]. One of their main observations was that randomized logspace computations are verifiable using only $O(\log n)$ random bits. More precisely, every problem in **BPL** has a streaming proof between a randomized logspace prover and a randomized logspace verifier, where the verifier uses only $O(\log n)$ random bits and has a read-once one-way access to the proof that is streamed by the prover. In other words, the prover provides a polynomial-length proof that is streamed to the verifier and the verifier can check whether the computation was performed correctly using only $O(\log n)$ random bits.

This raises the following intriguing possibility. Try to replace the random string of the prover by, say, the digits of π. In most cases, that should work and the computation should be performed correctly, as the digits of π seem unrelated to most computations. In the rare cases that the computation is not performed correctly, the verifier will figure that out, as the verification will fail with high probability, so no harm is done. Moreover, since the digits of π can be generated deterministically in small space, the prover is now deterministic so the verifier can fully simulate the prover. Since the verifier uses only $O(\log n)$ random bits, the verifier can just try all possibilities for these random bits so that the verifier is also deterministic[2], and thus the entire interaction is now simulated by a deterministic logspace algorithm.

This approach won't derandomize all randomized logspace computations, since the digits of π can be generated by a small space algorithm. The digits of π were not designed to fool randomized computations. The next logical step is to try to use sequences that were designed to fool randomized computations, namely, candidate constructions of pseudorandom generators, such as pseudorandom generators that are based on the hardness-versus-randomness paradigm [Sha81,Yao82,BM84,NW94, IW97, STV01, KvM02]. Such pseudorandom generators fool randomized computations, within a certain complexity class, assuming that certain widely-believed hardness assumptions hold.

Let G be a candidate construction for a pseudorandom generator, designed to fool randomized logspace computations, and assume that G uses logarithmic space and $O(\log n)$ random bits. We can try to replace the random string of the prover by pseudorandom sequences that are generated by G. Since, if we do so, both the prover and the verifier use a logarithmic number of random bits, the verifier can simulate the entire interaction by a deterministic logspace algorithm. If one of the $\text{poly}(n)$ possibilities for the $O(\log n)$ random bits of the generator results in a valid proof that the computation was performed correctly, the verifier will figure that out and accept that computation. If all $\text{poly}(n)$ possibilities fail, the verifier will alert that the generator failed. Thus, the algorithm never outputs an incorrect value.

If the generator G is based on the hardness-versus-randomness paradigm, a failure of the generator implies that the hardness assumption that the generator is based on is false. Moreover, proofs that are based on the hardness-versus-randomness paradigm are typically constructive, in the sense that they show that if the generator fails then one can construct a circuit that refutes the hardness assumption. If we can prove that constructing that circuit can be done in deterministic logspace then the verifier can obtain a circuit that refutes the hardness assumption that G is based on.

We use a variant of the hardness-versus-randomness pseudorandom generator of Klivans and van Melkebeek [KvM02] that builds on [NW94, IW97, STV01] to derandomize **BPL** (assuming an exponential hardness assumption). Based on this generator, we obtain the following result.

[2]Derandomizing the verifier by trying all possibilities for its random bits is not possible when the prover is randomized, or when the prover cannot be simulated by the verifier, since the verifier needs multi-access to the output of the prover in order to do that.

Theorem 1.1. *Let $\mathcal{L}$ be a language that can be decided in linear space and let $\epsilon > 0$ be a constant. Let $\mathcal{A}$ be the exponential hardness assumption that for every n, membership in $\mathcal{L}$ for inputs of length n cannot be decided by circuits of size smaller than $2^{\varepsilon n}$. Let $f : \{0,1\}^* \to \{0,1\}$ be a function computable by a randomized logspace algorithm R. Then, there exists a deterministic logspace algorithm D (explicitly given from R), such that on every input x of length n, the algorithm D outputs one of the following:*

1. *The correct value $f(x)$.*
2. *The string: "Unable to compute $f(x)$ because the hardness assumption $\mathcal{A}$ is false", followed by a (provenly correct) circuit of size smaller than $2^{\varepsilon n'}$ for membership in $\mathcal{L}$ for inputs of length n', for some $n' = \Theta(\log n)$; that is, a circuit that refutes $\mathcal{A}$.*

In other words, while the algorithms given by all previous derandomization results based on the hardness-versus-randomness paradigm rely blindly on the hardness assumption, and may output incorrect values if the hardness assumption is false, our algorithm D never outputs an incorrect value: If the hardness assumption is true, D always outputs the correct value $f(x)$. If the hardness assumption is false D still outputs the correct value $f(x)$, or alerts that the hardness assumption is false, and refutes the assumption.

In particular, if the hardness assumption used in Theorem 1.1 is true (and there are several such assumptions that are widely believed to be true), Theorem 1.1 gives a deterministic logspace algorithm that always outputs the correct value of $f(x)$ and that value is certified to be correct. In that sense, if the hardness assumption is true, the algorithm given by Theorem 1.1 effectively functions as a full derandomizer for the class **BPL**.

We note that in previous works, the, so called, reconstruction step, in which a circuit that refutes the hardness assumption is constructed (when the generator fails), required the use of randomness in multiple places and was not known to be computable in logspace. Our main technical contribution in the proof of Theorem 1.1 is carefully designing the pseudorandom generator and proving that for that generator, all parts of the reconstruction step can be done in deterministic logspace. We view this result, that the reconstruction can be done in deterministic logspace, as a separate contribution of our work.

Let us go back to the observation that every problem in **BPL** has a streaming proof between a randomized logspace prover and a randomized logspace verifier, where the verifier uses only $O(\log n)$ random bits and has a read-once one-way access to the proof that is streamed by the prover [GRZ23]. The proof is based on a protocol where the prover computes and streams the probability to reach each state of the branching program, underlying a randomized algorithm, and the verifier checks that these probabilities are consistent between each two consecutive time steps.

While we can use this approach to prove Theorem 1.1, we give here a slightly different and more direct proof, where the verification is done by verifying that the distribution of each bit that the pseudorandom generator outputs, conditioned on reaching each state of the underlying branching program, is close to uniform. These conditional probabilities are computed directly by checking all possible outputs of the pseudorandom generator. This is possible because the generator uses only a logarithmic number of random bits and hence the number of possibilities is polynomial in n. This approach is related to the work of Nisan [Nis93], who used a similar approach to check if a given polynomial-size set of strings is sufficiently random to simulate a randomized computation with high accuracy, in his proof that **BPL** $\subset$ **ZP*L** (where **ZP*L** is zero-error randomized logspace, where the machine has *two-way* access to the random tape).

The discussion above implies that the output of a candidate pseudorandom generator G (that uses logarithmic space and $O(\log n)$ random bits) can be verified as being sufficiently random

for a given randomized logspace computation. With this in mind, it is natural to try to find a pseudorandom generator that will be sufficiently good for a given randomized logspace computation, by an exhaustive search over all possible generators (using the fact that the generator is described by a constant size Turing machine). The final goal is to obtain a universal derandomizer, that will do at least as good as the best pseudorandom generator.

We explore this idea and discover that an even stronger result can be proved. We explicitly construct a universal derandomizer U for **prBPL** (**promise-BPL**, the class of promise problems solvable by bounded-error randomized logspace algorithms) that runs in the best possible deterministic space bound on **prBPL**.

More precisely, we give a deterministic algorithm U that takes as an input a randomized logspace algorithm R and an input x and simulates the computation of R on x. Under the widely believed assumption $\mathbf{prBPL} = \mathbf{L}$, the space used by U is at most $C_R \cdot \log n$ (where C_R is a constant depending on R). More generally, for every constant $c \geq 1$, if $\mathbf{prBPL} \subseteq \mathbf{SPACE}[(\log(n))^c]$ then the space used by U is at most $C_R \cdot (\log(n))^c$. We emphasize that the point here is that U is deterministic and is explicitly given, rather than an existential result. We remark that a similar result is not known in the time bounded case, and seems hard to obtain. We also remark that the best currently known space bound on **BPL** is $\mathbf{prBPL} \subseteq \mathbf{SPACE}[(\log(n))^{1.5-o(1)}]$ [SZ99, Hoz21].

Theorem 1.2. *Let U be the deterministic algorithm that is explicitly given in Section 5, that takes as an input a randomized logspace algorithm R and an input x. Assume that the probability that R accepts on x is either $\leq 1/4$ or $\geq 3/4$. Then, if the probability that R accepts on x is $\leq 1/4$, the output of U on input R, x is 0 and if the probability that R accepts on x is $\geq 3/4$, the output of U on input R, x is 1. Moreover, for every constant $c \geq 1$, if $\mathbf{prBPL} \subseteq \mathbf{SPACE}[(\log(n))^c]$ then the space used by U is at most $C_R \cdot (\log(n))^c$, (where C_R is a constant depending on R).*

We note that one can bound the space used by U, in Theorem 1.2, also by $C \cdot (\log(N))^c$, where C is a universal constant and N is an upper bound on both the length and width of the branching program underlying the computation of R on x (under the assumption $\mathbf{prBPL} \subseteq \mathbf{SPACE}[(\log(n))^c]$). (See Theorem 5.1).

Another prior work that is related to our work, as well as to [Nis93,GRZ23], is the work of Cheng and Hoza [CH22]. Cheng and Hoza proved that an optimal hitting set generator (the one-sided analogue of a pseudorandom generator) for logspace would imply $\mathbf{BPL} = \mathbf{L}$ (whereas the direct conclusion of such a hitting set generator would only be $\mathbf{RL} = \mathbf{L}$) [CH22]. To prove this result, they show how to use the hitting set generator to guess (approximations of) the probability to reach each state of a branching program, and they then check that these probabilities are consistent between each two consecutive time steps (similarly to and prior to [GRZ23]).

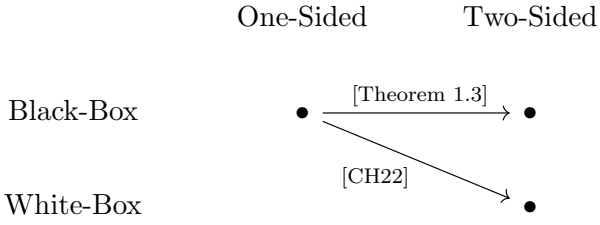
The proof given by Cheng and Hoza uses the explicit description of the underlying branching program. Our final result is an extension of their result to the case where the branching program is not given explicitly, but rather one only has oracle access to it, that is, access as a black box.

Theorem 1.3 (Informal: formally stated and proved in Section 6)**.** *Assume that optimal explicit hitting set generators for width n, length n ordered branching programs exist. Then optimal deterministic samplers for width n, length n ordered branching programs (with oracle access to the branching program) exist.*

The proof of this result relies on developing a "local consistency" test that can be implemented (using a hitting set) given black-box access to a branching program, in contrast to all prior tests in the literature [Nis93, CH22, GRZ23].

We remark that Cheng and Hoza [CH22] prove a version of this result for *constant* width branching programs (in addition to their non-black-box result on length n, width n programs that

capture **BPL**). They state a black-box equivalence in the **BPL** vs **L** regime as an open question, which we resolve. Our result complements equivalent results in the **BPP** vs **P** regime; several prior results [ACR96, BF99, ACRT99, GVW11, CH22] show that a hitting set for general circuits implies a deterministic sampler for general circuits. Thus, we close the gap in understanding between time-bounded and space-bounded derandomization with regards to this question.



We hope that our progress can eventually be used to get an equivalence in the *white-box* regime, that is, that $\mathbf{prRL} = \mathbf{L} \implies \mathbf{prBPL} = \mathbf{L}$. Such a result was established in the time-bounded regime by [BF99].

A common theme in all of our results is that our proofs exploit, and further demonstrate, the intriguing idea that in some settings randomized logspace computations can be verified.

1.1 Related Work

There have been four decades of work attempting to derandomize randomized logspace, that is, prove $\mathbf{BPL} = \mathbf{L}$. This work has taken (at least) two major forms: constructions of pseudorandom generators (PRGs) and their generalizations [Nis90, INW94, NZ96, GR14, FK18, MRT19, HZ20] and white-box derandomizations [SZ99, RR99, Rei08, RTV06, AKM+20, Hoz21]. This has resulted in a varied landscape, with explicit constructions of PRGs that obtain highly nontrivial but (presumably) suboptimal seed lengths, white-box derandomizations, and candidate constructions. We emphasize that these candidate constructions consist of both generators whose security follows from a certain hardness assumption [KvM02], and candidates that are not known to follow from a hardness assumption (for instance, the XOR of two small-bias distributions has been proposed as a candidate by Reingold and Vadhan [LV17]).

As mentioned above, besides [KvM02], the works most relevant to ours are [Nis93,CH22,GRZ23]. All these works have an element of verification that a randomized computation was performed correctly (in various forms and for various purposes), an idea that is also central in our work.

2 Preliminaries

We first define notation related to pseudorandom generators and branching programs.

Definition 2.1. Given a distribution D over a space $[S]$, let $x \leftarrow D$ represent drawing $x \in [S]$ from D. We let U_n denote the uniform distribution over $\{0,1\}^n$.

Definition 2.2. Given a pseudorandom generator (PRG) $G : \{0,1\}^s \to \{0,1\}^n$ and a function $f : \{0,1\}^n \to \mathbb{R}$, we use $\mathbb{E}[f]$ and $\mathbb{E}_G[f]$ to denote the expectation of f under uniformly distributed inputs and pseudorandom inputs generated by G respectively, that is,

$$\mathbb{E}[f] = \mathop{\mathbb{E}}_{x \leftarrow U_n}[f(x)], \qquad \mathop{\mathbb{E}}_{G}[f] = \mathop{\mathbb{E}}_{y \leftarrow U_s}[f(G(y))].$$

And we say that G **ε-fools** f if $|\mathbb{E}[f] - \mathbb{E}_G[f]| \leq \varepsilon$.

Definition 2.3. An ordered branching program (OBP) B of length n and width w is a directed acyclic graph whose vertices (or states) $V(B)$ are partitioned into $n+1$ layers $V_0, \ldots, V_n$ where $|V_i| \leq w$. For each $i < n$ and $v \in V_i$, there are two outgoing edges, labeled with 0 and 1 respectively, that lead into V_{i+1}. V_0 constains a single state v_{st} which is the starting state, and each state in V_n is labeled with a real number as the output of the branching program. Unless otherwise specified, we assume that the labels are either 0 or 1. In that case, we assume without loss of generality there is a single state labeled with 1, which we denote v_{acc}.

For each $v \in V_i$, $\sigma \in \{0,1\}^k$ and $u \in V_{i+k}$, we say $B[v,\sigma] = u$ if B transitions from state v to state u following the edges labeled by the bits in σ. We can think of B as a function on $\{0,1\}^n$ such that for every $x \in \{0,1\}^n$, $B(x)$ is the label on the output state $B[v_0, x]$. For each $v \in V_i$, let $B_{\to v}$ be an OBP of length i and width w such that $B_{\to v}(x) = 1$ if and only if $B[v_0, x_{1..i}] = v$.

For each $v \in V_i$, let

$$p_{\to v} = \Pr[B[v_{st}, U_i] = v], \qquad p_{v\to} = \Pr[B[v, U_{n-i}] = v_{\text{acc}}].$$

3 Effective Hardness to Randomness

We prove Theorem 1.1 in several stages. In the first stage, we show a testing procedure that, given a candidate PRG and an ordered branching program, either certifies that the PRG fools the branching program, or outputs a branching program that acts as a next-bit predictor for G. We then show how to go from such a next-bit predictor to a counterexample to the hardness assumption.

3.1 Verifiable PRGs for Logspace

We first show that there is a logspace verifier for PRGs (with logarithmic seed) against logspace OBPs, which detects when a PRG fails and outputs an example OBP that the PRG fails to fool. To formalize this, we recall the notion of a next-bit-predictor.

Definition 3.1. Given a function $G : \{0,1\}^s \to \{0,1\}^n$, a branching program $T : \{0,1\}^i \to \{0,1\}$ for $i < n$ is an ε-**next-bit-predictor** for G if $\Pr_{x \leftarrow U_s}[T(G(x)_{1..i}) = G(x)_{i+1}] > 1/2 + \varepsilon$.

Note that the uniform distribution is 0-next-bit-predictable, even for a computationally unbounded distinguisher.

We prove in this section the following lemma:

Lemma 3.2. *For every error function $\varepsilon(n)$ computable in space $O(\log n)$, there is a deterministic algorithm that, given as input an OBP B of length n and width w, and the black-box oracle access to a PRG $G : \{0,1\}^s \to \{0,1\}^n$, runs in space $O(s + \log(nw))$, and either*

1. *Confirms that G $\varepsilon \cdot n$-fools B; Or*
2. *Outputs an OBP T of length at most n and width w that is an $\varepsilon/2$-next-bit predictor for G.*

The main idea behind this proof has appeared before for different purposes [Nis93, CH22, GRZ23], and in fact (modifications of) all these results can be used to prove Lemma 3.2. However, we give a self-contained proof.

To prove Lemma 3.2, we first define a series of potential distinguishers, with the property that each can be evaluated in logspace. Each distinguisher measures the bias of the next bit in the PRG upon reaching a particular state.

Definition 3.3. Given an OBP B of length n, for every $i < n$ and $v \in V_i$, let $N_v : \{0,1\}^{i+1} \to \{-1, 0, 1\}$ be the function defined as:

$$N_v(x) = \begin{cases} 1 & \text{if } B_{\to v}(x) = 1 \text{ and } x_{i+1} = 1 \\ -1 & \text{if } B_{\to v}(x) = 1 \text{ and } x_{i+1} = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, N_v is computable in logspace given B and v.

When x is uniformly random, $B_{\to v}(x)$ and x_{i+1} are independent, and therefore $\mathbb{E}[N_v] = 0$ for all v. Consequentially, our verifier checks that $|\mathbb{E}_G[N_v]|$ is small for all v, where we feed the first $i+1$ bits of the PRG output to N_v. We first show its soundness:

Lemma 3.4. *Given an OBP B of length n, suppose that for every i, $\sum_{v \in V_i} |\mathbb{E}_G[N_v]| \le \varepsilon$. Then G $\varepsilon \cdot n$-fools B.*

Proof. As every edge from layer V_i goes into layer V_{i+1}, for every $i < n$ we have

$$\begin{aligned} &\sum_{v \in V_{i+1}} \left| \mathbb{E}[B_{\to v}] - \mathop{\mathbb{E}}_{G}[B_{\to v}] \right| \\ &\le \sum_{v \in V_i} \sum_{b \in \{0,1\}} \left| \Pr_{x \leftarrow U_n}[B_{\to v}(x) = 1 \wedge x_{i+1} = b] - \Pr_{x \leftarrow G(U_s)}[B_{\to v}(x) = 1 \wedge x_{i+1} = b] \right|. \end{aligned}$$

Notice that by the definition of N_v, we have

$$\begin{aligned} \mathbb{E}[N_v] &= \Pr_{x \leftarrow U_n}[B_{\to v}(x) = 1 \wedge x_{i+1} = 1] - \Pr_{x \leftarrow U_n}[B_{\to v}(x) = 1 \wedge x_{i+1} = 0] \\ &= 2 \Pr_{x \leftarrow U_n}[B_{\to v}(x) = 1 \wedge x_{i+1} = 1] - \mathbb{E}[B_{\to v}] \\ &= \mathbb{E}[B_{\to v}] - 2 \Pr_{x \leftarrow U_n}[B_{\to v}(x) = 1 \wedge x_{i+1} = 0], \end{aligned}$$

and the above holds similarly under pseudorandomness generated by G. Therefore we further have

$$\begin{aligned} \sum_{v \in V_{i+1}} \left| \mathbb{E}[B_{\to v}] - \mathop{\mathbb{E}}_{G}[B_{\to v}] \right| &\le \sum_{v \in V_i} \left| \mathbb{E}[B_{\to v}] - \mathop{\mathbb{E}}_{G}[B_{\to v}] \right| + \sum_{v \in V_i} \left| \mathbb{E}[N_v] - \mathop{\mathbb{E}}_{G}[N_v] \right| \\ &= \sum_{v \in V_i} \left| \mathbb{E}[B_{\to v}] - \mathop{\mathbb{E}}_{G}[B_{\to v}] \right| + \sum_{v \in V_i} \left| \mathop{\mathbb{E}}_{G}[N_v] \right|. \end{aligned}$$

With the assumption that $\sum_{v \in V_i} |\mathbb{E}_G[N_v]| \le \varepsilon$ and the fact that $\mathbb{E}[B_{\to v_0}] = \mathbb{E}_G[B_{\to v_0}] = 1$, we conclude that $\sum_{v \in V_n} |\mathbb{E}[B_{\to v}] - \mathbb{E}_G[B_{\to v}]| \le \varepsilon \cdot n$. As the output labels are binary, this means that $|\mathbb{E}[B] - \mathbb{E}_G[B]| \le \varepsilon \cdot n$, i.e. G $\varepsilon \cdot n$-fools B. □

Proof of Lemma 3.2. For every $i < n$, the algorithm iterates through every $v \in V_i$ and all the possible seeds for G, computes $\sum_{v \in V_i} |\mathbb{E}_G[N_v]|$ and checks if it is at most ε. This can be done in space $O(s + \log(nw))$. If all such checks pass, we have by Lemma 3.4 that G $\varepsilon \cdot n$-fools B.

Otherwise, we find some $i < n$ such that $\sum_{v \in V_i} |\mathbb{E}_G[N_v]| > \varepsilon$. Let T be an OBP of length i that is the same as B from layer V_0 to V_i, such that the output label on each $v \in V_i$ is 1 if $\mathbb{E}_G[N_v] \ge 0$,

and 0 if $\mathbb{E}_G[N_v] < 0$. Such an OBP is of size at most that of B, and can be constructed in space $O(s + \log(nw))$. We have

$$\begin{aligned}
&\Pr_{x \leftarrow G(U_s)}[T(x_{1..i}) = x_{i+1}] \\
&\quad = \sum_{\substack{v \in V_i \\ \mathbb{E}_G[N_v] \geq 0}} \Pr_{x \leftarrow G(U_s)}[B_{\to v}(x) = 1 \wedge x_{i+1} = 1] + \sum_{\substack{v \in V_i \\ \mathbb{E}_G[N_v] < 0}} \Pr_{x \leftarrow G(U_s)}[B_{\to v}(x) = 1 \wedge x_{i+1} = 0] \\
&\quad = \sum_{v \in V_i} \frac{1}{2} \left(\mathop{\mathbb{E}}_G[B_{\to v}] + \left| \mathop{\mathbb{E}}_G[N_v] \right| \right) \\
&\quad > \frac{1}{2}(1 + \varepsilon).
\end{aligned}$$

□

3.2 Refutable Hardness Assumptions in Logspace

Lemma 3.2 shows that, given an alleged PRG for logspace, we can use it to either successfully derandomize a logspace comptation, or explicitly output a counterexample to the PRG. The results of the hardness-versus-randomness paradigm claim that PRGs exist under certain hardness assumptions. Combining these results with Lemma 3.2, we can derandomize logspace computations given any alleged hard function, or determine that the hardness assumption does not hold. However, Theorem 1.1 requires a stronger guarantee from the algorithm - if the hardness assumption does not hold, the algorithm needs to output a small circuit that falsifies this assumption. Obtaining this result is the primary contribution of Section 4.

We first recall the result of Klivans and van Melkebeek [KvM02].

Theorem 3.5 ([KvM02])**.** *If there is a family of boolean functions $f \in \mathsf{SPACE}[n]$ that is not computable by circuits of size $2^{\varepsilon n}$ for some $\varepsilon > 0$, then $\mathsf{BPL} = \mathsf{L}$.*

Their proof is based on the worst-case hardness vs. randomness results by Imagliazzo and Wigderson [IW97], and shows how every step in the construction of the Imagliazzo-Wigderson PRG can be executed in deterministic logspace. However, their proof (and all other proofs of the hardness vs. randomness paradigm) does not show that given a branching program (or circuit) that distinguishes the PRG from random (i.e. contradicts the original hardness assumption), there is an efficient deterministic logspace algorithm to produce a circuit for the supposedly hard function. This is for two reasons. First, the conversion from a distinguisher to a next bit predictor (which we address in Lemma 3.2). Even once we obtain such a predictor, prior approaches used space- and randomness-inefficient probabilistic method arguments to go from a predictor to a worst-case correct circuit for the original function. Our primary contribution in Section 4 is to carefully design the PRG and develop an efficient *reconstruction* procedure, given a distinguisher for the constructed PRG.

This leads to the following theorem:

Theorem 3.6. *For every family of boolean functions $f \in \mathsf{SPACE}[n]$ and $\varepsilon > 0$, there is a deterministic algorithm that, given as the input an OBP B of length n and width $w = n$, runs in space $O(\log n)$, and either*

1. *Outputs $\mathbb{E}[B]$ with $1/4$ error; Or*
2. *Outputs a circuit $\mathcal{C}$ of size $2^{\varepsilon m}$ that computes f on $\{0,1\}^m$ where $m = \Theta(\log n)$.*

Proof. Let $G : \{0,1\}^s \to \{0,1\}^n$ be the generator of Theorem 4.1 with $\varepsilon = \varepsilon$ and $f = f$ and let $m = m_0$ be the instance size of f used to construct G.

We then apply Lemma 3.2 on B and G with $\varepsilon = 1/(4n)$. Of the two possible outcomes:

1. If it is certified that G $\varepsilon \cdot n$-fools B. In this case the algorithm computes and outputs $\mathbb{E}_G[B]$ which approximates $\mathbb{E}[B]$ within additive error $1/4$.

2. Otherwise we get for some $i < n$ an explicit OBP T of length i and width w, such that $\Pr_{x \leftarrow U_s}[T(G(x)_{1..i}) = G(x)_{i+1}] > \frac{1}{2}(1 + \varepsilon)$. In other words, T is an $\varepsilon/2 = 1/8n$ next-bit predictor against G of size at most n^2, and T can be evaluated in space $O(\log n)$. Then by Theorem 4.1, we can construct in space $O(\log n)$ a circuit $\mathcal{C}$ for f on inputs of size $m = \Theta(\log n)$ of size at most $2^{\varepsilon m}$. □

Now Theorem 1.1 follows:

Proof of Theorem 1.1. Given a randomized logspace algorithm R with error probability at most $1/10$ and input $x \in \{0,1\}^n$, let B be the branching program representing how R uses its random bits on input x, which can be constructed in logspace. By assumption R uses $s = O(\log n)$ bits of space and hence at most $2^{s+O(1)} = \mathrm{poly}(n)$ random coins, and hence B has length and width $\mathrm{poly}(n)$. Pad B to have length and width n^c and apply Theorem 3.6 with $f = \mathcal{L}$ and $\varepsilon := \varepsilon$. Then we either obtain an estimate of $\mathbb{E}[B]$ up to $\pm 1/4$ (which suffices to decide the language by correctness of R) or a counterexample (in the form of a circuit of size at most $2^{\varepsilon n'}$ to the hardness of $\mathcal{L}$ on inputs of size $n' = \Theta(\log n^c) = \Theta(\log n)$ bits, as desired. □

We prove Theorem 4.1 in the following section.

4 Efficient Reconstructive Derandomization

We first state our main theorem of this section.

Theorem 4.1. *Given $\varepsilon > 0$, and a family of functions $f_m : \{0,1\}^m \to \{0,1\} \in \mathsf{SPACE}[m]$, there is a family of explicit generators $G : \{0,1\}^s \to \{0,1\}^n$ with $s = O(\log n)$ computable in space $O(\log n)$, and a deterministic logspace algorithm that, given $n \in \mathbb{N}$ and a $1/(8n)$-next-bit predictor $\mathcal{B}$ for G of size at most n^2 which is evaluable in space $O(\log n)$, outputs a circuit $\mathcal{C}$ of size $2^{\varepsilon \cdot m_0}$ for f_{m_0} with $m_0 = \Theta(\log n)$.*

We prove this theorem in four stages. Following the framework of [IW97], we first assume that f is a (worst-case) hard function, and construct a PRG via hardness amplifications and the Nisan-Wigderson PRGs [NW94]. The detailed steps are slightly different from those in [IW97], and we adapt the following strategy:

(a) From f, construct (by low-degree extension) a function f' that is hard-on-average on a 0.99 fraction of inputs.

(b) From f', construct (by derandomized XOR Lemma) a function f'' (with multiple bits of output) that is hard-on-average on a $2^{-\Omega(m)}$ fraction of inputs.

(c) From f'', construct (by Goldreich-Levin) a function f''' with single-bit output that is hard-on-average on a $1/2 + 2^{-\Omega(m)}$ fraction of inputs.

(d) Use f''' to instantiate a Nisan-Wigderson pseudorandom generator $G : \{0,1\}^s \to \{0,1\}^n$ for $s = O(m)$.

We make sure that f', f'', f''' and G are all computable within $O(\log n)$ space.

Furthermore, we prove that every step can be made logspace reconstructive, in the sense that given a counterexample to the conclusion (i.e. a small circuit that obtains some advantage) we can produce a counterexample to the assumption in deterministic logspace. This requires modifying the standard reconstruction algorithms for the first three steps, all of which use randomness-inefficient applications of the probabilistic method. Over the next four subsections, we state and prove the necessary components of the reconstructive PRG, and in Section 4.6, combine these results to conclude Theorem 4.1.

4.1 Preliminaries

First, we recall some notation related to the advantage of circuits.

Definition 4.2. Given $f : \{0,1\}^n \to \{0,1\}^m$ and a circuit $\mathcal{C}$, let $\mathtt{SUC}(\mathcal{C}, f) = \Pr_{x \leftarrow U_n}[\mathcal{C}(x) = f(x)]$. For $m = 1$, let $\mathtt{ADV}(\mathcal{C}, f) = 2\mathtt{SUC}(\mathcal{C}, f) - 1$. Let $\mathtt{ADV}_s(f) = \max_{\mathcal{C}:|\mathcal{C}| \le s} \mathtt{ADV}(\mathcal{C}, f)$ and likewise for $\mathtt{SUC}_s(f)$.

We will repeatedly make use of an averaging sampler in order to make probabilistic method arguments randomness efficient. We first recall the definition of an averaging sampler, and then recall the classical result in [RVW01] that there exist highly efficient averaging samplers, even with exponentially small error.

Definition 4.3. Given $m \in \mathbb{N}$ and $\varepsilon, \delta > 0$, we say that $\mathtt{SAMP} : \{0,1\}^l \to (\{0,1\}^m)^t$ is a t-query (m, ε, δ)-**averaging sampler** with seed length l if for every $g : \{0,1\}^m \to [0,1]$ we have

$$\Pr_{q_1,\ldots,q_t \leftarrow \mathtt{SAMP}(U_l)} \left[\left| \mathop{\mathbb{E}}_{i \in [t]} [g(q_i)] - \mathbb{E}[g] \right| \le \varepsilon \right] \ge 1 - \delta.$$

Proposition 4.4 ([RVW01])**.** *Given $m \in \mathbb{N}$ and $\varepsilon > 0$, there exists $t = \mathrm{poly}(m/\varepsilon)$ and a t-query $(m, \varepsilon, 2^{-2m})$-averaging sampler with seed length $4m$. Moreover, the sampler is evaluable in space $O(m)$.*

Another tool that is repeatedly used in our proof is the combinatorial design, which is a family of subsets $S_1, \ldots, S_n \subseteq [s]$ such that $|S_i| = \alpha s$ for some constant $\alpha \in (0,1)$ and all $i \in [n]$, while $|S_i \cap S_j| \le 2\alpha^2 s$ for all $i \ne j$. The design will be used at two places: once in derandomized XOR Lemma (Section 4.3) and once in the Nisan-Wigderson PRG (Section 4.5). While the application in Section 4.3 only requires a linear-sized design, the application in Section 4.5 requires an exponential-sized design that is deterministically constructible in linear pace. The later was formally given in [KvM02], so we concurrently use it for both applications.

Proposition 4.5 ([KvM02])**.** *For every $\alpha \in (0,1)$, there is $\beta \in (0,1)$ such that for $s \in \mathbb{N}$ one can deterministically generate in space $O(s)$ a combinatorial design of size $n = 2^{\beta s}$ over $[s]$, that is, a family of subsets $S_1, \ldots, S_n \subseteq [s]$ such that $|S_i| = \alpha s$ and $|S_i \cap S_j| \le 2\alpha^2 s$ for all $1 \le i < j \le n$.*

4.2 Derandomizing the Polynomial Decoder

For step (a) in Theorem 4.1, we need to convert a worst-case hard function to one with constant average-case hardness.

Lemma 4.6. *Given $f : \{0,1\}^m \to \{0,1\}$, there is $g : \{0,1\}^{m'} \to \{0,1\}$ where $m' = \Theta(m)$ such that, for every circuit $\mathcal{B}$ such that $\mathtt{SUC}(\mathcal{B}, g) > 0.99$, there is a circuit $\mathcal{C}$ of size $m^{O(1)} \cdot |\mathcal{B}|$ such that*

$$\mathcal{C}(x) = f(x), \ \forall x \in \{0,1\}^m.$$

Moreover, when f is computable in space $O(m)$, g is also computable in space $O(m)$, and there is a deterministic $O(m)$-space algorithm that, given the circuit $\mathcal{B}$ which is evaluable in space $O(m)$, prints $\mathcal{C}$, and $\mathcal{C}$ is also evaluable in space $O(m)$.

The proof for Lemma 4.6 is inspired by [STV01], where we encode f through Reed-Muller codes and switch to boolean domain via Hadamard codes. However, since we only need the resulting function to be average-case hard on a constant fraction of inputs, the code can be directly decoded instead of list-decoded, and we derandomize the decoding procedure with samplers.

We need the following two facts. The first is a folklore fact on constructing low-degree extension, whose proof can be found at [GKR15, Proposition 2.2]:

Proposition 4.7. *Given a finite field $\mathbb{F}$ and a subset $H \subseteq \mathbb{F}$, and oracle access to a function $f : H^\ell \to \{0,1\}$, one can compute in space $O(\log |\mathbb{F}| + \log \ell)$ an ℓ-variable polynomial $p : \mathbb{F}^\ell \to \mathbb{F}$ that coincides with f on H^ℓ, and the degree of p in each variable is smaller than $|H|$.*

The second fact concerns decoding Reed-Solomon codes:

Proposition 4.8. *Given a finite field $\mathbb{F}$ with $|\mathbb{F}| = N$, whose elements can be canonically listed as $a_1, \ldots, a_N$ where $a_1 = 0$, there exists a circuit $\mathtt{DEC} : \mathbb{F}^N \to \mathbb{F}^N$ that satisfies the following: If there exists a univariate polynomial $q : \mathbb{F} \to \mathbb{F}$ of degree at most $d < N$, such that $q(a_i) = b_i$ for at least $(N + d)/2$ of $i \in [N]$, then*

$$\mathtt{DEC}(b_1, \ldots, b_N) = (q(a_1), \ldots, q(a_N)).$$

Furthermore, $\mathtt{DEC}$ is of size $\mathrm{poly}(N)$ and depth $\mathrm{polylog}(N)$, and can be uniformly constructed in space $O(\log N)$ given the arithmetics in $\mathbb{F}$.

Proof. The circuit $\mathtt{DEC}$ instantiates the Berlekamp-Welch algorithm [WB86, GS92]. The algorithm involves solving systems of $O(N)$ linear equations on $O(N)$ variables, for which Csanky's algorithm [Csa76] can be implemented in logspace-uniform-**NC**. □

Proof of Lemma 4.6. We assume without loss of generality that m is a power of 2. Let $\ell = m/\log m$, and $\mathbb{F}$ be a finite field of characteristic 2 and size m^2. Take $H \subset \mathbb{F}$ to be a subset of size m, and we identify the domain $\{0,1\}^m$ of f with H^ℓ as $2^m = |H|^\ell$. The arithmetics in $\mathbb{F}$ can be done in time $O(|\mathbb{F}|)$ and space $O(m)$, and so does the bijection between $\{0,1\}^m$ and H^ℓ (and its reverse).

Let $p : \mathbb{F}^\ell \to \mathbb{F}$ be the polynomial in Proposition 4.7, and let $g : \mathbb{F}^{\ell+1} \to \{0,1\}$ be the function defined as

$$g(x_1, \ldots, x_\ell, y) = \langle p(x_1, \ldots, x_\ell), y \rangle,$$

where $\langle \cdot, \cdot \rangle$ stands for inner product in $\mathbb{F}_2$ when taking the binary representation of the two arguments in $\mathbb{F}$. It is clear that g can be computed in space $O(m)$, and the input of g has length $(\ell + 1) \log |\mathbb{F}| = O(m)$ when represented in binary.

Now assume there is a circuit $\mathcal{B}$ such that $\mathtt{SUC}(\mathcal{B}, g) > 0.99$. We first construct the circuit $\mathcal{B}' : \mathbb{F}^\ell \to \mathbb{F}$ such that the i-th bit of the output is

$$\mathcal{B}'_i(x_1, \ldots, x_\ell) = \mathtt{MAJ}_{z \in \mathbb{F}}(\mathcal{B}(x_1, \ldots, x_\ell, e_i + z) - \mathcal{B}(x_1, \ldots, x_\ell, z)).$$

Here e_i is the element in $\mathbb{F}$ whose binary representation has 1 on the i-th bit and 0 elsewhere.

Claim 4.9. $\mathtt{SUC}(\mathcal{B}', p) \geq 0.96$.

Proof. Since $\mathtt{SUC}(\mathcal{B}, g) > 0.99$, there are at least a 0.96-fraction of $(x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$ such that $\mathcal{B}$ coincide with g on more than $3/4$ of $y \in \mathbb{F}$, which contains both z and $(e_i + z)$ with probability larger than $1/2$ for a random $z \in \mathbb{F}$. In such cases we have $\mathcal{B}'_i(x_1, \ldots, x_\ell) = \langle p(x_1, \ldots, x_\ell), e_i \rangle$ for every i, and thus $\mathcal{B}'(x_1, \ldots, x_\ell) = p(x_1, \ldots, x_\ell)$. □

From $\mathcal{B}'$, we reconstruct the circuit $\mathcal{C} : \{0,1\}^m \to \{0,1\}$ as follows. Let $\mathtt{SAMP} : \{0,1\}^{8m} \to (\mathbb{F}^\ell)^t$ be the sampler in Proposition 4.4 with $\varepsilon = 0.01$ and thus $t = \mathrm{poly}(m)$. We think of $\mathtt{SAMP}$ as sampling t random vectors $v = (v_1, \ldots, v_\ell) \in \mathbb{F}^\ell$, and given the input $x = (x_1, \ldots, x_\ell) \in H^\ell$ for $\mathcal{C}$, each vector v represents a line $\{x + \lambda v \mid \lambda \in \mathbb{F}\}$. On each line, $p(x + \lambda v)$ is a univariate polynomial on λ of degree at most $\ell|H| = m^2/\log m$, and we use the decoder circuit $\mathtt{DEC}$ in Proposition 4.8 to decode the Reed-Solomon code given by $\mathcal{B}'$ on the line. We let the value of $\mathcal{C}(x)$ to be the most common (breaking ties arbitrarily) decoded value among the t lines. Notice that this process depends on the seed of the sampler, and we actually go through all the seeds and choose the one that makes $\mathcal{C}(x)$ correctly compute f on all $x \in H^\ell$.

Formally, we present this linear space reconstruction algorithm as Algorithm 1.

Algorithm 1: RM_RECON$(f, \mathcal{B})$

Let $\mathtt{SAMP} : \{0,1\}^{8m} \to (\mathbb{F}^\ell)^t$ be the sampler of Proposition 4.4 with $\varepsilon = 0.01$.
for $y \in \{0,1\}^{8m}$ **do**
  Let $v_1, \ldots, v_t \leftarrow \mathtt{SAMP}(y)$.
  Let $\mathcal{C} : \{0,1\}^m \to \{0,1\}$ be the circuit

$$\mathcal{C}(x) = \mathtt{MAJ}_{i \in [t]}(\mathtt{DEC}_1((\mathcal{B}'(x + \lambda v_i))_{\lambda \in \mathbb{F}})).$$

  if $\mathcal{C}(x) = f(x)$ *for all* $x \in \{0,1\}^m$ **then return** $\mathcal{C}$
end

The circuit $\mathcal{C}$ constructed in the algorithm is of size $2t|\mathbb{F}|^2|\mathcal{B}| + m^{O(1)} = m^{O(1)} \cdot |\mathcal{B}|$, and has additional depth $\mathrm{polylog}(m)$ compared to that of $\mathcal{B}$. Therefore $\mathcal{C}$ can be evaluated in space $O(m)$.

Now we prove that the algorithm always returns a valid circuit $\mathcal{C}$. Notice that for uniformly random $v \in \mathbb{F}^\ell$, $x + \lambda v$ is also uniformly random after given x and $\lambda \neq 0$. Since $\mathtt{SUC}(\mathcal{B}', p) \geq 0.96$, it means that there are at least a 0.84-fraction of $v \in \mathbb{F}^\ell$ such that $\mathcal{B}'$ coincide with p on $x + \lambda v$ for at least $3/4$ of $\lambda \in \mathbb{F}, \lambda \neq 0$. Recall that the degree of $q(\lambda) = p(x + \lambda v)$ is at most $\ell|H| = |\mathbb{F}|/\log m$, and therefore by Proposition 4.8 we conclude that for every $x \in \{0,1\}^\ell$,

$$\Pr_{v \in \mathbb{F}^\ell}\left[\mathtt{DEC}((\mathcal{B}'(x + \lambda v))_{\lambda \in \mathbb{F}}) = (p(x + \lambda v))_{\lambda \in \mathbb{F}}\right] \geq 0.84,$$

in which case we have $\mathtt{DEC}_1((\mathcal{B}'(x + \lambda v))_{\lambda \in \mathbb{F}}) = p(x)$. Viewing this probability as an expectation of the indicator function on v, by the guarantee of the sampler in Proposition 4.4 we have

$$\Pr_{v_1, \ldots, v_t \leftarrow \mathtt{SAMP}(U_{8m})}\left[\Pr_{i \in [t]}\left[\mathtt{DEC}_1((\mathcal{B}'(x + \lambda v_i))_{\lambda \in \mathbb{F}}) = p(x)\right] \geq 0.51\right] \geq 1 - 2^{-4m}.$$

By a union bound over $x \in \{0,1\}^m$, there must exist a $y \in \{0,1\}^{8m}$ such that $\mathcal{C}(x) = p(x) = f(x)$ for all $x \in \{0,1\}^m$. Therefore the algorithm always returns such a circuit $\mathcal{C}$. Moreover, the algorithm can be implemented to run in space $O(m)$, as we can enumerate over seeds to the sampler and construct the circuit (as a function of the sampler output) in space $O(m)$, and test if the circuit correctly computes f in this space bound. □

4.3 Derandomizing the Derandomized XOR Lemma

Our next step follows the approach of Impagliazo and Wigderson [IW97], who use a derandomized XOR lemma to produce from a function that is hard on a constant fraction of inputs, a function that is hard on any exponentially small fraction of inputs. The construction is identical to the one in [IW97], except that we modify the reconstruction algorithm and analysis to make the circuit $\mathcal{C}$ constructible in deterministic space $O(m)$.

Lemma 4.10. *For every $\gamma \in (0,1)$, there is an $O(m)$-space computable function $G : \{0,1\}^{m'} \to (\{0,1\}^m)^m$, where $m' = \Theta(m/\gamma)$, that satisfies the following: Given $f : \{0,1\}^m \to \{0,1\}$, and a circuit $\mathcal{B}$ satisfying $\mathtt{SUC}(\mathcal{B}, f^m \circ G) \geq 2^{-\gamma m}$, there exists a circuit $\mathcal{C}$ of size $2^{O(\gamma m)} \cdot |\mathcal{B}|$ such that*

$$\mathtt{SUC}(\mathcal{C}, f) > 0.99.$$

Moreover, when f is computable in space $O(m)$, there is a deterministic $O(m)$-space algorithm that, given the circuit $\mathcal{B}$ which is evaluable in space $O(m)$, prints $\mathcal{C}$, and $\mathcal{C}$ is also evaluable in space $O(m)$.

We first give the construction of the function G, which is called a *direct-product generator* in [IW97]. As in [IW97], it consists of two components: an expander walk and a combinatorial design. For the expander walk, we need an explicit expander where the neighbors of a vertex can be efficiently computed:

Proposition 4.11 (see e.g. [LPS88])**.** *There is a constant $\lambda \in (0,1)$, such that for every $m \in \mathbb{N}$, there exists a 4-regular graph E_m on the vertex set $\{0,1\}^m$ with spectral expansion (second largest eigenvalue of the normalized adjacency matrix) at most λ, such that given any vertex $v \in \{0,1\}^m$, its neighbors can be computed in time $\mathrm{poly}(m)$ and space $O(\log m)$.*

Define the expander walk function $\mathtt{EW} : \{0,1\}^{3m} \to (\{0,1\}^m)^m$ as follows: Given the input $v \in \{0,1\}^m$ and $d = (d_1, \ldots, d_m) \in [4]^m$, the output is sequence of vertices $v_1, \ldots, v_m$ in E_m that starts with $v_1 = v$, and take v_{i+1} to be the d_i-th neighbor of v_i. On the other hand, let $S_1, \ldots, S_m \subseteq [s]$ be the first m sets in the combinatorial design from Proposition 4.5 with $\alpha = \gamma/2$ and $s = m/\alpha$. Then we defined the function $G : \{0,1\}^{3m+s} \to (\{0,1\}^m)^m$ as:

$$G(r, v, d) = \big((r|_{S_1}) \oplus \mathtt{EW}(v,d)_1, \ldots, (r|_{S_m}) \oplus \mathtt{EW}(v,d)_m\big).$$

Here $r|_S$ is the part of $r \in \{0,1\}^s$ on indices S, and $\oplus$ is bit-wise XOR. From the definition we have that G can be computed in time $\mathrm{poly}(m)$ and space $O(m)$. The input length of G is $m' = 3m + 2m/\gamma = O(m/\gamma)$.

Now given $f : \{0,1\}^m \to \{0,1\}$, assume there is a circuit $\mathcal{B}$ such that $\mathtt{SUC}(\mathcal{B}, f^m \circ G) \geq 2^{-\gamma m}$. Before we move on and show how to reconstruct the circuit $\mathcal{C}$ efficiently and deterministically from $\mathcal{B}$, let us first review the reconstruction step in [IW97]. For $i \in [m]$, $x \in \{0,1\}^m$, $a \in \{0,1\}^{s-m}$, $v \in \{0,1\}^m$ and $d \in [4]^m$, let $h(i,x,a,v,d) = (r,v,d)$ where $r \in \{0,1\}^s$ such that

$$r|_{S_i} = x \oplus \mathtt{EW}(v,d)_i \text{ and } r|_{\overline{S_i}} = a.$$

The function h is called the *restricting function* of G. Given $x \in \{0,1\}^m$, with i, a, v and d chosen uniformly at random, they build a circuit $\mathcal{F}$ that first simulates $\mathcal{B}$ to compute $\mathcal{B}(h(i,x,a,v,d)) = (y_1, \ldots, y_m)$. Then it computes a number t defined as

$$t = \left|\left\{j \neq i \mid y_j \neq f\big(G_j \circ h(i,x,a,v,d)\big)\right\}\right|,$$

and outputs y_i with probability 2^{-t}, while outputting a random bit with probability $1 - 2^{-t}$. To compute t, for each $j \neq i$, $f(G_j \circ h(i, x, a, v, d))$ is computed through a non-uniformly constructed look-up table for f of size $2^{\gamma m}$, containing the values of $f(x_j)$ for all possible j-th output x_j of $G \circ h$ with the fixed i, a, v and d.

We could not resort to non-uniformity to construct the look-up table. Nevertheless, when f is computable in space $O(m)$, we can compute the entire table in space $O(m)$ and hardwire it to the circuit. Even better, when i, a, v and d are given, each output x_j of $G \circ h$ is fixed except for γm bits (corresponding to the coordinates in $S_i \cap S_j$), so we only need to go through all $2^{\gamma m}$ possibilities for these bits to compute the table.

The circuit $\mathcal{F}$ presented above uses a string R of $|R| = O(m)$ random bits, including i, a, v, d along with $w \in \{0,1\}^{m+1}$, the randomness used to decide the final output. It was proved in [IW97] that:

Proposition 4.12 ([IW97, Theorem 15])**.** *Suppose that* $\mathtt{SUC}(\mathcal{B}, f^m \circ G) \geq 2^{-\gamma m}$*. There exists* $c > 0$ *(that depends on* γ*), such that the fraction of inputs* $x \in \{0,1\}^m$ *with*

$$\Pr_R[\mathcal{F}(x, R) = f(x)] \geq 1/2 + 2^{-\gamma m}/c$$

is more than 0.99.

Therefore, the final circuit $\mathcal{C}$ takes $O(m \cdot 2^{2\gamma m})$ independent copies of $\mathcal{F}$ and outputs their majority, and there exists a fixing of the randomness that provides the final deterministic circuit $\mathcal{C}$. We could not afford to store exponentially many random bits if they are independently sampled. Instead, we employ the efficient sampler in Proposition 4.4 that uses only $O(m)$ random bits as the seed to generate $2^{O(\gamma m)}$ samples, and we can enumerate over all the seeds to find the one that makes $\mathtt{SUC}(\mathcal{C}, f) > 0.99$. As shown in the proof below, such seed always exists.

Proof of Lemma 4.10. Let $\mathcal{F} : \{0,1\}^{m+|R|} \to \{0,1\}$ be the circuit described above, and $c > 0$ be the constant in Proposition 4.12. We give the formal description of the linear-space algorithm for the reconstruction procedure as Algorithm 2.

Algorithm 2: XOR_RECON$(f, \mathcal{B})$

Let $\mathtt{SAMP} : \{0,1\}^{4|R|} \to (\{0,1\}^{|R|})^t$ be the sampler of Proposition 4.4 with $\varepsilon = 2^{-\gamma m}/(2c)$.
for $y \in \{0,1\}^{4|R|}$ **do**
  Let $R_1, \ldots, R_t \leftarrow \mathtt{SAMP}(y)$.
  Let $\mathcal{C} : \{0,1\}^m \to \{0,1\}$ be the circuit

$$\mathcal{C}(x) = \mathtt{MAJ}(\mathcal{F}(x, R_1), \ldots, \mathcal{F}(x, R_t)).$$

  if $\mathtt{SUC}(\mathcal{C}, f) > 0.99$ **then return** $\mathcal{C}$
end

By Proposition 4.4 we have $t = \mathrm{poly}(m/\varepsilon) = 2^{O(\gamma m)}$ for $\varepsilon = 2^{-\gamma m}/(2c)$. From the description we know that $\mathcal{F}$ has size $|\mathcal{B}| + 2^{\gamma m} \cdot m^{O(1)}$, and therefore $\mathcal{C}$ has size $t|\mathcal{F}| + m^{O(1)} = 2^{O(\gamma m)} \cdot |\mathcal{B}|$. When $\mathcal{B}$ is evaluable in space $O(m)$, $\mathcal{C}$ is clearly also evaluable in space $O(m)$.

By the guarantee of the averaging sampler $\mathtt{SAMP}$ in Proposition 4.4, for every $x \in \{0,1\}^m$:

$$\Pr_{R_1,\ldots,R_t \leftarrow \mathtt{SAMP}(U_{4|R|})} \left[\left| \mathop{\mathbb{E}}_{i \in [t]} [\mathcal{F}(x, R_i)] - \mathop{\mathbb{E}}_{R} [\mathcal{F}(x, R)] \right| \leq \varepsilon \right] \geq 1 - 2^{-2|R|}.$$

By Proposition 4.12, there exists a subset $V \subseteq \{0,1\}^m$ such that $|V| > 0.99 \cdot 2^m$, such that for every $x \in V$:

$$\left| \mathop{\mathbb{E}}_{R}[\mathcal{F}(x,R)] - f(x) \right| \leq 1/2 - 2^{-\gamma m}/c = 1/2 - 2\varepsilon.$$

Therefore for every $x \in V$, it is implied that

$$\Pr_{R_1,\ldots,R_t \leftarrow \mathtt{SAMP}(U_{4|R|})} \left[\left| \mathop{\mathbb{E}}_{i \in [t]}[\mathcal{F}(x,R_i)] - f(x) \right| \leq 1/2 - 2\varepsilon + \varepsilon \right] \geq 1 - 2^{-2|R|},$$

which means that

$$\Pr_{R_1,\ldots,R_t \leftarrow \mathtt{SAMP}(U_{4|R|})} [\mathtt{MAJ}(\mathcal{F}(x,R_1),\ldots,\mathcal{F}(x,R_t)) = f(x)] \geq 1 - 2^{-2|R|} > 1 - 1/|V|.$$

By a union bound over $x \in V$, there must exist a $y \in \{0,1\}^{4|R|}$ such that $\mathcal{C}(x) = f(x)$ for all $x \in V$, which satisfies $\mathtt{SUC}(\mathcal{C}, f) > 0.99$. Therefore the algorithm always returns a valid $\mathcal{C}$. Moreover, the algorithm runs in space $O(m)$, as it enumerates the seeds of length $O(|R|) = O(m)$, constructs and evaluates the circuit $\mathcal{C}$ and makes oracle calls to f, which all can be done in space $O(m)$. □

4.4 Derandomizing the Goldreich-Levin Theorem

Lemma 4.13. *Given $f : \{0,1\}^m \to \{0,1\}^m$, let $g : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ be defined as $g(x,r) = \langle f(x), r \rangle$. Then, given $\delta > 0$, there is $\delta' \geq \Omega(\delta^3/m)$ so that, for every $\mathcal{B}$ satisfying $\mathtt{ADV}(\mathcal{B}, g) > \delta$, there is a circuit $\mathcal{C}$ of size at most $|\mathcal{B}| \cdot (m/\delta)^{O(1)}$ satisfying*

$$\mathtt{SUC}(\mathcal{C}, f) > \delta'.$$

Moreover, when f is computable in space $O(m)$, there is a deterministic $O(m)$-space algorithm that, given the circuit $\mathcal{B}$ which is evaluable in space $O(m)$, prints $\mathcal{C}$, and $\mathcal{C}$ is also evaluable in space $O(m)$.

Note that the original Goldreich-Levin theorem [GL89] does not guarantee (and in fact does not give) an efficient deterministic reconstructor, as it is not randomness efficient. A later work of Hoza and Klivans [HK18] achieves this, though with a significantly more involved proof. As such, we directly show this using small-bias spaces, which we define now:

Definition 4.14. A function $G : \{0,1\}^t \to \{0,1\}^k$ is an **ε-biased generator** if $G(U_t)$ is a ε-biased probability space over $\{0,1\}^k$, which formally means that for every $T \in \{0,1\}^k$,

$$\Pr_{y \leftarrow U_t} [\langle T, G(y) \rangle = 1] \in [1/2 - \varepsilon, 1/2 + \varepsilon].$$

We recall that small-bias generators exist with good seed length, and moreover these generators can be evaluated in small space:

Proposition 4.15 ([NN93])**.** *Given $k \in \mathbb{N}$ and $\varepsilon > 0$, there is an $O(t)$-space evaluable ε-biased generator $\mathtt{BIAS} : \{0,1\}^t \to \{0,1\}^k$ with seed length $t = O(\log(k/\varepsilon))$.*

We require a basic Fourier-analytic lemma, that states that a small-bias space fools the conjunction of k parities.

Lemma 4.16. *Let* $\mathtt{BIAS} : \{0,1\}^t \to \{0,1\}^k$ *be an* ε*-biased generator. Then for every collection* $T_1, \ldots, T_d \in \{0,1\}^k$ *and* $v_1, \ldots, v_d \in \{0,1\}$ *we have*

$$\left| \mathop{\mathbb{E}}_{r \leftarrow \mathtt{BIAS}(U_t)} \left[\bigwedge_{i \in [d]} (\langle T_i, r \rangle \oplus v_i) \right] - \mathop{\mathbb{E}}_{r \leftarrow U_k} \left[\bigwedge_{i \in [d]} (\langle T_i, r \rangle \oplus v_i) \right] \right| \leq 2\varepsilon.$$

Proof. We have

$$\begin{aligned} \bigwedge_{i \in [d]} (\langle T_i, r \rangle \oplus v_i) &= 1 - 2 \cdot 2^{-d} \sum_{S \subseteq [d]} \bigoplus_{i \in S} \neg (\langle T_i, r \rangle \oplus v_i) \\ &= 1 - 2 \cdot 2^{-d} \sum_{S \subseteq [d]} \left(\left\langle \bigoplus_{i \in S} T_i, r \right\rangle \oplus \bigoplus_{i \in S} \neg v_i \right) \end{aligned}$$

and as $\mathtt{BIAS}$ fools all such parities to error ε in the summation over $S \subseteq [d]$, we have that the total error is at most 2ε. □

Proof of Lemma 4.13. If $\delta < 2^{-m}$, we can choose $\delta' = 2^{-m}$ and the lemma trivially holds for a circuit $\mathcal{C}$ outputting a constant. Therefore, from now on we assume that $\delta \geq 2^{-m}$. We formally state our algorithm as Algorithm 3, with δ' to be determined later. Note that $\ell = O(m)$, and therefore in the ε-biased generator $\mathtt{BIAS} : \{0,1\}^t \to \{0,1\}^{\ell \times m}$ we have $t = O(\log(\ell m/\varepsilon)) = O(m)$ with $\varepsilon = 2^{-4m-1}$, and the algorithm runs in space $O(t + \ell + m) = O(m)$.

Algorithm 3: GL_RECON(f, B)

Let $\ell \leftarrow \lceil \log_2(128m/\delta^2 + 1) \rceil$.
Let $\mathtt{BIAS} : \{0,1\}^t \to \{0,1\}^{\ell \times m}$ be the generator of Proposition 4.15 with $\varepsilon = 2^{-4m-1}$.
for $y \in \{0,1\}^t$ **do**
  Let $r_1, \ldots, r_\ell \leftarrow \mathtt{BIAS}(y)$.
  for $(b_1, \ldots, b_\ell) \in \{0,1\}^\ell$ **do**
    Let $\mathcal{C} : \{0,1\}^m \to \{0,1\}^m$ be the circuit that for each $i \in [m]$:

$$\mathcal{C}_i(x) = \mathtt{MAJ}_{J \subseteq [\ell] : J \neq \emptyset}(b^J \oplus \mathcal{B}(x, r^J \oplus e_i)).$$

    if $\mathtt{SUC}(\mathcal{C}, f) > \delta'$ **then return** $\mathcal{C}$.
  end
end

We view the output of $\mathtt{BIAS}$ as a tuple of ℓ vectors:

$$\mathtt{BIAS}(y) = (r_1, \ldots, r_\ell), \quad r_i \in \{0,1\}^m.$$

For convenience, let $\vec{r} := (r_1, \ldots, r_\ell)$ and $\vec{b} := (b_1, \ldots, b_\ell)$. For every $J \subseteq [\ell]$, let:

$$r^J := \bigoplus_{i \in J} r_i, \qquad b^J = \bigoplus_{i \in J} b_i.$$

Note that in the original GL algorithm, all r_i's are i.i.d. uniformly over $\{0,1\}^m$. We first argue that our distribution over r^J's satisfies (approximately) the two properties used in the analysis of the original algorithm:

Claim 4.17. *The following two properties hold:*

1. *For every non-empty J, r^J is 2^{-2m}-close to U_m in ℓ_1-distance.*
2. *For every non-empty J and J' where $J \neq J'$, $(r^J, r^{J'})$ is 2^{-2m} close to U_{2m} in ℓ_1-distance.*

Proof. For $i \in [m]$, the i-th bit of r^J can be written as $\langle T_{i,J}, \texttt{BIAS}(y)\rangle$ where $T_{i,J}$ indicates a non-empty subset of bits. From Lemma 4.16 we know that for every $v \in \{0,1\}^m$,

$$\left| \Pr_{r \leftarrow \texttt{BIAS}(U_t)}[r^J = v] - \Pr_{r \leftarrow U_{\ell m}}\left[\bigwedge_{i\in[m]} (\langle T_{i,J}, r\rangle = v_i)\right]\right| \leq 2\varepsilon.$$

Notice that $\{T_{i,J}\}_{i\in[m]}$ are linearly independent, and thus $(\langle T_{i,J}, r\rangle)_{i\in[m]}$ is uniformly distributed over $\{0,1\}^m$. Therefore taking the sum over $v \in \{0,1\}^m$ we have that r^J is $2\varepsilon \cdot 2^m \leq 2^{-2m}$-close to U_m in ℓ_1 distance.

When $J \neq J'$ are both non-empty, $\{T_{i,J}\}_{i\in[m]} \cup \{T_{i,J'}\}_{i\in[m]}$ are still linearly independent. For the same reason as above, $(r^J, r^{J'})$ is $2\varepsilon \cdot 2^{2m} = 2^{-2m}$-close to U_{2m} in ℓ_1 distance. □

Now recall that for $i \in [m]$ the ith bit of the output of $\mathcal{C}$ is

$$\mathcal{C}_i(x) = \texttt{MAJ}_{J:J\neq\emptyset}(b^J \oplus \mathcal{B}(x, r^J \oplus e_i)).$$

Thus $\mathcal{C}$ has size $|\mathcal{C}| \leq (|\mathcal{B}| + O(\ell)) \cdot 2^\ell m \leq |\mathcal{B}| \cdot O(2^\ell \ell m) = |\mathcal{B}| \cdot (m/\delta)^{O(1)}$ as claimed. To analyze the performance of $\mathcal{C}$, let

$$S := \{x \in \{0,1\}^m : \Pr_{z \leftarrow U_m}[\mathcal{B}(x,z) = g(x,z)] \geq 1/2 + \delta/2\}.$$

By a standard averaging argument, $|S| \geq (\delta/2) \cdot 2^m$.

Claim 4.18. *For every $x \in S$ and $i \in [m]$,*

$$\Pr_{(r_1,\ldots,r_\ell)\leftarrow\texttt{BIAS}(U_t)}\left[\left|\{J : \mathcal{B}(x, r^J \oplus e_i) = g(x, r^J \oplus e_i)\}\right| \leq \frac{1}{2}(2^\ell - 1)\right] \leq \frac{1}{2m}.$$

Proof. For the remainder of the proof we fix x and i. Let $A \subset \{0,1\}^m$ be the set of values r on which $\mathcal{B}(x,r) = g(x,r)$. By the fact that $x \in S$ we have $|A| \geq (1/2 + \delta/2) \cdot 2^m$. Furthermore, for each $y \in \{0,1\}^t$ (where y is the input to $\texttt{BIAS}$) let

$$\zeta_J(y) = \mathbb{I}[r^J \oplus e_i \in A]$$

and observe that $\zeta_J = 1$ is equivalent to $\mathcal{B}(x, r^J \oplus e_i) = g(x, r^J \oplus e_i)$, i.e. $\mathcal{B}$ computes the inner product with $f(x)$ correctly on that input. Now observe that by Claim 4.17,

$$\mathop{\mathbb{E}}_y[\zeta_J] = \Pr_y[\zeta_J(y) = 1] \geq 1/2 + \delta/2 - 2^{-2m} \geq 1/2 + \delta/4.$$

We now bound the variance of the number of such places where we compute the inner product correctly. Let

$$\begin{aligned}\sigma^2 = \mathrm{Var}\left(\sum_J \zeta_J\right) &= \sum_{J,J'} \mathrm{Cov}(\zeta_J, \zeta_{J'}) \\ &\leq \sum_J \mathrm{Var}(\zeta_J) + \sum_{J,J'} 2^{-2m} \\ &\leq 2^\ell + 2^{2\ell} \cdot 2^{-2m} \leq 2^{\ell+1}\end{aligned}$$

where the first inequality follows from Claim 4.17. Now the result follows by Chebyshev's inequality and a union bound. For convenience let $d = 2^\ell - 1$, and the probability in the claim equals:

$$\begin{aligned}\Pr_y\left[\sum_J \zeta_J \le \frac{d}{2}\right] &\le \Pr_y\left[\left|\sum_J \zeta_J - \mathbb{E}[\zeta_J]\cdot d\right| \ge \left(\frac{d\delta}{4\sigma}\right)\cdot\sigma\right] \\ &\le \frac{16\sigma^2}{\delta^2(2^\ell-1)^2} \le \frac{32\sigma^2}{\delta^2 2^{2\ell}} \le \frac{64}{\delta^2 2^\ell} \le \frac{1}{2m}.\end{aligned}$$

□

Notice that when $\mathcal{B}(x, r^J \oplus e_i) = g(x, r^J \oplus e_i)$ and for every $j \in [\ell]$, $b_j = g(x, r_j)$, we have

$$b^J \oplus \mathcal{B}(x, r^J \oplus e_i) = g(x, r^J) \oplus g(x, r^J \oplus e_i) = g(x, e_i) = f_i(x).$$

Thus, using a union bound over $i \in [m]$ on Claim 4.18, we have that for every $x \in S$,

$$\begin{aligned}\Pr_{\substack{\vec{r}\leftarrow \mathtt{BIAS}(U_t)\\ \vec{b}\leftarrow U_l}}[\mathcal{C}(x) = f(x)] \ge\ & \Pr_{\vec{r}\leftarrow \mathtt{BIAS}(U_t)}\left[\forall i\in[m], \left|\{J : \mathcal{B}(x, r^J\oplus e_i) = g(x, r^J\oplus e_i)\}\right| > \frac{1}{2}(2^l-1)\right] \\ & \cdot \Pr_{\vec{b}\leftarrow U_\ell}\left[\forall j\in[\ell], b_j = g(x, r_j)\right] \\ \ge\ & \frac{1}{2}\cdot \Pr_{\vec{b}\leftarrow U_\ell}\left[\forall j\in[\ell], b_j = g(x, r_j)\right] \ge 2^{-\ell-1}.\end{aligned}$$

Thus, there is an assignment of y and $\vec{b}$ such that $\mathcal{C}$ computes f correctly on at least $|S|\cdot 2^{-\ell-1} \ge 2^m \cdot \delta 2^{-\ell-2}$ inputs. Moreover, we can find such a circuit by enumerating the assignments to y and $\vec{b}$, and verifying the success probability by evaluating $\mathcal{C}$ and f over all $x \in \{0,1\}^m$. Therefore letting

$$\delta' = \delta 2^{-\ell-2} = \Omega(\delta^3/m)$$

completes the proof. □

4.5 Space-Efficient Nisan-Wigderson PRG

We recall the argument of [KvM02] that there is a space-efficient implementation of the Nisan-Wigderson [NW94] PRG, using the linear-space constructible combinatorial design (Proposition 4.5). While we rephrase their result in our notation, we make no changes to the construction, as (in contrast to all other steps) the existing implementation satisfies our desired reconstruction property.

Lemma 4.19. *Given $\rho > 0$ and $n \in \mathbb{N}$ and a family of functions $f_m : \{0,1\}^m \to \{0,1\} \in \mathsf{SPACE}[m]$, there exists an $m = \Theta(\log n)$ and $G : \{0,1\}^s \to \{0,1\}^n$ with $s = O(m)$ such that, given a circuit $\mathcal{B}$ which is a next-bit predictor for G with advantage ε, there is a circuit $\mathcal{C}$ of size $|\mathcal{B}| + O(n2^{\rho m})$ satisfying*

$$\mathtt{ADV}(\mathcal{C}, f_m) > \varepsilon.$$

Moreover, there is a deterministic $O(m)$-space algorithm that, given the circuit $\mathcal{B}$ which is evaluable in space $O(m)$, prints $\mathcal{C}$, and $\mathcal{C}$ is also evaluable in space $O(m)$.

Proof of Lemma 4.19. Fix $\alpha \in (0,1)$ such that $\alpha \le \rho/2$, and let $\beta \in (0,1)$ be the constant in Proposition 4.5. Choose $s = O(\log n)$ such that $2^{\beta s} = n$, and let $m = \alpha s$. Let $\mathcal{S} = (S_1, \ldots, S_n)$ be the design of Proposition 4.5 over $[s]$ with parameter α, and let $f_m : \{0,1\}^m \to \{0,1\}$ be the function on inputs of size $m = O(\log n)$.

We let $G(x) := f(x_{S_1})f(x_{S_2})\dots f(x_{S_n})$. Now suppose $\mathcal{B}$ is an ε-next-bit predictor for bit i of G, i.e.

$$\Pr_{x\leftarrow U_s}[\mathcal{B}(G(x)_{1..i}) = G(x)_{i+1}] > \frac{1}{2} + \varepsilon.$$

Then let $S := S_{i+1}$ and $T := [s] \setminus S_{i+1}$ and write the above inequality as

$$\Pr_{(x_S,x_T)\leftarrow U_s}[\mathcal{B}(G(x_S \cup x_T)_{1..i}) = f(x_S)] > \frac{1}{2} + \varepsilon.$$

For each fixing of x_T, we let the circuit $\mathcal{C}$ to be $\mathcal{C}(x_S) = \mathcal{B}(G(x_S \cup x_T)_{1..i})$. Then we have

$$\mathop{\mathbb{E}}_{x_T}[\mathtt{ADV}(\mathcal{C}, f_m)] > \varepsilon.$$

Thus, the algorithm can enumerate over all possible assignments to x_T in space $|T| = O(m)$, and for each assignment check the advantage of $\mathcal{C}$. Once the algorithm has found the fixing of x_T such that the restricted circuit has advantage at least ε, for every $j \le i$, the j-th bit of the output of $G(x_S \cup x_T)$, which is $f(x_{S_j})$, depends on $|S \cap S_j| \le 2\alpha^2 s = \rho m$ bits of x_S, and hence we can output a ($O(m)$-space constructible) circuit for $f(x_{S_j})$ of size at most $O(2^{\rho m})$, and hence the total size of $\mathcal{C}$ is at most $|\mathcal{B}| + O(n2^{\rho m})$. □

4.6 Putting It All Together

Proof of Theorem 4.1. Given ε, we first do the construction steps. For each $m \in \mathbb{N}$:

1. Let $f' : \{0,1\}^{m_1} \to \{0,1\}$ be the function g of Lemma 4.6 applied to f_m.
2. Let $f'' : \{0,1\}^{m_2} \to \{0,1\}^{m_1}$ be the function $f'^{m_1} \circ G$ of Lemma 4.10 applied to f'_{m_1} with the constant γ to be chosen later.
3. Let $f''' : \{0,1\}^{m_3} \to \{0,1\}$ be the function g of Lemma 4.13 applied to f''_{m_2} with the constant δ to be chosen later.
4. Let $G : \{0,1\}^s \to \{0,1\}^n$ be the function of Lemma 4.19 applied to f'''_{m_3} and $\mathcal{B}$ with the constant ρ to be chosen later.

Notice that m_1, m_2, m_3 and s are all $\Theta(m)$, and the functions f', f'', f''' and G are all computable in space $O(m)$.

Suppose now we are given a $1/(8n)$ next-bit predictor $\mathcal{B}$ for G of size n^2. As n is given, we decide the value of $m_3 = \Theta(\log n)$ through Lemma 4.19, which in turn decides the value of $m = \Theta(\log n)$. The reconstruction steps go as follows:

4. By Lemma 4.19, we can construct in space $O(m)$ a circuit $\mathcal{C}_3$ such that $\mathtt{ADV}(\mathcal{C}_3, f'''_{m_3}) > 1/(8n)$, and $\mathcal{C}_3$ has size $s_3 = n^2 + O(n2^{\rho m_3}) \le 2^{c_3\rho m}$ for some constant $c_3 > 0$.
3. By Lemma 4.13, where we now set $\delta = 1/(8n)$, we can construct in space $O(m)$ a circuit $\mathcal{C}_2$ such that $\mathtt{SUC}(\mathcal{C}_2, f''_{m_2}) > \Omega(\delta^3/m_2) \ge 2^{-c_2\rho m}$, and $\mathcal{C}_2$ has size $s_2 = s_3 \cdot (m_2/\delta)^{O(1)} \le 2^{c_2\rho m}$ for some constant $c_2 > 0$.
2. By Lemma 4.10, where we now set $\gamma = c_2\rho$, we can construct in space $O(m)$ a circuit $\mathcal{C}_1$ such that $\mathtt{SUC}(\mathcal{C}_1, f'_{m_1}) > 0.99$ and $\mathcal{C}_1$ has size $s_1 = s_2 \cdot 2^{O(\gamma m_1)} \le 2^{c_1\rho m}$ for some constant $c_1 > 0$.
1. By Lemma 4.6, we can construct in space $O(m)$ a circuit $\mathcal{C}$ such that $\mathcal{C}(x) = f_m(x)$ for every $x \in \{0,1\}^m$, and $\mathcal{C}$ has size $s = s_1 \cdot m^{O(1)} \le 2^{c_0\rho m}$ for some constant $c_0 > 0$. By choosing $\rho = \varepsilon/c_0$, we obtain the final result. □

5 Universal Derandomization of BPL

Here we state the main theorem of this section, that there exists a universal derandomizer for logspace computation.

Theorem 5.1. *There is a deterministic machine* UnivDerand *such that:*

- *On input* 1^n *and an OBP* B *of length and width at most* n*, outputs* $\delta := \text{UnivDerand}(1^n, B)$ *satisfying* $|\delta - \mathbb{E}[B]| < n^{-1}$*.*
- *For every space-constructible function* $S : \mathbb{N} \to \mathbb{N}$ *satisfying* $S(n) \geq \log n$*,* UnivDerand *runs in space* $O(S(n))$ *if and only if* $\mathsf{prBPL} \subseteq \mathsf{SPACE}[O(S(n))]$*.*

We first give the intuitive explanation of the algorithm executed by the machine UnivDerand. It enumerates over Turing machines $\langle i \rangle$ and space bounds j. At each step, UnivDerand runs $\langle i \rangle$ on input $(1^n, B_{\to v})$ for every v, where $B_{\to v}$ for $v \in V_i$ is the program that is identical to B in the first i layers, then accepts if the program reaches state v. If $\langle i \rangle$ ever touches more than j spaces on the work tape, UnivDerand halts and increments i or j. Otherwise, we have a set of estimates $\{\widetilde{p_{\to v}}\} := \{\langle i \rangle(1^n, B_{\to v})\}$ (and note that we can generate these estimates on the fly in space $O(j + \log n)$). We then submit these estimates to the local consistency test of Cheng and Hoza [CH22], and if the test passes, we return the estimate of the probability of reaching the accepting state.

Theorem 5.2 ([CH22])**.** *There is a deterministic logspace algorithm* LCTest *that takes as input* 1^n *and an OBP* B *with length and width at most* n *and the estimates* $\{\widetilde{p_{\to v}}\}_{v \in V}$*. If for every* v*,* $|\widetilde{p_{\to v}} - p_{\to v}| \leq n^{-3}$*, the algorithm accepts, and moreover if the algorithm accepts,* $|\widetilde{p_{\to v}} - p_{\to v}| \leq n^{-1}$ *for every* v*.*

Note that the true probabilities $p_{\to v}$ only appear in the statement of the theorem, and are not part of the input to the testing algorithm. We can now give the formal description of the algorithm as Algorithm 4. By soundness of the test LCTest, if UnivDerand returns a value, the value must be a good approximation of the acceptance probability, so it suffices to show this occurs (and occurs in the desired space bound).

Algorithm 4: $\text{UnivDerand}(1^n, B)$

```
for j ← 0, 1 ..., do
    for i ← 0, 1, ..., j do
        for r ← 1 · n^{-5}/2, 2 · n^{-5}/2, ..., 2n^2 · n^{-5}/2 do
            Compute b ← LCTest(1^n, B, {⟨i⟩(1^n, B_{→v}, r)}_{v∈V(B)});
            whenever ⟨i⟩ uses more than j space or more than 2^j time do
                Abort the simulation of ⟨i⟩ and pass to the next r.
            end
            if b = 1 then return ⟨i⟩(1^n, B, r).
        end
    end
end
```

To do so, we rely on the promise search problem f_c with parameter $c \in \mathbb{N}$ (which we define as a function outputting a value in $[0, 1]$ for convenience) defined as follows. Given 1^n, an ordered

branching program B of length and width at most n, and a rounding threshold r with the promise that

$$\left|\mathbb{E}[B] - k \cdot n^{-c+2} + r\right| > \frac{n^{-c}}{6},\ \forall k \in \mathbb{Z},$$

i.e. $\mathbb{E}[B] + r$ is polynomially bounded away from every multiple of n^{-c+2}, the problem asks to output a (pseudo-deterministic) number $f_c(1^n, B, r)$ that is within n^{-c+2} distance of $\mathbb{E}[B]$. The presence of the rounding value, inspired by the approach of Saks and Zhou [SZ99], is because when $\mathbb{E}[B]$ is very close to a threshold, it becomes hard to determining whether the expectation is above or below the cutoff.

We prove in Proposition 5.3 that the task of computing f_c is **promise-BPL** complete for every $c \geq 3$. Therefore, if **prBPL** $\subseteq$ **SPACE**$[S(n)]$, there is a machine $\langle i \rangle$ that computes f_c in space $j = O(S(n))$. Finally, to accommodate the presence of the rounding threshold, UnivDerand additionally enumerates over a polynomial number of choices for r. We show that there exists a proper $c \in \mathbb{N}$ such that for every B, a good r that satisfies the promise of f_c exists. This is essentially proved via the argument of Saks and Zhou [SZ99]. Hence, the algorithm will always find a tuple (i, j, r) such that we obtain good estimates of $\mathbb{E}[B_{\to v}]$ for every v, and thus the machine will halt and return the correct value.

Proposition 5.3. *For every $c \in \mathbb{N}$ with $c \geq 3$, let f_c be the problem where, given 1^n and an ordered branching program B of length and width at most n, and $r \in [0, 1]$ such that for every $k \in \mathbb{Z}$,*

$$\left|\mathbb{E}[B] - k \cdot n^{-c+2} + r\right| > \frac{n^{-c}}{6},$$

*return with probability at least $2/3$ the same number δ that satisfies $|\mathbb{E}[B] - \delta| \leq n^{-c+2}$. Then f_c is **prBPL**-complete under **L** reductions.*

Proof Sketch. Fix arbitrary $c \geq 3$. We first prove $f_c \in$ **prBPL**. Let $\mathcal{R}(1^n, B, r)$ be an algorithm that takes n^{2c+1} random walks from v_{st} over B, and let γ be the fraction of these walks which reach v_{acc}. Let $k \in \mathbb{Z}$ be the largest value such that $\gamma + r \geq k \cdot n^{-c+2}$, and return $\delta = k \cdot n^{-c+2}$. Since this algorithm clearly runs in randomized logspace, it suffices to show that, for B and r that satisfy the promise, there is some fixed k that $\mathcal{R}$ identifies with probability over $2/3$. Note that by the promise, we have that for some $k_0 \in \mathbb{Z}$,

$$k_0 \cdot n^{-c+2} + \frac{n^{-c}}{6} < \mathbb{E}[B] + r < (k_0 + 1) \cdot n^{-c+2} - \frac{n^{-c}}{6}.$$

On the other hand, using concentration bounds we can show that with probability at least $2/3$,

$$|(\mathbb{E}[B] + r) - (\gamma + r)| = |\mathbb{E}[B] - \gamma| \leq \frac{n^{-c}}{6}.$$

In this case $\mathcal{R}$ always identifies $k = k_0$ since $k_0 \cdot n^{-c+2} < \gamma + r < (k_0 + 1) \cdot n^{-c+2}$.

We now prove that f_c is **prBPL**-hard. We recall the standard **prBPL**-complete problem: Given an OBP B of length and width n, determine if $\mathbb{E}[B] < 1/3$ or $\mathbb{E}[B] > 2/3$, where the promise is that one of these cases holds. We reduce this problem to f_c as follows. Let $T_B : \{0,1\}^{dn} \to \{0,1\}$ be the OBP defined as

$$T_B(x_1, \ldots, x_d) = \texttt{MAJ}(B(x_1), \ldots, B(x_d))$$

where $d = O(c \log n)$ such that if $\mathbb{E}[B] < 1/3$ then $\mathbb{E}[T_B] < n^{-c}/6$, and if $\mathbb{E}[B] > 2/3$ then $\mathbb{E}[T_B] > 1 - n^{-c}/6$. Observe that T_B has length and width $N = \text{poly}(n)$ and is constructible in deterministic logspace given B. Thus, let the input to f_c be $(1^N, T_B, n^{-c})$, which satisfies the promise of f_c, and hence if the answer is less than $1/2$ we determine that $\mathbb{E}[B] < 1/3$, and otherwise determine that $\mathbb{E}[B] > 2/3$. □

We first prove that the values that the machine returns are accurate (assuming the machine returns a value).

Lemma 5.4. *For every B, if* UnivDerand *halts on input $(1^n, B)$, then $|\textsc{UnivDerand}(1^n, B) - \mathbb{E}[B]| \leq n^{-1}$.*

Proof. This follows from Theorem 5.2 applied to $\widetilde{p_{\to v}} = \langle i \rangle(1^n, B_{\to v}, r)$. □

We next prove the machine halts in the claimed space bound.

Lemma 5.5. *For every space-constructible function $S : \mathbb{N} \to \mathbb{N}$ with $S(n) \geq \log n$,* UnivDerand *runs in space $O(S(n))$ if* $\mathbf{prBPL} \subseteq \mathbf{SPACE}[O(S(n))]$.

Proof. We prove that $\textsc{UnivDerand}(1^n, B)$ halts and returns a value with $i + j \leq c \cdot S(n)$ for an absolute constant c (in particular, $i, j < \infty$), which suffices to establish the lemma by the composition of space-bounded algorithms.

By Proposition 5.3, there is a Turing machine $\langle i \rangle$ deciding the language f_5 in $\mathbf{SPACE}[O(S(n))]$. We now show that there exists $r \in \{1 \cdot n^{-5}/2, 2 \cdot n^{-5}/2, \ldots, 2n^2 \cdot n^{-5}/2\}$ such that

$$\left|\mathbb{E}[B_{\to v}] - k \cdot n^{-3} + r\right| > n^{-5}/6 \qquad (\star)$$

for every k and v. There are n^2 different values $\mathbb{E}[B_{\to v}]$ over v in the vertex set $V(B)$ of the branching program, and for each v, there is at most one assignment to r such that $(\star)$ fails to hold for some $k \in \mathbb{Z}$. As there are $2n^2$ possible values for r, there must be one such that $(\star)$ holds for all k and v.

Finally, let $j = O(S(|B|))$ be such that $\langle i \rangle(1^n, B_{\to v}, r)$ halts using at most j space for every v. Such a j exists per assumption and the fact that the input $(1^n, B_{\to v}, r)$ satisfies the promise of Proposition 5.3 for every v. Thus, upon reaching the tuple (i, j, r), the set of estimates $\widetilde{p_{\to v}} = \langle i \rangle(1^n, B_{\to v}, r)$ must satisfy $|\widetilde{p_{\to v}} - \mathbb{E}[B_{\to v}]| \leq n^{-3}$ for every $v \in V(B)$. Then running $\textsc{LCTest}(1^n, B, \{\widetilde{p_{\to v}}\}_{v \in V(B)})$ (where we wait for the test to request a particular value $\widetilde{p_{\to v}}$ and then recompute it from $\langle i \rangle$, avoiding the need to store all n^2 values) will result in LCTest accepting, and hence UnivDerand halts in the claimed space bound. Moreover, the returned value $\delta = \langle i \rangle(1^n, B, r)$ satisfies that $|\delta - \mathbb{E}[B]| \leq n^{-1}$. □

We finally prove the converse.

Lemma 5.6. *For every space-constructible function $S : \mathbb{N} \to \mathbb{N}$ satisfying $S(n) \geq \log n$,* $\mathbf{prBPL} \subseteq \mathbf{SPACE}[O(S(n))]$ *if* UnivDerand *runs in space $S(n)$.*

Proof. By Proposition 5.3 it suffices to solve f_3 using a logspace reduction to UnivDerand. Given $(1^n, B, r)$ as the input (where r is the rounding threshold, which we will ignore), let $\delta := \textsc{UnivDerand}(1^n, B)$ be the value returned by UnivDerand on B. By Lemma 5.4 we have $|\delta - \mathbb{E}[B]| < n^{-1}$, and hence δ is a desired deterministic output for f_3. □

We can then conclude the proof of Theorem 5.1.

Proof of Theorem 5.1. Let UnivDerand be the algorithm as defined above. Theorem 5.1 follows from Lemma 5.4 (and the fact that it returns a value follows from Lemma 5.5). The if direction of Theorem 5.1 follows from Lemma 5.5, and the only if direction follows from Lemma 5.6. □

Finally, we conclude the proof of Theorem 1.2.

Proof of Theorem 1.2. Let U be the algorithm that, given the description of a randomized logspace algorithm R and an input x where $|x| = n$, constructs (in deterministic logspace) the ordered branching program $B := R(x, \cdot)$ of length and width at most $m = \text{poly}(n)$ that represents the action of R over its random bits. Then let U call $\textsc{UnivDerand}(1^m, B)$, and if the value returned is less than $1/2$ return 0, and otherwise return 1. By the promise on R we have either $\Pr[B(U_n) = 1] > 3/4$ or $\Pr[B(U_n) = 1] < 1/4$, and as in both cases we estimate the expectation of B up to error $1/n$ by Theorem 5.1, we correctly decide which case we are in, and the space consumption follows from that of Theorem 5.1. □

6 Hitting Sets Imply Samplers for Ordered Branching Programs

We now prove that hitting sets imply black-box two-sided derandomization of ordered branching programs. To do so, we first formally define hitting sets and deterministic samplers:

Definition 6.1. Given a class of functions $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}\}$, an *$\varepsilon$-hitting set generator (HSG)* $H : \{0,1\}^s \to \{0,1\}^n$ for $\mathcal{F}$ satisfies that for every $f \in \mathcal{F}$ with $\mathbb{E}[f] \geq \varepsilon$, there exists $y \in \{0,1\}^s$ where $f(H(y)) = 1$. We say H is *explicit* if there is a uniform algorithm that computes $H(x)$ in space $O(s)$ given 1^n and x.

Definition 6.2. Given a class of functions $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}\}$, an *$\varepsilon$-(deterministic) sampler* `SAMP` with space complexity $s(n)$ for $\mathcal{F}$ is a deterministic algorithm that runs in space $s(n)$ and, given oracle access to $f \in \mathcal{F}$, makes queries to f and outputs an estimate δ satisfying $|\delta - \mathbb{E}[f]| \leq \varepsilon$.

A deterministic sampler captures the idea of a derandomization algorithm that only accesses the branching program in a black-box fashion, and such a notion has been explored before in the context of small-space derandomization [HU22, CH22, PV22].

We now give a formal statement of Theorem 1.3. We state it in terms of dependence on the seed length of the HSG, as our result generically converts a hitting set to a sampler with comparable space complexity.

Theorem 6.3. *Suppose there is a uniformly constructible family $\mathcal{H} = \{H_1, \ldots, \}$ where $H_n : \{0,1\}^{s(n)} \to \{0,1\}^n$ is an explicit $1/2$-hitting set with seed length $s(n)$ for width n, length n OBPs. Then for every $\varepsilon > 0$, there is a uniformly computable deterministic ε-sampler with space complexity $O(s(nw/\varepsilon))$ for width w, length n OBPs.*

We prove this by developing a local consistency test that can be implemented given black-box access to a branching program.

6.1 Proof Overview

The proof of Theorem 6.3 relies on developing a local consistency test that can be implemented given black-box access to a branching program (whereas all previous tests required access to the internal states of the program). We first describe how we can access the internal states of the program in a black-box manner.

Given a branching program $B : \{0,1\}^n \to \{0,1\}$ and a hitting set $H : \{0,1\}^s \to \{0,1\}^n$, for each seed $x \in \{0,1\}^s$ and layer $i \in [n]$, the program reaches some state v on input $H(x)_{1..i}$. We can index this state in a black-box fashion by writing down (x, i). However, as potentially many seeds may reach the same state v, we would like to collapse these duplicates back together. Since we cannot examine layer i of the program, we can instead attempt to test if x and x' reach the same state, by plugging in every HSG output and see if the programs starting from (x, i) and (x', i) behave differently.

Definition 6.4 (Informal statement of Definition 6.9)**.** For $x, x' \in \{0,1\}^s$ and $i \in [n]$, tuples (x, i) and (x', i) are *indistinguishable* if for every $y \in \{0,1\}^s$,

$$B(H(x)_{1..i}H(y)_{1..n-i}) = B(H(x')_{1..i}H(y)_{1..n-i}).$$

It is not the case that indistinguishable tuples always reach the same state. However, Cheng and Hoza were able to show the following:

Lemma 6.5 ([CH22] (Informal))**.** *Suppose states v and v' are reached by indistinguishable tuples. Then the probability of accepting in B starting from v is similar to that of accepting starting from v'.*

Thus, the states have similar behavior from layer i onward. Unfortunately, it is not the case that indistinguishable states always have indistinguishable out-edges. Thus, a naive attempt to learn the program using query access would print *both* out-edges and output a nondeterministic branching program, a model that is provably **NL**-hard to derandomize. It is likewise unclear how to select a single edge to print in a way that maintains the acceptance probability of the program. In the constant-width regime, Cheng and Hoza [CH22] circumvented this by remembering $O(\log n)$ bits of information about every state in layer $i+1$ while constructing layer i, allowing them to choose a good out-edge. However, this does not seem feasible for super-constant width. Instead, we develop a local consistency test that can tolerate conflating indistinguishable states.

Suppose for every tuple (x, i) we are given an estimate $\tilde{p}_{x,i}$, which is supposedly close to the true probability of accepting from $v := B[v_{st}, H(x)_{1..i}]$.

Definition 6.6 (Black-Box Local Consistency Test (Informal))**.** Given black-box access to an ordered branching program $B : \{0,1\}^n \to \{0,1\}$ and a hitting set $H : \{0,1\}^s \to \{0,1\}^n$ and estimates $\{\tilde{p}_{x,i}\}_{x \in \{0,1\}^s, i \in [n]}$, verify that the following conditions hold:

1. For every pair of indistinguishable tuples $(x, i), (x', i)$, we have $|\tilde{p}_{x,i} - \tilde{p}_{x',i}| \leq O(\varepsilon)$.
2. For every tuple (x, i), let $(x_0, i+1)$ and $(x_1, i+1)$ be arbitrary tuples that are indistinguishable from $B[v_{st}, H(x)_{1..i}0]$ and $B[v_{st}, H(x)_{1..i}1]$ respectively. Then

$$\left| \tilde{p}_{x,i} - \frac{\tilde{p}_{x_0,i+1} + \tilde{p}_{x_1,i+1}}{2} \right| \leq O(\varepsilon).$$

If all such conditions hold, output the estimate $\tilde{p}_{0,0}$, and otherwise reject.

We think of all our tests as having a completeness and soundness component, where completeness means that a set of estimates which are sufficiently close to the true probabilities are gauranteed to pass, and soundness means that the test passing implies the returned estimate is close to the true value (where the precise parameters are discussed later).

The tests of Definition 6.6 can be implemented in space $O(s + \log n)$ given H and black-box access to B, as we can enumerate over the seeds of the hitting set and layers in the program, and all such tests are "local", in the sense that they deal with at most two layers and a constant number of seeds.

If every state is distinguishable from every other, and H hits every state in the program, the test of Definition 6.6 is equivalent to the following white-box local consistency test:

Definition 6.7 (White-Box Local Consistency Test (Informal))**.**

1. For every v, all estimates of the accepting probability from v must be within ε of each other.

2. For every v, estimates of the accepting probability from $v, v_0 := B[v, 0]$, and $v_1 := B[v, 1]$ (which we denote $\widetilde{p_v}, \widetilde{p_{v_0}}$, and $\widetilde{p_{v_1}}$) must satisfy $\widetilde{p_v} \approx (\widetilde{p_{v_1}} + \widetilde{p_{v_0}})/2$.

If all such conditions hold, output an arbitrary estimate $\widetilde{p_{v_{st}}}$, and otherwise reject.

The test of Definition 6.7 clearly accepts if the estimates are exactly (or within $\varepsilon/2$ of) the true probabilities of accepting from each vertex. Likewise, soundness is not difficult to show. However, this idealized version of the test in Definition 6.6 not exactly happen, for two reasons:

1. We may impose Item (1) checks between tuples that reach different, yet indistinguishable, states, and likewise for the 0 and 1 states of Item (2).

2. We may fail to impose Item (2) checks between v and $B[v, 0]$ and $B[v, 1]$ if no string output by the hitting set reaches one of the latter states.

Issue (1) must be dealt with in the proof of completeness (as we add some tests not in the white-box tester) and (2) in the proof of soundness (as we sometimes fail to impose tests that should be present). Issue (1) is the easier of the two to deal with. Since indistinguishable states have similar probability of accepting by Lemma 6.5, good estimates for the accepting probabilities of indistinguishable vertices will still be within $O(\varepsilon)$ of each other. Issue (2), in contrast, seemingly presents a real issue for the soundness. For state $v := B[v_{st}, H(x)_{1..i}]$ where there is no seed x' where $B[v_{st}, H(x')_{1..i+1}] = B[v, 0]$, we could run *no* local consistency test to verify $\tilde{p}_{x,i}$. In fact, the estimate of the probability of accepting from v could be arbitrarily wrong, and we would have no ability to detect it. However, we observe that every such v has low probability of being reached from the start state. This is because if no (ε-)HSG output reaches $B[v, 0]$, v must have probability of being reached from the start state at most 2ε. But then a very bad estimate of the probability of accepting from v only changes the overall probability of accepting by at most $O(\varepsilon)$ (and such an argument can be run for all non-verified states simultaneously). Ultimately, we are able to show that the lack of these checks can only change the accepting probability at the start state by $O(\varepsilon)$, which is tolerable.

Putting it all together, we show a black-box tester that, given estimates $\tilde{p}_{x,i}$ for the probability of accepting from $B[v_{st}, H(x)_{1..i}]$ for every x and i, either outputs an approximation of the expectation of the program or rejects the input. To conclude, we use an idea of Cheng and Hoza to find a good set of estimates $\tilde{p}_{x,i}$ using a hitting set. First, to obtain a better result for nontrivial yet suboptimal hitting set generators, we slightly modify the tester to take in $n \cdot w$ estimates, corresponding (essentially) to an estimate for the acceptance probability from every state in the original branching program. Then we show (essentially using the argument of [CH22]), that there is a branching program T of length $\text{poly}(nw/\varepsilon)$ and width $\text{poly}(nw/\varepsilon)$ that divides its input into $n \times w$ blocks, and uses the block labeled with v as a long random string to estimate $\tilde{p}_v$ for every state v in the program, and accepts if all these estimates are within ε of the true acceptance probability. The program uses the true probabilities to check if the empirical average of the samples is within ε of the true values, but we do not need to explicitly construct it - we only need that it exists, and hence our HSG family will contain some string hitting it. Finally, we argue that we can compute the associated empirical averages with oracle access to B, rather than T. A string that hits T will produce good estimates $\tilde{p}_v$ for every v, and our black-box tester will accept on these estimates. Then we can simply enumerate over hitting set strings, and return the first accepted estimate.

6.2 Black-Box Local Consistency Tests

We now formally state the black-box local consistency test:

Theorem 6.8. *There is a deterministic space $O(s+\log n)$ algorithm that, given an explicit ε-HSG $H : \{0,1\}^s \to \{0,1\}^n$ for length n, width w^2 branching programs, oracle access to an OBP B of width w and length n, and estimates $\{\tilde{p}_{x,i}\}_{x\in\{0,1\}^s, i\in[n]}$, either outputs a value or rejects. Moreover:*

1. *If for every x, we have $|p_{v\to} - \tilde{p}_{x,i}| \le 2\varepsilon$ where $v = B[v_{st}, H(x)_{1..i}]$ for every $i < n$ and $\tilde{p}_{x,n} = B(H(x))$, then the algorithm outputs a value.*
2. *If the algorithm outputs δ, then $|\mathbb{E}[B] - \delta| \le 6\varepsilon n$.*

We remark that, despite this result being a black-box test versus the white-box local consistency test of Cheng and Hoza, it obtains an improved soundness loss (of εn rather than εnw), which is relevant in the regime where the branching program has width much larger than length. This is notable as obtaining optimal error samplers in the Nisan-Zuckerman regime [NZ96] (where optimal-error hitting sets are already known) is a well known open question. Unfortunately, we do not obtain this result, as the argument that we can obtain good accepting probability estimates using a hitting set (Lemma 6.20) requires a hitting set for ordered programs of length $nw \gg n$.

We first define notation related to using H to traverse the branching program:

Definition 6.9. For every $x \in \{0,1\}^s$ and $i \in [n]$, let

$$v_i(x) := B[v_{st}, H(x)_{1..i}].$$

Note that this implies $v_i(x) = v_i(x')$ if $B[v_{st}, H(x)_{1..i}] = B[v_{st}, H(x')_{1..i}]$, i.e. the two seeds reach the same vertex in layer i. For convenience, we write $p_{x,i} := p_{v_i(x)\to}$. Moreover, for states $u, u' \in V_i$ we write $u \sim u'$ if the two states are *indistinguishable* under H, i.e. for all $y \in \{0,1\}^s$,

$$B[u, H(y)_{1..n-i}] = B[u', H(y)_{1..n-i}].$$

We can now define the consistency test implemented by the algorithm.

Definition 6.10 (Local Consistency Test)**.** Given B and H and the estimates $\tilde{p}_{x,i}$, let the test be as follows:

1. For every $x, i \in \{0,1\}^s \times [n]$ and for every $x_0, x_1 \in \{0,1\}^s$ such that $B[v_i(x), 0] \sim v_{i+1}(x_0)$ and $B[v_i(x), 1] \sim v_{i+1}(x_1)$, require
$$\left|\tilde{p}_{x,i} - \frac{\tilde{p}_{x_1,i+1} + \tilde{p}_{x_0,i+1}}{2}\right| \le 5\varepsilon.$$
2. For every $x, x' \in \{0,1\}^s$ and $i \in [n]$ such that $v_i(x) \sim v_i(x')$, require $|\tilde{p}_{x,i} - \tilde{p}_{x',i}| \le 5\varepsilon$.
3. For every $x \in \{0,1\}^s$, require $\tilde{p}_{x,n} = B(H(x))$.

Note that given H and oracle access to B and the estimates $\tilde{p}_{x,i}$, we can compute all such tests in space $O(s + \log n)$. We first show this test is complete:

Lemma 6.11. *Suppose for every x, $|\tilde{p}_{x,i} - p_{x,i}| \le 2\varepsilon$ for $i < n$ and $\tilde{p}_{x,n} = B(H(x))$. Then the test of Definition 6.10 passes.*

To show this we require the following, which follows from arguments about the mass of the set difference.

Claim 6.12 (Lemma 3.2 [CH22])**.** *If $v \sim v'$, then $|p_{v\to} - p_{v'\to}| \le \varepsilon$.*

We can then prove completeness.

Proof of Lemma 6.11. Consider an arbitrary Item 1 test:

$$\left|\tilde{p}_{x,i} - \frac{\tilde{p}_{x_1,i+1} + \tilde{p}_{x_0,i+1}}{2}\right|$$

Let $v := v_i(x)$ and for $b \in \{0,1\}$ let $v_b := B[v,b]$ be the state actually reached following edge b from state v. Then for $b \in \{0,1\}$ let $u_b := v_{i+1}(x_b)$ be the state reached by x_b in layer $i+1$. Note that u_b does not necessarily equal v_b, as we could be conflating different states in layer $i+1$, but $u_b \sim v_b$. Thus:

$$\begin{aligned}
\left|\tilde{p}_{x,i} - \frac{\tilde{p}_{x_1,i+1} + \tilde{p}_{x_0,i+1}}{2}\right| &\leq 4\varepsilon + \left|p_{v\to} - \frac{p_{u_0\to} + p_{u_1\to}}{2}\right| && \text{(Assumption)} \\
&\leq 5\varepsilon + \left|p_{v\to} - \frac{p_{v_0\to} + p_{v_1\to}}{2}\right| && \text{(Claim 6.12)} \\
&= 5\varepsilon.
\end{aligned}$$

The proof of Item 2 is analogous, again using Claim 6.12, and Item 3 is immediate. □

We now show soundness. The key issue is dealing with states v such that *no* x satisfies $v = v_i(x)$, because we cannot guarantee consistency for these states. However, these states are precisely those that the HSG fails to hit, which must mean they have low probability of being reached from the start vertex, and hence their estimates being wrong does only a small amount of harm.

Lemma 6.13. *Suppose the test of Definition 6.10 passes with estimates $\tilde{p}_{x,i}$. Then $|\tilde{p}_{0,0} - p_{v_{st}\to}| \leq 6\varepsilon n$.*

To prove Lemma 6.13, in the following three lemmas we assume that the test of Definition 6.10 passes. We first define states that are not verified, and show that the probability of reaching such states are small.

Definition 6.14. For every $x \in \{0,1\}^s$ and $i < n$, let $v = v_i(x)$ be an *unverified* state if there is some $b \in \{0,1\}$ such that there is no $x' \in \{0,1\}^s$ satisfying $B[v,b] = v_{i+1}(x')$, and otherwise let v be *verified*. Let $v_n(x)$ be verified for every x. Note that for an unverified state there still could be x' such that $B[v,b] \sim v_{i+1}(x')$, but we do not use this in the proof of soundness.

Lemma 6.15. *Let T be the event of reaching an unverified state in B. Then $\Pr[T(U_n) = 1] < 2\varepsilon$.*

Proof. Let R be the width $w+1$ program that is the same as B except it accepts if and only if we reach a state not hit by the HSG H. We have $\Pr[R(U_n) = 1] < \varepsilon$ by the goodness of the HSG. Furthermore, conditioned on reaching an unverified state in B, we have probability at least $1/2$ of reaching a state the HSG does not hit. Thus, $\varepsilon > \Pr[R(U_n) = 1] \geq \Pr[T(U_n) = 1]/2$. □

We now construct a branching program such that the estimates for unverified states are consistent with the true probabilities of these states.

Lemma 6.16. *There exists an ordered branching program $Q : \{0,1\}^n \to \{0,1\}$ on a superset of the vertices of B such that:*

1. $|\mathbb{E}[Q] - \mathbb{E}[B]| \leq 2\varepsilon$.

2. *Q is identical to B when restricted to edges between verified states, and edges from verified states to unverified states.*

3. *For every unverified state v in B, for every $x \in \{0,1\}^s$ such that $v = v_i(x)$, we have $|\tilde{p}_{x,i} - q_{v\to}| \leq 5\varepsilon$, where $q_{v\to}$ is the probability of accepting from v in Q.*

Proof. We first construct Q. Let N be the set of unverified states of B. For every $v \in N$ in layer i, note that for every $x, x' \in \{0,1\}^s$ satisfying $v = v_i(x) = v_i(x')$ we have $|\tilde{p}_{x,i} - \tilde{p}_{x',i}| \leq 5\varepsilon$ by Item 2. Let $q_{v\to}$ be a number satisfying $|q_{v\to} - \tilde{p}_{x,i}| \leq 5\varepsilon$ for every such x. We now modify B by wiring both edges from v to a new (arbitrarily complex) set of states such that v now has probability of accepting exactly $q_{v\to}$,[3] and we do this for every unverified v. Let Q be this new branching program. It is clear by construction that Q satisfies Property 2. Furthermore, by Lemma 6.15 we have Property 1. □

We now prove Lemma 6.13 by showing that the estimates $\tilde{p}_{x,i}$ are consistent with the modified program Q.

Lemma 6.17. *Let Q be defined as in Lemma 6.16. Then for every v in B and every x such that $v = v_i(x)$, we have $|\tilde{p}_{x,i} - q_{v\to}| \leq 5\varepsilon \cdot (n-i)$. In particular, $|\tilde{p}_{0,0} - q_{v_{st}\to}| \leq 5\varepsilon n$.*

Proof. The case $i = n$ holds by Item 3 of Definition 6.10 (and the fact that all final layer states are unmodified). Now assume this holds for layer $i+1$. Then for every $v = v_i(x)$ in layer i, we have two possibilities:

- Case 1: v is unverified. In this case, $|\tilde{p}_{x,i} - q_{v\to}| \leq 5\varepsilon$ by Lemma 6.16.
- Case 2: v is verified. For $b \in \{0,1\}$, let $v_b := B[v,b]$ and by Lemma 6.16 we know that v_b also coincides with $Q[v,b]$. Let x_b be such that $v_b = v_{i+1}(x_b)$ (and note that such x_0, x_1 exist because we are not in Case 1). Then:

$$\begin{aligned}
|\tilde{p}_{x,i} - q_{v\to}| &= \left|\tilde{p}_{x,i} - \frac{q_{v_0\to} + q_{v_1\to}}{2}\right| \\
&\leq 5\varepsilon + \left|\frac{\tilde{p}_{x_0,i+1} + \tilde{p}_{x_1,i+1}}{2} - \frac{q_{v_0\to} + q_{v_1\to}}{2}\right| && \text{(Item 1)} \\
&\leq 5\varepsilon + \frac{1}{2}\left|q_{v_0\to} - \tilde{p}_{x_0,i+1}\right| + \frac{1}{2}\left|q_{v_1\to} - \tilde{p}_{x_1,i+1}\right| \\
&\leq 5\varepsilon \cdot (n-i) && \text{(Induction.)}
\end{aligned}$$ □

We now conclude the proof of Lemma 6.13.

Proof of Lemma 6.13. We have:

$$\begin{aligned}
|\mathbb{E}[B] - \tilde{p}_{0,0}| &\leq 2\varepsilon + |\mathbb{E}[Q] - \tilde{p}_{0,0}| && \text{(Lemma 6.16)} \\
&\leq 2\varepsilon + 5\varepsilon n && \text{(Lemma 6.17).}
\end{aligned}$$ □

Finally we conclude Theorem 6.8.

[3]Technically this may not be possible without making Q a probabilistic branching program. However, this construction purely exists to analyze the probabilities $q_{v\to}$, so we ignore this minor complication.

Proof of Theorem 6.8. Given $H : \{0,1\}^s \to \{0,1\}^n$ and the estimates $\{\tilde{p}_{x,i}\}_{x\in\{0,1\}^s, i\in[n]}$, we run the tests as specified in Definition 6.10. All such tests can be implemented in space $O(s + \log n)$, as we now explain. Given $x \in \{0,1\}^s$ and $b \in \{0,1\}$, we can determine if $x' \in \{0,1\}^s$ satisfies $B[v_{st}, H(x)_{1..i}b] \sim v_{i+1}(x')$ by enumerating over $y \in \{0,1\}^s$ and computing the predicate

$$\bigwedge_{y\in\{0,1\}^s} \left[B(H(x)_{1..i}bH(y)_{1..n-i-1}) = B(H(x')_{1..i+1}H(y)_{1..n-i-1})\right].$$

This can be implemented in space $O(s + \log n)$ given black-box access to B, and hence we can determine which Item 1 tests on $\{\tilde{p}_{x,i}\}$ to run in the desired space bound. Similar reasoning applies to the Item 2 and Item 3 tests.

Finally, if all such tests pass, output $\tilde{p}_{0,0}$. By Lemma 6.11 we have that the completeness condition holds, and by Lemma 6.13 we have that the soundness condition holds. □

6.3 Putting It All Together

We now prove Theorem 6.3 from Theorem 6.8. It remains to show that we can generate a good set of estimates $\{\tilde{p}_{x,i}\}$ using a hitting set. We first show that we can modify Theorem 6.8 to only take in nw estimates, rather than $n \cdot 2^s$. This is not required for Theorem 6.3, but it improves the parameters in the case that H is a highly nontrivial yet non-optimal hitting set.

To do so, we first observe that the indistinguishably relation induces a set of equivalence classes on the seeds:

Definition 6.18. Given an OBP B of width w and length n and $H : \{0,1\}^s \to \{0,1\}^n$, for every i let $C_{1,i}, \ldots, C_{w,i} \subset \{0,1\}^s$ be the (possibly empty) equivalence classes of $\{0,1\}^s$ under the indistinguishably relations $x \sim_i x'$ iff $v_i(x) \sim v_i(x')$. Let $y_{j,i} \in \{0,1\}^s$ be the lexicographically first element of $C_{j,i}$ (and order the equivalence classes so that $y_{1,i} < y_{2,i} < \ldots < y_{w,i}$ for every i). Moreover, let $v_{j,i} := v_i(y_{j,i})$. Note that given B and H, $y_{j,i}$ (and hence an HSG output that reaches $v_{j,i}$) is constructible in space $O(s + \log n)$ given i, j.

Corollary 6.19. *There is a deterministic space $O(s+\log n)$ algorithm that, given an explicit ε-HSG $H : \{0,1\}^s \to \{0,1\}^n$ for length n, width w^2 branching programs, oracle access to an OBP B of width w and length n, and estimates $\{\tilde{p}_{j,i}\}_{j\in[w],i\in[n-1]}$, either outputs a value or rejects. Moreover:*

1. *If $|p_{v_{j,i}\to} - \tilde{p}_{j,i}| \le \varepsilon$ for every j, i, where $v_{j,i}$ is as defined in Definition 6.18, then the algorithm outputs a value.*
2. *If the algorithm outputs δ, $|\mathbb{E}[B] - \delta| \le 6\varepsilon n$.*

Proof. The tester simply takes in the estimate $\tilde{p}_{j,i}$ for $p_{v_{j,i}\to}$, copies it to be the estimate for every seed in the j-th equivalence class for layer i, perfectly computes $p_{x,n} := B(H(x))$ for every $x \in \{0,1\}^s$, and runs Theorem 6.8. Clearly if the tester outputs a value it is within $6\varepsilon n$ of δ, as we simply restrict the inputs to Theorem 6.8. Furthermore, note that for an arbitrary $v := v_i(x)$ in equivalence class $C_{j,i}$, we have by Claim 6.12:

$$|p_{v\to} - \tilde{p}_{j,i}| \le \varepsilon + |p_{v_{j,i}\to} - \tilde{p}_{j,i}| \le 2\varepsilon$$

and hence if $|p_{v_{j,i}\to} - \tilde{p}_{j,i}| \le \varepsilon$ is satisfied for all i and j we satisfy the completeness condition of Theorem 6.8, and so the tester will return a value. □

We now argue that there is a hitting set string that can be used to produce good estimates for $p_{v_{j,i}\to}$, where $v_{j,i}$ is as defined in Definition 6.18. The argument that such an output exists is a straightforward modification of the proof in Cheng and Hoza [CH22] that there exists an HSG output inducing estimates that satisfy their local consistency test.

Lemma 6.20 ([CH22])**.** *For every OBP B of length n and width w and $H : \{0,1\}^s \to \{0,1\}^n$ and $\varepsilon > 0$, there exists $t = O(\log(nw)/\varepsilon^2)$ and an OBP $\mathtt{EST} : \{0,1\}^{n\times w\times tn} \to \{0,1\}$ of length and width* $\mathrm{poly}(nw/\varepsilon)$ *defined as follows:*

$$\mathtt{EST}(z_{1,1},\dots,z_{w,n}) = \bigwedge_{i\in[n],j\in[w]} \mathtt{EST}_{j,i}(z_{j,i})$$

where $\mathtt{EST}_{j,i}(z_{j,i})$ computes as follows. It interprets $z_{j,i}$ as t samples of length n, and computes

$$\mathtt{EST}_{j,i}(s_1,\dots,s_t) = \mathbb{I}\left[\Pr_{k\in[t]}[B[v_{j,i},s_k] = v_{acc}] \in [p_{v_{j,i}\to} - \varepsilon, p_{v_{j,i}\to} + \varepsilon]\right].$$

(where $v_{j,i}$ is as defined in Definition 6.18 in terms of H). Then $\mathbb{E}[\mathtt{EST}] > 1/2$, and for every z such that $\mathtt{EST}(z) = 1$, for every j,i, using the samples in block $z_{j,i}$ of z to estimate the acceptance probability from $v_{j,i}$ produces an estimate with at most ε additive error.

Proof Sketch. It is clear that $\mathtt{EST}_{j,i}$ can be computed by an ordered branching program of the claimed length and width, by duplicating the subprogram of B starting from $v_{j,i}$ and counting the number of satisfied trials using an additional $O(\log(t))$ bits of memory, and accepting if the final count is within the specified range. Thus the conjunction $\mathtt{EST}$ can be computed in the claimed space bound. We then choose t sufficiently large such that a random input satisfies all these checks with overwhelming probability. We note that $\mathtt{EST}$ is defined in terms of the exact probabilities of acceptance, which the tester does not have, but we only need that the program exists, not that we can construct it. □

Then the proof of Theorem 6.3 follows.

Proof of Theorem 6.3. By a standard reduction (see e.g. [CH22]), $\mathcal{H}$ implies an explicit family of ε-hitting sets for length n, width w OBPs with seed length $s(\mathrm{poly}(nw/\varepsilon)) = O(s(nw/\varepsilon))$ (where the final equality follows as for any $s(n) = \Omega(\log^2 n)$ the theorem is trivial by the fact that the Nisan PRG exists, so we may assume this is not the case).

Let H be a $\varepsilon/(6n)$-HSG for length n and width w^2 OBPs with seed length $O(s(nw/\varepsilon))$. Let H_2 be a 1/3-HSG for length $n^2wt = \mathrm{poly}(nw/\varepsilon)$, width $(nw/\varepsilon)^c$ OBPs, where t is as in Lemma 6.20 with $\varepsilon := \varepsilon/(6n)$. By choice of parameters, H_2 has seed length $s_2 := O(s(nw/\varepsilon))$. The sampler enumerates over every $z \in \{0,1\}^{s_2}$. For every such z, the sampler calls the tester of Corollary 6.19, and when an estimate $\tilde{p}_{j,i}$ for $p_{v_{j,i}\to} = \mathbb{E}[B[v_{j,i}, U_{n-i}]]$ is required by the tester, we use the j,i block of $H_2(z)$ (as done by $\mathtt{EST}_{j,i}$ in Lemma 6.20) to compute the estimate. Note that we can find $y_{j,i}$, the lexicographically first seed in equivalence class j in layer i, in logspace, and by definition $v_{j,i} = v_i(y_{j,i})$. Thus, we can compute $\tilde{p}_{j,i}$ by enumerating over the samples $s_1,\dots,s_t$ in block j,i in $H_2(z)$ and returning

$$\mathop{\mathbb{E}}_{k\in[t]}[B(H(y_{j,i})_{1..i}s_k)] = \mathop{\mathbb{E}}_{k\in[t]}[B[v_{j,i},s_k]].$$

If the tester accepts, return the value that the tester outputs, and otherwise increment z. The space complexity is $O(\log(n) + s_2)$ by composition of space-bounded algorithms.

Now suppose the sampler returns a value. By Item 2 of Theorem 6.8, the returned estimate is within $\varepsilon/(6n) \cdot 6n = \varepsilon$ of the true expectation. To show the sampler returns a value, note that by

Item 1 of Theorem 6.8 it suffices to argue that we give the tester a series of inputs $\{\tilde{p}_{j,i}\}$ such that $|p_{v_{j,i}\to} - \tilde{p}_{j,i}| \le \varepsilon/(6n)$ for every i, j. But these are precisely the estimates generated by a string x such that $\texttt{EST}(x) = 1$, and H_2 hits this program by choice of parameters, so we conclude. □

References

[ACR96] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. Hitting sets derandomize BPP. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings*, volume 1099 of *Lecture Notes in Computer Science*, pages 357–368. Springer, 1996.

[ACRT99] 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.

[AKM+20] AmirMahdi Ahmadinejad, Jonathan A. Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil P. Vadhan. High-precision estimation of random walks in small space. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1295–1306. IEEE, 2020.

[BF99] Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 1999.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM J. Comput.*, 13(4):850–864, November 1984.

[CH22] Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. *Theory of Computing*, 18(21):1–32, 2022.

[Csa76] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM J. Comput.*, 5(4):618–623, 1976.

[FK18] Michael A. Forbes and Zander Kelley. Pseudorandom generators for read-once branching programs, in any order. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 946–955. IEEE Computer Society, 2018.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 25–32, New York, NY, USA, 1989. Association for Computing Machinery.

[GR14] Anat Ganor and Ran Raz. Space pseudorandom generators by communication complexity lower bounds. In *APPROX-RANDOM*, volume 28 of *LIPIcs*, pages 692–703. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

[GRZ23] Uma Girish, Ran Raz, and Wei Zhan. Is untrusted randomness helpful? In *14th Innovations in Theoretical Computer Science Conference, ITCS*, 2023.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Information processing letters*, 43(4):169–174, 1992.

[GVW11] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. Simplified derandomization of BPP using a hitting set generator. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 59–67. Springer, 2011.

[HK18] William M. Hoza and Adam R. Klivans. Preserving randomness for adaptive algorithms. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Hoz21] William M. Hoza. Better pseudodistributions and derandomization for space-bounded computation. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 28:1–28:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[HU22] William M. Hoza and Chris Umans. Targeted pseudorandom generators, simulation advice generators, and derandomizing logspace. *SIAM J. Comput.*, 51(2):17–281, 2022.

[HZ20] William M. Hoza and David Zuckerman. Simple optimal hitting sets for small-success RL. *SIAM J. Comput.*, 49(4):811–820, 2020.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 356–364. ACM, 1994.

[IW97] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229. ACM, 1997.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

[LPS88] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[LV17] Chin Ho Lee and Emanuele Viola. Some limitations of the sum of small-bias distributions. *Theory Comput.*, 13(1):1–23, 2017.

[MRT19] Raghu Meka, Omer Reingold, and Avishay Tal. Pseudorandom generators for width-3 branching programs. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 626–637. ACM, 2019.

[Nis90] Noam Nisan. Psuedorandom generators for space-bounded computation. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 204–212. ACM, 1990.

[Nis93] Noam Nisan. On read-once vs. multiple access to randomness in logspace. *Theor. Comput. Sci.*, 107(1):135–144, 1993.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.

[PV22] Edward Pyne and Salil P. Vadhan. Deterministic approximation of random walks via queries in graphs of unbounded size. In Karl Bringmann and Timothy Chan, editors, *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, pages 57–67. SIAM, 2022.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.

[RR99] Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 159–168. ACM, 1999.

[RTV06] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 457–466. ACM, 2006.

[RVW01] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1), January 2001.

[Sha81] Adi Shamir. The generation of cryptographically strong pseudo-random sequences. In *CRYPTO*, page 1. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981.

[STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

[SZ99] Michael E. Saks and Shiyu Zhou. $BP_HSpace(S) \subseteq DSPACE(S^{3/2})$. *J. Comput. Syst. Sci.*, 58(2):376–403, 1999.

[WB86] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91. IEEE Computer Society, 1982.