

Multi-Access Randomness Saves Space, Even for Halting Algorithms

William M. Hoza
Department of Computer Science
The University of Chicago
williamhoza@uchicago.edu

Abstract

We prove that every language in $P^{\#P}$ can be decided by a bounded-error randomized algorithm that uses only $O(\log n)$ bits of work space. The algorithm is guaranteed to halt for every input and every setting of the random tape. However, there is a catch: the algorithm uses a *multi-access* random tape, i.e., the algorithm scans both forward and backward across a read-only tape filled with an unlimited number of random bits. Prior work on log-space algorithms with multi-access randomness either focuses on polynomial-time algorithms, or else permits algorithms that sometimes run forever. Our algorithm always halts, but it uses exponential time. Thus, our work identifies a natural model of computation in which randomness is intrinsically useful, assuming $L \neq P^{\#P}$.

1 Introduction

When is randomness useful for efficient computation, and when is it not? The relationship between randomness and space complexity is remarkably subtle. There are multiple “reasonable” models of randomized space-bounded computation, and some are more powerful than others.

1.1 The Standard Model

The standard model of randomized log-space computation is a Turing machine with a read-only input tape, a read-write work tape of size $O(\log n)$, and a read-only random tape that extends infinitely in one direction. In the standard model, we assume that the random tape is *read-once*, i.e., the machine may scan to the right over the tape, but not to the left. Furthermore, we assume that the algorithm *always halts*, for every input and every setting of the random tape. The set of languages that can be decided by such an algorithm with a bounded probability of error is typically denoted BPL, but in this paper, we use Saks’ alternative notation BP_HL [Sak96]. The subscript “H” serves to emphasize the assumption that the machine always halts.

It is conjectured that $\text{L} = \text{BP}_H\text{L}$, i.e., randomness is not intrinsically useful in this setting. This conjecture has been proven under plausible assumptions [KvM02; DT23; PR23; PRZ23; DPT24; Sha24; DPTW25; PT25; PT26]. There are also unconditional partial derandomizations of BP_HL , such as Nisan’s proof that $\text{BP}_H\text{L} \subseteq \text{SC}$ [Nis94], Saks and Zhou’s proof that $\text{BP}_H\text{L} \subseteq \text{L}^{3/2}$ [SZ99] (with a lower-order improvement by Hoza [Hoz21]), and Cheng and Wang’s proof that BP_HL is contained in log-space-uniform AC^1 [CW24]. See also Hoza’s survey [Hoz22].

1.2 Variant Models

Machines that are not required to halt are potentially more powerful. Define BP_∞L just like BP_HL , except we drop the requirement that the machine always halts. We only require that the machine halts and outputs the correct answer with high probability.¹ We have $\text{NL} \subseteq \text{BP}_\infty\text{L}$ [Gil77]. Therefore, it is reasonably likely that $\text{L} \neq \text{BP}_\infty\text{L}$, so randomness can be useful for space-efficient computation, although it might be considered “cheating” to use algorithms that don’t always halt. On the upper bound side, Borodin, Cook, and Pippenger’s work implies $\text{BP}_\infty\text{L} \subseteq \text{TC}^1$ [BCP83; AO96], and Saks conjectured that $\text{BP}_\infty\text{L} = \text{NL}$ [Sak96].

We can also grant our machines another superpower: the ability to go back and re-read old random bits. Following Nisan [Nis93], we use a superscript asterisk to denote machines with this ability. For example, we define $\text{BP}_\infty^*\text{L}$ just like BP_∞L , except now the random tape head can scan both to the right and to the left. This “multi-access random tape” represents an enormous database of pre-generated random bits, whereas a read-once random tape represents a single coin. Non-halting algorithms with multi-access random tapes are extremely powerful. Karpinski and Verbeek’s work implies $\text{BP}_\infty^*\text{L} = \text{PSPACE}$ [KV85; KV86]. Certainly $\text{L} \neq \text{PSPACE}$ by the Space Hierarchy Theorem, so this is a case where randomized algorithms *provably* have an advantage over deterministic algorithms.

There is also a line of work that studies a weaker model in which the machine has multi-access random bits, but the machine is required to halt in polynomial time [BNS92; Nis93; VW97; BVW98; FF07; Rav07; PR23; Sha24]. Let BP_T^*L denote the class of languages that can be decided in this model with a bounded probability of error. The subscript T (“Time”) is not standard notation,

¹It turns out that we can assume without loss of generality that the machine halts with probability 1 [Sim81; RST84]. However, this does not imply that it halts for *every* setting of the random tape. For example, consider a machine that halts on every setting of the random tape except the all-zeroes sequence.

but it serves to emphasize the assumption that the machine runs in polynomial time. We have $\text{BP}_T^*L = \text{BP} \cdot L$, using Schöning’s generic BP-operator [Sch89]. We expect that $\text{BP}_T^*L = L$; one can prove $\text{BP}_T^*L = L$ under plausible assumptions [KvM02; PR23; Sha24].

1.3 Our Model: Halting Algorithms with Multi-Access Random Bits

In this paper, we study a model in which the machine has multi-access random bits, and the machine is required to halt, but its time complexity is unconstrained. Let BP_H^*L denote the class of languages that can be decided with bounded error probability in this model.

The BP_H^*L model has apparently been neglected in the literature, perhaps because a log-space machine with *read-once* randomness that always halts is automatically guaranteed to run in polynomial time ($\text{BP}_H^*L = \text{BP}_T^*L$). Nisan suggested that the same should hold for machines with multi-access random bits [Nis93], but this is not correct. We will show in this paper that a BP_H^*L algorithm might run for exponential time before halting, and our main result gives strong evidence that the BP_H^*L model is much more powerful than the BP_T^*L model.

Theorem 1.1 (Main result). $\text{P}^{\#P} \subseteq \text{BP}_H^*L$.

Recall that $\text{P}^{\#P}$ is the set of problems that can be solved in polynomial time using an oracle that tells how many solutions there are to a given instance of an arbitrary NP search problem. The class $\text{P}^{\#P}$ is very powerful. It contains P and NP (trivially), the rest of the polynomial hierarchy [Tod91], and BQP [BV97; Aar05; MP26]. Therefore, Theorem 1.1 provides strong evidence (albeit not a proof) that $L \neq \text{BP}_H^*L$, in which case this is another model where randomness is useful.

The time complexity of our BP_H^*L algorithm is $2^{\text{poly}(n)}$, which is the best we should hope for, given that we simulate a class as powerful as $\text{P}^{\#P}$. In contrast, Karpinski and Verbeek’s non-halting algorithm for solving arbitrary problems in PSPACE has a *doubly* exponential expected running time [KV85; KV86].²

The best upper bound we have on the power of our model is simply $\text{BP}_H^*L \subseteq \text{BP}_\infty^*L = \text{PSPACE}$. (See Fig. 1.) We leave it as an open problem to pin down our model’s power more precisely.

1.4 Proof Overview

The proof of Theorem 1.1 has two parts. First, we prove $\text{PP} \subseteq \text{BP}_H^*L$. Then, we prove $\text{P}^{\text{BP}_H^*L} = \text{BP}_H^*L$. Theorem 1.1 follows, because $\text{P}^{\text{PP}} = \text{P}^{\#P}$ [Gil77].

1.4.1 Overview of the Proof That $\text{PP} \subseteq \text{BP}_H^*L$

As a warm-up, let us sketch a proof that $\text{SAT} \in \text{BP}_H^*L$. Say we are given an n -variable CNF formula ϕ . We will search the random tape for a satisfying assignment. If we find one, we can verify it in log space easily enough. The challenge is to know when to give up looking. We need to give ourselves exponential time, so we can find a rare satisfying assignment if one exists, but we also need to make sure we eventually halt, even if ϕ is unsatisfiable.

To solve this problem, we sample positive integers $H_1, H_2, \dots$ with $\Pr[H_i = h] = 2^{-h}$. We halt if we find either (a) a large value $H_i \geq h_* = \Theta(n)$, or else (b) an interval in which some value h occurs $\Theta(n)$ times and no value larger than h occurs at all. One of these two events is bound to happen eventually, because the only way to avoid event (b) is to use larger and larger H_i values,

²Karpinski and Verbeek’s approach is to search the random tape for a place where the entire exponential-length computation history of the PSPACE computation just happens to occur [KV85; KV86]. A monkey with a typewriter will almost surely produce the works of Shakespeare if we wait long enough.

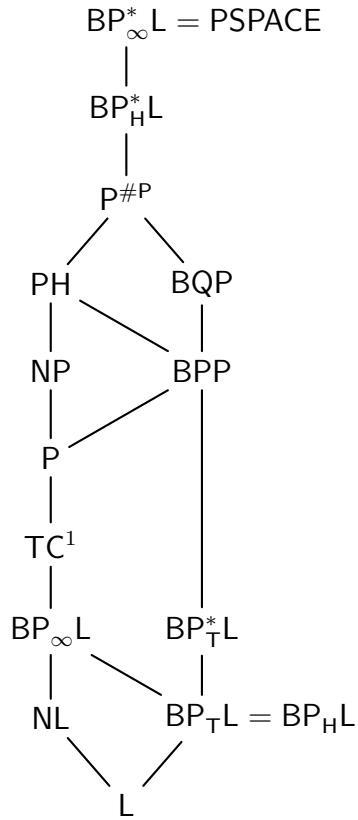


Figure 1: Models of bounded-error randomized log-space computation and where they fit in the broader complexity landscape. Upward edges represent containment.

eventually exceeding h_* . However, with high probability, it takes $2^{\Omega(n)}$ samples before either event occurs, giving us a suitable probabilistic “timer.”

The generalization to PP is similar. We use our timer construction to generate exponentially many random assignments for a given CNF formula ϕ . We use these random samples to estimate the fraction of assignments that satisfy ϕ . With high probability, the sampling error is much less than 2^{-n} , so one should be able to infer the exact number of satisfying assignments of ϕ . A difficulty is that we do not have enough work space to count how many random assignments we generated or how many of them satisfy ϕ . To resolve this difficulty, we count mod p for various small primes p , and then we use space-efficient algorithms for converting the Chinese remainder representation of a number to its standard binary representation [CDL01; HAB02].

1.4.2 Overview of the Proof That $\text{PBP}_{\text{H}}^* \text{L} = \text{BP}_{\text{H}}^* \text{L}$

In general, proving $\text{P}^{\mathcal{C}} = \mathcal{C}$ is more or less equivalent to proving that $\mathcal{C}$ is closed under polynomial-depth *composition*, i.e., for every length-preserving function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ that can be computed in $\mathcal{C}$, the n -fold composition $f^{(n)}(x) = f(f(f(\cdots(f(x))\cdots)))$ can also be computed in $\mathcal{C}$. Composing space-efficient algorithms is a fascinating subject. (See Pyne and Tell’s recent work [PT25].) Deterministic log-space is closed under constant-depth composition, but it is probably not closed under polynomial-depth composition, because that would imply $\text{L} = \text{P}$. In contrast, we show that $\text{BP}_{\text{H}}^* \text{L}$ is closed under polynomial-depth composition.

Our approach is to use the read-only random tape to implement a polynomial-size stack. Given a stack, composition is straightforward: we compute $f^{(n)}(x)$ by simulating the algorithm for f . When the algorithm tries to make a query to its input, we pause the computation, push its $O(\log n)$ -bit configuration onto the stack and erase it from our work tape, recursively compute the desired bit of $f^{(n-1)}(x)$, and then pop the configuration back off the stack and continue the simulation.

To implement a stack, we once again sample positive integers $H_1, H_2, \dots$, each with a geometrically decreasing probability distribution. We use these integers to divide the random tape into intervals, each of which is divided into subintervals, each of which is divided into subsubintervals, etc., almost like the classic “skip list” data structure [Pug90]. Suppose the random tape head is currently located at the beginning of some interval I . To push a value v onto the stack, we move to the v -th subinterval of I , and carry out future computations within that subinterval. Later, to pop the value off the stack, we work our way back to the beginning of the larger interval I , counting subintervals as we go to recover v . This technique is similar to a construction by Küçük, Say, and Yakaryılmaz [KSY14].

Annoyingly, our algorithm for computing $f^{(n)}(x)$ is actually not guaranteed to halt, because the inner algorithm might give inconsistent answers to the outer algorithm’s input queries, in which case the outer algorithm might get stuck in a loop. However, we are able to show that the algorithm uses only $2^{\text{poly}(n)}$ random bits with high probability, and this turns out to be enough. Using a “computation graph search” technique due to Sipser [Sip80], we show that log-space algorithms that use $2^{\text{poly}(n)}$ multi-access random bits can be converted into log-space algorithms that always halt. In contrast, note that Karpinski and Verbeek’s non-halting algorithm for solving arbitrary problems in PSPACE typically uses $2^{2^{\text{poly}(n)}}$ random bits [KV85; KV86].

1.5 More Related Work

The literature on randomized space-bounded computation is vast, and we certainly have not discussed every variant model that has been studied. For example, there is a line of work studying randomized log-space algorithms that make a few passes over their random tapes [DPS11; MPV15; DMPV16].

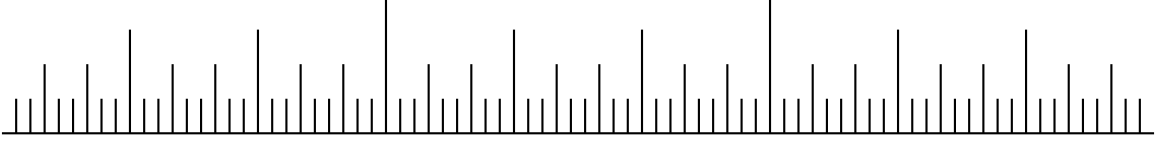


Figure 2: The (d, b) -ruler where $d = 4$ and $b = 3$.

Also, we have focused exclusively on algorithms with bounded error probabilities (“BP” algorithms). One-sided error models, zero-error models, and unbounded-error models are also fascinating. See, for example, Saks’ survey [Sak96].

2 Preliminaries

We assume the reader is familiar with standard complexity classes such as P, PP, and #P. For clarity, we elaborate about our computational model below.

Definition 2.1 (Randomized Turing machine). For the purposes of this paper, a *Turing machine with a random tape* is a three-tape Turing machine. The first tape is a read-only finite-length tape containing the input and end markers. The second tape is a read-write work tape. The third tape is a read-only tape that extends infinitely to the right, pre-filled with uniform random bits. There is a marker at the left end of the random tape. We say that the machine has a *read-once random tape* if the random tape head never moves left; otherwise, we say it has a *multi-access random tape*.

Definition 2.2 (The BP_H^*L complexity class). Let $L \subseteq \{0, 1\}^*$. Then $L \in \text{BP}_H^*L$ if there exists a Turing machine M with a multi-access random tape such that:

- For every input and every setting of the random tape, M halts.
- For every n -bit input and every setting of the random tape, M uses $O(\log n)$ cells of its work tape.
- For every $x \in L$, we have $\Pr[M \text{ accepts } x] \geq 2/3$.
- For every $x \notin L$, we have $\Pr[M \text{ accepts } x] \leq 1/3$.

3 Rulers, Hyporulers, and Hyperrulers

As discussed in Section 1.4, our algorithms involve sampling numbers $H_1, H_2, H_3, \dots$ with a geometrically decreasing probability distribution. Intuitively, the intention is that these numbers will approximate the *ruler sequence*, defined below.

Definition 3.1 (Ruler). Let $d \geq 0$ and $b \geq 1$. The (d, b) -ruler is the sequence

$$\vec{h} = (\vec{h}^{(1)}, d, \vec{h}^{(2)}, d, \dots, d, \vec{h}^{(b)}),$$

where each $\vec{h}^{(i)}$ is a copy of the $(d - 1, b)$ -ruler. In the base case, when $d = 0$, the $(0, b)$ -ruler is the empty sequence. (See Fig. 2.)

Of course, we cannot expect that random integers $H_1, H_2, \dots$ will exactly match the ruler sequence. Instead, we want to argue that they are in some sense “bounded by” the ruler sequence. We will need two notions, which we call *hyporulers* and *hyperrulers*.

3.1 Hyporulers

The probabilistic timer construction that we described in [Section 1.4](#) uses the notion of a *hyporuler*.

Definition 3.2 (Hyporuler). Let $d \geq 0$ and $b \geq 1$. A (d, b) -hyporuler is a sequence of the form

$$\vec{h} = (\vec{h}^{(1)}, d, \vec{h}^{(2)}, d, \dots, d, \vec{h}^{(t)}), \quad (1)$$

where $t \leq b$ and each $\vec{h}^{(i)}$ is a $(d-1, b)$ -hyporuler. In the base case, when $d = 0$, the empty sequence is the unique $(0, b)$ -hyporuler.

The following alternative characterization is helpful for recognizing and reasoning about hyporulers.

Lemma 3.3 (Characterizing hyporulers). *Let $d \geq 0$, let $b \geq 1$, and let $\vec{h} = (h_1, h_2, \dots, h_N) \in [d]^N$. The following are equivalent.*

1. $\vec{h}$ is a (d, b) -hyporuler.
2. Let $j \in [N]$. Let i be the largest value in $[j]$ such that $h_i > h_j$, or $i = 0$ if no such i exists. Then h_j appears at most $b-1$ times among $h_{i+1}, h_{i+2}, h_{i+3}, \dots, h_j$.

Proof. (1 $\implies$ 2) If $d = 0$, this is trivial. Now assume $d \geq 1$, so $\vec{h}$ has the form given in [Eq. \(1\)](#). If $i+1, i+2, \dots, j$ are all contained within a single $\vec{h}^{(k)}$, then we are done by induction. Otherwise, we must have $i = 0$ and $h_j = d$, in which case we are done because a (d, b) -hyporuler has at most $b-1$ occurrences of the value d .

(2 $\implies$ 1) Condition 2 clearly implies that d appears at most $b-1$ times among $h_1, h_2, \dots, h_N$. Therefore, we can write $\vec{h}$ in the form [Eq. \(1\)](#), where $t \leq b$ and each $\vec{h}^{(i)}$ is a sequence of elements of $[d-1]$ satisfying Condition 2. By induction, this implies that $\vec{h}^{(i)}$ is a $(d-1, b)$ -hyporuler, so we are done. $\square$

The following lemma says that if we sample $H_1, H_2, \dots$ with geometrically decreasing probability distributions, then they will probably form a long hyporuler.

Lemma 3.4 (Generating hyporulers). *Let $d \geq 0$ and $b \geq 1$. Let $H_1, \dots, H_N$ be independent and identically distributed random variables, where $\Pr[H_i = h] = 2^{-h}$ for every $h \in [d]$. Then*

$$\Pr[(H_1, \dots, H_N) \text{ is not a } (d, b)\text{-hyporuler}] \leq N \cdot (2^{-d} + 2^{-(b-1)}).$$

Proof. For each $j \in [N]$, we have

$$\Pr[H_j \notin [d]] = 1 - \sum_{h=1}^d 2^{-h} = 2^{-d}.$$

By the union bound, the probability that any H_j is not in $[d]$ is at most $N \cdot 2^{-d}$.

Now we turn to Condition 2 in [Lemma 3.3](#). Fix $j \in [N]$ and any value $h_j \in [d]$ for H_j . For each $i < j$, we have

$$\Pr[H_i \notin [h_j]] = 1 - \sum_{h=1}^{h_j} 2^{-h} = 2^{-h_j} = \Pr[H_i = h_j].$$

Think of sampling in the order $H_{j-1}, H_{j-2}, H_{j-3}, \dots, H_1$. The probability that the event “ $H_i = h_j$ ” occurs at least $b-1$ times before any occurrence of the event “ $H_i \notin [h_j]$ ” is at most $2^{-(b-1)}$, because the two events have equal probabilities. The lemma follows by a union bound over the N different values for j . $\square$

It is important that a hyporuler cannot go on forever. The following lemma gives a quantitative length bound.

Lemma 3.5 (Hyporuler length bound). *Let $d \geq 0$, let $b \geq 1$, and let $\vec{h} \in \mathbb{N}^N$. If $\vec{h}$ is a (d, b) -hyporuler, then $N \leq b^d - 1$.*

Proof. If $d = 0$, the statement is trivial. If $d > 0$, then by the definition of hyporulers and induction, we have $N \leq b - 1 + b \cdot (b^{d-1} - 1) = b^d - 1$. $\square$

3.2 Hyperrulers

Our algorithm for computing $f^{(n)}(x)$, where f is computable in $\text{BP}_{\text{H}}^*\text{L}$, uses the notion of a *hyperruler*.

Definition 3.6 (Hyperruler). Let $d \geq 0$ and $b \geq 1$. A (d, b) -hyperruler is a sequence of the form

$$\vec{h} = (\vec{h}^{(1)}, d, \vec{h}^{(2)}, d, \dots, d, \vec{h}^{(b)}, \vec{z}),$$

where each $\vec{h}^{(i)}$ is a $(d-1, b)$ -hyperruler and $\vec{z}$ is an arbitrary sequence of elements of $\{0, 1, \dots, d\}$. In the base case, when $d = 0$, a $(0, b)$ -hyperruler is an arbitrary-length sequence of zeroes.

The following lemma says that if we sample $H_1, H_2, \dots$ from a geometric distribution, then they will probably form a long hyperruler.

Lemma 3.7 (Generating hyperrulers). *Let $0 \leq d < M$, $b \geq 2$, and $\delta \in (0, 1/2]$. Let $H_1, H_2, H_3, \dots$ be independent and identically distributed nonnegative integers, where $\Pr[H_i \geq h] = \delta^h$ for each $h \in \{0, 1, \dots, d+1\}$. Let N be the first value such that $H_N > d$. Then except with probability $2\delta \cdot b^d$, the sequence $(H_1, H_2, \dots, H_{N-1})$ is a (d, b) -hyperruler.*

Proof. We prove it by induction on d with a failure probability bound of

$$\delta \cdot (b^1 + b^2 + \dots + b^d).$$

In the base case, when $d = 0$, this is trivial. Now assume $d > 0$. For each j , we have

$$\Pr[H_j \geq d+1 \mid H_j \geq d] = \frac{\delta^{d+1}}{\delta^d} = \delta.$$

Therefore, the probability of getting a value larger than d before we get $b-1$ occurrences of d is at most $\delta \cdot (b-1)$. By induction, in each region leading up to one of the first b occurrences of values that are at least d , the probability of not getting a $(d-1, b)$ -hyperruler is at most $\delta \cdot (b^1 + \dots + b^{d-1})$. Therefore, by the union bound, the overall failure probability is

$$\delta \cdot (b-1) + b \cdot \delta \cdot (b^1 + \dots + b^{d-1}) \leq \delta \cdot (b^1 + \dots + b^d). \quad \square$$

4 Simulating PP in the $\text{BP}_{\text{H}}^*\text{L}$ Model

In this section, we prove $\text{PP} \subseteq \text{BP}_{\text{H}}^*\text{L}$. As discussed in [Section 1.4](#), this is the first step of the proof that $\text{P}^{\#P} \subseteq \text{BP}_{\text{H}}^*\text{L}$.

Theorem 4.1. $\text{PP} \subseteq \text{BP}_{\text{H}}^*\text{L}$.

Proof. We will prove $\text{MAJSAT} \in \text{BP}_{\text{H}}^*\text{L}$. (Recall that MAJSAT is the problem of determining whether a given n -variable CNF formula has at least 2^{n-1} distinct satisfying assignments.) The theorem will follow, because MAJSAT is PP-complete with respect to deterministic, log-space reductions [\[Gil77\]](#).

The algorithm. Suppose we are given an n -variable CNF formula ϕ . Let R be the contents of our random tape, and write R in the form

$$R = (X^{(1)}, Y^{(1)}, X^{(2)}, Y^{(2)}, \dots),$$

where $X^{(j)} \in \{0, 1\}^n$ and $Y^{(j)} \in \{0, 1\}^{4n}$ for every j . For each j , let H_j be the location of the first “1” in $Y^{(j)}$, or $H_j = \infty$ if $Y^{(j)}$ is all zeroes, so $H_j \in [4n] \cup \{\infty\}$. Let N be the first value such that $(H_1, H_2, \dots, H_N)$ is not a $(4n, 4n)$ -hyporuler; such an N exists by [Lemma 3.5](#). Let

$$S = \sum_{j=1}^N \phi(X^{(j)}).$$

If $2^{n+2} \cdot S \geq (2^{n+1} - 1) \cdot N$, then halt and accept. Otherwise, halt and reject.

Correctness. By [Lemma 3.4](#), we have

$$\Pr[N < 2^{3n}] \leq 2^{3n} \cdot (2^{-4n} + 2^{-(4n-1)}) = o(1).$$

Fix any choice of $Y^{(1)}, Y^{(2)}, \dots$ such that $N \geq 2^{3n}$. Then by Hoeffding’s inequality, with respect to the random choice of $X^{(1)}, \dots, X^{(N)}$, we have

$$\Pr \left[\left| \frac{S}{N} - \frac{|\phi^{-1}(1)|}{2^n} \right| > 2^{-(n+2)} \right] \leq 2 \exp(-2 \cdot 2^{-2 \cdot (n+2)} \cdot 2^{3n}) = o(1).$$

Fix any choice of $X^{(1)}, X^{(2)}, \dots, X^{(N)}$ such that the event above does not occur. Then:

- If $\phi \in \text{MAJSAT}$, then $|\phi^{-1}(1)| \geq 2^{n-1}$, so $S/N \geq 1/2 - 2^{-(n+2)}$, so $2^{n+2} \cdot S \geq N \cdot (2^{n+1} - 1)$ and the algorithm accepts.
- If $\phi \notin \text{MAJSAT}$, then $|\phi^{-1}(1)| \leq 2^{n-1} - 1$, so $S/N \leq 1/2 - 2^{-n} + 2^{-(n+2)} < 1/2 - 2^{-(n+2)}$, so $2^{n+2} \cdot S < N \cdot (2^{n+1} - 1)$ and the algorithm rejects.

Efficiency. We compute S and N using the Chinese remainder representation. Let $p_1 < p_2 < \dots < p_{8n}$ be the first $8n$ prime numbers. We can find and store p_i using $O(\log n)$ bits of space, since $p_{8n} = O(n \cdot \log n)$ by the Prime Number Theorem. We can compute $N \bmod p_i$ as follows, based on [Lemma 3.3](#):

1. Initialize $c \leftarrow 0$.
2. For $j = 1, 2, 3, \dots$:
 - (a) Increment $c \leftarrow c + 1 \bmod p_i$.
 - (b) If $H_j = \infty$, then $c = N \bmod p_i$ and we are done. Otherwise:
 - (c) Store H_j in our work tape. Scan backward over the random tape, comparing each H_i to H_j . Each time we encounter an i such that $H_i = H_j$, increment a counter q in our workspace.
 - (d) If q reaches $4n$, then $c = N \bmod p_i$ and we are done. If, instead, we encounter some i such that $H_i > H_j$ or the beginning of the random tape, we need to move on to the next value of j , but first we must restore the random tape head location. To do so, scan forward over the random tape. Each time we encounter an i such that $H_i = H_j$, decrement q . Finally, when q reaches zero, we are ready to move on to the next value of j .

We can compute $S \bmod p_i$ similarly, with one change: for each j , we evaluate $\phi(X^{(j)})$ in $O(\log |\phi|)$ space, and we only increment c if $\phi(X^{(j)}) = 1$.

We have

$$\prod_{i=1}^{8n} p_i \geq \prod_{i=4n+1}^{8n} p_i > (4n)^{4n} \geq N$$

by Lemma 3.5. Therefore, the values $(N \bmod p_1, \dots, N \bmod p_{8n})$ uniquely determine N by the Chinese Remainder Theorem, and similarly $(S \bmod p_1, S \bmod p_2, \dots, S \bmod p_{8n})$ uniquely determine S . Furthermore, there are log-space algorithms for converting a nonnegative integer from Chinese remainder representation to standard binary representation [CDL01; HAB02], so we can recover each bit of N and S from their residues modulo $p_1, \dots, p_{8n}$.³ Finally, given that we can compute the bits of S and N , we can determine whether $2^{n+2} \cdot S \geq (2^{n+1} - 1) \cdot N$ in $O(\log n)$ additional space using standard grade-school algorithms. $\square$

5 BP_{HL}^* Is Closed Under Polynomial-Time Turing Reductions

In this section, we prove $\text{P}^{\text{BP}_{\text{HL}}^*} = \text{BP}_{\text{HL}}^*$. The proof has several steps.

- First, in Section 5.1, we bound the space and time complexity of amplification in the BP_{HL}^* model.
- Next, in Section 5.2, we show how to compute the t -fold composition $f^{(t)}(x)$ for any length-preserving function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ that is computable in BP_{HL}^* , where t is polynomially large. Annoyingly, our algorithm isn't guaranteed to halt, but the good news is that our algorithm uses $O(\log n)$ bits of space and uses only $2^{\text{poly}(n)}$ multi-access random bits with high probability.
- Then, in Section 5.3, we show that any algorithm with the features described above can be converted into a genuine BP_{HL}^* algorithm.
- Finally, in Section 5.4, we put the pieces together to conclude that $\text{P}^{\text{BP}_{\text{HL}}^*} = \text{BP}_{\text{HL}}^*$.

5.1 Time Complexity and Amplification in the BP_{HL}^* Model

Lemma 5.1 (Amplification in the BP_{HL}^* model). *Let $L \in \text{BP}_{\text{HL}}^*$. There is a randomized Turing machine M with a multi-access random tape such that:*

- *If $x \in L$ and $k \in \mathbb{N}$, then $\Pr[M \text{ accepts } (x, k)] \geq 1 - 2^{-k}$.*
- *If $x \notin L$ and $k \in \mathbb{N}$, then $\Pr[M \text{ accepts } (x, k)] \leq 2^{-k}$.*
- *For every input (x, k) and every setting of the random tape, M halts within T time steps and uses at most S bits of work space, where*

$$T = \text{poly}(k) \cdot 2^{\text{poly}(|x|)}$$

$$S = O(\log |x| + \log k).$$

³A more elementary approach is to guess the numbers N and S randomly and verify the guesses by comparing values mod $p_1, p_2, \dots, p_{8n}$. There is only a $2^{-\Theta(n \log n)}$ chance of success, but we can repeat the entire experiment $2^{\Theta(n \log n)}$ times until it succeeds, using two levels of probabilistic timers.

Proof. Let M_0 be the machine witnessing $L \in \text{BP}_{\text{HL}}^*$. We do $k' = \Theta(k)$ repetitions of M_0 and take a majority vote. For repetition i , we use the random bits located at positions congruent to $i \bmod k'$ on M 's random tape. The error probability and space complexity are clear. To complete the proof, we need to show that M_0 halts within $2^{\text{poly}(n)}$ time on n -bit inputs.

For every n -bit input x , there is a deterministic two-way automaton A_x with $\text{poly}(n)$ states that simulates M_0 's behavior on x , given the contents of the random tape. Every deterministic q -state two-way automaton can be simulated by a q' -state deterministic one-way automaton where $q' = 2^{O(q \cdot \log q)}$ [Rab57; She59; Kap05]. Furthermore, a q' -state deterministic one-way automaton that always halts must halt within q' steps. (Otherwise, by the pigeonhole principle, it would repeat a state, e.g., it would be in state u both after reading y and after reading yz . But in this case, it would run forever on the input $yzzz \dots$) Therefore, M_0 visits at most $2^{\text{poly}(n)}$ cells on its random tape. Finally, M_0 can only visit each cell of the random tape at most $\text{poly}(n)$ times, because otherwise, by another application of the pigeonhole principle, M_0 would repeat a configuration and get stuck in an infinite loop. Therefore, the time complexity of M_0 is at most $2^{\text{poly}(n)} \cdot \text{poly}(n) = 2^{\text{poly}(n)}$. $\square$

5.2 Composing BP_{HL}^* Algorithms

Lemma 5.2 (Composing BP_{HL}^* algorithms). *Let $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$, and assume f is length-preserving. Define*

$$\begin{aligned} L &= \{(x, i) : f(x)_i = 1\} \\ L' &= \{(x, 1^t, i) : f^{(t)}(x)_i = 1\}, \end{aligned} \tag{2}$$

where $f^{(t)}$ denotes the t -fold composition of f with itself and y_i denotes the i -th bit of y . Assume $L \in \text{BP}_{\text{HL}}^*$. Then there is a randomized log-space Turing machine M with a multi-access random tape such that:

- Given any n -bit $(x, 1^t, i) \in L'$, except with probability 0.1, M accepts after visiting at most $2^{\text{poly}(n)}$ cells of its random tape.
- Given any n -bit $(x, 1^t, i) \notin L'$, except with probability 0.1, M rejects after visiting at most $2^{\text{poly}(n)}$ cells of its random tape.

Proof. By Lemma 5.1, we can compute each bit of $f(x)$ with error 2^{-k} in space $O(\log |x| + \log k)$ and time $T = \text{poly}(k) \cdot 2^{\text{poly}(|x|)}$. Because T grows slowly with k , we can fix a value $k = \text{poly}(|x|, t)$ such that $(T + 1)^{2t} < 0.01 \cdot 2^k$. Note that the space complexity of computing $f(x)$ with error 2^{-k} is $O(\log |x| + \log t)$ and the time complexity is $T = \text{poly}(t) \cdot 2^{\text{poly}(|x|)}$. Now let us show how to compute $f^{(t)}(x)$.

Let R be the contents of our random tape. Write R as

$$R = (X_1, Y^{(1)}, X_2, Y^{(2)}, \dots),$$

where $X_i \in \{0, 1\}$ and $Y^{(i)} \in \{0, 1\}^{(2t+1) \cdot k}$ for each i . Define the “rank” H_i to be the maximum $h \in \{0, 1, \dots, 2t + 1\}$ such that the first hk bits of $Y^{(i)}$ are zero. Let $\delta = 2^{-k}$, and observe that $\Pr[H_i \geq h] = \delta^h$ for each $h \in \{0, 1, \dots, 2t + 1\}$.

The algorithm. Suppose r and s are currently stored in our work tape, and our current goal is to compute the bit $f^{(r)}(x)_s$. If $r = 0$, we have $f^{(0)}(x)_s = x_s$. More interestingly, suppose $r > 0$. We simulate the error- δ computation of f . We use our work tape to store the simulated work tape, the

internal state of the simulated machine, and the locations of the simulated input and work tape heads. For randomness, we use the bits X_i on the random tape such that $H_i = 2r$. When the simulation of f moves its random tape head to the right or left, we scan to the right or left to reach the next/previous i such that $H_i = 2r$. If we are scanning left and we reach i such that $H_i > 2r$, we treat it as the left end of the simulated random tape. If we are scanning right and we reach i such that $H_i > 2r$, we halt and output “error.” (See Fig. 3.)

Eventually, the simulation of f may query some position of its input, say position s' . We pause the simulation of f . We encode its current configuration (simulated work tape contents, internal state, locations of input tape and work tape heads, and the value s) as a polynomially bounded number Q . Then, in order to effectively “push” Q onto a simulated stack, we scan our random tape head to the right, counting until we have observed Q distinct values i such that $H_i = 2r - 1$. (If, during this process, we reach a value i such that $H_i \geq 2r$, we halt and output “error.”) Then we erase Q from our work tape and start a new simulation of f , with r replaced with $r - 1$ and s replaced with s' . We run this simulation as before, continuing the recursion.

Eventually, our simulation of f will halt, outputting a value $f^{(r)}(x)_s = b$. At this point, if $r = t$ and $s = i$, we can simply output b and be done. On the other hand, if $r < t$, then we need to give this value b to an outer simulation of f that is currently paused. We therefore scan our random tape head to the left, counting how many values i we see such that $H_i = 2r + 1$ before we arrive at a location i such that $H_i \geq 2r + 2$. Let Q be the number of occurrences of $2r + 1$ we observed. Then Q represents the configuration of the paused outer simulation of f . We use Q to restore the outer simulation, we provide it with the value b from the inner simulation, and we continue executing the outer simulation.

Space complexity. We only store $O(\log |x| + \log t)$ bits of information on our work tape: r is $O(\log t)$ bits, s is $O(\log |x|)$ bits, and Q is $O(\log |x| + \log t)$ bits.

Correctness. There are two “bad events” that can happen.

- Let N be the first value such that $H_N = 2t + 1$. The sequence $\vec{H} = (H_1, H_2, \dots, H_{N-1})$ might not be a $(2t, T + 1)$ -hyperruler. By Lemma 3.7, this happens with probability at most $2\delta \cdot (T + 1)^{2t} < 0.02$. Fix any choice of $Y^{(1)}, Y^{(2)}, \dots$ such that $\vec{H}$ is a $(2t, T + 1)$ -hyperruler.
- One of the f computations might output the incorrect answer. To bound the probability of this event, note that since $\vec{H}$ is a $(2t, T + 1)$ -hyperruler, we can write it as $\vec{H} = (\vec{H}^{(1)}, 2t, \vec{H}^{(2)}, 2t, \dots, 2t, \vec{H}^{(T+1)}, \vec{Z})$, where each $\vec{H}^{(i)}$ is a $(2t - 1, T + 1)$ -hyperruler. Let $i_1, i_2, \dots, i_T$ be the locations of the first T occurrences of $2t$. Recursively define

$$I_{2t}(\vec{H}) = \{(i_1, i_2, \dots, i_T)\} \cup \bigcup_{j=1}^T I_{2t-1}(\vec{H}^{(j)}),$$

where $I_0(\vec{H}) = \emptyset$ and we always use “global” locations (measuring from the beginning of the random tape, not the beginning of some $\vec{H}^{(i)}$ subsequence). If one of the f computations outputs the incorrect answer, then there is some $r \in [t]$, some $s \in [|x|]$, and some sequence $(i'_1, i'_2, \dots, i'_T) \in I_{2t}(\vec{H})$ such that $H_{i'_1} = H_{i'_2} = \dots = H_{i'_T} = 2r$, and the computation of f fails when the input is the true value $f^{(r-1)}(x)$, the randomness is $(X_{i'_1}, \dots, X_{i'_T})$, and the desired output bit is position s . By the union bound, the probability of this event is at most

$$\delta \cdot |I_{2t}(\vec{H})| \cdot |x| = \delta \cdot (1 + T + T^2 + \dots + T^{2t-1}) \cdot T = \delta T \cdot \frac{T^{2t}}{T - 1} \leq \delta \cdot (T + 1)^{2t} < 0.01.$$

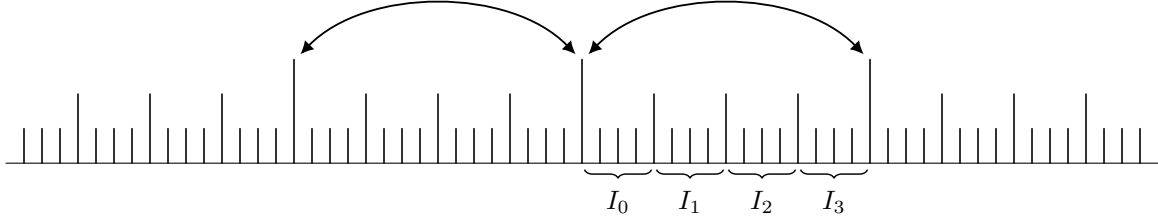


Figure 3: Composing BP_H^*L algorithms. We use the high-rank positions on the random tape to simulate the outer algorithm's random tape, shown here with curved arrows. When the outer algorithm makes a query to its input, we pause it and encode its configuration as a number Q . Then we move to the right to find the corresponding interval I_Q and we erase the value Q from our work tape. We simulate the inner algorithm entirely within the interval I_Q . When the inner algorithm finishes, we count backward to recover the number Q , allowing us to restore the configuration of the outer algorithm and continue simulating it. In this diagram, we pretend that the ranks form an exact ruler for simplicity, but in reality they will form a hyperruler.

Assuming neither of the bad events occurs, our algorithm M outputs the correct answer. (Assume without loss of generality that $T > Q_{\max}$, where $Q_{\max}$ is a polynomial bound on any Q value that might occur during the computation.)

Randomness complexity. The number of cells of the random tape that we visit is at most $\text{poly}(n) \cdot N$. We have $\mathbb{E}[N] = (1/\delta)^{2t+1} = 2^{\text{poly}(n)}$. Markov's inequality completes the proof. $\square$

There are two reasons that the algorithm M described above might not halt.

- The head might scan to the right indefinitely if we get exceedingly unlucky values for H_i , e.g., all zeroes. This issue is minor, and could be resolved using a hyporuler timer, just like in the proof of $\text{PP} \subseteq \text{BP}_H^*L$.
- The inner computations might give inconsistent responses to the outer computations' input queries. This could potentially cause an outer computation to get stuck in an infinite loop.⁴ This issue is more serious, but we will show how to resolve it in the next subsection.

5.3 Converting Exponential-Randomness Algorithms Into Halting Algorithms

Lemma 5.3. *Let L be a language. Assume there exists a randomized log-space Turing machine M with a multi-access random tape such that:*

- *Given any n -bit $x \in L$, except with probability 0.1, M accepts after visiting at most $2^{\text{poly}(n)}$ cells of its random tape.*
- *Given any n -bit $x \notin L$, except with probability 0.1, M rejects after visiting at most $2^{\text{poly}(n)}$ cells of its random tape.*

*Then $L \in \text{BP}_H^*L$.*

⁴Theoretically, it could also potentially cause the outer computation to use more than $O(\log n)$ bits of work space, but we can easily modify the f algorithm to ensure that it uses $O(\log n)$ bits of work space even if it receives inconsistent answers to input queries.

Proof. Without loss of generality, we assume that M has a unique accepting configuration, e.g., with an empty work tape with each head at the beginning of its tape. To design a $\text{BP}_H^* \text{L}$ algorithm for L , we use a technique due to Sipser [Sip80], combined with the “hyporuler” trick we used in the proof that $\text{PP} \subseteq \text{BP}_H^* \text{L}$ (Theorem 4.1). Let $x \in \{0, 1\}^n$ be the input. Let c be a constant that is large enough that with probability 0.9, M outputs the correct answer on x after visiting at most 2^{n^c} cells of its random tape. Let R be the contents of our random tape. Write R in the form

$$R = (X_1, Y^{(1)}, X_2, Y^{(2)}, \dots),$$

where $X_j \in \{0, 1\}$ and $Y^{(j)} \in \{0, 1\}^{n^c+4}$ for every j . We will use $X = (X_1, X_2, X_3, \dots)$ as the contents of M ’s simulated random tape. Let H_j be the location of the first “1” in $Y^{(j)}$, or $H_j = \infty$ if $Y^{(j)}$ is all zeroes, so $H_j \in [n^c + 4] \cup \{\infty\}$. Let N be the smallest number such that $(H_1, H_2, \dots, H_N)$ is not a $(n^c + 4, n^c + 5)$ -hyporuler. Such an N exists by Lemma 3.5.

Let V be the set of all possible configurations of M on input x and random tape X , where “configuration” means the internal state of M , the contents of its work tape, and the locations of all its heads. Let $G = (V, E)$ be the computation digraph, i.e., an edge (u, v) means that v is the unique successor configuration of u . Let u_{acc} be the unique accepting configuration.

Let G' be the induced subgraph of G consisting of all configurations u in which M ’s random tape head location is less than N . Let G'' be the induced subgraph of G' consisting of all vertices u from which u_{acc} is reachable. This graph G'' is a tree, rooted at u_{acc} , in which edges are oriented from child to parent.

The algorithm. Do a depth-first search through G'' , starting at the root node u_{acc} . If we ever find the initial configuration u_{start} of M , we accept. If we complete the depth-first search without ever finding u_{start} , we reject.

Halting. The algorithm always halts, because the tree G'' is finite, since we require random tape head location to be less than N .

Correctness. We consider two “bad events.”

- It might be that M fails on input x and random tape X , in the sense that it outputs the wrong answer, or it never halts, or it visits more than 2^{n^c} cells of its random tape. By assumption, the probability of this bad event is at most 0.1.
- We might have $N \leq 2^{n^c}$. By Lemma 3.4, the probability of this bad event is at most $2^{n^c} \cdot (2^{-(n^c+4)} + 2^{-(n^c+4)}) \leq 1/8$.

If neither of these bad events occurs, then our algorithm outputs the correct answer.

Efficiency. To implement the depth-first search, we need to keep track of what vertex $v \in V$ we are currently visiting. The vertex v encodes the internal state of M , the contents of its work tape, and the locations of all its heads. We use our work tape to store everything except the location of its random tape head. If M ’s random tape head is located at position j , then we ensure that our random tape head is located at position $(j - 1) \cdot (n^c + 5) + 1$, so it is currently observing the symbol X_j .

To traverse an edge of G'' forward, we simulate a single step of M . To traverse an edge backward, we consider all candidate predecessors, and simulate a single step of M forward from each one to determine which candidate predecessors are actual predecessors. In both cases, to ensure that we

do not leave the subgraph G'' , we must check whether M 's simulated random tape head location j is less than N . We do so using the same method we used in the proof of [Theorem 4.1](#) to check for hyporulers. We repeat the method here for convenience:

- We check whether $H_j > n^c + 4$.
- We store H_j in our work tape and scan backward over the random tape and check whether we see $n^c + 5$ distinct i such that $H_i = H_j$ before we see an i such that $H_i > H_j$. This tells us whether $(H_1, \dots, H_j)$ is a $(n^c + 4, n^c + 5)$ -hyporuler. Then, to restore the random tape head location, we scan forward over the tape until we have seen the appropriate number of i such that $H_i = H_j$.

□

5.4 Simulating a $\text{BP}_{\text{H}}^*\text{L}$ Oracle

Theorem 5.4. $\text{BP}_{\text{H}}^*\text{L}$ is closed under polynomial-time Turing reductions. That is, $\text{P}^{\text{BP}_{\text{H}}^*\text{L}} = \text{BP}_{\text{H}}^*\text{L}$.

Proof. Clearly, $\text{BP}_{\text{H}}^*\text{L} \subseteq \text{P}^{\text{BP}_{\text{H}}^*\text{L}}$. To prove the reverse inclusion, we can encode a configuration z of the $\text{P}^{\text{BP}_{\text{H}}^*\text{L}}$ machine on an n -bit input as a string of length $\ell(n) = \text{poly}(n)$. For the purposes of this proof, the “configuration” includes the input, the contents of all the tapes, the locations of all the heads, and the internal state of the machine. For each configuration z , let $f(z)$ be the “next” configuration, i.e., f simulates a single step of the $\text{P}^{\text{BP}_{\text{H}}^*\text{L}}$ computation. By our choice of encoding, f is length-preserving. Furthermore, the language L in [Eq. \(2\)](#) is in $\text{BP}_{\text{H}}^*\text{L}$: a non-query step can be computed in deterministic log-space, and a query step is clearly a $\text{BP}_{\text{H}}^*\text{L}$ computation. We can simulate the whole $\text{P}^{\text{BP}_{\text{H}}^*\text{L}}$ computation by computing the appropriate bit of $f^{(\text{poly}(n))}(z_0)$, where z_0 is the initial configuration, which we can do in $\text{BP}_{\text{H}}^*\text{L}$ by [Lemmas 5.2](#) and [5.3](#). □

Proof of Theorem 1.1. We have $\text{P}^{\#P} = \text{P}^{\text{PP}} \subseteq \text{P}^{\text{BP}_{\text{H}}^*\text{L}} = \text{BP}_{\text{H}}^*\text{L}$, where the first equation is a standard exercise [[Gil77](#)], the inclusion is by [Theorem 4.1](#), and the last equation is by [Theorem 5.4](#). □

AI Disclosure

I used GPT-6 Astra to generate initial versions of proofs, simplify proofs, catch and fix bugs, generate figures, and help with writing. I take responsibility for every part of the paper.

Acknowledgments

I thank OpenAI for giving me complimentary access to ChatGPT Pro. I thank Alicia Torres Hoza for helpful comments on a draft of this paper.

References

- [Aar05] Scott Aaronson. “Quantum computing, postselection, and probabilistic polynomial-time”. In: *Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.* 461.2063 (2005), pp. 3473–3482. DOI: [10.1098/rspa.2005.1546](#).
- [AO96] Eric Allender and Mitsunori Ogihara. “Relationships among PL, #L, and the determinant”. In: *RAIRO Inform. Théor. Appl.* 30.1 (1996), pp. 1–21. DOI: [10.1051/ita/1996300100011](#).

- [BCP83] A. Borodin, S. Cook, and N. Pippenger. “Parallel computation for well-endowed rings and space-bounded probabilistic machines”. In: *Inform. and Control* 58.1-3 (1983), pp. 113–136. DOI: [10.1016/S0019-9958\(83\)80060-6](https://doi.org/10.1016/S0019-9958(83)80060-6).
- [BNS92] László Babai, Noam Nisan, and Mária Szegedy. “Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs”. In: *J. Comput. System Sci.* 45.2 (1992), pp. 204–232. DOI: [10.1016/0022-0000\(92\)90047-M](https://doi.org/10.1016/0022-0000(92)90047-M).
- [BV97] Ethan Bernstein and Umesh Vazirani. “Quantum complexity theory”. In: *SIAM J. Comput.* 26.5 (1997), pp. 1411–1473. DOI: [10.1137/S0097539796300921](https://doi.org/10.1137/S0097539796300921).
- [BVW98] Ronald V. Book, Heribert Vollmer, and Klaus W. Wagner. “Probabilistic type-2 operators and ‘almost’-classes”. In: *Comput. Complexity* 7.3 (1998), pp. 265–289. DOI: [10.1007/s000370050012](https://doi.org/10.1007/s000370050012).
- [CDL01] Andrew Chiu, George Davida, and Bruce Litow. “Division in logspace-uniform NC¹”. In: *Theor. Inform. Appl.* 35.3 (2001), pp. 259–275. DOI: [10.1051/ita:2001119](https://doi.org/10.1051/ita:2001119).
- [CW24] Kuan Cheng and Yichuan Wang. “BPL $\subseteq$ L-AC¹”. In: *Proceedings of the 39th Computational Complexity Conference (CCC)*. 2024, 32:1–32:14. DOI: [10.4230/LIPIcs.CCC.2024.32](https://doi.org/10.4230/LIPIcs.CCC.2024.32).
- [DMPV16] Peter Dixon, Debasis Mandal, A. Pavan, and N. V. Vinodchandran. “A Note on the Advice Complexity of Multipass Randomized Logspace”. In: *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*. 2016, 31:1–31:7. DOI: [10.4230/LIPIcs.MFCS.2016.31](https://doi.org/10.4230/LIPIcs.MFCS.2016.31).
- [DPS11] Matei David, Periklis A. Papakonstantinou, and Anastasios Sidiropoulos. “How strong is Nisan’s pseudo-random generator?” In: *Inform. Process. Lett.* 111.16 (2011), pp. 804–808. DOI: [10.1016/j.ipl.2011.04.013](https://doi.org/10.1016/j.ipl.2011.04.013).
- [DPT24] Dean Doron, Edward Pyne, and Roei Tell. “Opening up the distinguisher: a hardness to randomness approach for BPL = L that uses properties of BPL”. In: *Proceedings of the 56th Symposium on Theory of Computing (STOC)*. 2024, pp. 2039–2049. DOI: [10.1145/3618260.3649772](https://doi.org/10.1145/3618260.3649772).
- [DPTW25] Dean Doron, Edward Pyne, Roei Tell, and R. Ryan Williams. “When connectivity is hard, random walks are easy with non-determinism”. In: *Proceedings of the 57th Symposium on Theory of Computing (STOC)*. 2025, pp. 1108–1117. DOI: [10.1145/3717823.3718303](https://doi.org/10.1145/3717823.3718303).
- [DT23] Dean Doron and Roei Tell. “Derandomization with Minimal Memory Footprint”. In: *Proceedings of the 38th Computational Complexity Conference (CCC)*. 2023, 11:1–11:15. DOI: [10.4230/LIPIcs.CCC.2023.11](https://doi.org/10.4230/LIPIcs.CCC.2023.11).
- [FF07] Abraham D. Flaxman and Alan M. Frieze. “The diameter of randomly perturbed digraphs and some applications”. In: *Random Structures Algorithms* 30.4 (2007), pp. 484–504. DOI: [10.1002/rsa.20172](https://doi.org/10.1002/rsa.20172).
- [Gil77] John Gill. “Computational complexity of probabilistic Turing machines”. In: *SIAM J. Comput.* 6.4 (1977), pp. 675–695. DOI: [10.1137/0206049](https://doi.org/10.1137/0206049).
- [HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. “Uniform constant-depth threshold circuits for division and iterated multiplication”. In: *J. Comput. System Sci.* 65.4 (2002), pp. 695–716. DOI: [10.1016/S0022-0000\(02\)00025-9](https://doi.org/10.1016/S0022-0000(02)00025-9).

- [Hoz21] William M. Hoza. “Better Pseudodistributions and Derandomization for Space-Bounded Computation”. In: *Proceedings of the 25th International Conference on Randomization and Computation (RANDOM)*. 2021, 28:1–28:23. DOI: [10.4230/LIPIcs.APPROX/RANDOM.2021.28](https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.28).
- [Hoz22] William M. Hoza. “Recent progress on derandomizing space-bounded computation”. In: *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS* 138 (2022), pp. 114–143. URL: <http://bulletin.eatcs.org/index.php/beatcs/article/view/728>.
- [Kap05] Christos Kapoutsis. “Removing bidirectionality from nondeterministic finite automata”. In: *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Springer, Berlin, 2005, pp. 544–555. DOI: [10.1007/11549345_47](https://doi.org/10.1007/11549345_47).
- [KSY14] Uğur Küçük, A. C. Cem Say, and Abuzer Yakaryılmaz. “Finite automata with advice tapes”. In: *Internat. J. Found. Comput. Sci.* 25.8 (2014), pp. 987–1000. DOI: [10.1142/S012905411440019X](https://doi.org/10.1142/S012905411440019X).
- [KV85] Marek Karpinski and Rutger Verbeek. “There is no polynomial deterministic space simulation of probabilistic space with a two-way random-tape generator”. In: *Inform. and Control* 67.1-3 (1985), pp. 158–162. DOI: [10.1016/S0019-9958\(85\)80032-2](https://doi.org/10.1016/S0019-9958(85)80032-2).
- [KV86] Marek Karpinski and Rutger Verbeek. “On the power of two-way random generators and the impossibility of deterministic polyspace simulation”. In: *Inform. and Control* 71.1-2 (1986), pp. 131–142. DOI: [10.1016/S0019-9958\(86\)80020-1](https://doi.org/10.1016/S0019-9958(86)80020-1).
- [KvM02] Adam R. Klivans and Dieter van Melkebeek. “Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses”. In: *SIAM J. Comput.* 31.5 (2002), pp. 1501–1526. DOI: [10.1137/S0097539700389652](https://doi.org/10.1137/S0097539700389652).
- [MP26] David Miloschewsky and Supartha Podder. *Modifications of Quantum Computation and Adaptive Queries to PP*. 2026. DOI: [10.48550/arXiv.2507.03692](https://doi.org/10.48550/arXiv.2507.03692).
- [MPV15] Debasis Mandal, A. Pavan, and N. V. Vinodchandran. “On Probabilistic Space-Bounded Machines with Multiple Access to Random Tape”. In: *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. 2015, pp. 459–471. DOI: [10.1007/978-3-662-48054-0_38](https://doi.org/10.1007/978-3-662-48054-0_38).
- [Nis93] Noam Nisan. “On read-once vs. multiple access to randomness in logspace”. In: *Theoret. Comput. Sci.* 107.1 (1993), pp. 135–144. ISSN: 0304-3975,1879-2294. DOI: [10.1016/0304-3975\(93\)90258-U](https://doi.org/10.1016/0304-3975(93)90258-U).
- [Nis94] Noam Nisan. “ $RL \subseteq SC$ ”. In: *Comput. Complexity* 4.1 (1994), pp. 1–11. ISSN: 1016-3328,1420-8954. DOI: [10.1007/BF01205052](https://doi.org/10.1007/BF01205052).
- [PR23] Rafael Pass and Oren Renard. *Characterizing the Power of (Persistent) Randomness in Log-space*. ECCC preprint TR23-211. 2023. URL: <https://eccc.weizmann.ac.il/report/2023/211/>.
- [PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. “Certified hardness vs. randomness for log-space”. In: *Proceedings of the 64th Symposium on Foundations of Computer Science (FOCS)*. 2023, pp. 989–1007. DOI: [10.1109/FOCS57990.2023.00061](https://doi.org/10.1109/FOCS57990.2023.00061).
- [PT25] Edward Pyne and Roei Tell. *Composing Low-Space Algorithms*. ECCC preprint TR25-140. 2025. URL: <https://eccc.weizmann.ac.il/report/2025/140/>.

- [PT26] Edward Pyne and Roei Tell. “Using hardness vs randomness to design low-space algorithms”. In: *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS* 148 (2026), pp. 38–76. URL: <https://bulletin.eatcs.org/index.php/beatcs/article/view/880>.
- [Pug90] William Pugh. “Skip lists: a probabilistic alternative to balanced trees”. In: *Commun. ACM* 33.6 (1990), 668–676. DOI: [10.1145/78973.78977](https://doi.org/10.1145/78973.78977).
- [Rab57] M. O. Rabin. “Two-Way Finite Automata”. In: *Proceedings of the Summer Institute of Symbolic Logic*. 1957, pp. 366–369.
- [Rav07] Bala Ravikumar. “On some variations of two-way probabilistic finite automata models”. In: *Theoret. Comput. Sci.* 376.1-2 (2007), pp. 127–136. DOI: [10.1016/j.tcs.2007.01.017](https://doi.org/10.1016/j.tcs.2007.01.017).
- [RST84] Walter L. Ruzzo, Janos Simon, and Martin Tompa. “Space-bounded hierarchies and probabilistic computations”. In: *J. Comput. System Sci.* 28.2 (1984), pp. 216–230. DOI: [10.1016/0022-0000\(84\)90066-7](https://doi.org/10.1016/0022-0000(84)90066-7).
- [Sak96] M. Saks. “Randomization and derandomization in space-bounded computation”. In: *Proceedings of the 11th Computational Complexity Conference (CCC)*. 1996, pp. 128–149. DOI: [10.1109/CCC.1996.507676](https://doi.org/10.1109/CCC.1996.507676).
- [Sch89] Uwe Schöning. “Probabilistic complexity classes and lowness”. In: *J. Comput. System Sci.* 39.1 (1989), pp. 84–100. DOI: [10.1016/0022-0000\(89\)90020-2](https://doi.org/10.1016/0022-0000(89)90020-2).
- [Sha24] Yakov Shalunov. “Leakage-Resilient Hardness Equivalence to Logspace Derandomization”. In: *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. 2024, 83:1–83:16. DOI: [10.4230/LIPIcs.MFCS.2024.83](https://doi.org/10.4230/LIPIcs.MFCS.2024.83).
- [She59] J. C. Shepherdson. “The reduction of two-way automata to one-way automata”. In: *IBM J. Res. Develop.* 3 (1959), pp. 198–200. DOI: [10.1147/rd.32.0198](https://doi.org/10.1147/rd.32.0198).
- [Sim81] Janos Simon. “Space-bounded probabilistic Turing machine complexity classes are closed under complement (Preliminary Version)”. In: *Proceedings of the 13th Symposium on Theory of Computing (STOC)*. 1981, pp. 158–167. DOI: [10.1145/800076.802469](https://doi.org/10.1145/800076.802469).
- [Sip80] Michael Sipser. “Halting space-bounded computations”. In: *Theoret. Comput. Sci.* 10.3 (1980), pp. 335–338. DOI: [10.1016/0304-3975\(80\)90053-5](https://doi.org/10.1016/0304-3975(80)90053-5).
- [SZ99] Michael Saks and Shiyu Zhou. “ $\text{BP}_{\text{H}}\text{SPACE}(S) \subseteq \text{DSPACE}(S^{3/2})$ ”. In: *Journal of Computer and System Sciences* 58.2 (1999), pp. 376–403. DOI: [10.1006/jcss.1998.1616](https://doi.org/10.1006/jcss.1998.1616).
- [Tod91] Seinosuke Toda. “PP is as hard as the polynomial-time hierarchy”. In: *SIAM J. Comput.* 20.5 (1991), pp. 865–877. DOI: [10.1137/0220053](https://doi.org/10.1137/0220053).
- [VW97] H. Vollmer and K.W. Wagner. “On operators of higher types”. In: *Proceedings of the Twelfth Computational Complexity Conference (CCC)*. 1997, pp. 174–184. DOI: [10.1109/CCC.1997.612313](https://doi.org/10.1109/CCC.1997.612313).