Derandomizing Logspace with a Small Shared Hard Drive

Edward Pyne
MIT, Cambridge, MA, USA

Abstract

We obtain new catalytic algorithms for space-bounded derandomization. In the catalytic computation model introduced by (Buhrman, Cleve, Koucký, Loff, and Speelman STOC 2013), we are given a small worktape, and a larger catalytic tape that has an arbitrary initial configuration. We may edit this tape, but it must be exactly restored to its initial configuration at the completion of the computation. We prove that

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{CSPACE}\left[S, S^2\right]$$

where $\mathbf{BPSPACE}[S]$ corresponds to randomized space S computation, and $\mathbf{CSPACE}[S, C]$ corresponds to catalytic algorithms that use $O(S)$ bits of workspace and $O(C)$ bits of catalytic space. Previously, only $\mathbf{BPSPACE}[S] \subseteq \mathbf{CSPACE}\left[S, 2^{O(S)}\right]$ was known. In fact, we prove a general tradeoff, that for every $\alpha \in [1, 1.5]$,

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{CSPACE}\left[S^\alpha, S^{3-\alpha}\right].$$

We do not use the algebraic techniques of prior work on catalytic computation. Instead, we develop an algorithm that branches based on if the catalytic tape is conditionally random, and instantiate this primitive in a recursive framework. Our result gives an alternate proof of the best known time-space tradeoff for $\mathbf{BPSPACE}[S]$, due to (Cai, Chakaravarthy, and van Melkebeek, Theory Comput. Sys. 2006). As a final application, we extend our results to solve search problems in $\mathbf{CSPACE}\left[S, S^2\right]$. As far as we are aware, this constitutes the first study of search problems in the catalytic computing model.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Catalytic computation, space-bounded computation, derandomization

Digital Object Identifier 10.4230/LIPIcs.CCC.2024.4

Funding Supported by a Jane Street Graduate Research Fellowship.

Acknowledgements I am grateful to William Hoza for bringing the reference [5] to my attention. I thank Dean Doron, Ian Mertz, Roei Tell, Ryan Williams, and anonymous reviewers for helpful discussions and comments on the manuscript.

1 Introduction

In the catalytic logspace (**CL**) model, introduced by Buhrman, Cleve, Koucký, Loff, and Speelman [3], there is a machine M with $O(\log n)$ bits of standard working memory, and n^c bits of catalytic memory. This catalytic memory has an arbitrary initial configuration (perhaps data on a shared hard drive), and must be returned to exactly this configuration at the end of the computation. Remarkably, [3] showed that **CL** is likely to be strictly more powerful than **L**. In particular, it contains logspace-uniform $\mathbf{TC}^1$ and thus **NL**. Motivated by this striking result, there have been several further works exploring the power of catalytic computation [4, 12, 10, 8, 2, 9].

We parameterize catalytic computation by time, space, and catalytic space (similar notions have been considered before, e.g. [2]).

▶ **Definition 1.** *Let* **CTISP** $[T(n), S(n), C(n)]$ *be the set of languages recognized by catalytic machines that use* $O(S(n))$ *workspace and* $O(C(n))$ *catalytic space on inputs of size* n*, and run in time* $\mathrm{poly}(T(n))$ *in the worst case.*

Note that the worst-case runtime must hold over every catalytic tape, as well as every input.

Prior work has studied the ability of catalytic space to substitute for *randomness*, in particular in the setting of derandomizing space-bounded computation. Let **BPL** be the set of languages recognized by randomized machines that run in space $O(\log n)$ on inputs of size n, and make two-sided error. The result of [3] implies that

$$\mathsf{BPL} \subseteq \mathsf{CTISP}\,[n, \log n, n^c]$$

for some constant c. In fact, we are aware of two other proofs of this fact. An unpublished result (see the recent survey of Mertz [15] for a sketch) proves it by treating the catalytic tape as a set of random walks, and the third follows from recent work on certified derandomization for **BPL** [20, 11].

As our main result, we improve the amount of catalytic space needed to simulate **BPL** by a superpolynomial amount.

▶ **Theorem 2.**

$$\mathsf{BPL} \subseteq \mathsf{CTISP}\left[n, \log n, \log^2 n\right].$$

Our simulation of **BPL** is as time- and space- efficient as the frontier result of Nisan [18], which proves that $\mathsf{BPL} \subseteq \mathsf{TISP}[n, \log^2 n]$, and moreover almost all the space used is catalytic. Next, we incorporate this algorithm into a recursive framework to derive a (time-efficient) tradeoff between the catalytic- and non-catalytic space consumption.

▶ **Theorem 3.** *For every* $\alpha \in [1, 1.5]$*,*

$$\mathsf{BPL} \subseteq \mathsf{CTISP}\left[2^{\log^{\alpha}(n)}, \log^{\alpha}(n), \log^{3-\alpha}(n)\right].$$

This result immediately gives a new proof of the best known result on time-space trandeoffs for **BPL** due to Cai, Chakaravarthy, and van Melkebeek [5]. They prove **BPL** is contained in $\mathsf{TISP}[2^{\log^{\alpha} n}, \log^{3-\alpha} n]$ for every $\alpha \in [1, 1.5]$. In fact, our result shows that for every $\alpha < 1.5$, we can achieve a simulation with equivalent time and total space, but where the majority of the space used can be made catalytic.

Interestingly, while previous work on algorithms for catalytic computation [3, 8] primarily used algebraic techniques involving reversible computation over a ring, our results take a completely different approach based on conditional compressibility. We also develop a new approach for efficient composition of catalytic algorithms, building on the well-known composition of space-bounded algorithms. We hope that our techniques will have broader applications, both inside and beyond the model of catalytic computation. As a final application, we show that our ideas can be used to give nontrivial catalytic search algorithms.

1.1 Proof Overview for Theorem 2

A canonical (promise)-**BPL** complete problem is that of estimating transition probabilities in read-once branching programs:

▶ **Definition 4.** *A **read-once branching program (ROBP)** $\overline{B}$ of width w and length n and alphabet $\{0,1\}^t$ is defined by a function $B : [w] \times \{0,1\}^t \to [w]$.*[1] *For $x \in (\{0,1\}^t)^n$, define*

$$\overline{B}[i,x] = B[B[\dots B[B[i,x_1],x_2]\dots],x_{n-1}],x_n].$$

It is well known that to derandomize **BPL**, it suffices to estimate $\Pr_{x\leftarrow U_n}[\overline{B}[1,x]=1]$ up to error 1/3 for an ROBP $\overline{B}$ of length n, width n and alphabet $\{0,1\}$.

The Result of Nisan

We now recall the result of [18], which itself begins with the PRG of [16]. In this PRG, we draw $\ell = \log n$ hash functions $h_1,\dots,h_\ell$ from a pairwise independent hash family on $t = O(\log nw)$ bits. We recursively define the PRG as follows. Let $\mathrm{NIS}_0(x) = x$ for $x \in \{0,1\}^t$, and let $\mathrm{NIS}_{i+1}(x) = (\mathrm{NIS}_i(x), \mathrm{NIS}_i(h_{i+1}(x)))$. To analyze this PRG, fix a branching program $\overline{B} : (\{0,1\}^t)^n \to \{0,1\}$ of width w, with transition function B. Viewing this construction from the bottom up, the first hash function h_1 is good if for every every $a,b \in [w]$,

$$\Pr_{x,x'\leftarrow U_t}[B[B[a,x],x']=b] \approx \Pr_{x\leftarrow U_t}[B[B[a,x],h_1(x)]=b],$$

i.e. the distribution $(x, h_1(x))$ is indistinguishable from the distribution (x,x') by the composition of B with itself. Since B can only pass $\log w$ bits of information from the first to the second half, this occurs with probability $1 - w^{-c}$ over $h_1 \leftarrow \mathcal{H}$ (assuming $t = O(\log w)$ is sufficiently large). The ultimate PRG is analyzed recursively using ℓ applications of essentially the same idea. Concretely, at the second level of the construction, we now want a hash function h_2 that fools the length $n/2$ program with transition function $B'[a,x] = B[B[a,x],h_1(x)]$.

While the Nisan PRG randomly selects ℓ hash functions at once, the insight of [18] was that, given a specific program $\overline{B}$ that we want to fool, we can *search* for good hash functions level by level. At level i, we find a hash function h_i that fools the relevant transition function. As this test is easy to implement in time $2^{O(t)}$ for a fixed h and there are $2^{O(t)}$ such h to test, we can find such a good hash function in time $2^{O(t)} = \mathrm{poly}(n)$ per level, giving a polynomial runtime overall.

An Algorithm From Conditional Compression

We transform this algorithm into a *catalytic* algorithm as follows. Suppose we have a branching program $\overline{B}$ of width w and length $n = 2^\ell$, and a catalytic tape $\mathbf{w}$, with an arbitrary initial configuration. We interpret $\mathbf{w}$ as holding 2ℓ hash functions $h_1,\dots,h_{2\ell}$, each over $t = O(\log nw)$ bits (and note that each function can have description size exactly $2t$, as there exists a pairwise independent hash family on t bits of size 2^{2t}). Let $V \in \{0,1,*\}^{2\ell}$ and initialize $V = *^{2\ell}$ to indicate the status of each block. We then iterate through this list. Letting the ith hash function be $\tilde{h}$ and the previous good hash functions be $\vec{h_p}$, we check if $\tilde{h}$ is a good hash function, using the test as before.

- If $\tilde{h}$ is good, we set $V_i = 1$, indicating $\tilde{h}$ is part of the list of good hashes.
- If $\tilde{h}$ is not good, it must lie in the set $\mathbf{BAD}(\vec{h_p})$ of hash functions that fail to fool the current transition function. But as almost all h are good, the index of $\tilde{h}$ in $\mathbf{BAD}(\vec{h_p})$ is a concise description of $\tilde{h}$! We can then replace $\tilde{h}$ with this index, and free up $\Omega(\log nw)$ bits on this block of the tape. Finally, set $V_i = 0$ to indicate we have compressed this block.

[1] The standard definition of ROBPs permits the transition function to differ between the layers. However, as we will always be dealing with programs where $w \geq n$, and we are insensitive to polynomial losses in the width, we can assume all transition functions are the same for clarity.

At the end of this phase, we have either found ℓ good hash functions, or have freed up $\ell \cdot \Omega(\log nw)$ bits on the tape. In the latter case, we can simply search for a good set of hash functions (on slightly fewer bits), exactly as in the algorithm of [18], and store these in the free space of the compressed blocks. Thus, in both cases we obtain a sequence of hash functions that together constitute a good PRG for $\overline{B}$, and hence can construct a generator NIS that does a good job estimating walk probabilities on $\overline{B}$. The final step of estimating these walks can be performed in space $O(t + \log nw) = O(\log nw)$ with read-only access to the tape $\mathbf{w}$.

Finally, to return the tape to its original configuration, we work backwards over the compressed blocks, i.e. indices i where $V_i = 0$. For each block, we determine the preceeding good hash functions $\vec{h_p}$, read the index of the original hash (i.e. tape configuration) in $\mathbf{BAD}(\vec{h_p})$, then find the hash with this index by enumeration and write it to the tape.

1.2 Proof Overview for Theorem 3

To obtain a smooth tradeoff between the catalytic and non-catalytic space, our next idea is to unify this with efficient *composition* of catalytic algorithms:

Composition of Catalytic Algorithms

Recall that in the conventional composition of space-bounded algorithms, we can compute the composition of two algorithms running in space $S(n)$ in space $c \cdot S(n)$, for some constant $c > 1$. Our key observation is that for catalytic algorithms, we can obtain composition with *no* increase in the length of the catalytic tape:

▶ **Theorem 5** (Composition of Catalytic Space-Bounded Algorithms)**.** *Given two catalytic algorithms $\mathcal{M}_1, \mathcal{M}_2$ computing f_1, f_2 respectively, each using space $S(n) \geq \log n$, catalytic space $C(n)$, and time $T(n)$, there is a catalytic algorithm $\mathcal{M}'$ using time* $\text{poly}(T(n))$*, space $O(S(n))$, and catalytic space $C(n)$ that computes $f_2 \circ f_1$.*

The proof of this result modifies the standard composition of space-bounded algorithms. To compute $\mathcal{M}_2(\mathcal{M}_1(x))$, we begin to simulate $\mathcal{M}_2^{\mathbf{w}}(f_1(x))$ (where the superscript notation denotes running the machine with catalytic tape $\mathbf{w}$). Whenever $\mathcal{M}_2$ reads a bit of the input, we simulate $\mathcal{M}_1^{\mathbf{w}'}(x)$ to obtain the relevant bit of $f_1(x)$, where $\mathbf{w}'$ is the current configuration of the catalytic tape of $\mathcal{M}_2$. Since $\mathcal{M}_1$ is guaranteed to produce the correct answer for every starting tape, we have that $\mathcal{M}_1^{\mathbf{w}'}(x) = f_1(x)$. Moreover, as $\mathcal{M}_1$ is catalytic, it resets the tape to $\mathbf{w}'$ before returning, so $\mathcal{M}_2$ does not notice the call has occurred, and can continue its computation.

We remark that we are not able to apply this theorem as-is due to issues with (essentially) f_1 being a relation with multiple valid outputs, so the actual statement we prove is more involved. In particular, we must deal with safety reverting the catalytic tape if an intermediate call to $\mathcal{M}_1$ fails.

Derandomization via Repeated Powering

Going from this to Theorem 3 requires a further ingredient, which is given by a variant of the Saks-Zhou recursive powering scheme. Saks-Zhou [21] divides computing the nth power of an $n \times n$ stochastic matrix M (a **prBPL** complete problem) into r_2 iterations of computing the 2^{r_1}th power, for any $r_1 r_2 = \log n$. For convenience, let $M_0 = M$ and $M_i = M^{2^{r_1 \cdot i}}$ for $i \in [r_2]$. In the original algorithm, all levels share a single set of hash functions $\vec{h} = (h_1, \ldots, h_{r_1})$, each on $O(\log n)$ bits. A random set of hash functions will do a good job computing M_i

from M_{i-1} for every i, and so we can reuse this fixed set of hash functions at every level.[2] Unfortunately, such an argument is incompatible with searching for good hash functions one by one. Since we use every hash function to produce an approximation to M_1, if we later discover a hash function is bad at powering M_i for $i \geq 1$, seemingly we must destroy all partial progress and try a new set of hash functions. Thus, the Saks-Zhou algorithm must enumerate over $\vec{h} = (h_1, \ldots, h_{r_1})$ all at once, incurring a runtime of $2^{\Omega(r_1 \cdot \log n)}$. As the algorithm incurs a runtime of $2^{\Omega(r_2 \cdot \log n)}$ merely from the recursive composition of space bounded algorithms, the total runtime is at least $2^{\Omega(\max\{r_1, r_2\} \cdot \log n)} = 2^{\Omega(\log^{3/2} n)}$ for any setting of parameters r_1 and r_2. We note that the work of [5] also avoids this issue, and we explain their differing approach in more detail in Section 1.2.

Composing Conditional Compression Algorithms

Our catalytic algorithm allows for a more efficient approach. We follow the same recursive powering scheme as Saks-Zhou, but at each level use the algorithm of Theorem 2 that treats $\mathbf{w}$ as a list of $2r_1$ candidate hash functions.[3] Whenever we request an entry of a smaller power, we call the next level algorithm. If that level sees that the hash functions currently on the tape are good, it uses them to compute the requested entry. If not, it temporarily compresses the tape, finds good hash functions in time $\text{poly}(n)$, uses them to compute the requested entry, then resets the tape to exactly the same configuration the calling algorithm was expecting before returning. Thus, every level can either use the tape as-is, if it is suitable, or quickly compute a better set of hash functions on the fly and revert before returning control. This eliminates the $2^{r_1 \log n}$ term in the runtime. Moreover, the $O(r_1 \log n)$ bits used to store the hash functions can be treated as catalytic space, resulting in an algorithm that uses only $O(r_2 \log n)$ bits of workspace.

Finally, for every $\alpha \in [1, 1.5]$, we can choose $r_1 = \log^{2-\alpha}(n)$ and $r_2 = \log^{\alpha-1}(n)$ and obtain a algorithm that uses $O(r_1 \log n) = O(\log^{3-\alpha} n)$ catalytic space, $O(r_2 \log n) = O(\log^{\alpha} n)$ workspace, and runs in time $\text{poly}\left(n^{r_2}\right) = 2^{O(\log^{\alpha} n)}$, as claimed.

Such an approach runs into a subtle technical issue. Since the algorithm at level i may be called many times with different starting catalytic tapes, we must ensure that the algorithm returns the *same* approximate power each time, as otherwise the composition would not be well defined. To fix this, we first define a notion of catalytic algorithms that are allowed to return $\perp$ for some initial catalytic tapes, in addition to a fixed output that is independent of the catalytic tape. We then show how these algorithms can be composed, while still maintaining the ability to revert the tape to the original configuration in the worst case. Finally, we adopt the strategy of Saks and Zhou [21], and randomly perturb (or "shift") the matrices at each level. In our case, if a level of the algorithm determines that a shift is bad (i.e. could produce ambiguous behavior) it aborts and returns $\perp$. We show with high probability over the shifts, this will never occur (i.e. we will not return $\perp$) no matter the tape, and so we can compose the algorithm with itself and find the desired output.

Showing that we can successfully avoid permanent damage to the tape in the case that the shifts are bad requires further work. In particular, we ensure that our catalytic algorithms can be *reverted* from any point, where our notion of reversibility requires that we do not introduce

[2] There are additional complications from reusing the hash functions, but they are not the primary reason for the high time complexity.

[3] We give a "non-black-box" explanation of the final algorithm here as it illustrates the actual idea, but our proof uses a black-box statement regarding composition of catalytic algorithms.

any new configurations of the catalytic tape. We show that we can achieve this notion without a substantial time cost, and moreover it is compatible with recursive composition. Using this tool, we are able to return to the original tape configuration of a subroutine ever returns $\perp$.

Comparison With [5]

We briefly overview the techniques of [5], which achieves the best known time-space tradeoff for **BPL**, but in which all $O(\log^{3-\alpha} n)$ bits of space must be standard workspace. They likewise give a version of the Saks-Zhou result that does not incur the n^{r_2} factor in runtime, which we now explain. Their result follows the following recursive framework. We start with a set of hash functions $\vec{h} = (h_1, \ldots, h_\ell)$ that produce a good approximation of M_i (which we denote $\widetilde{M_i}$) from M_{i-1} for every $i \leq r$, but does not necessarily produce a good approximation of M_{r+1} from M_r. We then search for a new set of hash functions $\vec{h}' = (h'_1, \ldots, h'_\ell)$ with the following two properties. First, $\vec{h}'$ is good at approximately powering M_i for every $i \leq r$ (in particular, it produces a good approximation $\widetilde{M'_{r+1}}$). Second, after applying the random shift and round operation to the approximations $\widetilde{M_i}, \widetilde{M'_i}$ for $i \leq r$ produced by the old and new sets of hash functions, we obtain *the same* matrices. After doing so, we replace $\vec{h}$ with $\vec{h}'$ and increment r. The latter requirement allows us to make progress, as we can gradually find sets of hash functions that are good for greater powers, without destroying progress by altering the "results" of prior computation. However, this approach does not give a catalytic algorithm (in particular, it does not exploit the fact that bad hash functions are compressible).

1.3 Search Problems in Catalytic Space

Finally, we show how our compression-based techniques can be extended to solve search problems in catalytic space. As far as we are aware, we are the first to study catalytic search algorithms. Previous work of Sivakumar [22] showed that many search problems, such as producing a Johnson-Lindenstrauss sketch of a collection of vectors, can be reduced in logspace to solving the following problem, which we call "mutual ROBP hitting".

▶ **Definition 6** (Mutual ROBP Hitting). *Given a list of branching programs $\left(\overline{B}^1, \ldots, \overline{B}^n\right)$ each of length and width n, such that $\mathbb{E}\left[\overline{B}^j(U_n)\right] \geq 1 - 1/n^2$ for every j, produce a fixed $x \in \{0,1\}^n$ such that*

$$\bigwedge_{j \in [n]} \overline{B}^j(x) = 1.$$

We remark that this problem is (as of now) possibly harder than the task of producing x that satisfies a single ROBP, as it is not known to lie in **BPL**. Despite this, we show that we can solve this problem in the same asymptotic bound as Theorem 2:

▶ **Theorem 7.** *There is a* **CTISP** $\left[n, \log n, \log^2 n\right]$ *algorithm that solves the mutual ROBP hitting problem.*

Previously, this problem was known to be solveable in **TISP**$[n, \log^2 n]$, so we again show that almost all of the required space can be made catalytic.

We can immediately apply the reduction of [22] to obtain search algorithms for well-studied problems. As one application, we obtain a catalytic algorithm which produces a Johnson-Lindenstrauss transform: a low-dimensional embedding of a collection of vectors that approximately preserves their ℓ_2 distance. The problem of deterministically producing such an embedding has been extensively studied [22, 1, 13, 14].

▶ **Corollary 8.** *There is a* **CTISP** $\left[n, \log n, \log^2 n\right]$ *algorithm that, given* $\varepsilon > 0$ *and a collection of vectors* $v_1, \ldots, v_n \in \mathbb{R}^n$*, outputs vectors*[4] $\tilde{v}_1, \ldots, \tilde{v}_n \in \mathbb{R}^{O(\log(n)/\varepsilon^2)}$ *such that for every* $i, j \in [n]$*, we have*

$$(1-\varepsilon)\|v_i - v_j\|_2^2 \leq \|\tilde{v}_i - \tilde{v}_j\|_2^2 \leq (1+\varepsilon)\|v_i - v_j\|_2^2.$$

We prove Theorem 7 by extending our compress-or-random approach to producing a set of hash functions that is good for a *polynomial number* of ROBPs at once. In more detail, suppose the algorithm of Theorem 2 is now given a list of branching programs $(B^1, \ldots, B^n)$. The algorithm is structured as before, but now for each new hash function $\tilde{h}$, we test if $\tilde{h}$ is good for all of the input ROBPs. If yes, we again add it to our good sublist. Otherwise, let B^j be the first program that $\tilde{h}$ was not good for, and let $\mathbf{BAD}^j$ be the set of bad hash functions for this program, given the current prefix (and note that the prefix is by construction good for all programs). We now compress $\tilde{h}$ by recording j (which we require for the decompression algorithm to recover the hash function) together with the index of $\tilde{h}$ in $\mathbf{BAD}^j$. Again as before, once we have processed all blocks, either we have found a set of hash functions that are good for all ROBPs, or we have freed up $\Omega(\log^2 n)$ space. In that case, we again search for a set of hash functions that is good for every program simultaneously. This only incurs a mild constant factor loss in the length of each hash function, so our algorithm has the same asymptotic performance.

1.4 Roadmap

In Section 2, we formally define the catalytic computation model, and prove Theorem 5 and Theorem 7. In Section 3, we prove Theorem 2, and in Section 4, we prove Theorem 3. In Appendix A we provide proofs of some cited lemmas.

2 Catalytic Machines and Composition

We first formally define a catalytic Turing machine.

▶ **Definition 9** (Catalytic Turing Machine [3])**.** *A Turing machine* $\mathcal{M}$ *is a* ***catalytic machine*** *using time* $T(n)$*, workspace* $S(n)$*, and catalytic space* $C(n)$ *if it has a work tape, a read-only input tape, a write-only output tape, and a catalytic tape* $\mathbf{w}$*. We require that for every input* x *with* $|x| = n$ *and every* $\mathbf{w}$*,* $\mathcal{M}^{\mathbf{w}}(x)$ *halts in time at most* $T(n)$*, using at most* $S(n)$ *cells on the worktape and* $C(n)$ *cells on* $\mathbf{w}$*. Moreover, the final configuration of* $\mathbf{w}$ *must be equal to its initial configuration, for every* x *and* $\mathbf{w}$*.*

We now define the notion of a catalytic machine that computes a function. We furthermore define the notion of partially computing a function, where on some tapes $\mathbf{w}$ the machine can output a special failure symbol $\perp$.

▶ **Definition 10.** *For a function* $f : \{0,1\}^* \to \{0,1\}^*$*, we say a catalytic machine* $\mathcal{M}$ ***(catalytically) computes*** f *if for every* x *and* $\mathbf{w}$*,* $\mathcal{M}^{\mathbf{w}}(x) = f(x)$*, and at the end of the computation* $\mathbf{w}$ *is in its original state. We say that* $\mathcal{M}$ ***partially (catalytically) computes*** f *if for every* x *and* $\mathbf{w}$*,* $\mathcal{M}^{\mathbf{w}}(x) \in \{\perp, f(x)\}$*, and at the end of the computation (no matter the output)* $\mathbf{w}$ *is in its original state.*

[4] We assume the input and output are specified to $O(\log n)$ bits of precision.

Partial catalytic computation is trivial without further restrictions (as $\mathcal{M}$ can always output $\perp$), but we require it as an intermediate step in our analyses. We require a further condition on our machines, that they can revert the catalytic tape at any time without the catalytic tape traversing any new configurations:[5]

▶ **Definition 11.** *A catalytic machine $\mathcal{M}$ is* ***reversible*** *if for every x and initial configuration $\mathbf{w}$, at any point during the execution of $\mathcal{M}^{\mathbf{w}}(x)$, the machine can receive an external REVERT signal. Let $P = P(\mathbf{w})$ denote all prior configurations of the catalytic tape during the execution of $\mathcal{M}^{\mathbf{w}}(x)$. After this signal, $\mathcal{M}$ must reset $\mathbf{w}$ to the original configuration, and moreover every intermediate configuration of $\mathbf{w}$ during this process must lie in P. We require any time bound on $\mathcal{M}$ to hold even in the case that $\mathcal{M}$ is given the REVERT command at an arbitrary point.*

2.1 Composition of Catalytic Algorithms

We state the main result of this section, which is that catalytic algorithms can be composed without increasing the *catalytic* space usage. We must be careful when dealing with partial catalytic machines, and in this case we only obtain composition if the machines are reversible (Definition 11).

▶ **Theorem 12** (Composition of Partial Catalytic Machines)**.** *Suppose reversible catalytic machines $\mathcal{M}_1, \mathcal{M}_2$ partially compute $f_1, f_2 : \{0,1\}^n \to \{0,1\}^n$ respectively using workspace $S(n) \geq \log(n)$, catalytic space $C(n)$, and time $T(n)$. Then there is a reversible catalytic machine $\mathcal{M}$ that partially computes $f_2 \circ f_1$ using workspace $2S(n) + O(\log(Sn))$, catalytic space C, and time $\operatorname{poly}(T(n))$. Moreover, $\mathcal{M}^{\mathbf{w}}(x) = \perp$ only if $\mathcal{M}_2^{\mathbf{w}}(f_1(x)) = \perp$, or there exists $\mathbf{w}'$ such that $\mathcal{M}_1^{\mathbf{w}'}(x) = \perp$.*

Proof. We proceed roughly following the standard proof for composition of space-bounded algorithms. We maintain two sections on the worktape of size S for $\mathcal{M}_1$ and $\mathcal{M}_2$ (and auxiliary state of size $O(\log(Sn))$ to keep track of the location of read and write heads, and the FSM configuration of both machines), and a single catalytic tape $\mathbf{w}$.

The Simulation. We now begin to simulate $\mathcal{M}_2^{\mathbf{w}}$. We first verify that $\mathcal{M}_1^{\mathbf{w}}(x) \neq \perp$. As in the conventional composition of space-bounded algorithms, every time $\mathcal{M}_2$ reads its input, we run $\mathcal{M}_1^{\mathbf{w}}(x)$ on a separate section of the worktape and return the relevant bit of its output, where $\mathbf{w}$ is the same catalytic tape used by $\mathcal{M}_2$, in whatever its current configuration is at the time of the tape read. Moreover, every time $\mathcal{M}_2$ writes to the catalytic tape, resulting in a configuration $\mathbf{w}'$, we run $\mathcal{M}_1^{\mathbf{w}'}(x)$ and verify that it does not produce $\perp$.

Computing the Function. In the case that $\mathcal{M}_1^{\mathbf{w}'}(x) = f_1(x)$ for every configuration $\mathbf{w}'$ that is encountered in this simulation and $\mathcal{M}_2^{\mathbf{w}}(f_1(x)) = f_2(f_1(x))$, it is easy to see that $\mathcal{M}^{\mathbf{w}}(x) = f_2(f_1(x))$. Moreover, it is clear that in this case we successfully reset the tape. Otherwise, consider the first point at which $\mathcal{M}_1^{\mathbf{w}'}(x) = \perp$. We first undo the most recent change to the catalytic tape, and send the REVERT command to $\mathcal{M}_2$. Once $\mathcal{M}_2$ has finished reverting, return $\perp$. We claim that $\mathcal{M}_2$ successfully reverts the tape. This follows from the reversibility of $\mathcal{M}_2$, and the fact that all calls $\mathcal{M}_2$ makes to its input during this

[5] There are existing results related to transforming catalytic algorithms into reversible catalytic algorithms [15]. However, they do not appear to maintain worst-case runtime over the catalytic tape, which is crucial for our results.

process are correctly answered by $\mathcal{M}_1$. The latter property follows as every time $\mathcal{M}_2$ queries its input during the revert process, $\mathbf{w}$ is in a state that was encountered during the forward pass, and hence $\mathcal{M}_1(x)$ produced $f_1(x)$ when initialized with this catalytic configuration (as otherwise we would have aborted sooner).

Reversibility. Essentially the same argument establishes that $\mathcal{M}$ is reversible. If we receive the REVERT command, let $P(\mathbf{w})$ be the states of the catalytic tape that have been encountered so far. First send REVERT to $\mathcal{M}_1$ (if operating), and once it has completed send the REVERT command to $\mathcal{M}_2$. Moreover, while $\mathcal{M}_2$ is reverting, we claim that $\mathbf{w}$ remains in $P(\mathbf{w})$. This follows from the fact that $\mathcal{M}_2$ is reversible (as any configurations it creates will lie in $P(\mathbf{w})$), and moreover every time $\mathcal{M}_2$ queries its input, any computation done by $\mathcal{M}_1$ will likewise keep $\mathbf{w}$ in $P(\mathbf{w})$, as we already called $\mathcal{M}_1$ with this starting configuration in the forward pass (and $\mathcal{M}_1$ will not produce $\perp$, as otherwise we would have already aborted). Thus, the composed algorithm is reversible.

Time and Space. We now argue the space and time are as claimed. There are a constant number of pointers (which we maintain on the worktape) to track the number of bits output by $\mathcal{M}_1$, current tape heads, and other information. The fact that the catalytic tape size is preserved is immediate. The call overhead adds at most a polynomial factor in the runtime, as we run $\mathcal{M}_1$ at most once per step of $\mathcal{M}_2$. Finally, if $\mathcal{M}_1$ computes $f_1(x)$ for every catalytic tape and $\mathcal{M}_2$ computes correctly on $\mathbf{w}$, we successfully compute $f_2 \circ f_1$ as claimed. ◀

We derive an easy corollary in the case of multiple composition:

▶ **Corollary 13.** *Suppose a reversible catalytic machine $\mathcal{M}$ partially computes $f : \{0,1\}^n \to \{0,1\}^n$ using workspace $S(n) \geq \log n$, catalytic space C, and time 2^S. Then there exists a reversible catalytic machine $\mathcal{M}'$ that partially computes f^ℓ using workspace $O(\ell \cdot S)$, catalytic space C, and time $2^{O(\ell \cdot S)}$. Moreover, for x where $\mathcal{M}^{\mathbf{w}}(f^i(x)) = f(f^i(x))$ for every $\mathbf{w}$ and $i \in \{0, \ldots, \ell - 1\}$, $\mathcal{M}'^{\mathbf{w}}(x) = f^\ell(x)$.*

3 Catalytic Derandomization From Conditional Compression

In this section we prove Theorem 2. We state all our results in terms of catalytic algorithms for the stochastic matrix powering problem, as it is easily compatible with the recursive framework we implement later. Recall a nonnegative matrix is **stochastic** (resp. **substochastic**) if all row sums are 1 (resp. at most 1). For a set S, let U_S be the uniform distribution over S, and let $U_n = U_{\{0,1\}^n}$.

▶ **Theorem 14.** *There is a* **CTISP** $\left[n, \log n, \log^2 n\right]$ *algorithm that, given n and a stochastic matrix $M \in [0,1]^{n \times n}$ where each entry is specified with $O(\log n)$ bits of precision, outputs $\widetilde{M} \in [0,1]^{n \times n}$ such that $\left\|\widetilde{M} - M^n\right\|_1 \leq 1/n$.*

We recall the existence of efficient algorithms which canonicalize (sub)stochastic matrices, essentially reducing the stochastic matrix powering problem to producing a PRG that fools a branching program.

▶ **Lemma 15** ([21, 19, 7]). *There is a constant $c > 0$ and a space $O(\log nw/\varepsilon)$ algorithm which, given $\varepsilon > 0$ and $n, w \in \mathbb{N}$ where $w \geq n$ and a substochastic matrix $M \in [0,1]^{w \times w}$ with $O(\log w)$ bits of precision, returns a branching program*

$$\overline{B} : [(w/\varepsilon)^c] \times \{0,1\}^m \to [(w/\varepsilon)^c]$$

of width $w' = (w/\varepsilon)^c$ and length $m = n \cdot O(\log(w/\varepsilon))$ where m is a power of two. Moreover, letting $\widetilde{M} \in [0,1]^{w \times w}$ be the (substochastic) matrix where for $i, j \in [w]$ we define[6] $\widetilde{M}_{i,j} = \Pr_{x \leftarrow U_n}[\overline{B}[i,x] = j]$*, we have* $\left\| \widetilde{M} - M^n \right\|_1 \leq \varepsilon$.

As this is not the way these results are stated, we provide a translation in Appendix A. We next define the Nisan PRG, and recall several auxiliary lemmas.

The Nisan PRG

Given a branching program, we first define the larger alphabet program obtained from duplicating each edge:

▶ **Definition 16.** *For $t \in \mathbb{N}$, for $\overline{B} : [w] \times \{0,1\}^n \to [w]$ of width w, let $\overline{B}_t : [w] \times (\{0,1\}^t)^n \to [w]$ be the branching program of length n and width w over alphabet $\{0,1\}^t$ with transition function $B_t[a, y] = B[a, y_1]$, where y_1 is the first bit of $y \in \{0,1\}^t$. Note that $\overline{B}_t$ can be constructed in space $O(\log tnw)$ given $\overline{B}$, and furthermore for every $i, j \in [w]$, $\Pr_{x \leftarrow U_n}[\overline{B}[i,x] = j] = \Pr_{x \leftarrow U_{(\{0,1\}^t)^n}}[\overline{B}_t[i,x] = j]$.*

We recall a pairwise independent hash family with a very efficient description:

▶ **Observation 17.** *For every $t \in \mathbb{N}$, there exists a pairwise independent hash family $\mathcal{H} : \{0,1\}^t \to \{0,1\}^t$ such that $|\mathcal{H}| = 2^{2t}$, and $h \in \mathcal{H}$ (which we associate with $h \in \{0,1\}^{2t}$) can be evaluated in space $O(t)$.*

Given a (hash) function $h : \{0,1\}^t \to \{0,1\}^t$ and a program $\overline{B}$, we define an operator that applies a single level of the Nisan construction with hash function h.

▶ **Definition 18.** *Given $\overline{B}_t : [w] \times (\{0,1\}^t)^n \to [w]$ and $h : \{0,1\}^t \to \{0,1\}^t$, let $\overline{B}_{t,h} : [w] \times (\{0,1\}^t)^{n/2} \to [w]$ be the width w, length $n/2$ program with transition function*

$$B_{t,h}[a, x] = B_t[B_t[a, x], h(x)].$$

Using a recursive application of hash functions, we can define the Nisan PRG as follows.

▶ **Definition 19.** *For $(h_1, \ldots, h_\ell) \in \mathcal{H}_t$, define $\mathrm{NIS}_{(h_1,\ldots,h_\ell)} : \{0,1\}^t \to \{0,1\}^{t \cdot n}$ inductively as follows. Let $\mathrm{NIS}_0(x) = x_1$, and for $j \in [\ell]$*

$$\mathrm{NIS}_{(h_1,\ldots,h_j)}(x) = (\mathrm{NIS}_{(h_1,\ldots,h_{j-1})}(x) || \mathrm{NIS}_{(h_1,\ldots,h_{j-1})}(h_j(x))).$$

Note that $B[\cdot, \mathrm{NIS}_{(h_1,\ldots,h_\ell)}(\cdot)]$ and $B_{t,(h_1,\ldots,h_\ell)}[\cdot,\cdot]$ (as defined in Definition 18) are equal as functions.

To analyze the Nisan PRG, we define the notion of a hash function being good for composing two functions, and a PRG being good for a function.

▶ **Definition 20.** *For every $n, w, t \in \mathbb{N}$ and $\delta > 0$ and $\overline{f} : [w] \times (\{0,1\}^t)^n \to [w]$ and $G : \{0,1\}^t \to (\{0,1\}^t)^n$, we say that G is δ-**good** for $\overline{f}$ if for every $i, j \in [w]$,*

$$\left| \Pr_{x \leftarrow U_t}[\overline{f}[i, G(x)] = j] - \Pr_{x \leftarrow U_{(\{0,1\}^t)^n}}[\overline{f}[i,x] = j] \right| \leq \delta.$$

*Moreover, we say $h \in \mathcal{H}$ is δ-**good** (and δ-**bad** otherwise) for $f : [w] \times \{0,1\}^t \to [w]$ if $G(x) = (x||h(x))$ is δ-good for $\overline{f}[i, (x, y)] = f[f[i, x], y]$.*

[6] Note that this truncates the size from w' back to w.

We recall that a random hash function is good with high probability.

▶ **Lemma 21** ([16]). *For every f, $\Pr_{h \leftarrow \mathcal{H}_t}[h$ is δ-good for $f] \geq 1 - w^5(1/\delta)^2/2^t$.*

(We provide a proof in Appendix A.) Moreover, a hybrid argument establishes the following.

▶ **Lemma 22** ([16]). *For every $\overline{B}_t : [w] \times (\{0,1\}^t)^n \to [w]$ and $\vec{h} = (h_1, \dots, h_\ell)$, suppose for every $i \in [\ell]$, h_i is δ-good for $B_{t,h_1,\dots,h_{i-1}}$. Then $\mathrm{NIS}_{\vec{h}}$ is $\delta \cdot nw$-good for $\overline{B}$.*

Catalytic Derandomization

We now state the main result that powers both of our derandomizations.

▶ **Theorem 23.** *There is a pair of reversible catalytic algorithms $\mathcal{A}, \mathcal{D}$ that run in workspace $O(\log nw/\varepsilon)$, catalytic space $O(\log(n) \cdot \log(nw/\varepsilon))$, and time $\mathrm{poly}(nw/\varepsilon)$ and act as follows. Given $\varepsilon > 0$ and a length $n = 2^\ell$, width w ROBP $\overline{B} : [w] \times \{0,1\}^n \to [w]$ where $w \geq n$:*

- *The machine $\mathcal{A}^{\mathbf{w}}(\overline{B})$ outputs $V \in \{0,1\}^{2\ell}$ and $t = O(\log nw/\varepsilon)$ and sets the catalytic tape to $\mathbf{w}'$, such that $(\mathbf{w}', V)$ contains a (read-only) data structure supporting access to hash functions $\vec{h} = (h_1, \dots, h_\ell)$ each on t bits, such that $\mathrm{NIS}_{\vec{h}}$ is ε-good for $\overline{B}$.*
- *The machine $\mathcal{D}^{\mathbf{w}'}(\overline{B}, V)$ sets the final catalytic tape configuration to $\mathbf{w}$.*

For the extension to search problems, we require a version which takes in a list of ROBP, and constructs a set of hash functions that is simultaneously good for all of them. Our proof is for this more general notion, which immediately implies Theorem 23:

▶ **Theorem 24.** *There is a pair of reversible catalytic algorithms $\mathcal{A}, \mathcal{D}$ that run in workspace $O(\log nw/\varepsilon)$, catalytic space $O(\log(n) \cdot \log(nw/\varepsilon))$, and time $\mathrm{poly}(nw/\varepsilon)$ and act as follows. Given $\varepsilon > 0$ and a set of w length $n = 2^\ell$, width w ROBPs*

$$\mathcal{B} = \left(\overline{B}^1, \dots, \overline{B}^w\right)$$

with $\overline{B}^j : [w] \times \{0,1\}^n \to [w]$ where $w \geq n$:

- *The machine $\mathcal{A}^{\mathbf{w}}(\mathcal{B})$ outputs $V \in \{0,1\}^{2\ell}$ and $t = O(\log nw/\varepsilon)$ and sets the catalytic tape to $\mathbf{w}'$, such that $(\mathbf{w}', V)$ contains a (read-only) data structure supporting access to hash functions $\vec{h} = (h_1, \dots, h_\ell)$ each on t bits, such that $\mathrm{NIS}_{\vec{h}}$ is ε-good for $\overline{B}^j$ for every $j \in [w]$.*
- *The machine $\mathcal{D}^{\mathbf{w}'}(\mathcal{B}, V)$ sets the final catalytic tape configuration to $\mathbf{w}$.*

To make our compression and decompression algorithms work, we require that we can determine if a hash function is good for a branching program at a certain level of the Nisan construction, given pointers to the hash functions at the previous levels:

▶ **Proposition 25.** *There is a space $O(t + \log(w/\delta))$ algorithm that, given $n, w, t \in \mathbb{N}$ with $w \geq n$ and $\tilde{h} \in \{0,1\}^{2t}$ and $\overline{B} : [w] \times \{0,1\}^n \to [w]$ and (read only) $\mathbf{w}$ and pointers $p_1, \dots, p_r$ such that $\mathbf{w}_{[p_i, \dots, p_i + 2t]} = h_i$ represents a hash function on t bits, returns if $\tilde{h}$ is δ-good for $B_{t,(h_1,\dots,h_r)}$.*

We give a proof in Appendix A, as it essentially follows from the argument of [18]. We can then prove the theorem:

Proof of Theorem 24. We assume without loss of generality that n, w and $1/\varepsilon$ are powers of two. Set

$$t_0 = 60\log(w/\varepsilon), \qquad t_1 = 25\log(w/\varepsilon), \qquad \delta = \varepsilon/nw \geq \varepsilon/w^2,$$

and note that we choose t_1 large enough such that a good series of hash functions (for all $\overline{B}^j$ simultaneously) on t_1 bits always exists. The algorithm works as follows. First, virtually divide the catalytic tape as:

$$\mathbf{w} = \left(\mathbf{w}^1 || \mathbf{w}^2 || \cdots || \mathbf{w}^{2\ell}\right)$$

where $|\mathbf{w}^i| = 2t_0$, which we think of as initially holding $h : \{0,1\}^{t_0} \to \{0,1\}^{t_0}$. Note that $|\mathbf{w}| = \ell \cdot 4t_0 = O(\log(n)\log(nw/\varepsilon))$ as claimed.

Next, initialize $V \in \{0,1,*\}^{2\ell}$ to indicate if each block is compressed, uncompressed, or unprocessed respectively. The first two cases correspond to the following two formats of the block:

$$\mathbf{w}^i = \begin{cases} h & V_i = 1 \\ \left(z||j||0^{50\log(w/\varepsilon)}\right) & V_i = 0 \end{cases}$$

Informally, the first corresponds to block i originally containing a good hash function for $\mathcal{B}$, and the second corresponds to block i originally containing a bad hash function for $\overline{B}^j$, which is thus compressible (in fact, z represents a compressed version of the original data). We define notation for the set of blocks in each configuration:

▶ **Definition 26.** *For $b \in \{0,1\}$, let $I^b(V) \subseteq [2\ell]$ correspond to the indices such that $V_i = b$, and let $S^b(V) = |I^b(V)|$.*

Next, we initialize a counter $i = 1$ for the current block. We then iterate over $i = \{1,\ldots,2\ell\}$ until $\max\{S^1(V), S^0(V)\} = \ell$.[7] For each i, the algorithm works as follows. Let $\tilde{h} = \mathbf{w}^i$ be the hash function (on t_0 bits) obtained from the current block. We then test if $\tilde{h}$ is δ-good for

$$f^j = B^j_{t_0,\vec{h_p}}, \text{ for every } j \in [w].$$

Where

$$\vec{h_p} = \left(\mathbf{w}^{I^1(V)_1}, \ldots, \mathbf{w}^{I^1(V)_{S^1(V)}}\right)$$

corresponds to the hash functions on the preceding good blocks, and $B^j_{t_0,\vec{h_p}}$ is defined as in Definition 18 applied to $\overline{B}^j$ and $\vec{h_p}$. As the index set $I^1(V)$ is easy to generate given V, this test can be performed in space $O(\log nw/\varepsilon)$ without modifying the catalytic tape (and hence also in time $\text{poly}(nw/\varepsilon)$), by Proposition 25.

Given the results of this test, we break into cases depending on if $\tilde{h}$ is good:

- If $\tilde{h}$ is δ-good for f^j for every $j \in [w]$, set $V_i = 1$.
- If $\tilde{h}$ is δ-bad for some f^j, set $V_i = 0$ and let j be the first index for which this holds. Next, by enumeration over strings $h \in \{0,1\}^{2t_0}$ (which we can do using the workspace), determine the index of $\tilde{h}$ in the set

  $$\mathbf{BAD}_{i,j} = \left\{h \in \{0,1\}^{2t} : h \text{ is } \delta\text{-bad for } B^j_{t_0,\vec{h_p}}\right\}$$

  where we again perform this test using Proposition 25. Letting the index of $\tilde{h}$ in this set be z, write

  $$\mathbf{w}^i = \left(z||j||0^{50\log(w/\varepsilon)}\right)$$

[7] If we exit before $i = 2\ell$, set the remaining indices of V to an arbitrary value, which we ignore for clarity of presentation.

(we perform this write operation left to right, and will revert it right to left). We denote the final $50\log(w/\varepsilon)$ bits as free space.

Finally, we claim that we can in fact write these quantities in space $|\mathbf{w}^i| = 2t_0$. The index j requires $\log(w)$ bits. Moreover, we have

$$\begin{aligned}|\mathbf{BAD}_{i,j}| &= 2^{2t_0} \cdot \Pr_{h\leftarrow\mathcal{H}}[h \text{ is } \delta\text{-bad for } f] \\ &\leq 2^{2t_0} \cdot w^5(1/\delta)^2/2^{t_0} \qquad \text{(Lemma 21)} \\ &\leq 2^{2t_0} \cdot (w/\varepsilon)^{7-60}\end{aligned}$$

And thus $\log|\mathbf{BAD}_{i,j}| \leq 2t_0 - 51\log(w/\varepsilon) = |\mathbf{w}^i| - 51\log(w/\varepsilon)$. Therefore, we can record all required information as claimed.

After processing all blocks, we obtain a catalytic tape $\mathbf{w}'$ and one of two cases:

- If $S^1(V) = \ell$, there exist $\vec{h} = (h_1, \ldots, h_\ell)$ corresponding to the hash functions (on t_0 bits) in $I^1(V)$, and these functions are easy to recover from V.
- Else, we must have $S^0(V) = \ell$.

  For $i \in I^0(V)$, let F_i be the $50\log(w/\varepsilon)$ free bits in $\mathbf{w}^i$. Note that a description of a hash function $h : \{0,1\}^{t_1} \to \{0,1\}^{t_1}$ is of size $|F_i|$. Iterating over $i \in I^0(V)$ in increasing order, we find (via brute force enumeration) a hash function $\tilde{h}$ that is δ-good for

  $$f^j = B^j_{t_1,\vec{h_p}}, \text{ for every } j \in [w].$$

  Where

  $$\vec{h_p} = \left(\mathbf{w}_{F_{I^0(V)_1}}, \ldots, \mathbf{w}_{F_{I^0(V)_{i-1}}}\right)$$

  corresponds to the (δ-good) hash functions stored on the free space in the preceding indices of $I^0(V)$. Next, store $\tilde{h}$ in $\mathbf{w}_{F_i}$. Such a good hash function always exists, by our choice of t_1 and Lemma 21, and moreover testing if each candidate is good can be computed in the desired space and time by Proposition 25. After this processing,

  $$\left(\mathbf{w}_{F_{I^0(V)_1}}, \ldots, \mathbf{w}_{F_{I^0(V)_\ell}}\right)$$

  contain ℓ good hash functions, which we can clearly access in read-only fashion given given V and $\mathbf{w}'$.

Thus, in both cases we obtain a set of hash functions $\vec{h} = (h_1, \ldots, h_\ell)$ on $t = O(\log nw/\varepsilon)$ bits that is δ-good for every one of the relevant tests, so by Lemma 22 we have that $\mathrm{NIS}_{\vec{h}}$ is $\delta \cdot nw \leq \varepsilon$-good for $\overline{B}^j$ for every j.

Decompression and Reversibility. It suffices to show that at any point, the algorithm can revert the tape to the original configuration $\mathbf{w}$ (and then $\mathcal{D}(\mathcal{B}, V)$ simply issues the REVERT command). No matter the present configuration, we iterate through $I^0(V)$ in descending order. Letting the current index be $i \in I^0(V)$, recall this block is of form $(\mathbf{w}')^i = (z||j||*)$. First write $0^{50\log(w/\varepsilon)}$ to the last indices (in reverse order to satisfy reversibility), such that we reach the configuration after compressing the block. Then enumerate over $h \in \{0,1\}^{2t_0}$ using workspace $O(t_0 + \log(w/\varepsilon))$, until we find the hash with index z in $\mathbf{BAD}_{i,j}$, where $\mathbf{BAD}_{i,j}$ and $B^j_{t_0,\vec{h_p}}$ are defined as before (which we still have access to because we reset the tape in reverse order), and we determine membership by Proposition 25.

Once we find this h, write $(\mathbf{w}')^i = h$ (in the reverse order to satisfy reversibility) and proceed to the next highest index in $I^0(V)$. After this process has completed, it is clear from construction that $\mathbf{w}$ has been reset to the original configuration, and that the tape never reaches a new intermediate configuration during this process.

Time and Space. In every step of the computation, we perform at most $\text{poly}(2^{t_0}nw/\varepsilon)$ work to determine if a hash function is good, find the index of a bad hash function, or find a good hash function. Moreover, as at every point we store at most a constant number of hash functions on the worktape, the space consumption follows. ◀

It is easy to go from Theorem 23 to Theorem 14.

Proof of Theorem 14. Let $\overline{B} : [\text{poly}(n)] \times \{0,1\}^{n^c} \to [\text{poly}(n)]$ be the ROBP obtained from applying Lemma 15 to M with $n = n$, $w = n$, and $\varepsilon = 1/2n$. We then call Theorem 23 with $\overline{B} = \overline{B}$ and $\varepsilon = 1/2n^2$. Let $\vec{h} = (h_1, \ldots, h_{c \log n})$ be the hash functions obtained from this call, which we have implicit access to via the current state of the catalytic tape $\mathbf{w}'$ and V, and let $t = O(\log n)$ be the domain of the hash functions. Then enumerate over $x \in \{0,1\}^t$ and for $i, j \in [w]$ let

$$\widetilde{M}_{i,j} = \Pr_{x \leftarrow U_t}\left[\overline{B}[i, \text{NIS}_{\vec{h}}(x)] = j\right].$$

By Lemma 15 and Theorem 23, we have the guarantee that

$$\left\|\widetilde{M} - M^n\right\|_1 \le \frac{1}{2n} + n \cdot \frac{1}{2n^2} = 1/n.$$ ◀

Finally, we can use Theorem 24 to prove Theorem 7.

Proof of Theorem 7. The algorithm works as follows. Given $\mathcal{B} = \left(\overline{B}^1, \ldots, \overline{B}^n\right)$ of width and length n where $\mathbb{E}\left[\overline{B}^j(U_n)\right] \ge 1 - 1/n^2$ for every j, we call Theorem 24 with $\mathcal{B} = \mathcal{B}$ and $\varepsilon = 1/2n^2$. Let $\vec{h} = (h_1, \ldots, h_{\log n})$ be the hash functions obtained from this call, which we have implicit access to via the current state of the catalytic tape $\mathbf{w}'$ and V, and let $t = O(\log n)$ be the length of the domain of the hash functions. We then claim there is some $x \in \{0,1\}^t$ such that for all $j \in [n]$,

$$\overline{B}^j(\text{NIS}_{\vec{h}}(x)) = 1.$$

We have this as

$$\begin{aligned}
\Pr_{x \leftarrow U_t}\left[\bigwedge_{j \in [n]} \overline{B}^j\left(\text{NIS}_{\vec{h}}(x)\right)\right] &\ge 1 - \sum_{j \in [n]} \Pr_{x \leftarrow U_t}\left[\overline{B}^j\left(\text{NIS}_{\vec{h}}(x)\right) = 0\right] \\
&\ge 1 - \sum_{j \in [n]} \left(\Pr_{x \leftarrow U_t}\left[\overline{B}^j(U_n) = 0\right] + \frac{1}{2n^2}\right) \\
&\ge 1 - n(2/n^2) > 0
\end{aligned}$$

where the second inequality follows from our choice of ε. Then it is simple to find such an x by enumeration, whereupon we compute and return $\text{NIS}_{\vec{h}}(x) \in \{0,1\}^n$, which is precisely the solution for the mutual ROBP hitting problem. Finally, we use Theorem 24 to reset the tape. ◀

4 Catalytic Recursive Matrix Powering

We now transform Theorem 23 into a parameterized algorithm for matrix powering.

▶ **Theorem 27.** *There is a catalytic machine that, given r_1, r_2 such that $r_1 r_2 = \log(n)$ and a stochastic matrix $M \in [0,1]^{n \times n}$ where each entry is specified with $l = O(\log n)$ bits of precision, uses workspace $O(r_2 \cdot \log(n))$, catalytic space $O(r_1 \cdot \log(n))$, and time $n^{O(r_2)}$, and outputs $\widetilde{M}$ such that $\|\widetilde{M} - M^n\| \leq 1/n$.*

Theorem 27 immediately implies Theorem 3 by setting $r_2 = \log^{\alpha-1}(n)$ and $r_1 = \log^{2-\alpha}(n)$ for $\alpha \in [1, 1.5]$, and using the standard transformation of estimating the acceptance probability of a **BPL** machine via stochastic matrix powering.

We first prove there exists an algorithm which computes a single intermediate power. We must be careful to ensure that the algorithm satisfies the requirements of (partial) catalytic computation. In particular, if the machine ever outputs an answer (rather than $\perp$), this must be the only possible answer for this input, over all possible catalytic tapes. Simultaneously, we must ensure that the vast majority of inputs never return $\perp$ no matter the initial catalytic tape configuration.

We achieve this dual guarantee using an idea from Saks and Zhou [21]. For a given input matrix M, we additionally take in a shift $s \in [0, \delta]$ for $\delta = 1/\operatorname{poly}(w/\varepsilon)$. After computing an approximate power of M, we add s to each entry, and then truncate each entry to $O(\log w/\varepsilon)$ bits of precision. In fact, we first verify that our shifted approximate power is sufficiently far from the rounding threshold, and if not return $\perp$. By doing so, we algorithmically verify that we will never round to different thresholds over different $\mathbf{w}$. Unfortunately, for some pairs (M, s) it may be the case that we detect possible mis-rounding for some tapes $\mathbf{w}$, even if all possible approximations lie inside a single rounding interval. This can result in returning $\perp$ on some tapes and a (consistent) value otherwise. However, we show that we can choose the magnitude of s such that with high probability over s this does not occur, and we always return a (consistent) value.

▶ **Theorem 28.** *For every $n, w \in \mathbb{N}$ and $\varepsilon > 0$ where $w \geq n \geq \log w$ and $2^{-n} > \varepsilon > 0$, there is a reversible catalytic machine $\mathcal{P}$ that uses workspace $O(\log w/\varepsilon)$, catalytic space $O(\log(n)\log(nw))$, and time $\operatorname{poly}(nw/\varepsilon)$. The machine takes input $s \in \{0,1\}^{O(\log(w/\varepsilon))}$ and a substochastic matrix $M \in [0,1]^{w \times w}$, where each entry of M is specified with $l = O(\log w/\varepsilon)$ bits of precision. Moreover:*

- *For every (s, M), there is substochastic $\widetilde{M_s}$ (defined without reference to $\mathbf{w}$) with l bits of precision satisfying $\left\|\widetilde{M_s} - M^n\right\|_1 \leq \varepsilon$ such that for every $\mathbf{w}$,*

$$\mathcal{P}^{\mathbf{w}}(s, M) \in \left\{\perp, \widetilde{M_s}\right\}.$$

- *For every M, $\Pr_s[\exists \mathbf{w},\ \mathcal{P}^{\mathbf{w}}(s, M) = \perp] \leq 1/w^2$.*

Proof. Let $\overline{B} : [(w/\delta)^c] \times \{0,1\}^{m = n \cdot O(\log(w/\delta))} \to [(w/\delta)^c]$ be the result of Lemma 15 applied to M with $n = n$ and $\varepsilon = \delta$ to be chosen later. We compose the output of this algorithm with Theorem 23, applied with $\overline{B} = \overline{B}$ and $w' = (w/\delta)^c$ and $n' = m = O(n^3)$ and $\varepsilon = \delta$ to be chosen later. Let $\vec{h} = (h_1, \ldots, h_\ell)$ be the hash functions obtained from this call, which we have implicit access to via the current state of the catalytic tape $\mathbf{w}'$ and V, and let $t = O(\log w/\delta)$ be the domain of the hash functions. Then enumerate over $x \in \{0,1\}^t$ and for every $i, j \in [w]$ let

$$\widetilde{M}_{i,j} = \Pr_{x \leftarrow U_t}\left[\overline{B}\left[i, \mathrm{NIS}_{\vec{h}}(x)\right] = j\right].$$

Next, define $\tau = \left(s \cdot 2^{-2k}\right) \cdot J$ where $J = 1^{w \times w}$, interpreting $s \in [2^k]$. We next check if any entry of $\widetilde{M} + \tau$ is within 2δ of a multiple of 2^{-l}, our rounding threshold. In this case, run $\mathcal{D}^{\mathbf{w}'}(\overline{B}, V)$ to reset the tape and return $\perp$. Otherwise, let

$$\widetilde{M_s} = \left\lfloor \widetilde{M} + \tau \right\rfloor_l$$

where $\lfloor\cdot\rfloor_l$ rounds each entry down to l bits of precision, and decreases the largest entry per row such that the final matrix is substochastic. Let this matrix be $\widetilde{M_s}$. Finally, run $\mathcal{D}^{\mathbf{w}'}(\overline{B}, V)$, and return $\widetilde{M_s}$.

Accuracy. By our choice of error in Theorem 23 and Lemma 15, we have that

$$\left\|\widetilde{M} - M^n\right\|_1 \leq 2w\delta$$

and moreover $\widetilde{M}$ has each row sum at most 1. Furthermore, perturbing by τ and rounding down the largest entry causes an ℓ_1 error of at most $2w \cdot 2^{-k}$. Finally, rounding each entry down to a multiple of 2^{-l} causes a total error of at most $w \cdot 2^{-l}$, so

$$\left\|\widetilde{M_s} - M^n\right\| \leq 2w\delta + 2w \cdot 2^{-k} + w \cdot 2^{-l} \leq \varepsilon$$

Where the final inequality comes from choosing

$$l = O(\log(w/\varepsilon)), \quad k = 10 \cdot l, \quad \delta = 2^{-2k}.$$

Uniqueness. We claim that for every (s, M), there is at most 1 possible non-$\perp$ output over all choices of $\mathbf{w}$ (which we denote $\widetilde{M_s}$ in the theorem statement). Let $\widetilde{M_{\mathbf{w}}}, \widetilde{M_{\mathbf{w}'}}$ be the result of Theorem 23 on M initialized with catalytic tapes $\mathbf{w}, \mathbf{w}'$. By the accuracy guarantee of Theorem 23, for every i, j we have

$$\left|(\widetilde{M_{\mathbf{w}}} + \tau)_{i,j} - (\widetilde{M_{\mathbf{w}'}} + \tau)_{i,j}\right| \leq \left|(\widetilde{M_{\mathbf{w}}} + \tau)_{i,j} - (M + \tau)_{i,j}\right| + \left|(M + \tau)_{i,j} - (\widetilde{M_{\mathbf{w}'}} + \tau)_{i,j}\right| \leq 2\delta$$

Thus, if $(\widetilde{M_{\mathbf{w}'}} + \tau)_{i,j}$ is greater than 2δ from a multiple of 2^{-l}, we can be certain that no tape $\mathbf{w}'$ will induce an estimate that falls on the other size of the threshold, and hence all non-$\perp$ outputs will be rounded consistently.

Success Probability. Furthermore, we argue that for every M, with probability at least $1 - 1/w^2$ over s we return $\widetilde{M_s}$ (not $\perp$) for every initial tape configuration. Fixing arbitrary s and $i, j \in [w]$, if $(M + \tau)_{i,j}$ is at least 3δ from every multiple of 2^{-l}, every $\mathbf{w}$ will induce an estimate $(\widetilde{M} + \tau)_{i,j}$ that is at least 2δ from every multiple of 2^{-l}, and hence for every $\mathbf{w}$ we will not produce $\perp$ due to this entry. This occurs for every i, j simultaneously with probability at least

$$w^2 \cdot 2^l \cdot \frac{6\delta \cdot 2^k + 2}{2^k} \ll 1/w^2.$$

Time and Space. It is clear the algorithm runs in the claimed time and space bound, given Theorem 23.

Reversibility. As the only components of the algorithm that write to the catalytic tape are calls to Theorem 23, reversibility follows immediately from the equivalent result for that algorithm. ◀

We can then prove the main result.

▶ **Theorem 27.** *There is a catalytic machine that, given r_1, r_2 such that $r_1 r_2 = \log(n)$ and a stochastic matrix $M \in [0,1]^{n \times n}$ where each entry is specified with $l = O(\log n)$ bits of precision, uses workspace $O(r_2 \cdot \log(n))$, catalytic space $O(r_1 \cdot \log(n))$, and time $n^{O(r_2)}$, and outputs $\widetilde{M}$ such that $\|\widetilde{M} - M^n\| \leq 1/n$.*

Proof. Let $\vec{s} = (s_1, \ldots, s_{r_2}) \in \{0,1\}^{r_2 \cdot O(\log n)}$ be a vector of random shifts. Let $\widetilde{M_0} = M$ and for $i \in [r_2]$ recursively define

$$\widetilde{M_i} = f\left(\widetilde{M_{i-1}}, s_i\right),$$

where f is the function defined by Theorem 28 with $\varepsilon = 1/n^3$ and $n = 2^{r_1}$. An easy inductive proof [19] establishes that, letting

$$\left\|\widetilde{M_i} - M^{2^{r_1 \cdot i}}\right\|_1 = \delta_i$$

we have $\delta_{i+1} \leq 1/n^3 + 2^{r_1} \cdot \delta_{i-1} \leq 2^{r_1+1} \cdot \delta_{i-1}$, and hence $\delta_{r_2} \leq 1/n$.

The final algorithm iterates over $\vec{s}$ and computes $\widetilde{M_{r_2}}$ by applying recursive composition of space-bounded machines Corollary 13 to the algorithm of Theorem 28 as defined above.[8] The algorithm returns the first non-$\perp$ output. The fact that the algorithm is catalytic follows from Corollary 13 and Theorem 28. Next, we claim there is some $\vec{s}$ where the algorithm returns a value. Note that s_i is chosen obliviously to $\widetilde{M_{i-1}}$, and so with probability at least $1/n^2$ over s_i, on input $(\widetilde{M_{i-1}}, s_i)$ the algorithm returns $\widetilde{M_i}$ (i.e. not $\perp$) when run with every possible catalytic tape. Thus, there is some $\vec{s}$ where every level computes correctly.

Time and Space. Every application of Theorem 28 occurs with parameters $n = 2^{r_1}$ and $w = n$ and $\varepsilon = 1/n^3$, such that the algorithm uses workspace $O(r_1 + \log(n)) = O(\log n)$, catalytic space $O(r_1 \cdot (r_1 + \log(n))) = O(r_1 \cdot \log n)$, and time $\mathrm{poly}(n)$, and moreover the shift s for each level is of length $O(\log n)$. Applying Corollary 13, we obtain that the composed algorithm uses workspace $O(r_2 \cdot \log(n) + |\vec{s}|) = O(r_2 \cdot \log n)$, catalytic space $O(r_1 \cdot \log n)$, and runs in time $n^{O(r_2)}$ as claimed. ◀

References

1 Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi:10.1016/S0022-0000(03)00025-4.

2 Sagar Bisoyi, Krishnamoorthy Dinesh, and Jayalal Sarma. On pure space vs catalytic space. *Theor. Comput. Sci.*, 921:112–126, 2022. doi:10.1016/j.tcs.2022.04.005.

3 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014*, pages 857–866. ACM, 2014.

4 Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space: Non-determinism and hierarchy. *Theory Comput. Syst.*, 62(1):116–135, 2018. doi:10.1007/s00224-017-9784-7.

5 Jin-yi Cai, Venkatesan T. Chakaravarthy, and Dieter van Melkebeek. Time-space tradeoff in derandomizing probabilistic logspace. *Theory Comput. Syst.*, 39(1):189–208, 2006. doi:10.1007/S00224-005-1264-9.

6 Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. *Theory of Computing*, 18(21):1–32, 2022. doi:10.4086/toc.2022.v018a021.

[8] We can define the machine to take the entire shift vector and a pointer to the index it should use, such that we are recursively composing exactly the same machine, but we suppress this for clarity.

7 Gil Cohen, Dean Doron, Ori Sberlo, and Amnon Ta-Shma. Approximating iterated multiplication of stochastic matrices in small space. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 35–45. ACM, 2023.

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

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

10 Samir Datta, Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Randomized and symmetric catalytic computation. In *Computer Science – Theory and Applications – 15th International Computer Science Symposium in Russia, CSR 2020*, pages 211–223. Springer, 2020.

11 Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for BPL = L that uses properties of BPL. *Electron. Colloquium Comput. Complex.*, TR23-208, 2023. arXiv:TR23-208.

12 Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Unambiguous catalytic computation. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019*, volume 150 of *LIPIcs*, pages 16:1–16:13, 2019.

13 Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit johnson-lindenstrauss families. In Leslie Ann Goldberg, Klaus Jansen, R. Ravi, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, volume 6845 of *Lecture Notes in Computer Science*, pages 628–639. Springer, 2011. doi:10.1007/978-3-642-22935-0_53.

14 Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014. doi:10.1145/2559902.

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

16 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. doi:10.1145/100216.100242.

17 Noam Nisan. On read-once vs. multiple access to randomness in logspace. *Theor. Comput. Sci.*, 107(1):135–144, 1993. doi:10.1016/0304-3975(93)90258-U.

18 Noam Nisan. RL <= SC. *Comput. Complex.*, 4:1–11, 1994. doi:10.1007/BF01205052.

19 Aaron (Louie) Putterman and Edward Pyne. Near-optimal derandomization of medium-width branching programs. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 23–34, 2023.

20 Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 989–1007. IEEE, 2023. doi:10.1109/FOCS57990.2023.00061.

21 Michael E. Saks and Shiyu Zhou. $BP_H Space(S) \subseteq DSPACE(S^{3/2})$. *J. Comput. Syst. Sci.*, 58(2):376–403, 1999. doi:10.1006/jcss.1998.1616.

22 D. Sivakumar. Algorithmic derandomization via complexity theory. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 619–626. ACM, 2002. doi:10.1145/509907.509996.

A Proofs of Lemmas

We first prove that a hash function drawn from a pairwise independent hash family is good for a function with high probability. To do this, we recall the hash mixing lemma:

▶ **Lemma 29** ([16]). *Let $A, A' \subseteq \{0,1\}^t$ be arbitrary subsets of density $\rho = |A|/2^t$ and $\rho' = |A'|/2^t$. Then for every $\delta > 0$,*

$$\Pr_{h \leftarrow \mathcal{H}}\left[\left|\Pr_{x \leftarrow U_t}[x \in A, h(x) \in A'] - \rho\rho'\right| \geq \delta\right] \leq (1/\delta^2)/2^t.$$

Proof of Lemma 21. For every $i, k \in [w]$, let $A_{i,k} = \{x \in \{0,1\}^t : f[i,x] = k\}$ and let $\rho_{i,k} = |A_{i,k}|/2^t$. Note that for every i, j,

$$\Pr_{x,x' \leftarrow U_t}[f[f[i,x],x'] = j] = \sum_{k \in [w]} \rho_{i,k}\rho_{k,j}.$$

Thus, for every h such that for every i, k, j, $\Pr_{x \leftarrow U_t}[x \in A_{i,k}, h(x) \in A_{k,j}] - \rho_{i,k}\rho_{k,j}| \leq \delta/w$, we have that h is δ-good for f. By Lemma 29 this event occurs with probability $1-(w/\delta)^2/2^t$ over $h \leftarrow \mathcal{H}$ for each pair of sets, and thus with probability $1-w^3(w/\delta)^2/2^t = 1-w^5(1/\delta)^2/2^t$ for every tuple (i,j,k). ◀

We recall there is a logspace algorithm which tests if a hash function is good, given oracle access to the function we wish to fool. We remark that there is work [17, 6, 20] on testing if an entire PRG is good for a branching program, but we need a much weaker claim.

▶ **Lemma 30** ([18]). *There is a space $O(t + \log(w/\delta))$ algorithm that, given oracle access to $f : [w] \times \{0,1\}^t \to [w]$ and $h \in \mathcal{H}_t$ and $\delta > 0$, tests if h is δ-good for f.*

Proof. The algorithm enumerates over $i, j \in [w]$. For every i, j, the algorithm computes $p_{i,j} = \mathbb{E}_{x,x' \leftarrow U_t}[f[f[i,x],x']]$ (i.e. the correct probability) by enumeration over x, x' in space $O(t + \log w)$. Then it computes $\tilde{p}_{i,j} = \mathbb{E}_{x \leftarrow U_t}[f[f[i,x],h(x)]]$ and rejects if the estimate is greater than δ from the true value. Correctness and total space consumption are immediate. ◀

We can then prove that we can test if a hash function is good, given $\overline{B}$ and pointers to preceding hash functions.

▶ **Proposition 25.** *There is a space $O(t + \log(w/\delta))$ algorithm that, given $n, w, t \in \mathbb{N}$ with $w \geq n$ and $\tilde{h} \in \{0,1\}^{2t}$ and $\overline{B} : [w] \times \{0,1\}^n \to [w]$ and (read only) $\mathbf{w}$ and pointers $p_1, \ldots, p_r$ such that $\mathbf{w}_{[p_i,\ldots,p_i+2t]} = h_i$ represents a hash function on t bits, returns if $\tilde{h}$ is δ-good for $B_{t,(h_1,\ldots,h_r)}$.*

Proof. By Lemma 30, it suffices to show that given $i \in [w]$ and $x \in \{0,1\}^t$, we can compute $B_{t,(h_1,\ldots,h_r)}[i,x]$. To do this, the algorithm maintains $v \in [w]$ as its current position in the branching program (initialized to $v = i$) and $i \in [n]$ to track the current layer. To determine the next position, it suffices to determine the ith block of the output of $\text{NIS}_{(h_1,\ldots,h_r)}(x)$. It is well known that this can be computed in space $O(t + \log nw)$ given read-only access to the set of hash functions (by walking down the binary expansion of i, denoted $\langle i \rangle$, and applying h_j if $\langle i \rangle_j = 1$), which we have via the pointers. ◀

Finally, we provide a translation of our quantization statement. We first recall a strict specialization of the statement of [19]:

▶ **Lemma 31.** *There exists a* ***canonicalizer*** *algorithm* $\mathtt{C_t}$ *that, given* $n, w \in \mathbb{N}$ *with* $w \geq n$*, takes in* $\varepsilon > 0$ *and a sub-stochastic matrix* $M \in \mathbb{R}^{w \times w}$ *with each entry represented by at most* $O(\log w/\varepsilon)$ *bits, runs in space* $O(\log w/\varepsilon)$*, and returns a branching program* $\overline{B}$ *of length* n *and width* $w+1$ *with alphabet* $\{0,1\}^t$ *for* $t = O(\log(w/\varepsilon))$*. Moreover, letting* $\widetilde{M} \in [0,1]^{w \times w}$ *be the matrix where for* $i, j \in [w]$ *we have*

$$\widetilde{M}_{i,j} = \Pr_{x \leftarrow U_{(\{0,1\}^t)^n}} \left[\overline{B}[i,x] = j\right]$$

then

$$\left\|\widetilde{M} - M^n\right\|_1 \leq \varepsilon.$$

We reduce the alphabet (and slightly increase the length) as follows. We transform $\overline{B}$ into a branching program of width $(w+1) \cdot 2^t = \text{poly}(w/\varepsilon)$ and length $n \cdot t = n \cdot O(\log w/\varepsilon)$, where each set of t layers reads t bits, interprets these bits as $\sigma \in \{0,1\}^t$, and takes the transition labeled with σ in $\overline{B}$.