Electronic Colloquium on Computational Complexity, Report No. 106 (2024)

The Structure of Catalytic Space: Capturing Randomness and Time via Compression

James Cook
Toronto, Canada
falsifian@falsifian.org

Jiatu Li*
MIT
jiatuli@mit.edu

Ian Mertz†
University of Warwick
ian.mertz@warwick.ac.uk

Edward Pyne‡
MIT
epyne@mit.edu

June 17, 2024

Abstract

In the catalytic logspace (**CL**) model of (Buhrman et. al. STOC 2013), we are given a small work tape, 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. This model is of interest from a complexity-theoretic perspective as it gains surprising power over traditional space. However, many fundamental structural questions remain open.

We substantially advance the understanding of the structure of **CL**, addressing several questions raised in prior work. Our main results are as follows.

1. We unconditionally derandomize catalytic logspace: **CBPL** = **CL**.
2. We show time and catalytic space bounds can be achieved separately if and only if they can be achieved simultaneously: any problem in **CL** ∩ **P** can be solved in polynomial time-bounded **CL**.
3. We characterize deterministic catalytic space by the intersection of randomness and time: **CL** is equivalent to polytime-bounded, *zero-error* randomized **CL**.

Our results center around the *compress–or–random* framework. For the second result, we introduce a simple yet novel *compress–or–compute* algorithm which, for any catalytic tape, either compresses the tape or quickly and successfully computes the function at hand. For our first result, we further introduce a *compress–or–compress–or–random* algorithm that combines runtime compression with a second compress–or–random algorithm, building on recent work on distinguish-to-predict transformations and pseudorandom generators with small-space deterministic reconstruction.

*Supported by an MIT Akamai Fellowship.
†Supported by Royal Society University Research Fellowship URF\R1\191059.
‡Supported by a Jane Street Graduate Research Fellowship.

ISSN 1433-8092

1 Introduction

How useful is access to a full hard drive? The *catalytic logspace* (**CL**) model, introduced by Buhrman, Cleve, Koucký, Loff, and Speelman [BCK+14], models this question by augmenting a logspace machine M with a polynomially large extra work tape, called the *catalytic tape*; the catch is that this tape begins in some arbitrary initial configuration $\mathbf{w}$, and while it can be edited during the computation, at the end it *must* be reset to its initial $\mathbf{w}$.

Such a model naturally sits between **L** and **PSPACE**, but in many preexisting contexts [CMW+12, Liu13, EMP18, IN19] it was strongly assumed that full memory cannot be in any way useful for unrelated computation, and so it seemed likely that **CL** would be equal to **L**. Remarkably, however, [BCK+14] showed that **CL** is likely to be *strictly more powerful* than **L**: they showed that **CL** contains logspace-uniform $\mathbf{TC}^1$, a class known to contain non-deterministic logpsace (**NL**), randomized logspace (**BPL**), and more. Subsequently, there have been several further works exploring the power of catalytic computation [BKLS18, GJST19, DGJ+20, CM20, CM21, BDS22, CM22, CM23, Pyn23, LPT24] (see surveys of Koucký [Kou16] and Mertz [Mer23] for an overview).

In this work we study the structural complexity of catalytic computation. We show multiple unconditional relations between some of the most well-studied catalytic classes, and obtain new conditional results under substantially weaker assumptions than previously known.

1.1 Derandomizing Catalytic Space

Our first result relates to *randomized* catalytic computation. Introduced by [DGJ+20], the class **CBPL** is the natural extension of randomized logspace (**BPL**) to the catalytic setting. We note two important features of this model: the random coins are accessed in a read-once fashion (analogously to **BPL**), and the machine must always reset the catalytic tape, no matter the sequence of random bits.

The line of work on derandomizing randomized logspace (i.e. proving $\mathbf{BPL} = \mathbf{L}$) has been highly fruitful, resulting in the state of the art result of [SZ99, Hoz21] that randomized space s can be simulated deterministically in space $s^{3/2-o(1)}$. In the catalytic setting, however, derandomization—i.e. $\mathbf{CBPL} = \mathbf{CL}$, posed as an open question in Mertz [Mer23]—was only known assuming strong circuit lower bounds [DGJ+20].

We unconditionally derandomize catalytic computation.

Theorem 1.1.

$$\mathbf{CBPL} = \mathbf{CL}.$$

This is among the first unconditional derandomization results known for uniform computation. Note that $\mathbf{L} \subseteq \mathbf{CL} \subseteq \mathbf{PSPACE}$, and $\mathbf{PSPACE} = \mathbf{BPPSPACE}$ is known whereas $\mathbf{BPL} = \mathbf{L}$ is an open question, and so our results can be thought of as progress in this direction. However, we also know that $\mathbf{L} \subseteq \mathbf{CL} \subseteq \mathbf{ZPP}$, and both $\mathbf{BPL} = \mathbf{L}$ and $\mathbf{BPP} = \mathbf{P}$ are open, giving a curious case where a natural intermediate class can be derandomized.

1.2 The Power of Time-Bounded Catalytic Space

Our second setting is motivated by a central open question in catalytic computing: is **CL** contained in **P**? In their first paper introducing **CL**, [BCK+14] showed that $\mathbf{CL} \subseteq \mathbf{ZPP}$; thus $\mathbf{CL} \subseteq \mathbf{P}$ under the widely-believed (uniform) assumption that $\mathbf{ZPP} = \mathbf{P}$. However, putting aside the long-standing intractability of derandomizing **ZPP**, even finding such a derandomization does necessarily not give a *catalytic* polynomial time algorithm. In particular, let **CLP** be the set of problems solvable by

catalytic logspace algorithms that run in worst-case polynomial time; while $\mathsf{CLP} \subseteq \mathsf{CL} \cap \mathsf{P}$ is immediate, the converse was not known. The work of [DGJ+20] showed that $\mathsf{CL} = \mathsf{CLP}$ follows from strong circuit lower bounds, but no such conclusion was known from any uniform assumption.

We resolve this question and show that functions which are efficiently solvable with respect to both catalytic space and time *individually* admit algorithms which are efficient with respect to both *simultaneously*.

Theorem 1.2.

$$\mathsf{CL} \cap \mathsf{P} = \mathsf{CLP}.$$

We note two corollaries of this result. First, showing $\mathsf{CL} \subseteq \mathsf{P}$ is *equivalent* to making catalytic algorithms run in worst-case polynomial time.

Corollary 1.3.

$$\mathsf{CL} \subseteq \mathsf{P} \iff \mathsf{CL} = \mathsf{CLP}.$$

One can view this result in a positive sense, and as a barrier. On the positive side, a proof of $\mathsf{CL} \subseteq \mathsf{P}$ gives a further, potentially stronger result for free. On the barrier side, it is *provably impossible* to take advantage of the plentiful space afforded by P to simulate CL, without simultaneously improving the (catalytic) space-bounded inclusion.

Second, derandomization of ZPP scales *down* and implies a collapse of two subclasses CL and CLP of ZPP:

Corollary 1.4.

$$\mathsf{ZPP} = \mathsf{P} \implies \mathsf{CL} = \mathsf{CLP}.$$

Towards proving $\mathsf{CL} = \mathsf{CLP}$. Corollary 1.4 can be further strengthened to show that $\mathsf{CL} = \mathsf{CLP}$ follows from the derandomization of a syntactic subclass LOSSY of ZPP that has been studied by several recent works [Kor21, ILW23, Kor22, CTW23, LPT24].

Definition 1.5. The complexity class LOSSY is defined as the languages that are polynomial-time reducible to the following total search problem called $\mathsf{LossyCode}$: Given a pair of Boolean circuits $C : \{0,1\}^n \to \{0,1\}^{n-1}$ and $D : \{0,1\}^{n-1} \to \{0,1\}^n$, find some $x \in \{0,1\}^n$ such that $D(C(x)) \neq x$.

Indeed, we unconditionally prove that $\mathsf{CL} \subseteq \mathsf{LOSSY}$.

Theorem 1.6.

$$\mathsf{CL} \subseteq \mathsf{LOSSY} (\subseteq \mathsf{ZPP}).$$

Together with Corollary 1.3, we show that $\mathsf{LOSSY} = \mathsf{P}$ is sufficient to prove $\mathsf{CL} = \mathsf{CLP}$.

Corollary 1.7.

$$\mathsf{LOSSY} = \mathsf{P} \implies \mathsf{CL} = \mathsf{CLP}.$$

This strengthens Corollary 1.4 as $\mathsf{LOSSY} \subseteq \mathsf{ZPP}$, and it is not known whether $\mathsf{LOSSY} = \mathsf{ZPP}$, and constitutes the first improvement on the assumptions required to prove $\mathsf{CL} \subseteq \mathsf{P}$. Moreover, since $\mathsf{LOSSY} = \mathsf{P}$ follows from uniform space-time trade-off lower bounds (see [Kor22]), $\mathsf{CL} \subseteq \mathsf{P}$ and $\mathsf{CL} = \mathsf{CLP}$ also follow from the same assumptions.

1.3 Synthesis: Characterizing Catalytic Space Via Randomness and Time

Combining our two lines of work gives a surprising characterization of **CL**: in exchange for granting our catalytic machine *zero-error* randomness,[1] we may guarantee that it runs in worst-case polynomial time, and this characterization is exact. We follow our previous convention for **CLP** and dub the latter class **CZPLP**:

Theorem 1.8.

$$\mathbf{CL} = \mathbf{CZPLP}.$$

While Theorem 1.1 can be thought of as bounding the power of randomness in catalytic computing, the forward direction of Theorem 1.8 gives a highly nontrivial *use* for randomness in the catalytic model.

Theorem 1.8 makes progress towards resolving $\mathbf{CL} \subseteq \mathbf{P}$; not only do we weaken the derandomization hypothesis *sufficient* to resolve the question—$\mathbf{CZPLP} \subseteq \mathbf{ZPP}$ follows trivially from their respective definitions—but in fact our equivalence shows that such derandomization is *necessary* as well. Furthermore, in the *randomzied* setting we unconditionally resolve the question:

Corollary 1.9.

$$\mathbf{CZPL} = \mathbf{CZPLP}.$$

This gives another way to view Corollary 1.4 as a scaling down of derandomization for **ZPP**.

1.4 Technical Overview

Our results build on and substantially extend the *compress–or–random* approach to studying **CL** [Dul15, Mer23, Pyn23, DPT24, LPT24]. At a high level, prior results in this framework work as follows. Think of the catalytic tape as a candidate source of random bits. If the tape is sufficiently random, we can simply use it without modification for our desired task; otherwise, the tape must be (information theoretically) compressible. To use this dichotomy in the context of **CL**, we must have a way of *certifying* if the tape is random enough, and if not, must have a compression scheme that can be implemented in **CL**. The requirement for this efficient compression scheme previously limited the approach to studying **BPL** [Dul15, Pyn23, DPT24].

To explain the relevance of compress-or-random to the problem of ensuring (deterministic) catalytic algorithms have fast *runtime*, recall that [BCK+14] observed that over a random initial catalytic tape, a **CL** machine will halt in $\text{poly}(n)$ steps with high probability. Thus, every initial tape configuration that "causes" a high runtime is unusual, and in particular is information-theoretically compressible. This fact was known for a decade, but it was unclear how to make it algorithmically useful, required for the compress–or–random paradigm.

1.4.1 Compress–or–compute: compression from high runtime.

Our first main insight in this paper—in essence the only tool needed for our results on **CLP**—is a new way of compressing the tape in the compress–or–random argument when simulating a catalytic machine. For catalytic machine M deciding language L, our idea is as follows: if we run M in question on starting tape $\mathbf{w}$, it either quickly halts and returns the correct answer, or it runs for a long time; for concreteness, say M runs for at least $2n^c$ steps, where M's free work tape has

[1] Where on input x, we either compute $L(x)$ or return a special symbol $\perp$, and we return $\perp$ with probability at most $1/2$ for every input and starting tape.

length $c \log n$. In the former case we are done, as we have successfully and quickly computed the language L in question. In the latter case, we now think of the starting tape as having the form

$$(\mathbf{w} \circ i)$$

where $i \in [2n^c]$ is a timestep specified with $c \log n + 1$ bits. After running the machine M on starting tape $\mathbf{w}$ for i steps, the catalytic tape will be in configuration

$$(\mathbf{w}' \circ i)$$

for some new configuration $\mathbf{w}'$, and moreover M's work tape will have configuration $v \in \{0,1\}^{c \log n}$ for some v. Next, we set the catalytic tape to

$$(\mathbf{w}' \circ v \circ 0).$$

Our insight is that we can describe $(\mathbf{w} \circ i)$ (i.e. the original configuration of the tape) as "run M backwards from catalytic tape $\mathbf{w}'$ and work tape v, and count the number of elapsed steps until we reach a start state."

This describes $(\mathbf{w}, i)$ with $(\mathbf{w}', v)$ via a catalytic algorithm, and

$$|\mathbf{w} \circ i| = |\mathbf{w}' \circ v| - 1$$

from our choice of timestep size. Thus, we effectively compress the tape by one bit from the failure of the machine to halt quickly, and both the compression and decompression can be implemented *in-place* with $O(\log n)$ bits of auxiliary workspace[2]. A natural recursive extension of this algorithm results in a catalytic machine that either computes L in polynomial time, or frees up a polynomial amount of space on the catalytic tape; hence, we dub this strategy "compress–or–compute". This argument immediately yields Theorem 1.2: in **CLP**, either our **CL** machine computes the language quickly, or we free up enough space to run our **P** machine.

1.4.2 Compress–or–compress–or–random: derandomization through two different compressors.

The previous result compressed the tape from the runtime (with starting tape $\mathbf{w}$) being too large. To generalize this notion, we let the configuration (sub)graph from starting state $\mathbf{w}$, denoted $\mathcal{R}(\mathbf{w})$, be the set of states $(\mathbf{w}', v)$ reachable from starting tape $\mathbf{w}$, where $\mathbf{w}'$ represents the catalytic tape and v the workspace of the machine M. For a deterministic catalytic machine, this graph is (essentially) a line. For *randomized* machines, each state now has out-degree 2, corresponding to the transitions from reading random bit 0 and 1. However, it is still the case that over a random $\mathbf{w}$, the size of $\mathcal{R}(\mathbf{w})$ is bounded by $\text{poly}(n)$ with high probability.[3]

Naively, one would hope to adopt the same argument, compressing $\mathbf{w} \circ i$ by using i to index into $\mathcal{R}(\mathbf{w})$ if it is sufficiently large. However, it is not even clear how to explore this graph, and if $\mathcal{R}(\mathbf{w})$ is small, it is not clear how to decide the language (as we cannot simply examine the final state of a line).

We deal with both of these problems by using an *additional* application of the compress–or–random framework. For now, assume we have access to a collection of random walks $Y \subseteq \{0,1\}^n$ of size a large polynomial in n. We then consider the configuration subgraph

$$\mathcal{Y} := \mathcal{Y}(\mathbf{w}, Y) \subseteq \mathcal{R}(\mathbf{w})$$

[2]Due to the initial machine possibly having non-reachable states in the configuration graph, our real scheme is more complicated, but this captures the idea.

[3]This follows as the machine must reset the catalytic tape no matter the sequence of random bits.

representing states reached from initial configuration $\mathbf{w}$ with random coins specified by Y. Building on the deterministic case, we are able to label the states in this subgraph in a consistent fashion given Y. We again think of our catalytic tape as

$$(\mathbf{w} \circ i)$$

where $i \in [2n^c]$, and divide into two cases based on the size of $\mathcal{Y}$:

Large Graph Case. If $|\mathcal{Y}| \geq 2n^c$ we follow essentially the same compression strategy as Section 1.4.1: we interpret i as an index into $\mathcal{Y}$, traverse to the state specified by that index, and replace i with the work tape of M at this configuration, freeing up one bit of space.

We remark that *decompressing* in this case is not immediate. We are "at" a state

$$\sigma = (\mathbf{w}', v)$$

and wish to find the index of σ in $\mathcal{Y}$. Our compression scheme cannot store which string $y \in Y$ we used to reach σ (otherwise it is not compressing); therefore, the decompression algorithm cannot naively "walk backwards" to recover the initial tape $\mathbf{w}$ and the index i without this information. One may think of running the machine forwards from σ until it halts to recover the initial tape $\mathbf{w}$. However, this will destroy our intermediate configuration $(\mathbf{w}', v)$, leaving us unable to determine the index i. Fortunately, we do have access to the set Y in our decompression algorithm. We iterate over $y \in Y$ to find a string that takes us from σ to the (unique) backwards-reachable start state. Our algorithm, which uses ideas from reversible computation, satisfies the following: if a walk y does *not* take us to the start state, we reset the tape to σ and can try again.

Once we have identified a good walk y (which may be non-unique), we can describe σ via the index of y in Y using $O(\log n)$ bits, which allows us to temporarily "store" the configuration on the work tape. We can then use a further routine to determine the index of σ in $\mathcal{Y}$.

Small Graph Case. If $|\mathcal{Y}| \leq 2n^c$, we hope to decide L. Unfortunately, there are now two different reasons why our collection of explored states could be small: the configuration graph $\mathcal{R}(\mathbf{w})$ is actually small, or our collection of random walks Y does a bad job exploring it! In addition, at present we do not have a set of walks Y with which to instantiate this paradigm with.

To deal with all of these issues, we create the strings Y using an instantiation of the *Nisan-Wigderson generator* [NW94] developed by Doron, Pyne, and Tell [DPT24]. We denote the generator as:

$$\mathsf{NW}^f : \{0,1\}^{O(\log n)} \to \{0,1\}^n$$

where $f \in \{0,1\}^{\mathrm{poly}(n)}$ is a truth table. For every $D : \{0,1\}^n \to \{0,1\}$ that distinguishes the output of the PRG from uniform, i.e.

$$\left| \mathbb{E}\left[D\left(\mathsf{NW}^f(\mathbf{U}) \right) \right] - \mathbb{E}[D(\mathbf{U})] \right| \geq 1/10$$

there is a small circuit C such that C computes f when given oracle access to D. Following the approach of [DPT24], we use a new section of the *catalytic tape*, which we denote $\mathbf{m}$, as the truth table f. Unrolling the definition, if $\mathsf{NW}^{\mathbf{m}}$ fails to fool D, the section of tape $\mathbf{m}$ is compressible given access to D. If the distinguisher D tests if NW does a good job exploring the configuration graph, the failure to explore becomes another win condition.

To realize this approach, we must make the transformation from D to a small circuit for f implementable in catalytic logspace. We build off recent works studying related questions [PRZ23,

DPT24], while incorporating new ideas. There are two required steps in this transformation. The first is transforming the distinguisher D into a previous bit predictor[4] P for $\mathsf{NW}^{\mathbf{m}}$, i.e. a function satisfying

$$\Pr_{x \leftarrow \mathsf{NW}^{\mathbf{m}}(\mathbf{U})}[P(x_{>j}) = x_j] \geq \frac{1}{2} + \frac{1}{n^2}.$$

In our case, we consider the following distinguisher:

$$D(r) = \mathbb{I}\left[M \text{ leaves } \mathcal{Y} \text{ when reading random bits } r\right].$$

and note that (by definition) we have $D\left(\mathsf{NW}^{\mathbf{m}}(\mathbf{U})\right) = 0$, so if $\mathbb{E}[D(\mathbf{U})] \geq 1/10$ we have that D distinguishes $\mathsf{NW}^{\mathbf{m}}$ from uniform (in particular, this occurs if $\mathsf{NW}^{\mathbf{m}}$ does not do a good job exploring the graph). By Yao's Lemma [Yao86], we have that there exists a predictor for $\mathsf{NW}^{\mathbf{m}}$ with the following form:

$$P(r_{>}) = D(z \circ r_{>}) \oplus b \tag{1}$$

where $b \in \{0,1\}$ and $z \in \{0,1\}^*$, as long as the conventional hybrid argument is done *backwards*. Next, we make the same observation as [DPT24]: restricting the first bits of D to z simply corresponds to starting the random walk at a new location in $\mathcal{Y}$. As there are only $\mathrm{poly}(n)$ different places to start this walk, we can create a candidate predictor for each vertex, then determine if any such predictor achieves good advantage on $\mathsf{NW}^{\mathbf{m}}$. We remark that in the language of [DPT24, LPT24], we obtain a distinguish-to-predict transformation for this distinguisher.

If such a good predictor exists, we must compress $\mathbf{m}$ *in-place* with $O(\log n)$ auxiliary workspace. Luckily, such an algorithm was constructed recently by [DPT24].[5] If no such predictor exists, we must have that NW does a good job exploring the configuration graph, and can thus decide the language by taking a majority vote over the outputs of the PRG, without modifying the tape.

There is one more complication that we discuss here. Once we compress the generator, the above approach would lose the ability to evaluate the predictor (and hence we would not be able to decompress). This is because the predictor is defined in terms of the explored subgraph $\mathcal{Y}$, which itself depends on the outputs of the generator. To resolve this circularity, we use a *sequence* of generators $G_1, \ldots, G_{2n^c}$, each instantiated with its own section of catalytic tape. Let $\mathcal{Y}_i$ be the states explored by G_i. For every fixed starting tape $\mathbf{w}$, each additional generator either explores a new configuration (bringing us closer to the large graph case) or fails to do so, in which case

$$\mathcal{Y}_i \subseteq \bigcup_{j<i} \mathcal{Y}_j.$$

If this holds, note that we can describe all predictors of Equation (1) using outputs of the *previous* PRGs, which we do not attempt to compress, and hence we do not lose access as we compress G_i. Finally, for technical reasons we use two distinguishers, but both can be transformed to predictors using similar ideas.

We remark that [DPT24] did not have to deal with this complication for their proof of $\mathbf{BPL} \subseteq \mathbf{CL}$, as there the graph was always present on the read-only input tape, whereas here we have only implicit access.

[4] In the general case, i.e., the distinguisher D is a general circuit, this problem is known to be as hard as derandomization itself [LPT24].

[5] For this step, we can replace the Nisan-Widgerson generator of [DPT24] with a compression algorithm utilizing previous bit predictors implicit in Korten's proof of the **prBPP**-hardness of R-Lossy Code [Kor22]. This is because once we have obtained a predictor, we only need to compress a truth table of size n^c to $n^c - n$ bits. (Note that the generator in [DPT24] allows us to compress a truth table of size n^c to n bits, which is indeed an overkill.) The details will be given in a later version of the paper.

Putting the Cases Together. We now present one step (with mild simplifications) of our final algorithm. We interpret the catalytic tape as

$$\mathbf{w} \circ i \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2n^c}$$

and for $j \in [2n^c]$ instantiate the PRGs

$$G_j = \mathsf{NW}^{\mathbf{m}_j} : \{0,1\}^{O(\log n)} \to \{0,1\}^{2n^c}$$

and let Y be the set of strings output by the union of these PRGs. Then there are three cases:

1. If Y explores more than $2n^c$ states, we are in the large graph case. We compress $(\mathbf{w} \circ i)$ to $(\mathbf{w}' \circ v)$ (freeing up one bit on the tape) and proceed to the next iteration, without modifying Y.

2. If Y explores fewer than $2n^c$ states, there is some j such that G_j exclusively reaches states already reached by $G_1 \cup \ldots \cup G_{j-1}$. We then build candidate predictors

   $$P_1, \ldots, P_{\mathrm{poly}(n)},$$

   each of which can be concisely described using $G_1, \ldots, G_{j-1}$.

   (a) If there is some k such that P_k predicts G_j with good advantage, we compress $\mathbf{m}_j$ by $\mathrm{poly}(n)$ bits and can decide the language via a space-inefficient algorithm.

   (b) If no such predictor achieves good advantage, it must be the case that G_j is a good collection of random walks, so we can use its output to derandomize.

Thus in Item 1 and Item 2a we compress the tape by 1 and $\mathrm{poly}(n)$ bits respectively, and in Item 2b we use our PRG to derandomize the algorithm in the conventional way. Due to this structure, we call this approach compress–or–compress–or–random.

1.5 Future Questions

The most immediate question left open by our work is to show $\mathsf{CL} \subseteq \mathsf{P}$. We call attention to one angle suggested by our work: if Theorem 1.2 can be adapted to the zero-error randomized case, i.e. $\mathsf{CZPL} \cap \mathsf{P} = \mathsf{CZPLP}$, then the question is resolved by Theorem 1.8.

As for other catalytic models for which our techniques may find future traction, can we show that *nondeterministic* catalytic logspace (CNL) equals catalytic logspace? The requirement to restore the tape no matter the sequence of guesses can be used to show $\mathsf{CNL} \subseteq \mathsf{ZPP}$, and Buhrman et. al. [BKLS18] showed that CNL is closed under complement under strong circuit lower bounds. While our timestamp compression approach could still apply here—a polynomial amount of free space on the tape is sufficient to solve CNL directly—there are two barriers to approaching unconditional structural results: first, as in the randomized case the configuration graph has out-degree 2, which complicates the case of walking backward; and second, a good guess sequence may be exponentially unlikely, which makes us unable to apply the machinery of directed random walks to obtain win-win arguments.

1.6 Roadmap

In Section 2 we formally define catalytic classes and their configuration graphs. In Section 3 we prove Theorem 1.2 and Theorem 1.6. In Section 4 we prove Theorem 1.1. All major catalytic routines will have pseudocode which can be found in Appendix D.

2 Preliminaries

2.1 Notation

Let $[n] = \{1, 2, \ldots, n\}$. We use $\mathbf{U}_n$ to denote the uniform distribution over $\{0,1\}^n$, and may omit the subscript n if it is clear in the context.

For a string y and $i \in \mathbb{N}$, we use y_i to denote the i-th bit of y, $y_{\leq i}$ to denote the prefix of y of length i, and $y_{>i}$ to denote the suffix of y of length $|y| - i$. For two strings x and y, we use $x \circ y$ to denote the concatenation of x and y.

Let $\mathbb{I}[\phi]$ be the indicator function, i.e., $\mathbb{I}[\phi] = 1$ if ϕ is true, and $\mathbb{I}[\phi] = 0$ otherwise. For a language $L \subseteq \{0,1\}^*$, we define $L(x) := \mathbb{I}[x \in L]$.

2.2 Complexity Classes for Catalytic Computation

Our basic starting point is the notion of a catalytic machine, as defined by Buhrman et al. [BCK+14]:

Definition 2.1. A *catalytic machine* M is defined as a Turing machine in the usual sense—i.e. a read-only input tape, a write-only output tape, and a (space-bounded) read-write work tape—with an additional read-write tape known as the *catalytic tape*. Unlike the ordinary work tape, the catalytic tape is initialized to hold an arbitrary string $\mathbf{w}$, and M has the restriction that for any initial setting of the catalytic tape, at the end of its computation the catalytic tape must be returned to the original state $\mathbf{w}$.

It will be helpful to define a notion of one catalytic machine simulating another, especially when the operation of the "smaller" machine is only partial, i.e. we care about transitioning to some internal state of the smaller machine inside the larger one. While the defining feature of catalytic machines is to reset the catalytic tape, such partial situations do not necessarily preserve this feature; in fact, we often use these operations to prepare an intermediate state of interest.

Definition 2.2. Let $s < s', c < c'$, and let M and M' be catalytic machines using free space s (s') and catalytic space c (c', respectively). We say M' *simulates* M, or that M is a *catalytic subroutine* of M', if M' runs M on s free bits of its work tape and c bits of its catalytic tape; in particular, M will be run using the c bits written on M' as its initial catalytic tape.

We may further say that M' simulates M or M is a catalytic subroutine of M'

- *with access to* σ: M' can access string σ during the execution of M; this may be written on the catalytic or work tape outside of the space used to run M, or it may be derivable from the information written in this space via a separate catalytic subroutine
- *using additional workspace* k: M' uses an additional k bits of free work memory in its execution of M, which, like the free work memory of the catalytic subroutine M, may be reclaimed by M' afterwards
- *with end catalytic state* $\mathbf{w}$: the end result is to change the catalytic memory set aside for M into state $\mathbf{w}$ (note that this is unlike a typical run of a catalytic machine, which resets the memory to its original configuration)
- *returning* a: the end result is to write a to the free work memory of M'.

All other standard qualifiers, i.e. running in time t, can be applied as usual.

Recall the standard definitions of a randomized machine M computing a function f: on input x, M outputs

- *zero-sided* error: $f(x)$ with probability at least $2/3$ and $\perp$ otherwise
- *one-sided* error: if $f(x) = 1$, $f(x)$ with probability 1; if $f(x) = 0$, $f(x)$ with probability at least $2/3$ and $\overline{f(x)}$ otherwise
- *two-sided* error: $f(x)$ with probability at least $2/3$ and $\overline{f(x)}$ otherwise

where the probability is only with respect to the randomness of M and is independent of x. Note that the error probability $1/3$ can be set to be an arbitrary constant in $(0, 1/2)$, as one can apply standard the error reduction techniques.

Definition 2.3 ([DGJ+20])**.** A *randomized catalytic machine* M is defined as a catalytic Turing machine with access to a uniformly random string r. As in the standard model of randomized space-bounded computation, M may only access r in a *one-way* fashion, and must halt in finite time with certainty.

We define *zero-sided*, *one-sided*, and *two-sided* error as above; in particular, a randomized catalytic machine computes a function f, in any of these senses, iff the probability of success depends only on the randomness of r (in particular, it holds for every value of x and $\mathbf{w}$). Furthermore, we require that $\mathbf{w}$ is reset on every computation path, i.e. no matter the contents of the random tape and what M outputs.

This gives rise to a natural structural theory of catalytic space paralleling that of ordinary complexity theory.

Definition 2.4. We define catalytic variants of standard space-bounded classes as follows:

- $\mathsf{CSPACE}[s]$ is the class of functions that can be recognized by catalytic Turing machines using workspace $O(s)$ and catalytic space $2^{O(s)}$.
- $\mathsf{CZPSPACE}[,]\,\mathsf{CRSPACE}, \mathsf{CBPSPACE}[s]$[6] are the classes of functions that can be recognized by randomized catalytic Turing machines using workspace $O(s)$, catalytic space $2^{O(s)}$, and access to $2^{O(s)+2^{O(s)}}$ random bits, with zero-sided, one-sided, and two-sided error, respectively.
- $\mathsf{CTISP}[t, s]$ is the class of functions that can be recognized by catalytic Turing machines using time $O(t)$, workspace $O(s)$, and catalytic space $2^{O(s)}$.
- $\mathsf{CZPTISP}[t, s], \mathsf{CRTISP}, \mathsf{CBPTISP}[t, s]$ are the classes of functions that can be recognized by randomized catalytic Turing machines using time $O(t)$, workspace $O(s)$, catalytic space $2^{O(s)}$, and access to $O(t)$ random bits, with zero-sided, one-sided, and two-sided error, respectively.

Furthermore we define the following specifications of the above classes to the logspace setting, which is the instantiation of the most interest to the present work:

- $\mathsf{CL} := \mathsf{CSPACE}[\log n]$
- $\mathsf{CBPL} := \mathsf{CBPSPACE}[\log n]$
- $\mathsf{CLP} := \mathsf{CTISP}[\mathrm{poly}(n), \log n]$ (also called CSC^1 [DGJ+20])
- $\mathsf{CZPLP} := \mathsf{CZPTISP}[\mathrm{poly}(n), \log n]$

[6] While all published works on the subject of randomized catalytic space [DGJ+20, Mer23, Pyn23, DPT24] put C before e.g. BP in $\mathsf{CBPSPACE}[s]$, they first appear in an older, yet unpublished, work by Dulek, which reverses the order. Theorem 1.1, thankfully, all but obviates the need to solve this nomenclature issue.

Note that while some works consider the more general case of $\mathbf{CSPACE}[s, c]$, where the catalytic tape may take some variable length c different from 2^s (see e.g. [BDS22, Pyn23]), Definition 2.4 is the most standard setting and the one of interest to the current work.

While our main results are stated in terms of catalytic logspace, we will prove them in generality for different values of s and t. One subtlety here is that such values themselves need to be easily computable:

Definition 2.5. We say that a function $\ell := \ell(n)$ is *constructible* in space $s := s(n)$ if there exists a machine M_ℓ using space $s(n)$ which takes in 1^n and outputs the value $\ell(n)$.

We say ℓ is *space constructible* if ℓ is constructible in space $O(\ell)$; it is said to be *logspace constructible* if ℓ is constructible in space $O(\log \ell)$.

2.3 Configuration Graphs of Catalytic Machines

We define the configuration graph of a catalytic machine, and how one can traverse it using catalytic subroutines. Throughout this subsection, we assume that $s := s(n) \geq \log n$ is a space constructible function.

2.3.1 Deterministic Configuration Graphs

We start with defining and manipulating configuration graphs of deterministic catalytic machines, which we will use in Section 3.

Definition 2.6 (Configuration graphs of deterministic catalytic machines)**.** Let M be a deterministic catalytic machine that uses s bits of workspace and 2^s bits of catalytic space on inputs of length n, and let $x \in \{0,1\}^n$ be an input. The $\mathsf{configuration\ graph}$ $\mathcal{G}_x$ is a directed graph defined as follows:

- Each node is a configuration $(\mathbf{w}, v)$ of M, where $\mathbf{w} \in \{0,1\}^{2^s}$ is the catalytic tape configuration and $v \in \{0,1\}^s$ is the bits of auxiliary state.[7]
- We say a vertex σ is a $\mathsf{starting\ state}$ if $\sigma = (\mathbf{w}, 0^s)$ for some $\mathbf{w}$ (and WLOG assume that the machine starts in such a configuration). We let this state be denoted $\mathsf{start}(\mathbf{w})$.
- We say a vertex σ is a $\mathsf{halting\ state}$ if $\sigma = (\mathbf{w}, b \cdot 1^{s-1})$ for some $\mathbf{w}$ and $b \in \{0,1\}$ (and WLOG assume that the machine always returns b upon reaching such a configuration). We let this state be denoted $\mathsf{halt}(\mathbf{w}, b)$, and let $\mathsf{acc}(\mathbf{w}) = \mathsf{halt}(\mathbf{w}, 1)$.
- Each non-halting configuration $(\mathbf{w}, v)$ has a single out-edge to $(\mathbf{w}', v')$, which is the configuration of M after one step execution from the configuration $(\mathbf{w}, v)$ on input x. We define $\Gamma(\mathbf{w}, v) := (\mathbf{w}', v')$.

We may drop the subscript x and denote $\mathcal{G}_x$ by $\mathcal{G}$ when the input x is clear in the context.

We will need a way for a catalytic machine M' to simulate another catalytic machine M in a way that is reversible: after running the simulation forward for some number of steps, M' must be able to just as quickly run the simulation backward the same number of steps, restoring the catalytic tape to where it started. The simulation need not follow the same sequence of steps as M itself, but running it to the end must produce the same output as M. The following theorem makes this precise.

[7] The state description v should be of length $O(s)$ to keep track of the FSA configuration and tape head locations of M, but we ignore this technicality for the sake of simplicity.

Theorem 2.7. *For every catalytic machine M computing a language L using workspace $s := s(n) \geq \log n$ with configuration graph $\mathcal{G}$, there exist catalytic subroutines* $\mathsf{DetWalk}, \mathsf{DetRev}$ *that run in worst-case time* $\mathrm{poly}(2^s, k)$ *and work as follows. There exists a bijection*

$$\Pi : \mathcal{G}_x \times \{0,1\} \to \mathcal{G}_x \times \{0,1\}$$

so that for every $\mathbf{w}$*, the sequence*

$$(\mathsf{start}(\mathbf{w}), 0), \Pi[\mathsf{start}(\mathbf{w}), 0], \Pi^2[\mathsf{start}(\mathbf{w}), 0], \ldots$$

includes $(\mathsf{halt}(\mathbf{w}, b), 0) = ((\mathbf{w}, b1^{s-1}), 0)$*, where* $b = L(x)$*, and does not include* $(\mathsf{start}(\mathbf{w}'), 0)$ *for any* $\mathbf{w}' \neq \mathbf{w}$ *or* $(\mathsf{halt}(\mathbf{w}', b'), 0)$ *for any* $\mathbf{w}' \neq \mathbf{w}$ *or* $b' \neq L(x)$*.*

For every $k \in \mathbb{N}$ *and* $\mathbf{w}$*, either:*

1. $\mathsf{DetWalk}^{\mathbf{w}}(x, k)$ *returns* $L(x)$ *(and does not modify the catalytic tape).*
2. $\mathsf{DetWalk}^{\mathbf{w}}(x, k)$ *sets the catalytic tape to* $\mathbf{w}'$ *and returns* v', a'*, where* $((\mathbf{w}', v'), a') = \Pi^k[\mathsf{start}(\mathbf{w}), 0]$*. Moreover,* $\mathsf{DetRev}^{\mathbf{w}'}(x, v', a')$ *sets the catalytic tape to* $\mathbf{w}$ *and returns* k*.*

As this result involves technical manipulations of the configuration graph of catalytic machines, we defer the proof to Appendix B; the code can be found in Algorithms 1 and 2. The essential approach is to take an Eulerian tour through the configuration graph, verifying that all operations can be done in-place on the catalytic tape, and in polynomial time. As the theorem crucially uses that the configuration graph has out-degree 1, we must take a separate approach for the randomized case, which we discuss later.

2.3.2 Randomized Configuration Graphs

Similar to the deterministic case, we can define the configuration graph of a randomized catalytic machine and related notions, which we will use in Section 4. Some of the concepts and results have been implicit in literature of catalytic computation; nevertheless, we provide a self-contained description for completeness.

Definition 2.8 (Configuration graphs of randomized catalytic machines)**.** Let M be a randomized catalytic machine that uses s bits of workspace and 2^s bits of catalytic space on inputs of length n, and let $x \in \{0,1\}^n$ be an input. The configuration graph $\mathcal{G}_x$ is a directed graph defined as follows:

- Each node is a configuration $(\mathbf{w}, v)$ of M, where $\mathbf{w} \in \{0,1\}^{2^s}$ is the catalytic tape configuration and $v \in \{0,1\}^s$ is the bits of auxiliary state.[8]
- We say a vertex σ is a starting state if $\sigma = (\mathbf{w}, 0^s)$ for some $\mathbf{w}$ (and WLOG assume that the machine starts in such a configuration). We let this state be denoted $\mathsf{start}(\mathbf{w})$.
- We say a vertex σ is a halting state if $\sigma = (\mathbf{w}, b \cdot 1^{s-1})$ for some $\mathbf{w}$ and $b \in \{0,1\}$ (and WLOG assume that the machine always returns b upon reaching such a configuration). We let this state be denoted $\mathsf{halt}(\mathbf{w}, b)$, and let $\mathsf{acc}(\mathbf{w}) = \mathsf{halt}(\mathbf{w}, 1)$.
- Each non-halting configuration $(\mathbf{w}, v)$ has two out-edges to $(\mathbf{w}_0, v_0)$ and $(\mathbf{w}_1, v_1)$, where $(\mathbf{w}_b, v_b)$ is the configuration of M after one step execution from the configuration $(\mathbf{w}, v)$ on input x if the random bit probed by the machine in this step is $b \in \{0,1\}$. We define $\Gamma_b(\mathbf{w}, v) := (\mathbf{w}_b, v_b)$.

[8] As in the deterministic case, the state description v should be of length $O(s)$ to keep track of the FSA configuration and tape head locations of M, but we ignore this technicality for the sake of simplicity.

We may drop the subscript x and denote $\mathcal{G}_x$ by $\mathcal{G}$ when the input x is clear in the context.

The key difference between Definition 2.6 and Definition 2.8 is the outdegree of non-halting configurations, which increases due to conditioning on different random strings. This will greatly affect how we reconfigure the machine to travel forwards and backwards à la Theorem 2.7; furthermore, while a walk may end with an output being produced, this output is no longer guaranteed to be the correct value of $L(x)$, as the specific randomness r may cause our machine to err.

Definition 2.9. For a configuration graph $\mathcal{G}_x$ of a randomized catalytic machine, a configuration $\sigma \in \mathcal{G}_x$, and a string $r \in \{0,1\}^\ell$, we define $\mathcal{G}_x[\sigma, r]$ as the configuration σ_ℓ reached by taking a walk of length ℓ according to r, i.e.,

$$\sigma_0 := \sigma, \sigma_1 := \Gamma_{r_1}(\sigma_0), \ldots, \sigma_\ell := \Gamma_{r_\ell}(\sigma_{\ell-1}). \tag{2}$$

If the initial configuration σ is clear in the context, we may slightly abuse the notation to identify r and the walk specified by r in (2).

These walks will take the place of Π in the statement of Theorem 2.7, and will allow us to quantify the behavior of our forward and backward machines. (In contrast to Theorem 2.7, these walks exactly match computations of the original catalytic machine. For Theorem 2.7, it was necessary to invent an invertible transition function Π in order to allow running backward in a time-efficient way; here we avoid that complication but give no runtime guarantee.)

Theorem 2.10. *For every randomized catalytic machine M computing a language L using workspace $s := s(n) \geq \log n$ with configuration graph $\mathcal{G}$, there exist catalytic subroutines* $\mathsf{RandWalk}, \mathsf{RandRev}$ *that use $O(s + \log |r|)$ additional workspace and work as follows.*

- $\mathsf{RandWalk}^{\mathbf{w}}(x, v, r)$ *sets the catalytic tape to $\mathbf{w}'$ and returns v', where $\mathcal{G}_x[(\mathbf{w}, v), r] = (\mathbf{w}', v')$. In addition,* $\mathsf{RandWalk}^{\mathbf{w}}(x, v, r)$ *only requires one-way access to r.*
- *If there is a catalytic tape configuration $\mathbf{w}$ such that $\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = (\mathbf{w}', v')$,* $\mathsf{RandRev}^{\mathbf{w}'}(x, v', r)$ *accepts and leaves the catalytic tape in configuration $\mathbf{w}$; otherwise, it rejects and leaves the catalytic tape in configuration $\mathbf{w}'$.*

As Theorem 2.10 involves technical manipulations of configuration graphs, we defer the proof to Appendix C, while the code for $\mathsf{RandRev}$ can be found in Algorithm 3.

Lastly, we will need one other tool for randomized configuration graphs, namely to compare the results of two different walks, each starting from the same $\mathsf{start}(\mathbf{w})$ but generated by different random strings $r, r' \in \{0,1\}^*$.

Lemma 2.11. *There is a catalytic subroutine* $\mathsf{EQ}(x, r, r')$ *using $O(\log(n) + \log(|r|) + \log(|r'|))$ additional workspace such that* $\mathsf{EQ}^{\mathbf{w}}(x, r, r')$ *accepts if and only if $\mathcal{G}_x[\mathsf{start}(\mathbf{w}), r] = \mathcal{G}_x[\mathsf{start}(\mathbf{w}), r']$.*

Proof. Let $(\mathbf{w}_r, v_r) := \mathcal{G}[\mathsf{start}(\mathbf{w}), r]$ and $(\mathbf{w}_{r'}, v_{r'}) := \mathcal{G}[\mathsf{start}(\mathbf{w}), r']$; the goal of $T^{\mathbf{w}}(r, r')$ is to determine whether or not $(\mathbf{w}_r, v_r) = (\mathbf{w}_{r'}, v_{r'})$. The algorithm compares $(\mathbf{w}_r, v_r)$ and $(\mathbf{w}_{r'}, v_{r'})$ bit by bit. Let $\ell := \max\{|(\mathbf{w}_r, v_r)|, |(\mathbf{w}_{r'}, v_{r'})|\}$. For each $i \in \{1, 2, \ldots, \ell\}$, it works as follows:

1. Let $v_r := \mathsf{RandWalk}^{\mathbf{w}}(x, 0^s, r)$. The catalytic tape will be $\mathbf{w}_r$, where $(\mathbf{w}_r, v_r) := \mathcal{G}[\mathsf{start}(\mathbf{w}), r]$. Let b be the i-th bit of $(\mathbf{w}_r, v_r)$. We then call $\mathsf{RandRev}^{\mathbf{w}_r}(x, v_r, r)$ to reset the catalytic tape to $\mathbf{w}$.
2. Let $v_{r'} := \mathsf{RandWalk}^{\mathbf{w}}(x, 0^s, r')$. The catalytic tape will be $\mathbf{w}_{r'}$, where $(\mathbf{w}_{r'}, v_{r'}) := \mathcal{G}[\mathsf{start}(\mathbf{w}), r']$. Let b' be the i-th bit of $(\mathbf{w}_{r'}, v_{r'})$. We then call $\mathsf{RandRev}^{\mathbf{w}_{r'}}(x, v_{r'}, r)$ to reset the catalytic tape to $\mathbf{w}$.

3. The algorithm rejects if $b \neq b'$.

See Algorithm 4 for the pseudocode of EQ.

The correctness and the space complexity of the algorithm follow directly from the correctness and the space complexity of RandWalk and RandRev (see Theorem 2.10). □

3 Structural Results for CL

In this section, we introduce a reduction from **CL** to LossyCode, and derive new structural results: **CLP** = **CL** ∩ **P** and **CL** ⊆ **CZPLP**. Finally, we show that the reduction implies a "certified derandomization" for **CL**.

3.1 CL Reduces to Lossy Code

Our main compression algorithm for **CSPACE**$[s]$ is as follows:

Theorem 3.1. *Let $s := s(n) \geq \log n$ be space constructible. For every $L \in$ **CSPACE**$[s]$, there are catalytic subroutines* DetComp *and* DetDecomp *with worst-case* $\mathrm{poly}(2^s, B)$ *running time using additional workspace $O(s+\log B)$ that work as follows. Let $\mathbf{w}$ be a length-$(2^s+O(s)+B)$ configuration of the catalytic tape and $x \in \{0,1\}^n$ be an input. Then the subroutines work as follows:*

- $\mathsf{DetComp}^{\mathbf{w}}(1^B, x)$ *either returns $L(x)$ and resets the catalytic tape, or returns $\perp$ and sets the catalytic tape to be of form $\mathbf{w}' \circ 0^B$, where $|\mathbf{w}'| = 2^s + O(s)$.*
- $\mathsf{DetDecomp}^{\mathbf{w}' \circ 0^B}(1^B, x)$ *sets the catalytic tape to $\mathbf{w}$.*

Proof. Let M be a catalytic machine using s workspace that decides L and let $\mathcal{G}_x$ be the configuration graph of M on input x.

The compression algorithm DetComp**.** We implement DetComp as the following iterative algorithm. It maintains a counter $k \in \{0, \ldots, B\}$ and maintains the invariant that at the end of the k-th iteration, either C returns $L(x)$ and resets the catalytic tape, or it sets the catalytic tape to be of the form $\mathbf{w}'_k \circ 0^k$ such that $\mathsf{DetDecomp}^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ (which will be defined later) will set the catalytic back to $\mathbf{w}$.

Let $k := 0$ and $\mathbf{w}'_0 := \mathbf{w}$ at the beginning, and thus the invariant holds trivially. Assume that we are at the beginning of the $k+1$-st iteration for some $k \in [B]$. We know by the invariant that the catalytic tape is of the form $\mathbf{w}'_k \circ 0^k$. We parse $\mathbf{w}'_k$ as

$$\mathbf{w}'_k = \mathbf{m} \circ i \circ z$$

where $\mathbf{m}$ is of length 2^s, i is of length $s+2$ (which we interpret as a number in $[2^{s+2}]$), and z is the remaining string of length at least $B - k$. Next, we call the machine DetWalk of Theorem 2.7 with input parameters $\mathsf{DetWalk}^{\mathbf{m}}(x, i)$. Then one of two events occurs:

1. First, suppose DetWalk returns $L(x)$ and resets the catalytic tape to $\mathbf{m}$. If $k \geq 1$, we call the machine $D^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ so that by the invariant we will reset the catalytic tape to $\mathbf{w}$.
2. Otherwise, the machine halts with the section of catalytic tape in configuration $\mathbf{m}'$ and returns $(v', a') \in \{0,1\}^{s+1}$. In this case, we set the full catalytic tape to

   $$(\mathbf{w}'_{k+1} := \mathbf{m}' \circ (v' \circ a') \circ z) \circ 0^{k+1}$$

   where $\mathbf{m}'$ is of length 2^s, $(v' \circ a')$ is of length $s+1$, and z is as before. We increment k and go to the start of the loop.

If the loop counter reaches B, we halt and return $\perp$. See Algorithm 5 for the pseudocode of DetComp.

It is clear that all our loop invariants hold in each iteration; note that in each new iteration of the loop, i requires $s+2$ bits, which will be taken from $(v' \circ a')$ from the previous iteration plus one bit from z from the previous iteration. It is also clear that the runtime is bounded by $\mathrm{poly}(2^s)$ in both cases.

The decompression algorithm DetDecomp**.** As we mentioned above, the catalytic machine DetDecomp will satisfy the following property. Assume that $\mathsf{DetComp}^{\mathbf{w}}(1^B, x)$ does not halt in the k-th iteration. For its catalytic tape $\mathbf{w}'_k \circ 0^k$ at the end of the k-th iteration, $\mathsf{DetDecomp}^{\mathbf{w}'_k}(1^k, x)$ sets the catalytic tape to $\mathbf{w}$. For simplicity of presentation and analysis, we define $\mathsf{DetDecomp}^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ as a recursive algorithm, while it can be easily converted to an equivalent iterative algorithm.

Fix any k. The catalytic machine $\mathsf{DetDecomp}^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ works as follows. We interpret $\mathbf{w}'_k$ as

$$\mathbf{w}'_k = \mathbf{m}' \circ (v' \circ a') \circ z$$

where $\mathbf{m}'$ is of length 2^s and (v', a') is of length $s+1$. We then run the machine DetRev of Theorem 2.7 as $\mathsf{DetRev}^{\mathbf{m}'}(x, v', a')$. Let $\mathbf{m}$ and i be such that $\mathsf{DetWalk}^{\mathbf{m}}(x, i)$ sets the catalytic tape to $\mathbf{m}'$ and returns (v', a'). By Theorem 2.7, $\mathsf{DetRev}^{\mathbf{m}'}(x, v', a')$ will thus set the catalytic tape to $\mathbf{m}$ and return i. We then set the overall catalytic tape to

$$\mathbf{m} \circ i \circ z \circ 0^{k-1}.$$

By the definition of DetComp and DetDecomp, we can see that the computation of $\mathsf{DetDecomp}^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ as we described above is exactly the reverse simulation of the i-th iteration of $\mathsf{DetComp}^{\mathbf{w}}(1^B, x)$. Therefore, $\mathbf{m} \circ i \circ z \circ 0^{k-1}$ is the catalytic tape $\mathbf{w}'_{k-1} \circ 0^{k-1}$ after the first $k-1$ iterations of $\mathsf{DetComp}^{\mathbf{w}}(1^B, x)$. The algorithm DetDecomp then recursively calls $\mathsf{DetDecomp}^{\mathbf{w}'_{k-1} \circ 0^{k-1}}(1^{k-1}, x)$ and by the invariant the catalytic tape is reset to $\mathbf{w}$. See Algorithm 6 for the pseudocode of DetDecomp.

Analysis. Note that the correctness of the algorithm follows directly from the invariant in the iterative algorithm DetComp. Each of DetComp and DetDecomp has B iterations (which requires an $O(\log B)$-bits counter), in each of which it simulates or backward simulates M using $2^{O(s)}$ time and $O(s)$ space. Therefore, both DetComp and DetDecomp run in $\mathrm{poly}(2^s, B)$ time and require $O(s + \log B)$ workspace. □

An immediate corollary of Theorem 3.1 is that **CL** is contained in **LOSSY**. Recall that **LOSSY** is the class of languages reducible to the total search problem LossyCode [Kor22].

Definition 1.5. The complexity class **LOSSY** is defined as the languages that are polynomial-time reducible to the following total search problem called LossyCode: Given a pair of Boolean circuits $C : \{0,1\}^n \to \{0,1\}^{n-1}$ and $D : \{0,1\}^{n-1} \to \{0,1\}^n$, find some $x \in \{0,1\}^n$ such that $D(C(x)) \neq x$.

Theorem 1.6.

$$\mathbf{CL} \subseteq \mathbf{LOSSY} (\subseteq \mathbf{ZPP}).$$

Proof. Let $L \in \mathbf{CL}$ and M be a catalytic machine using $s := s(n) = O(\log n)$ bits of workspace that decides L. By Theorem 3.1 with $B = 1$, we can obtain the catalytic subroutines DetComp and DetDecomp that runs in worst-case $\mathrm{poly}(2^{O(\log n)}) = \mathrm{poly}(n)$ time. This implies that there are polynomial-time algorithms $\mathsf{DetComp}'$ and $\mathsf{DetDecomp}'$ such that:

- $\mathsf{DetComp}'(x, \mathbf{w})$ takes $x \in \{0,1\}^n$ and a catalytic tape configuration of length 2^s and simulates $\mathsf{DetComp}^{\mathbf{w}}(1, x)$. It outputs 0^{2^s-1} if $\mathsf{DetComp}^{\mathbf{w}}(1, x)$ does not output $\perp$, and otherwise it outputs the first $s-1$ bits of the catalytic tape after the simulation.

- $\mathsf{DetDecomp}'(x, \mathbf{w}')$ takes $x \in \{0,1\}^n$ and a string $\mathbf{w}'$ of length 2^s-1. It simulates $\mathsf{DetDecomp}^{\mathbf{w}' \circ 0}(1, x)$ and outputs the catalytic tape of D after the simulation.

Our reduction from L to $\mathsf{LossyCode}$ works as follows: Given any input $x \in \{0,1\}^n$, it constructs (by standard transformation of algorithms to circuits) a pair of circuits computing $\mathsf{DetComp}'(x, \cdot) : \{0,1\}^{2^s} \to \{0,1\}^{2^s-1}$ and $\mathsf{DetDecomp}'(x, \cdot) : \{0,1\}^{2^s-1} \to \{0,1\}^{2^s}$.

Let $\mathbf{w}^*$ be a solution to the $\mathsf{LossyCode}$ instance $(\mathsf{DetComp}'(x, \cdot), \mathsf{DetDecomp}'(x, \cdot))$, i.e.,

$$\mathsf{DetDecomp}'(x, \mathsf{DetComp}'(x, \mathbf{w}^*)) \neq \mathbf{w}^*.$$

By the correctness of the catalytic subroutines $\mathsf{DetComp}$ and $\mathsf{DetDecomp}$ (see Theorem 3.1), we know that $\mathsf{DetComp}^{\mathbf{w}^*}(1, x)$ outputs $L(x)$. We can then simulate $\mathsf{DetComp}^{\mathbf{w}^*}(1, x)$ in polynomial-time and outputs the answer. □

3.2 Structural Results for CL and CLP

We now use our compression algorithm in Theorem 3.1 to prove Theorem 1.2, our main structural result for time-bounded catalytic computing.

Theorem 3.2. *For all space constructible function $s := s(n) \geq \log n$ and logspace constructible function $t := t(n) \geq n$,*

$$\mathbf{CTISP}\left[2^{O(s)} \cdot t^{O(1)}, s + \log t\right] = \mathbf{CSPACE}\left[s + \log t\right] \cap \mathbf{DTIME}\left[2^{O(s)} \cdot t^{O(1)}\right].$$

In particular, $\mathbf{CLP} = \mathbf{CL} \cap \mathbf{P}$.

Proof. Fix any $s := s(n) \geq \log n$ and $t(n) \geq n$. The forward containment is immediate from the definitions, so it suffices to prove the other direction.

Let $L \in \mathbf{CSPACE}\left[s + \log t\right] \cap \mathbf{DTIME}\left[2^{O(s)} \cdot t^{O(1)}\right]$, M be a $\mathbf{CSPACE}\left[s + \log t\right]$ machine that decides L, and M' be a (possibly space inefficient) machine with running time $O(t^k \cdot 2^{k \cdot s})$ that decides L for some constant $k \geq 1$. We describe a catalytic machine for L as follows. We first simulate the machine $\mathsf{DetComp}$ of Theorem 3.1 for M with input $(x, 1^{2^{(k+1)s} \cdot t^{k+1}})$.

- If $\mathsf{DetComp}$ returns a value rather than $\perp$, we return that value and halt, where by Theorem 3.1 the catalytic tape has been successfully reset and the machine decides whether $x \in L$.

- Otherwise, we run the machine M' on the last $2^{(k+1)s} \cdot t^{k+1}$ bits on the catalytic tape (which are all zero after running C). It decides whether $x \in L$; we store the result on the work tape, set the last $2^{(k+1)s} \cdot t^{k+1}$ bits on the catalytic tape back to all zero, and call the decompression algorithm $\mathsf{DetDecomp}$ with input $(x, 1^{2^{(k+1)s} \cdot t^{k+1}})$. By the correctness of $\mathsf{DetComp}$ and $\mathsf{DetDecomp}$, the catalytic tape will be reset, and we can decide whether $x \in L$.

Recall that both $\mathsf{DetComp}$ and $\mathsf{DetDecomp}$ run in time

$$\mathrm{poly}(2^s, 2^{(k+1)s} \cdot t^{k+1}) = 2^{O(s)} \cdot t^{O(1)}$$

time and use workspace

$$O\left(s + \log t + \log\left(2^{(k+1)s} \cdot t^{k+1}\right)\right) = O(s + \log t).$$

This shows that our catalytic machine runs in $2^{O(s)} \cdot t^{O(1)}$ time and uses $O(s + \log t)$ workspace simultaneous, which implies that $L \in \mathsf{CTISP}\left[2^{O(s)} \cdot t^{O(1)}, s + \log t\right]$. □

From Theorem 3.2 we immediately obtain multiple corollaries.

Corollary 1.3.

$$\mathsf{CL} \subseteq \mathsf{P} \iff \mathsf{CL} = \mathsf{CLP}.$$

Proof. Suppose that $\mathsf{CL} \subseteq \mathsf{P}$, we know by Theorem 3.2 that $\mathsf{CLP} = \mathsf{CL} \cap \mathsf{P} = \mathsf{CL}$. On the other hand, $\mathsf{CL} = \mathsf{CLP}$ immediately implies that $\mathsf{CL} \subseteq \mathsf{P}$ as $\mathsf{CLP} \subseteq \mathsf{P}$. □

Corollary 1.4.

$$\mathsf{ZPP} = \mathsf{P} \implies \mathsf{CL} = \mathsf{CLP}.$$

Proof. Suppose that $\mathsf{ZPP} = \mathsf{P}$, we know that $\mathsf{CL} \subseteq \mathsf{ZPP} \subseteq \mathsf{P}$. This immediately implies that $\mathsf{CL} = \mathsf{CL} \cap \mathsf{P} = \mathsf{CLP}$. □

Corollary 1.7.

$$\mathsf{LOSSY} = \mathsf{P} \implies \mathsf{CL} = \mathsf{CLP}.$$

Proof. Suppose that $\mathsf{LOSSY} = \mathsf{P}$, we know that $\mathsf{CL} \subseteq \mathsf{LOSSY} \subseteq \mathsf{P}$ (see Theorem 1.6). This immediately implies that $\mathsf{CL} = \mathsf{CL} \cap \mathsf{P} = \mathsf{CLP}$. □

3.3 A New Uniform Upper Bound for CL

We now use the proof of Theorem 3.2 to obtain the first half of Theorem 1.8. Our only change will be to no longer assume that we are dealing with a language in P, but rather in ZPP. Buhrman et al. [BCK+14] showed that in fact such a containment holds without any further assumptions:

Theorem 3.3 ([BCK+14])**.** *For all space constructible functions $s := s(n) \geq \log n$,*

$$\mathsf{CSPACE}[s] \subseteq \mathsf{ZPTIME}\left[2^{O(s)}\right].$$

In particular, $\mathsf{CL} \subseteq \mathsf{ZPP}$.

This is sufficient to prove the forward direction of Theorem 1.8.

Theorem 3.4. *For all space constructible functions $s := s(n) \geq \log n$,*

$$\mathsf{CSPACE}[s] \subseteq \mathsf{CZPTISP}\left[2^{O(s)}, s\right]$$

In particular, $\mathsf{CL} \subseteq \mathsf{CZPLP}$.

Proof. Let $L \in \mathsf{CSPACE}[s]$ and M be a catalytic $O(s)$-space machine that decides L. Thus by Theorem 3.3 we know that $L \in \mathsf{ZPTIME}\left[2^{O(s)}\right]$. Let M' be the (possibly space inefficient) zero-error probabilistic machine that decides L in time $O(2^{k \cdot s})$ for some constant k.

Consider the following probabilistic catalytic machine for L. Given any input x, we first simulate the machine $\mathsf{DetComp}$ of Theorem 3.1 for M with input $(x, 1^{2^{2k \cdot s}})$.

- If DetComp returns a value rather than $\perp$, we return that value and halt. Note that by Theorem 3.1 the catalytic tape has been successfully reset and DetComp decides whether $x \in L$.
- Otherwise, we run the zero-error probabilistic machine M' on the last $2^{2k \cdot s}$ bits on the catalytic tape (which are all zero after running DetComp). This is possible as we can probe sufficiently many random bits, store them on the catalytic tape, and simulate M' using the stored random bits. It stores the output of M' (which is in $\{0, 1, \perp\}$), set the last $2^{(k+1)s}$ bits on the catalytic tape back to all zero, and call the decompression algorithm DetDecomp with input $(x, 1^{2^{(k+1)s}})$ to reset the catalytic tape. Then we output the stored output value of M'.

Note that in the former case, our algorithm decides whether $x \in L$ with certainty; in the latter case, our algorithm simulates M' so that it never makes mistake and outputs $\perp$ with a negligible probability. This concludes the correctness of the algorithm.

It is clear that both DetComp and DetDecomp run in time $\text{poly}(2^s)$ and use $O(s + \log(2^{O(s)})) = O(s)$ workspace. Therefore, the algorithm runs in time $2^{O(s)}$ and uses $O(s)$ workspace simultaneous, which implies that $L \in \mathbf{CZPTISP}\left[2^{O(s)}, s\right]$. □

4 Derandomizing CBPL

Our second set of results will be on derandomization for catalytic computation.

Theorem 4.1. *For all space constructible functions $s := s(n) \geq \log n$,*

$$\mathbf{CBPSPACE}\left[s\right] \subseteq \mathbf{CSPACE}\left[s\right]$$

In particular, $\mathbf{CBPL} \subseteq \mathbf{CL}$.

We require two sets of tools to prove Theorem 4.1: 1) subroutines to manipulate the configuration graphs of randomized catalytic algorithms; and 2) tools from pseudorandomness to generate random walks, and compress if these walks are non-random. The first was discussed in Section 2.3.2, while we address the latter shortly.

Throughout this section, let $s := s(n) \geq \log n$ be a space constructible function and M be a $\mathbf{CBPSPACE}\left[s\right]$ machine deciding a language L. Without loss of generality, we assume that M probes a random bit in each step of its execution. As with Theorem 3.4, we will need a uniform upper bound on CBPL in order to complete the proof, which was provided by Datta et al. [DGJ+20]:

Theorem 4.2 ([DGJ+20])**.** *For all space constructible functions $s := s(n) \geq \log n$,*

$$\mathbf{CBPSPACE}\left[s\right] \subseteq \mathbf{ZPTIME}\left[2^{O(s)}\right].$$

In particular, $\mathbf{CBPL} \subseteq \mathbf{ZPP}$.

This will be necessary for handling the compression case; in fact we will only need the weaker statement that $\mathbf{CBPSPACE}\left[s\right]$ can be computed in $\mathbf{SPACE}\left[2^{O(s)}\right]$.

Proof of Theorem 4.1. Let $L \in \mathbf{CBPSPACE}\left[s\right]$, let M be a randomized space s catalytic machine deciding L, and let $\mathcal{G}$ be the configuration graph of M. Let $S := 2^s$. Let $s_L \leq 2^{O(s)}$ be such that L can be computed in $\mathbf{SPACE}\left[s_L\right]$, where the bound follows from Theorem 4.2.

Our approach will be similar to that of Theorem 1.2, but rather than compress the timestamp along *one* path, we will compress based on the number of configurations seen along *many* paths, obtained from a set of random walks.

Definition 4.3. For a randomized configuration graph $\mathcal{G}_x$, initial configuration $\mathsf{start}(\mathbf{w})$, and collection of walks $Y \subseteq \{0,1\}^m$, we define the subgraph of states in $\mathcal{G}$ reached by walks in Y as follows. Let $\mathcal{Y} := \mathcal{Y}(\mathbf{w}, Y)$ be the graph with vertex set (which we abuse notation and denote $\mathcal{Y}$)[9]

$$\left\{(\mathbf{w}', v) \in \mathcal{G} : \exists y \in Y, i \in \mathbb{N} \text{ s.t. } \mathcal{G}[\mathsf{start}(\mathbf{w}), y_{\leq i}] = (\mathbf{w}', v)\right\} \cup \{\perp\}.$$

where $\perp$ is an auxiliary state which we define subsequently. Next, we define the transition function on $\mathcal{Y}$ as follows.

$$\mathcal{Y}[(\mathbf{w}, v), b] := \begin{cases} \mathcal{G}[(\mathbf{w}, v), b] & \mathcal{G}[(\mathbf{w}, v), b] \in \mathcal{Y} \\ \perp & \text{o.w.} \end{cases}$$

In words, all edges internal to $\mathcal{Y}$ are retained, and all edges that leave $\mathcal{Y}$ are routed to a new dummy state $\perp$.

As an analogue of Theorem 3.1, we can efficiently compress the catalytic tape only using the fact that the explored graph via a collection Y of walks is sufficiently large.

Theorem 4.4. *There are catalytic subroutines* $\mathsf{Size}, \mathsf{RandComp}, \mathsf{RandDecomp}$ *that work as follows. For every* $x \in \{0,1\}^n$*, collection of walks* $Y \subseteq \{0,1\}^m$*, and catalytic tape* $\mathbf{w}$ *of size* 2^s*, let* $\mathcal{Y} := \mathcal{Y}(\mathbf{w}, Y)$*. There is a bijection* $f : \mathcal{Y} \to [|\mathcal{Y}|]$ *such that:*

- $\mathsf{Size}^{\mathbf{w}}(x, Y)$ *outputs* $|\mathcal{Y}|$ *without changing the catalytic tape.*
- $\mathsf{RandComp}^{\mathbf{w}}(x, i, Y)$ *sets the catalytic tape to* $\mathbf{w}'$ *and returns* v'*, where* $f((\mathbf{w}', v')) = i$*.*
- $\mathsf{RandDecomp}^{\mathbf{w}'}(x, v', Y)$ *sets the catalytic tape to* $\mathbf{w}$ *and returns* $i := f((\mathbf{w}', v'))$*.*

Moreover, all subroutines use $O(s + \log |Y|)$ *additional workspace.*

Unlike in Theorem 3.1, we have no immediate guarantees if our compression fails, i.e. if the set $\mathcal{Y}$ is small. We will require a second algorithm, which will follow the classic "compress-or-random" argument of previous works. This no longer follows from a simple condition on the size of Y; we will require a well-structured set of walks generated from the output of a pseudorandom generator, which we later recall.

Theorem 4.5 (Deciding Small Configuration Graphs)**.** *There is a catalytic subroutine* Small *such that, under the promise that* $|\mathcal{Y}(\mathbf{w}, Y)| < 2S$*, we have*

$$\mathsf{Small}^{\mathbf{w} \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}}(x) = L(x)$$

(where Y *is to be defined later in terms of* $\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}$*, such that* Y *can be output in space* $O(s)$ *given read-only access to* $\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}$*) and moreover* Small *uses additional workspace* $O(s)$*.*

We prove Theorem 4.4 and Theorem 4.5 in the subsequent subsections. First, however, we use them to prove Theorem 4.1. At a high level, our goal is to create a $\mathbf{CSPACE}[s]$ machine M' that decides L. At each step we attempt find a configuration graph of sufficiently large size, in which case we use the algorithm $\mathsf{RandComp}$ of Theorem 4.4 to compress an additional bit. If this continues for enough rounds, then we use the deterministic algorithm for L in the free space; if not, then we must eventually have only configuration graphs of small size, in which case we use the algorithm Small of Theorem 4.5. Whichever case occurs, at the end we reset all bits compressed using the algorithm $\mathsf{RandDecomp}$ of Theorem 4.4 and return the answer $L(x)$.

[9]Statements like $(\mathbf{w}', v) \in \mathcal{Y}$ (as used in the compression loop) exclusively refer to the vertex set of $\mathcal{Y}$ excluding the dummy state $\perp$.

Initialization. Let $k \leftarrow 0$ be our counter for the number of bits compressed thus far. We start in the *compression loop.*

The Compression Loop. We interpret the catalytic tape as

$$\mathbf{w}_k = (\mathbf{w} \circ i \circ z \circ 0^k \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S})$$

where $\mathbf{w}$ has length 2^s, i has length $s+1$, and z has length $s_L - k$. Recall that $\mathcal{Y}(\mathbf{w}, Y)$ is the set of configurations reached using every possible PRG output as a random string, and Y is computable in logspace given read-only access to $\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}$. We call the catalytic subroutine $\mathsf{Size}^{\mathbf{w}}(x, Y)$ in Theorem 4.4 to computes $|\mathcal{Y}|$ then break into two cases:

- **The Small Graph Case.** If $|\mathcal{Y}| < 2S$, we call the subroutine $\mathsf{Small}^{\mathbf{w} \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}}(x)$ of Theorem 4.5, which returns $L(x)$. We then move to the *decompression loop.*
- **The Large Graph Case.** If $|\mathcal{Y}| \geq 2S$, we call the algorithm $\mathsf{RandComp}^{\mathbf{w}}(x, i, Y)$ of Theorem 4.4, which returns v and leaves the first section of $\mathbf{w}_k$ in configuration $\mathbf{w}'$. We then overwrite $\mathbf{w}_k$ with
$$\mathbf{w}_{k+1} = (\mathbf{w}' \circ v \circ z \circ 0^{k+1} \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S})$$
which can be done by replacing the first s bits of i with v and erasing the last bit, then left shifting z (such that $(v \circ z_1)$ will be parsed as the time index in the next stage of the loop). We then increment k and return to the start of the compression loop.

If the counter k reaches s_L, we use the $\mathbf{SPACE}[s_L]$ algorithm for L on the work tape $0^k = 0^{s_L}$. After this machine returns $L(x)$, we store $L(x)$, set this section of the tape to 0^n, and move to the *decompression loop* which will reset the catalytic tape.

The Decompression Loop. At the start of every iteration of the decompression loop, the current configuration of the first section of the catalytic tape is

$$\mathbf{w}_k = (\mathbf{w}' \circ v \circ z \circ 0^k \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S})$$

for $k \leq s_L$. If $k = 0$ then we end the algorithm and return the stored value of $L(x)$; otherwise we call the algorithm $\mathsf{RandDecomp}$ of Theorem 4.4 with $\mathsf{RandDecomp}^{\mathbf{w}'}(x, v, Y)$, which returns i and leaves the first section of $\mathbf{w}_k$ in configuration $\mathbf{w}$. We then overwrite $\mathbf{w}_k$ with

$$\mathbf{w}_{k-1} = (\mathbf{w}' \circ i \circ z \circ 0^{k-1} \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S})$$

which can be done by right shifting z, then replacing $v \circ 0$ with i. We then decrement k and return to the start of the decompression loop.

Correctness. It is clear that we correctly decide the language, and every string $\mathbf{w}_k$ in the compression case is identical to the same string $\mathbf{w}_k$ from the decompression case, and so we successfully reset the catalytic tape.

We also analyze the space usage of our algorithm. All subroutines in both the compression and decompression loop can be executed in space $O(s)$, and at the end of each iteration we can erase everything on the free work tape besides i and k, and thus our algorithm runs in free workspace $O(s)$. Lastly our catalytic tape needs to store $\mathbf{w}$, i, 0^k, and $\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}$; since all of these strings have length at most $2^{O(s)}$, our algorithm is a $\mathbf{CSPACE}[s]$ algorithm as required. $\square$

4.1 The Large Graph Case

We first handle Theorem 4.4, showing that a similar timestamp compression to Theorem 3.1 can be achieved.

Proof of Theorem 4.4. Let $\ell := |Y|$. We order the sub-strings of Y as follows. First, order Y in any efficiently comparable order (e.g., in lexicographic order) as $(y_1, \ldots, y_\ell) = Y$. Let $y_i^j = (y_i)_{\leq j}$ be the j-bit prefix of y_i. Let $z_1, z_2, \ldots, z_{m\ell}$ be the strings in this order:

$$y_1^1, \ldots, y_1^m, y_2^1, \ldots, y_2^m, \ldots, y_\ell^1, \ldots, y_\ell^m.$$

Let $(\mathbf{w}_k, v_k) := \mathcal{G}\left[\mathsf{start}(\mathbf{w}), z_k\right]$ be the configuration reached by the walk z_k. Clearly $\mathcal{Y} = \{(\mathbf{w}_k, v_k)\}_{k\in[m\ell]}$. Note that there might be $k_1 \neq k_2$ such that $(\mathbf{w}_{k_1}, v_{k_1}) = (\mathbf{w}_{k_2}, v_{k_2})$.

We define the mapping $f : \mathcal{Y} \to [|\mathcal{Y}|]$ as follows: For each $(\mathbf{w}_k, v_k) \in \mathcal{Y}$, let k' be the smallest number such that $(\mathbf{w}_{k'}, v_{k'}) = (\mathbf{w}_k, v_k)$, and define $f((\mathbf{w}_k, v_k)) := |\{(\mathbf{w}_j, v_j) \mid j \in [k']\}|$. Note that $\{(\mathbf{w}_j, v_j) \mid j \in [k']\}$ is the set of preceding states (in particular, it is not a multi-set). That is, if we remove all but the first appearance of σ in the list $(\mathbf{w}_1, v_1), \ldots, (\mathbf{w}_{m\ell}, v_{m\ell})$, $f(\sigma)$ is defined to be its rank in the list.

Let

$$\mathsf{First}(z_k) := \bigwedge_{k'<k} \mathbb{I}\left[(\mathbf{w}_{k'}, v_{k'}) \neq (\mathbf{w}_k, v_k)\right]$$

be the predicate that z_k is the first string in the ordering to reach the state $(\mathbf{w}_k, v_k)$. Note that we can determine this using the catalytic algorithm EQ in Lemma 2.11 without changing the contents of the catalytic tape.

Size counter and compression. Now we describe the subroutines $\mathsf{Size}^{\mathbf{w}}(x, Y)$ and $\mathsf{RandComp}^{\mathbf{w}}(x, i, Y)$, both follow from an iterative procedure.

Let $\mathsf{start}(\mathbf{w})$ be an initial configuration for M, we initialize a counter $i' \leftarrow 0$, which counts how many elements of $\mathcal{Y}$ it has seen so far, as well as another counter $k \leftarrow 1$. The iterative procedure maintains the invariant that at the beginning of each iteration, the size of the set $\{(\mathbf{w}_1, v_1), \ldots, (\mathbf{w}_{k-1}, v_{k-1})\}$ is i', until $k > m\ell$.

- If $\mathsf{First}(z_k)$ does not hold, we increment k and continue.
- If $\mathsf{First}(z_k)$ holds and $i' = i$, by the definition of f we know that $f(\mathbf{w}_k, v_k) = i$. The compression subroutine $\mathsf{RandComp}^{\mathbf{w}}(x, i, Y)$ sets its catalytic tape to $\mathbf{w}_k$ and computes v_k by calling the subroutine $\mathsf{RandWalk}^{\mathbf{w}}(x, 0, z_k)$ in Theorem 2.10, halts, and returns v_k.
- Otherwise, we increment i' and continue.

At the end, we know that $i' = S$. Therefore, the size counting subroutine $\mathsf{Size}^{\mathbf{w}}(x, 0^s, z_k)$ simply outputs i' and halts. The correctness follows directly from the invariant, and the additional workspace used by both algorithms is bounded by $O(s+\log|Y|)$ to call $\mathsf{RandWalk}^{\mathbf{w}}$ in Theorem 2.10 and store the counters i' and k. See Algorithms 7 and 8 for the pseudocode of Size and $\mathsf{RandComp}$ respectively.

Decompression. We now describe the decompression subroutine $\mathsf{RandDecomp}^{w'}(x, v', Y)$. Note that $(\mathbf{w}', v') \in \mathcal{G}_x$ is a configuration of M. By the compression algorithm, there is a $k \in [m\ell]$ such that $(\mathbf{w}', v') = (\mathbf{w}_k, v_k)$ and $\mathsf{First}(z_k)$ is true. We know by the definition of f that the decompression

subroutine should return the size of the set $\{(\mathbf{w}_1, v_1), \ldots, (\mathbf{w}_k, v_k)\}$ (again note that we consider this as a set, not a multiset) and resets the catalytic tape to $\mathbf{w}$.

We enumerate over k and run the subroutine $\mathsf{RandRev}^{\mathbf{w}'}(x, v', r)$ in Theorem 2.10 with $r := z_k$. If the algorithm accepts, we know that $(\mathbf{w}', v') = (\mathbf{w}_k, v_k)$ and $\mathsf{First}(z_k)$ is true; we store this value of k on the work tape. Otherwise, we increment k and continue, and in which case, the catalytic tape will be reset to $\mathbf{w}'$ by Theorem 2.10.

Once we have such a k, we know that the catalytic tape is set to $\mathbf{w}$ by Theorem 2.10. Therefore, the subroutine $\mathsf{RandDecomp}$ simply performs the iterative procedure used by $\mathsf{RandComp}$ and Size (which does not change the contents of the catalytic tape) to determine the size of the set $\{(\mathbf{w}_1, v_1), \ldots, (\mathbf{w}_k, v_k)\}$, returns the size, and halts. The correctness and space complexity analysis is straightforward. See Algorithm 9 for the pseudocode of $\mathsf{RandDecomp}$. □

4.2 The Small Graph Case

We now handle the case of small graphs, i.e. Theorem 4.5, which will complete our proof. To do so, we actually specify how we will produce Y: We recall a PRG of [DPT24] with deterministic approximate reconstruction. We first formally define a previous bit predictor:

Definition 4.6. A function $P : \{0,1\}^m \to \{0,1\}$ is an ε-previous bit predictor for a distribution $\mathbf{D}$ over $\{0,1\}^n$ (for $n > m$) if

$$\Pr_{x \leftarrow \mathbf{D}}[P(x_{>n-m}) = x_{n-m}] \geq \frac{1}{2} + \varepsilon.$$

We then recall the PRG:

Theorem 4.7. *There are universal constants $c_{\mathsf{NW}} > 1$ and $c > 0$ such that the following holds. There is an algorithm NW computing*

$$\mathsf{NW}^f : \{0,1\}^{O(\log N)} \to \{0,1\}^N$$

such that for any $f \in \{0,1\}^{N^{c_{\mathsf{NW}}}}$, we have the following:

1. ***Efficiency.*** *When given $\tau \in \{0,1\}^{O(\log N)}$ and oracle access to f, the generator runs in space $O(\log N)$ and outputs an N-bit string $\mathsf{NW}^f(\tau)$.*
2. ***Deterministic Reconstruction.*** *There are deterministic space $O(\log N)$ algorithms $\mathsf{Hint}, \mathsf{Decode}$ that act as follows.*
   - Hint*, given oracle access to f and oracle access to a $(1/N^2)$-previous bit predictor P for NW^f, returns $h \in \{0,1\}^{O(\log N)}$ and (the endpoints of) an interval $K \subseteq [N^{c_{\mathsf{NW}}}]$ of length N^c.*
   - Decode*, given oracle access $\widetilde{f}$ such that $\widetilde{f}_{\bar{K}} = f_{\bar{K}}$ and oracle access to P, satisfies for every $j \in K$:*

$$\mathsf{Decode}^{\widetilde{f},P}(h, j) = f_j.$$

The result is not written in this fashion in [DPT24], but can be adapted to have this form. We provide a formal proof from their result in Appendix A.

Proof of Theorem 4.5. We interpret our catalytic tape as

$$(\mathbf{w} \circ \mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{2S}).$$

where $\mathbf{w}$ has length 2^s and each $\mathbf{m}_i$ has length $2^{O(s)}$ (to be set shortly). Next, for every $i \in [2S]$ let

$$G_i := \mathsf{NW}^{\mathbf{m}_i} : \{0,1\}^{O(s)} \to \{0,1\}^{2S}$$

be the generator of Theorem 4.7 with $N := \max\left\{2S, s_L^{1/c}\right\}$ (recall that $s_L \leq 2^{O(s)}$ is the deterministic space complexity of L), and observe that we can set $|\mathbf{m}_i| = N^{c_{\mathsf{NW}}} = 2^{O(s)}$. Finally let

$$Y_i := G_i(\mathbf{U}), \qquad Y_{<i} := \bigcup_{j<i} Y_j$$

Note that each $Y_{<i}$ can be produced in logspace given read-only access to $\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{i-1}$.

For a catalytic tape $\mathbf{w}$ of length 2^s, recall that $\mathcal{Y} := \mathcal{Y}(\mathbf{w}, Y)$ is configuration subgraph reachable using strings in Y as randomness. Let $\mathcal{Y}_{<i} := \mathcal{Y}(\mathbf{w}, Y_{<i})$ be the subgraph reached by the output of the first $i-1$ PRGs, and $\mathcal{Y}_i := \mathcal{Y}(\mathbf{w}, Y_i)$ be the subgraph reached by the output of G_i.

Our goal is to either compress some string $\mathbf{m}_i$ using Theorem 4.7 or to use the walks Y to estimate the acceptance probability of our **CBPL** machine for L.

As we need to retain access to each candidate predictor after compressing the PRG, we wish to identify a PRG G_i whose output lies entirely inside the explored graph of the previous $i-1$ PRGs. Because there are $2S$ sets $\mathcal{Y}_i$ and

$$\left|\bigcup_{i\in[2S]} \mathcal{Y}_i\right| = |\mathcal{Y}(\mathbf{w}, Y)| < 2S,$$

there must exist some i where $\mathcal{Y}_i \subseteq \mathcal{Y}_{<i}$, and moreover we can determine this index efficiently:

Claim 4.8. *There is a catalytic machine* Pigeon *using $s(n)$ workspace such that* $\mathsf{Pigeon}^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{2S}}(x)$ *returns i such that $\mathcal{Y}_i \subseteq \mathcal{Y}_{<i}$.*

Proof. We instantiate a counter $i = 1$. For each value of this counter, we test if for every $y \in Y_i$ and prefix $y_<$ of y if there exists a $z \in Y_{<i}$ with some prefix $z_<$ such that

$$\mathcal{G}_x[\mathsf{start}(\mathbf{w}), y_<] = \mathcal{G}_x[\mathsf{start}(\mathbf{w}), z_<]$$

We can test this via the catalytic subroutine EQ^w of Lemma 2.11 for any fixed $y_<, z_<$, both of which can be computed in logspace given access to $(\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_i)$ by Theorem 4.7.

Overall Pigeon will act as follows. We loop over all y and prefix $y_<$ of y, for which we will attempt to test this condition. Given $y_<$, Pigeon tests every $z_<$ until we find $\mathcal{G}[\mathsf{start}(\mathbf{w}), y_<]$, in which case we move to the next prefix $y_<$; if we exhaust all prefixes $z_<$ without finding our current state $\mathcal{G}[\mathsf{start}(\mathbf{w}), y_<]$, we increment i, reset our catalytic tape, and start over. If we find a prefix $z_<$ for each prefix $y_<$ such that this property holds, then we reset our catalytic tape and return the current value of i. See Algorithm 10 for the pseudocode of Pigeon. □

For the remainder of the proof, let i be the value returned by Claim 4.8, and define

$$\mathcal{U} := \mathcal{Y}_{<i}.$$

Note that $\mathcal{U} \supseteq \mathcal{Y}_i$ by the guarantee of Claim 4.8. We now attempt to compress the i-th PRG, and if we fail to do so we derandomize. We instantiate a family of candidate previous-bit predictors, derived from two distinguishers. Let

$$L_1(r) := \mathbb{I}\left[\mathcal{U}[\mathsf{start}(\mathbf{w}), r] = \perp\right]$$

be the test function that determines if a walk leaves $\mathcal{U}$. Recall that in $\mathcal{U}$, all edges that would leave to states outside $\mathcal{U}$ instead route to a dummy state $\perp$. Moreover, let

$$L_2(r) := \mathbb{I}\left[\mathcal{U}[\mathsf{start}(\mathbf{w}), r] = \mathsf{acc}(\mathbf{w})\right]$$

be the test function that determines if a walk stays entirely inside $\mathcal{U}$ and reaches the accepting configuration. Note that as $\mathcal{U}$ is acyclic (except possibly for the states $\mathsf{halt}(\mathbf{w}, b)$) and by assumption has size at most $2S$, we may assume that L_1 and L_2 take exactly $2S$ bits of input, and hence their composition with G_i is well defined.

Both functions can be computed catalytically without access to $\mathbf{m}_i$, as we do not need to retain access to the i-th PRG. This is the reason we identify a PRG with output that does not explore new states.

Claim 4.9. *There are catalytic machines A_1, A_2 such that*

$$A_1^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r) = L_1(r), \quad A_2^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r) = L_2(r)$$

that use $O(s + \log|r|)$ workspace.

Proof. We first determine if the walk specified by r ever leaves the graph $\mathcal{U}$ (i.e. $L_1(r)$). Note that this is equivalent to the (negation of the) predicate that for every prefix $r_<$ of r, there exists a prefix $y_<$ of y in

$$\bigcup_{k\in[i-1]} G_k(\mathbf{U})$$

such that

$$\mathcal{G}[\mathsf{start}(\mathbf{w}), r_<] = \mathcal{G}[\mathsf{start}(\mathbf{w}), y_<],$$

hence as $\mathcal{G}[\mathsf{start}(\mathbf{w}), y_<] \in \mathcal{U}$ by definition, $r_<$ ends inside $\mathcal{U}$. Thus, to determine L_1 we enumerate over prefixes $r_<$ in logspace, and then enumerate over $y_<$ using that the generators are computable in logspace given $(\mathbf{m}_1 \circ \ldots \circ \mathbf{m}_{i-1})$. We then apply the algorithm of Lemma 2.11 with $r = r_<$, $r' = y_<$, which computes the desired information catalytically.

Finally, note that for every r where $L_1(r) = 0$, we have that $L_2(r) = \mathbb{I}\left[\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = \mathsf{acc}(\mathbf{w})\right]$, and so A_2 runs A_1 and accepts iff we never leave $\mathcal{U}$ and reach final state $\mathsf{acc}(\mathbf{w})$. See Algorithms 11 and 12 for the pseudocode of A_1 and A_2 respectively. □

Using these tools, we now move on to our overall algorithm Small for Theorem 4.5. First, we use Pigeon from Claim 4.8 to identify an i for which we will attempt to compress the i-th PRG. This will be stored on our work tape for the remainder of the algorithm, using $\log B = O(s)$ bits.

In order to apply Theorem 4.7, we build a family of candidate previous-bit predictors for G_i. Each will be defined by a prefix generated by a prior PRG $G_k \in \{G_1, \ldots, G_{i-1}\}$ plus a seed τ for G_k, which we then run up to a cutoff point l_1; after this prefix, we use the suffix of r starting at l_2. Formally, let

$$\mathcal{P} = \left\{P_{a,v=(b,k,j,l_1,l_2)}(r) : a \in \{1,2\}, b \in \{0,1\}, k \in [i-1], \tau \in \{0,1\}^{O(s)}, \ell_1 \in [2S], \ell_2 \in [2S]\right\}$$

be the family of potential predictors where

$$P_{a,v=(b,k,\tau,l_1,l_2)}(r) = L_a((G_k(\tau))_{\le l_1} \circ r_{>l_2}) \oplus b$$

and note that v can be described with $O(s)$ bits, and moreover each candidate predictor can be evaluated catalytically using Claim 4.9:

Claim 4.10. *There is a catalytic machine* Pred *satisfying*

$$\mathsf{Pred}^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(a,v,r) = P_{a,v}(r)$$

that uses $O(s+\log|r|)$ *workspace.*

Using Claim 4.10, Small enumerates over $a\in\{1,2\}$ and $v\in\{0,1\}^{O(s)}$, and computes

$$\alpha_{a,v} = \Pr_{x\leftarrow G_i(\mathbf{U})}[P_{a,v}(x_{>j}) = x_j] - 1/2.$$

If $\alpha_{a,v} < 1/N^2$ for every (a,v), we move to the *compute case*, and otherwise we move to the *walk compress case* for any (a,v) satisfying $\alpha_{a,v}\geq 1/N^2$. We describe each subroutine in turn now.

The Compute Case. Given that $\alpha_{a,v} < 1/N^2$ for every (a,v), we compute

$$\rho = \mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}), G_i(\mathbf{U})]].$$

If $\rho > 1/2$ we determine $L(x)=1$, and otherwise we determine $L(x)=0$. Recall that $\mathcal{G}$ denotes the configuration graph on input x, which satisfies

$$\mathbb{E}[\mathcal{G}[(w,0^s),\mathbf{U}] = \mathsf{acc}(\mathbf{w})] \in \begin{cases}[0,1/3), & L(x)=0\\ (2/3,1], & L(x)=1\end{cases}$$

and thus the correctness of our algorithm is given by the following lemma:

Lemma 4.11. *Suppose* $\alpha_{a,v} < 1/N^2$ *for every* a,v. *Then*

$$|\mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}), G_i(\mathbf{U})] = \mathsf{acc}(\mathbf{w})] - \mathbb{E}[\mathcal{G}[\mathsf{start}(\mathbf{w}),\mathbf{U}] = \mathsf{acc}(\mathbf{w})]| \leq \frac{2}{N}$$

and hence we correctly decide the language.

Proof. We have

$$\begin{aligned}
&|\mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}), G_i(\mathbf{U})] = \mathsf{acc}(\mathbf{w})] - \mathbb{E}[\mathcal{G}[\mathsf{start}(\mathbf{w}),\mathbf{U}] = \mathsf{acc}(\mathbf{w})]|\\
&\quad\leq |\mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}), G_i(\mathbf{U})] = \mathsf{acc}(\mathbf{w})] - \mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}),\mathbf{U}] = \mathsf{acc}(\mathbf{w})]| +\\
&\qquad |\mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}),\mathbf{U}] = \mathsf{acc}(\mathbf{w})] - \mathbb{E}[\mathcal{G}[\mathsf{start}(\mathbf{w}),\mathbf{U}] = \mathsf{acc}(\mathbf{w})]|\\
&\quad\leq |\mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}), G_i(\mathbf{U})] = \mathsf{acc}(\mathbf{w})] - \mathbb{E}[\mathcal{U}[\mathsf{start}(\mathbf{w}),\mathbf{U}] = \mathsf{acc}(\mathbf{w})]| + L_1(\mathbf{U})\\
&\quad= |\,\mathbb{E}[L_2(G_i(\mathbf{U}))] - \mathbb{E}[L_2(\mathbf{U})]| + |\,\mathbb{E}[L_1(G_i(\mathbf{U}))] - \mathbb{E}[L_1(\mathbf{U})]|.
\end{aligned}$$

where the second inequality follows as any walk on which $\mathcal{U}$ and $\mathcal{G}$ reach different locations must leave $\mathcal{U}$, and the equality follows as G_i does not hit L_1 and hence $\mathbb{E}[L_1(G_i(\mathbf{U}))]=0$.

For $a\in\{1,2\}$, define $\rho_a := |\,\mathbb{E}[L_a(G_i(\mathbf{U}))] - \mathbb{E}[L_a(\mathbf{U})]|$; thus it is enough to show that $\rho_a < 1/N$ to complete our proof. We will require a statement of Yao's Lemma [Yao82]:

Lemma 4.12. *For every function* $f:\{0,1\}^N\to\{0,1\}$ *and distribution* $\mathbf{D}$ *supported over* $\{0,1\}^N$, *define* $\rho := |\,\mathbb{E}[f(\mathbf{U}_N)] - \mathbb{E}[f(\mathbf{D})]|$. *Then there exists* $j\in[N]$, $z\in\{0,1\}^j$, *and* $b\in\{0,1\}$ *such that*

$$\Pr_{x\leftarrow\mathbf{D}}[f(z\circ x_{>j})\oplus b = x_j] \geq \frac{1}{2} + \frac{\rho}{N}$$

Applying Lemma 4.12, if $\rho_a \geq 1/N$ then there exists $z \in \{0,1\}^j$, $j \in [2S]$, $b \in \{0,1\}$ such that

$$\Pr_{x \leftarrow G_i(\mathbf{U})}[L_a(z \circ x_{>j}) \oplus b = x_j] \geq \frac{1}{2} + \frac{1}{N^2}$$

Recall that $\alpha_{a,v} < 1/N^2$ for all a, v, which means that for a, j, and b as above and any k, τ, and ℓ, we have

$$\Pr_{x \leftarrow G_i(\mathbf{U})}[L_a((G_k(\tau))_{\leq \ell} \circ x_{>j}) \oplus b = x_j] < \frac{1}{2} + \frac{1}{N^2}.$$

Thus it suffices to show that for every $z \in \{0,1\}^j$, we have that there are k, τ, and ℓ such that

$$L_a(z \circ x_{>j}) \equiv L_a((G_k(\tau))_{\leq \ell} \circ x_{>j})$$

where $\equiv$ denotes equality as functions of the input $x_{>j}$.

Following [DPT24], observe that walking according to z reaches a state $v_z \in \mathcal{U}$ (and $v_z \neq \perp$, as otherwise the predictor would output the same value on every input). But then there is some k and seed τ and timestep ℓ such that $\mathcal{U}[\mathsf{start}(\mathbf{w}), (G_k(\tau))_\ell] = v_z$, so we are done. □

The Walk Compress Case. We now assume that we have found some (a, v) such that $P_{a,v}$ predicts G_i with advantage at least $1/B$. By Claim 4.10, we have access to an algorithm Pred for computing $P_{a,v}$ which never accesses $\mathbf{m}_i$. Recall that we also have access to a deterministic $\mathbf{SPACE}[s_L]$ algorithm for computing L.

We run the algorithm Hint of Theorem 4.7 with $f = \mathbf{m}_i$ and $P = P_{a,v}$, and obtain $h \in \{0,1\}^{O(s)}$ and the indices of a subinterval $K \subseteq [N^{c_{\mathrm{NW}}}]$ of size $N^c \geq s_L$ (using that $N \geq s_L^{1/c}$), both of which we store on the work tape. We then set $(\mathbf{m}_i)_K = 0^{s_L}$, and run the $\mathbf{SPACE}[s_L]$ algorithm of M using this workspace. Once this algorithm halts, we store the answer on the work tape and move to restore $\mathbf{m}_i$. Letting $\mathbf{m}_i'$ be the modified section of catalytic tape, and recalling that we have h stored on our work tape, we run the algorithm of Decode of Theorem 4.7 with

$$h = h, \quad \widetilde{f} = \mathbf{m}_i', \quad P = P_{a,v}$$

and for every index $j \in K$, we use Decode to compute $(\mathbf{m}_i)_j$ and write it to the catalytic tape. Since no other sections of the catalytic tape were altered, this completes the restoration, and note that our work tape stores at most

$$|h| + \log N + 1 = O(s)$$

bits of information on the free work tape. This completes the description of Small, the proof of Theorem 4.5, and with it the proof of Theorem 1.1. □

4.3 Implications

An easy corollary of Theorem 4.1 is the following converse to Theorem 3.4, which finishes the proof of Theorem 1.8.

Theorem 4.13. *For all space constructible functions $s := s(n) \geq \log n$,*

$$\mathbf{CZPTISP}\left[2^{O(s)}, s\right] \subseteq \mathbf{CSPACE}[s]$$

In particular, $\mathbf{CZPLP} \subseteq \mathbf{CL}$.

Proof. By Theorem 4.1 and basic closure properties we have

$$\begin{aligned}\mathsf{CZPTISP}\left[2^{O(s)}, s\right] &\subseteq \mathsf{CZPSPACE}\left[s\right] \\ &\subseteq \mathsf{CBPSPACE}\left[s\right] \\ &\subseteq \mathsf{CSPACE}\left[s\right].\end{aligned}$$

□

The same equations, combined with Theorem 3.4, give us Corollary 1.9, which is our last result.

Theorem 4.14. *For all space constructible functions $s := s(n) \geq \log n$,*

$$\mathsf{CZPTISP}\left[2^{O(s)}, s\right] = \mathsf{CZPSPACE}\left[s\right]$$

In particular, $\mathsf{CZPL} = \mathsf{CZPLP}$.

Proof. The forward inclusion is immediate. For the reverse inclusion, by Theorem 4.1 and Theorem 3.4 we have

$$\begin{aligned}\mathsf{CZPSPACE}\left[s\right] &\subseteq \mathsf{CBPSPACE}\left[s\right] \\ &\subseteq \mathsf{CSPACE}\left[s\right] \\ &\subseteq \mathsf{CZPTISP}\left[2^{O(s)}, s\right].\end{aligned}$$

□

Acknowledgements

Jiatu Li and Edward Pyne thank Oliver Korten for bringing the connection between catalytic computation and LossyCode to our attention. James Cook and Ian Mertz thank Noah Fleming, Toniann Pitassi, and Morgan Shirley for early conversations regarding timestamp compression. We thank Roei Tell, Igor Oliveira, Ninad Rajgopal, Bruno Cavalar, and others for helpful conversations.

References

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Proc. 46 Annual ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014.

[BDS22] Sagar Bisoyi, Krishnamoorthy Dinesh, and Jayalal Sarma. On pure space vs catalytic space. *Theor. Comput. Sci.*, 921:112–126, 2022.

[BKLS18] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space: Non-determinism and hierarchy. *Theory Comput. Syst.*, 2018.

[CM20] 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.

[CM21] James Cook and Ian Mertz. Encodings and the tree evaluation problem. *Electron. Colloquium Comput. Complex.*, TR21-054, 2021.

[CM22] James Cook and Ian Mertz. Trading time and space in catalytic branching programs. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[CM23] James Cook and Ian Mertz. Tree evaluation is in space o(log n · log log n). *Electron. Colloquium Comput. Complex.*, TR23-174, 2023.

[CMW+12] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2):4:1–4:43, 2012.

[CTW23] Lijie Chen, Roei Tell, and Ryan Williams. Derandomization vs refutation: A unified framework for characterizing derandomization. In *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2023. To appear.

[DGJ+20] 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*, 2020.

[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 *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, 2024.

[Dul15] Yfke Dulek. Catalytic space: on reversibility and multiple-access randomness, 2015.

[EMP18] Jeff Edmonds, Venkatesh Medabalimi, and Toniann Pitassi. Hardness of function composition for semantic read once branching programs. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 15:1–15:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[GJST19] 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.

[Hoz21] William M. Hoza. Better pseudodistributions and derandomization for space-bounded computation. In *Proceedings of the 25th International Conference on Randomization and Computation (RANDOM)*, pages 28:1–28:23, 2021.

[ILW23] Rahul Ilango, Jiatu Li, and R. Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1076–1089, [2023] ©2023.

[IN19] Kazuo Iwama and Atsuki Nagao. Read-once branching programs for tree evaluation problems. *ACM Trans. Comput. Theory*, 11(1):5:1–5:12, 2019.

[Kor21] Oliver Korten. The hardest explicit construction. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 433–444. IEEE, 2021.

[Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *Proc. 37th Annual IEEE Conference on Computational Complexity (CCC)*, 2022.

[Kou16] Michal Koucký. Catalytic computation. *Bull. EATCS*, 118, 2016.

[Liu13] David Liu. Pebbling arguments for tree evaluation. *CoRR*, abs/1311.0293, 2013.

[LPT24] Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness, 2024.

[Mer23] Ian Mertz. Reusing space: Techniques and open problems. *Bulletin of EATCS*, 141(3), 2023.

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

[PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, 2023.

[Pyn23] Edward Pyne. Derandomizing logspace with a small shared hard drive. *Electron. Colloquium Comput. Complex.*, TR23-168, 2023.

[SZ99] Michael E. Saks and Shiyu Zhou. $\mathbf{BP_hSPACE}[s] \subseteq \mathbf{DSPACE}[s^{3/2}]$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Proc. 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

A An Instantiation of the Doron–Pyne–Tell PRG

We recall the precise statement in that work.

Theorem A.1 (Theorem 8.1 [DPT24])**.** *There is a universal constant $c_{\mathsf{NW}} > 1$ such that the following holds. There is an algorithm NW computing*

$$\mathsf{NW}^f : \{0,1\}^{O(\log N)} \to \{0,1\}^N$$

such that for any $f \in \{0,1\}^{N^{c_{\mathsf{NW}}}}$, we have the following:

1. ***Efficiency.*** *When given $s \in \{0,1\}^{O(\log N)}$ and oracle access to f, the generator runs in space $O(\log N)$ and outputs an N-bit string $\mathsf{NW}^f(s)$.*
2. ***Deterministic Reconstruction.*** *There are deterministic space $O(\log N)$ algorithms R, T, F that act as follows.*
   - *R, given oracle access to f and oracle access to a $(1/N^2)$ previous bit predictor P for NW^f, outputs $h \in \{0,1\}^{O(\log n)}$. Moreover, there is a subset $K \subseteq \{0,1\}^{N^{c_{\mathsf{NW}}}}$ of size $N^{c_{\mathsf{NW}}/100}$ that satisfies the following.*

- *T, given $h \in \{0,1\}^{O(\log n)}$ and $i \in [N^{c_{\mathsf{NW}}}]$, determines if $i \in K$.*
- *F, given $h \in \{0,1\}^{O(\log n)}$ and oracle access to $\widetilde{f}$ such that $\widetilde{f}_K = f_K$ and oracle access to P, satisfies*

$$\Pr_{j \in [N^{c_{\mathsf{NW}}}]}[F^{h,\widetilde{f},P}(j) = f_j] \geq 1 - N^{-c'}$$

for a constant $c' > 0$. Moreover, F only queries $\widetilde{f}$ at locations in K.

We then transform the algorithm as follows:

Theorem 4.7. *There are universal constants $c_{\mathsf{NW}} > 1$ and $c > 0$ such that the following holds. There is an algorithm NW computing*

$$\mathsf{NW}^f : \{0,1\}^{O(\log N)} \to \{0,1\}^N$$

such that for any $f \in \{0,1\}^{N^{c_{\mathsf{NW}}}}$, we have the following:

1. ***Efficiency.*** *When given $\tau \in \{0,1\}^{O(\log N)}$ and oracle access to f, the generator runs in space $O(\log N)$ and outputs an N-bit string $\mathsf{NW}^f(\tau)$.*
2. ***Deterministic Reconstruction.*** *There are deterministic space $O(\log N)$ algorithms $\mathsf{Hint}, \mathsf{Decode}$ that act as follows.*
   - Hint*, given oracle access to f and oracle access to a $(1/N^2)$-previous bit predictor P for NW^f, returns $h \in \{0,1\}^{O(\log N)}$ and (the endpoints of) an interval $K \subseteq [N^{c_{\mathsf{NW}}}]$ of length N^c.*
   - Decode*, given oracle access $\widetilde{f}$ such that $\widetilde{f}_{\bar{K}} = f_{\bar{K}}$ and oracle access to P, satisfies for every $j \in K$:*

$$\mathsf{Decode}^{\widetilde{f},P}(h, j) = f_j.$$

Proof. We pass through all parameters to Theorem A.1. Then the new algorithms $\mathsf{Hint}, \mathsf{Decode}$ work as follows. For Hint, we run the algorithm R with f and oracle access to P and obtain $h \in \{0,1\}^{O(\log n)}$, which we return. Next, we attempt to idenfity a subinterval K' of length N^c for c to be chosen later such that for every $j \in K'$:

- $j \notin K$ (which we can test in logspace using the algorithm T of Theorem A.1).
- The algorithm F decodes correctly at that location. Note that F only queries $\widetilde{f}$ at locations in K, so to determine this we can run the algorithm with $\widetilde{f} = f$ and determine if the returned value matches the actual value.

Once we have found such an interval, we return the start and end indices. Then our algorithm Decode on input j returns $F^{h,\widetilde{f}}(j)$, and by our construction of K it is clear that we have the desired behavior.

We claim that such a subinterval exists with polynomial length. Note that K is of size $N^{c_{\mathsf{NW}}/100}$ and the set of indices that are decoded incorrectly is of size $N^{c_{\mathsf{NW}}-c'}$ for a constant c'. Thus the set of excluded indices is at most $N^{c_{\mathsf{NW}}-c'} + N^{c_{\mathsf{NW}}/100} \leq N^{c_{\mathsf{NW}}-c'+1}$, so there is a subinterval of length at least N^c for a constant $c > 0$. □

B Manipulating Deterministic Configuration Graphs

We prove Theorem 2.7 in two stages. We first define the bijection and show how to take single steps over it, then show how to use these subroutines to obtain the final result.

Lemma B.1. *For every space s catalytic machine M computing a language L, there exist two catalytic subroutines $M_{\rightarrow}, M_{\leftarrow}$ that run in worst-case time* $\mathrm{poly}(n)$ *and work as follows. Let $\mathcal{G}_x$ be the configuration graph of M on input x. Then there exists a bijection*

$$\Pi : \mathcal{G}_x \times \{0,1\} \to \mathcal{G}_x \times \{0,1\}$$

so that for every $\mathbf{w}$, the sequence

$$(\mathsf{start}(\mathbf{w}), 0), \Pi[\mathsf{start}(\mathbf{w}), 0], \Pi^2[\mathsf{start}(\mathbf{w}), 0], \ldots$$

includes $(\mathsf{halt}(\mathbf{w}, b), 0) = ((\mathbf{w}, b1^{s-1}), 0)$*, where $b = L(x)$, and does not include* $(\mathsf{start}(\mathbf{w}'), 0)$ *for any $\mathbf{w}' \neq \mathbf{w}$ or* $(\mathsf{halt}(\mathbf{w}', b'), 0)$ *for any $\mathbf{w}' \neq \mathbf{w}$ or $b' \neq L(x)$. Moreover:*

- $M^{\mathbf{w}}_{\rightarrow}(x, v, a)$ *sets the catalytic tape to $\mathbf{w}'$ and returns v', a', where* $((\mathbf{w}', v'), a') = \Pi[(\mathbf{w}, v), a]$.
- $M^{\mathbf{w}}_{\leftarrow}(x, v, a)$ *sets the catalytic tape to $\mathbf{w}'$ and returns v', a', where* $((\mathbf{w}', v'), a') = \Pi^{-1}[(\mathbf{w}, v), a]$.

We can then use this lemma (with the same function Π) to prove the result:

Proof of Theorem 2.7. The bijection Π given by Lemma B.1 defines a path connecting $(\mathsf{start}(\mathbf{w}), 0)$ to $(\mathsf{halt}(\mathbf{w}, L(x)), 0)$ in the graph $\mathcal{G}_x \times \{0,1\}$. DetWalk explores the first k steps of that path by invoking $M_{\rightarrow}$ for up to k steps.

If DetWalk encounters a node of the form $(\mathsf{halt}(\mathbf{w}', b), 0)$, it halts and returns b (which we can be sure equals $L(x)$), leaving the catalytic tape set to $\mathbf{w}'$ (which we can be sure equals $\mathbf{w}$).

Otherwise, DetWalk returns the pair (v', a') returned by the k-th invocation of $M_{\rightarrow}$. At this point, $((\mathbf{w}', v'), a') = \Pi^k[\mathsf{start}(\mathbf{w}), 0]$ (where $\mathbf{w}'$ is the final state of the catalytic tape after the k-th invocation of $M_{\rightarrow}$).

Finally, DetRev follows the same path backward by invoking $M_{\leftarrow}$ instead of $M_{\rightarrow}$. It stops when it encounters a node of the form $(\mathsf{start}(\mathbf{w}'), 0)$ (we can be sure $\mathbf{w}' = \mathbf{w}$) and returns the number of steps taken. Since the path followed by Π contains no start nodes other than $(\mathsf{start}(\mathbf{w}), 0)$, we can be sure that DetRev stops at $\mathsf{start}(\mathbf{w})$ after exactly k steps. □

To prove Lemma B.1, we recall a folklore result:

Proposition B.2 (Folklore)**.** *There are deterministic catalytic subroutines* InDegree, InEdge, Rank *using $O(s)$ additional workspace that works as follows. Let $x \in \{0,1\}^n$ be an input of M and $(\mathbf{w}, v) \in \mathcal{G}_x$ be a configuration.*

- $\mathsf{InDegree}^{\mathbf{w}}(v)$ *returns the number of configurations $(\mathbf{w}', v')$ such that $\Gamma(\mathbf{w}', v') = (\mathbf{w}, v)$, with read-only access to the catalytic tape $\mathbf{w}$.*
- $\mathsf{InEdge}^{\mathbf{w}}(v, i)$ *updates the catalytic tape to $\mathbf{w}'$ and returns v', where $(\mathbf{w}', v')$ is the i-th node such such that $\Gamma(\mathbf{w}', v') = (\mathbf{w}, v)$ with respect to a consistent ordering.*
- $\mathsf{OutEdge}^{\mathbf{w}}_M(v)$ *updates the catalytic tape to $\mathbf{w}'$ and returns v', where $(\mathbf{w}', v) = \Gamma(\mathbf{w}, v)$.*
- $\mathsf{Rank}^{\mathbf{w}'}(v')$ *returns the rank of the node $(\mathbf{w}', v')$ in the list $\Gamma^{-1}(\mathbf{w}, v)$, where $(\mathbf{w}, v) := \Gamma(\mathbf{w}', v')$, with read-only access to the catalytic tape $\mathbf{w}'$.*

Proof of Lemma B.1. The machines work as follows. We use the machines of Proposition B.2 to traverse the configuration graph as an Eulerian tour. We maintain an auxiliary state $e \in \{0,1\}$ that tracks the current direction we are traversing in the up or down direction.

Functionality. The machine $M^{\mathbf{w}}_{\rightarrow}$ maintains state (v, e), where v is the workspace of a machine M that decides L, and $e \in \{\mathsf{DOWN} := 0, \mathsf{UP} := 1\}$ represents the next direction to traverse. Then the decision rule works as follows:

1. If $e = \mathsf{DOWN}$ and v is a halting state, halt and return $(v, e = \mathsf{UP})$.
2. If $e = \mathsf{DOWN}$ and v is not a halting state, let $(\mathbf{w}', v') = \Gamma(\mathbf{w}, v)$. First, let we determine the rank of $(\mathbf{w}, v)$ in the set $\Gamma^{-1}(\mathbf{w}', v')$, via the machine $\mathsf{Rank}^{\mathbf{w}}(v)$. We store the rank k on the work tape and execute $\mathsf{OutEdge}^{\mathbf{w}}(v)$, which updates the catalytic tape to $\mathbf{w}'$ and returns v', which we store. Now we call $\mathsf{InDegree}^{\mathbf{w}'}(v')$ to determine the in-degree d of the current configuration. If $d = k$, we halt with the catalytic tape in configuration $\mathbf{w}'$ and return $(v', e = \mathsf{DOWN})$. Otherwise, we call $\mathsf{InEdge}^{\mathbf{w}'}(v', k+1)$ which sets the catalytic tape to $\mathbf{w}''$ and returns v''. We halt in this configuration and return $(v'', e = \mathsf{UP})$.
3. If $e = \mathsf{UP}$, we call $\mathsf{InDegree}^{\mathbf{w}}(v)$ to determine the in-degree d of $(\mathbf{w}, v)$. If $d = 0$, we halt and return $(v, e = \mathsf{DOWN})$. Otherwise, we call $\mathsf{InEdge}^{\mathbf{w}}(v, 1)$, which updates the tape to $\mathbf{w}'$ and returns v', where $(\mathbf{w}', v')$ has rank 1 in the set of in-edges to $(\mathbf{w}, v)$. We then halt and return (v', UP).

The inverse $M^{\mathbf{w}}_{\leftarrow}(v)$ can be described as follows:

1. If $e = \mathsf{UP}$ and v is a halting state, halt and return $(v, e = \mathsf{DOWN})$. (This undoes case 1 of $M_{\rightarrow}$.)
2. If $e = \mathsf{UP}$ and v is not a halting state, let $(\mathbf{w}', v') = \Gamma(\mathbf{w}, v)$ and store the rank k of $(\mathbf{w}, v)$ in $\Gamma^{-1}(\mathbf{w}', v')$. Execute $\mathsf{OutEdge}^{\mathbf{w}}(v)$ to put $\mathbf{w}'$ on the tape and store v'. If $k = 1$, then return (v', UP); this undoes the second part of case 3. Otherwise ($k > 1$), execute $\mathsf{InEdge}^{\mathbf{w}}(v', k-1)$, updating the catalytic tape to $\mathbf{w}''$, and return the resulting state v''. This undoes the second part of case 2.
3. If $e = \mathsf{DOWN}$, then call $\mathsf{InDegree}^{w}(v)$ to compute the in-degree d. If $d = 0$ then return (v, UP); this undoes the first part of case 3. Otherwise, execute $\mathsf{InEdge}^{\mathbf{w}}(v, d)$, setting the catalytic tape to $\mathbf{w}'$ and returning v' such that $\Gamma(\mathbf{w}', v') = (\mathbf{w}, v)$, and return (v, DOWN). This undoes the first part of case 2.

Correctness. It can be seen by inspection that $M_{\leftarrow}$ and $M_{\rightarrow}$ are inverses of each other, so they define a bijection Π.

We begin by considering the configuration graph $\mathcal{G} = \mathcal{G}_x$ of the original catalytic machine M. Let $\mathcal{G}_{\text{un}}$ be the undirectified version, i.e. forgetting the edge directions. Fix an initial catalytic tape $\mathbf{w}$, and let $\mathcal{G}^{\mathbf{w}}_{\text{un}}$ be the connected component containing $\mathsf{start}(\mathbf{w})$.

Claim B.3. *$\mathcal{G}^{\mathbf{w}}_{\text{un}}$ is a tree, and it contains $\mathsf{start}(\mathbf{w})$ and $\mathsf{halt}(\mathbf{w}, L(x))$ and no other nodes $\mathsf{start}(\mathbf{w}')$ or $\mathsf{halt}(\mathbf{w}', b')$ for $\mathbf{w}' \neq \mathbf{w}$ or $b' \neq L(x)$.*

Proof. Every node in $\mathcal{G}$ has out-degree one except the sink nodes $\mathsf{halt}(\mathbf{w}', b')$. It follows that every connected component of $\mathcal{G}_{\text{un}}$ is either a tree containing exactly one node of the form $\mathsf{halt}(\mathbf{w}', b')$ or contains no such nodes. $\mathcal{G}$ has a path from $\mathsf{start}(\mathbf{w})$ to $\mathsf{halt}(\mathbf{w}, L(x))$, so $\mathcal{G}^{\mathbf{w}}_{\text{un}}$ is a tree which includes both those nodes and no other node of the form $\mathsf{halt}(\mathbf{w}', b')$.

It remains to show that $\mathcal{G}^{\mathbf{w}}_{\text{un}}$ contains no nodes $\mathsf{start}(\mathbf{w}')$ for $\mathbf{w}' \neq \mathbf{w}$. This is true because every $\mathsf{start}(\mathbf{w}')$ is connected to $\mathsf{halt}(\mathbf{w}', L(x))$, which we have just seen is not in $\mathcal{G}^{\mathbf{w}}_{\text{un}}$ for $\mathbf{w}' \neq \mathbf{w}$. □

Now, consider the sequence of nodes in $\mathcal{G} \times \{\mathsf{DOWN}, \mathsf{UP}\}$ given in the lemma statement:

$$((\mathbf{w}_t, v_t), e_t) := \Pi^t(\mathsf{start}(\mathbf{w}), \mathsf{DOWN})$$

To finish the proof, it suffices to observe that the sequence follows an Eulerian tour reaching every node in $\mathcal{G}^{\mathbf{w}}_{\text{un}}$. For completeness, we prove that the tour does indeed work as intended.

Say a node $((\mathbf{w}', v'), e')$ is *reached* if $((\mathbf{w}', v'), e') = ((\mathbf{w}_t, v_t), e_t)$ for some t. Since Π follows edges in $\mathcal{G}$, it is clear that only nodes in $\mathcal{G}^{\mathbf{w}}_{\text{un}}$ are reached. It remains to prove the converse: that every node in $\mathcal{G}^{\mathbf{w}}_{\text{un}}$ is reached. Or, more precisely, that $(\mathsf{halt}(\mathbf{w}, L(x)), \mathsf{DOWN})$ is reached.

Suppose for a contradiction that $(\mathsf{halt}(\mathbf{w}, L(x)), \mathsf{DOWN})$ is not reached. Since $(\mathsf{start}(\mathbf{w}), \mathsf{DOWN})$ is reached but $(\mathsf{halt}(\mathbf{w}, L(x)), \mathsf{DOWN})$ is not, there must be a pair of nodes $(\mathbf{w}_1, v_1), (\mathbf{w}_2, v_2) = \Gamma(\mathbf{w}_1, v_1)$ such that $((\mathbf{w}_1, v_1), \mathsf{DOWN})$ is reached but $((\mathbf{w}_2, v_2), \mathsf{DOWN})$ is not.

Let k be the rank of $(\mathbf{w}_1, v_1)$ in $\Gamma^{-1}(\mathbf{w}_2, v_2)$, and d be the in-degree of $(\mathbf{w}_2, v_2)$. If $k = d$, then $\Pi((\mathbf{w}_1, v_1), \mathsf{DOWN}) = ((\mathbf{w}_2, v_2), \mathsf{DOWN})$ (case 2, first part). By assumption this is not true, so $k < d$ and $\Pi((\mathbf{w}_1, v_1), \mathsf{DOWN}) = ((\mathbf{w}', v'), \mathsf{UP})$ where $(\mathbf{w}', v')$ has rank $k + 1$ in $\Gamma^{-1}(\mathbf{w}_2, v_2)$.

Since $((\mathbf{w}', v'), \mathsf{UP})$ is reached, eventually $((\mathbf{w}', v'), \mathsf{DOWN})$ must also be reached. Otherwise, the walk would get stuck in the subtree of $\mathcal{G}_{\text{un}}$ rooted at $(\mathbf{w}', v')$ without returning to $(\mathbf{w}_1, v_1)$, which is impossible because we know the permutation Π must eventually cycle back to nodes outside that subtree. The node after $((\mathbf{w}', v'), \mathsf{DOWN})$ is $((\mathbf{w}'', v''), \mathsf{UP})$ where $(\mathbf{w}'', v'')$ has rank $k + 2$ in $\Gamma^{-1}(\mathbf{w}_2, v_2)$.

We can only do this $d - k$ times in total before running out of nodes of higher rank in $\Gamma^1(\mathbf{w}_2, v_2)$. So, eventually, $((\mathbf{w}_{\text{last}}, v_{\text{last}}), \mathsf{DOWN})$ is reached, where $(\mathbf{w}_{\text{last}}, v_{\text{last}})$ has rank d in $\Gamma^{-1}(\mathbf{w}_2, v_2)$, and the next step after that is $((\mathbf{w}_2, v_2), \mathsf{DOWN})$: a contradiction. □

C Manipulating Randomized Configuration Graphs

In this section we prove Theorem 2.10. Note that RandWalk is easy to implement; it is clear that $\mathcal{G}_x[\sigma, r]$ can be computed by a catalytic subroutine given one-way access to r, by simulating the original catalytic machine M.

For RandRev, we need to deal with the fact that We say that a configuration σ is *reachable* if for some catalytic tape configuration $\mathbf{w}$ and $r \in \{0,1\}^*$, $\sigma = \mathcal{G}_x[\mathsf{start}(\mathbf{w}), r]$. (Note that $\mathsf{start}(\mathbf{w})$ is the initial configuration with catalytic tape $\mathbf{w}$.) Similar to deterministic catalytic computation, there are no two different initial configurations that reach the same configuration.

Proposition C.1. *For every randomized configuration graph $\mathcal{G}_x$, for every reachable configuration σ', there is exactly one initial configuration σ such that there exists $z \in \{0,1\}^*$ such that $\mathcal{G}[\sigma, z] = \sigma'$.*

Proof. Suppose for contradiction this was not the case. Then for z_1, z_2 we have

$$\mathcal{G}[\sigma_1, z_1] = \sigma' = \mathcal{G}[\sigma_2, z_2]$$

for a pair of starting configurations $\sigma_1 \neq \sigma_2$. Then it is easy to see that with non-zero probability, the machine does not reset the tape correctly on at least one of these starting states. □

Note that this means that a reachable configuration σ' uniquely specifies its initial configuration σ, even when we do not remember the specific *path* used to reach σ' from σ. This property is essential in our algorithm.

We state the following standard result that helps to efficiently traverse on $\mathcal{G}_x$ in reverse direction (mirroring the result of Proposition B.2 used for traversing the configuration graph of deterministic machines). The proof is omitted.

Proposition C.2 (Folklore)**.** *There are deterministic catalytic subroutines* $\mathsf{InDegree}$*,* InEdge*,* Rank *using $O(s)$ additional workspace that works as follows. Let $x \in \{0,1\}^n$ be an input of M and $(\mathbf{w}, v) \in \mathcal{G}_x$ be a configuration.*

- $\mathsf{InDegree}^{\mathbf{w}}(v, b)$ *returns the number of configurations $(\mathbf{w}', v')$ such that $\Gamma_b(\mathbf{w}', v') = (w, v)$, with read-only access to the catalytic tape $\mathbf{w}$.*
- $\mathsf{InEdge}^{\mathbf{w}}(v, i, b)$ *updates the catalytic tape to $\mathbf{w}'$ and returns v', where $(\mathbf{w}', v')$ is the i-th node such such that $\Gamma_b(\mathbf{w}', v') = (\mathbf{w}, v)$ with respect to a consistent ordering.*
- $\mathsf{Rank}^{\mathbf{w}'}(v', b)$ *returns the rank of the node in the list of $\Gamma_b^{-1}(\mathbf{w}, v)$ (according to the ordering of $\mathsf{InEdge}^{\mathbf{w}}(v, \cdot, b)$), where $(\mathbf{w}, v) := \Gamma_b(\mathbf{w}', v')$, with read-only access to the catalytic tape $\mathbf{w}'$.*

We are now ready to prove the $\mathsf{RandRev}$ case of Theorem 2.10, which we restate here:

Lemma C.3 (Reverse Traversal Given Walk)**.** *For a* $\mathbf{CL}$ *configuration graph $\mathcal{G}$, there is a catalytic subroutine $\mathsf{RandRev}^{\mathbf{w}'}(x, v', r)$ using $O(s + \log |r|)$ additional workspace that works as follows:*

1. *If there is a catalytic tape configuration $\mathbf{w}$ such that $\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = (\mathbf{w}', v')$, it accepts and with the catalytic tape in configuration $\mathbf{w}$.*
2. *Otherwise, it rejects and leaves the catalytic tape in configuration $\mathbf{w}'$.*

Proof. Let $\mathbf{w}'$ be the catalytic tape and v', r be the input. We define a directed graph $\mathcal{G}^*$ whose nodes are of the form $(\mathbf{w}_1, v_1, k) \in \{0,1\}^{2^s} \times \{0,1\}^s \times [|r|]$. Let $\sigma_1 = (\mathbf{w}_1, v_1)$, $\sigma_2 = (\mathbf{w}_2, v_2)$, and $k \in [1, |r| + 1]$, there is an edge $(\mathbf{w}_1, v_1, k) \mapsto (\mathbf{w}_2, v_2, k + 1)$ if $\sigma_2 = \Gamma_{r_k}(\sigma_1)$. Consider the induced sub-graph $\mathcal{G}^*_{\mathbf{w}', v'} \subseteq \mathcal{G}^*$ containing all nodes that can reach $(\mathbf{w}', v', |r| + 1)$ through edges in $\mathcal{G}^*$.

Claim C.4. *$\mathcal{G}^*_{\mathbf{w}', v'}$ is a directed tree rooted at $(\mathbf{w}', v', |r| + 1)$, where the direction of the edges is from the child to its parent.*

Proof. Clearly $(\mathbf{w}', v', |r| + 1)$ is in $\mathcal{G}^*_{\mathbf{w}', v'}$. Since every node in $\mathcal{G}^*_{\mathbf{w}', v'}$ can reach $(\mathbf{w}', v', |r| + 1)$ through edges in $\mathcal{G}^*$, we know that $\mathcal{G}^*_{\mathbf{w}', v'}$ is connected (forgetting the direction of edges). Moreover, $(\mathbf{w}', v', |r| + 1)$ has out-degree 0, and all other nodes has out-degree exactly 1. This implies that it is a directed tree going in the root $(\mathbf{w}', v', |r| + 1)$. □

It is clear that there is a catalytic tape configuration $\mathbf{w}$ such that $\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = (\mathbf{w}', v')$ if and only if $(\mathbf{w}, v, 1) \in \mathcal{G}^*_{\mathbf{w}', v'}$, where $(\mathbf{w}, v) = \mathsf{start}(w)$. By Proposition C.1, there is at most one such $\mathbf{w}$. Therefore, the task of finding such $\mathbf{w}$ reduces to finding a node of form $(\mathbf{w}, v, 1)$ satisfying $(\mathbf{w}, v) = \mathsf{start}(\mathbf{w})$ on the tree $\mathcal{G}^*_{\mathbf{w}', v'}$.

The algorithm $\mathsf{RandRev}$ performs a DFS on the tree from the root $(\mathbf{w}', v', |r| + 1)$. We maintain a counter k and a tape configuration v on the work tape. Suppose that the catalytic tape is $\mathbf{w}$, our DFS algorithm reaches the node $(\mathbf{w}, v, k)$. It works as follows.

1. Let $\mathbf{w}$ be the current catalytic tape. Suppose that $(\mathbf{w}, v) = \mathsf{start}(\mathbf{w})$ and $k = 1$, we conclude by the discussion above that $\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = (\mathbf{w}', v')$. The algorithm then halts and accepts.
2. By calling the catalytic subroutine $\mathsf{InDegree}^{\mathbf{w}}(v, r_{k-1})$ in Proposition C.2, we can check whether the current node $(\mathbf{w}, v, k)$ has any child. In case that there is at least one child, i.e., $k > 1$ and $\mathsf{InDegree}^{\mathbf{w}}(v, r_{k-1}) > 0$, the first child of $(\mathbf{w}, v, k)$ will be the next node to visit. Therefore, we move to that node by calling $\hat{v} \leftarrow \mathsf{InEdge}^{\mathbf{w}}(v, i, b)$ and setting $v \leftarrow \hat{v}$, $k \leftarrow k - 1$, and restart from Step 1.

3. Otherwise, we need to find the next node to visit in the DFS order. It jumps towards the root of the tree until there is an unvisited sibling of the current node $(\mathbf{w}, v, k)$. Concretely, it first computes its rank among the siblings by calling $j \leftarrow \mathsf{Rank}^{\mathbf{w}}(v, r_k)$. It then walks to its parent by calling $v \leftarrow \mathsf{RandWalk}^{\mathbf{w}}(x, v, r_k)$ and setting $k \leftarrow k + 1$.

   (a) If the in-degree of the parent (which can be obtained by calling $\mathsf{InDegree}^{\mathbf{w}}(v, r_{k-1})$) is not equal to the rank j, we move to the $(j+1)$-th child of the parent (by calling $\hat{v} \leftarrow \mathsf{InEdge}^{\mathbf{w}}(v, r_{k-1})$ and setting $v \leftarrow \hat{v}$, $k \leftarrow k - 1$) and restart from Step 1.

   (b) If we reach $k = |r| + 1$ without triggering Case 3a, it means that all nodes are visited. The algorithm then halts and rejects.

(See Catalytic Subroutine 3 for the pseudocode of the algorithm.)

To see the correctness of the algorithm, one can prove the invariant that when the algorithm reaches Step 1 for the i-th time, $(\mathbf{w}, v, k)$ is the i-th node on the tree $\mathcal{G}^*_{\mathbf{w}',v'}$ in its DFS order, where the children of a node are visited according to the consistent ordering given by Rank and InEdge in Proposition C.2. If there is a valid initial configuration from which $(\mathbf{w}', v', |r| + 1)$ is reachable, it will be visited in Line 5, which satisfies Item 1. Otherwise, it will eventually visit all the nodes and reach Line 17. By that time, $(\mathbf{w}, v, k)$ will have been reset back to $(\mathbf{w}', v', |r| + 1)$, and thus it satisfies Item 2. □

Remark C.5. We note that Lemma C.3 is different from its counterpart Lemma B.1 for deterministic computation in an important way. In Lemma B.1, the forward and backwards transitions are the exact reverse of each other. In Lemma C.3, the reverse simulation algorithm on input r could take much longer than the forwards simulation on r. This is because in Lemma B.1, we transform the forward simulation algorithm to "match" the reverse simulation. Here this modification is not possible, since we require a fixed (i.e. unmodified) forward algorithm that works for *every* walk r.

D List and Pseudocode of Catalytic Subroutines

D.1 Reference table

Subroutine	Reference	Usage	Pseudocode
Det. config. graphs	Section 2.3.1		
$\mathsf{DetWalk}^{\mathbf{w}}(x,k)$	Theorem 2.7	either returns $L(x)$ or sets the catalytic tape to $\mathbf{w}'$ and returns v', a', where $((\mathbf{w}', v'), a') = \Pi^k[\mathsf{start}(\mathbf{w}), 0]$	Algorithm 1
$\mathsf{DetRev}^{\mathbf{w}'}(x,v,a)$	Theorem 2.7	sets the catalytic tape to $\mathbf{w}$ and returns k, where $((\mathbf{w}', v'), a') = \Pi^k[\mathsf{start}(\mathbf{w}), 0]$	Algorithm 2
Rand. config. graphs	Section 2.3.2		
$\mathsf{RandWalk}^{\mathbf{w}}(x,v,r)$	Theorem 2.10	sets the catalytic tape to $\mathbf{w}'$ and returns v', where $\mathcal{G}_x[(\mathbf{w}, v), r] = (\mathbf{w}', v')$	N/A
$\mathsf{RandRev}^{\mathbf{w}'}(x,v',r)$	Theorem 2.10	if there is a catalytic configuration $\mathbf{w}$ such that $\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = (\mathbf{w}', v')$, accepts with the catalytic tape in configuration $\mathbf{w}$; otherwise rejects and leaves the catalytic tape in configuration $\mathbf{w}'$	Algorithm 3
$\mathsf{EQ}^{\mathbf{w}}(x,r,r')$	Lemma 2.11	accepts iff $\mathcal{G}_x[\mathsf{start}(\mathbf{w}), r] = \mathcal{G}_x[\mathsf{start}(\mathbf{w}), r']$	Algorithm 4
Det. results	Section 3		
$\mathsf{DetComp}^{\mathbf{w}}(1^B,x)$	Theorem 3.1	either returns $L(x)$ or sets the catalytic tape to $\mathbf{w}' \circ 0^B$, where $\lvert\mathbf{w}'\rvert = 2^s + O(s)$	Algorithm 5
$\mathsf{DetDecomp}^{\mathbf{w}'\circ 0^B}(1^B,x)$	Theorem 3.1	sets the catalytic tape to $\mathbf{w}$, where $\mathsf{DetDecomp}^{\mathbf{w}}(1^B,x)$ sets the catalytic tape to $\mathbf{w}' \circ 0^B$	Algorithm 6
Large graph case	Section 4.1		
$\mathsf{Size}^{\mathbf{w}}(x,Y)$	Theorem 4.4	returns $\lvert\mathcal{Y}(\mathbf{w}, Y)\rvert$	Algorithm 7
$\mathsf{RandComp}^{\mathbf{w}}(x,i,Y)$	Theorem 4.4	sets the catalytic tape to $\mathbf{w}'$ and returns v', where $f((\mathbf{w}', v')) = i$	Algorithm 8
$\mathsf{RandDecomp}^{\mathbf{w}'}(x,v',Y)$	Theorem 4.4	sets the catalytic tape to $\mathbf{w}$ and returns $i = f((\mathbf{w}', v'))$	Algorithm 9
PRG tools	Section 4.2		
$\mathsf{NW}^f(s)$	Theorem 4.7	candidate pseudorandom generator	N/A
$\mathsf{Hint}^{f,P}$	Theorem 4.7	assuming P is a previous bit predictor for NW^f, returns $h \in \{0,1\}^{O(\log N)}$ and an interval $K \subseteq [N^{c_{\mathsf{NW}}}]$ of length N^c	N/A
$\mathsf{Decode}^{a,\widetilde{f},P}(j)$	Theorem 4.7	assuming $\widetilde{f}_{\bar{K}} = f_{\bar{K}}$ and P is a previous bit predictor for NW^f, returns f_j for $j \in K$	N/A
Small graph case	Section 4.2		
$\mathsf{Pigeon}^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{2S}}(x)$	Claim 4.8	returns i such that $\mathcal{Y}_i \subseteq \mathcal{Y}_{<i}$	Algorithm 10
$A_1^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r)$	Claim 4.9	returns $L_1(r) = \mathbb{I}[\mathcal{U}[\mathsf{start}(\mathbf{w}), r] = \perp]$	Algorithm 11
$A_2^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r)$	Claim 4.9	returns $L_2(r) = \mathbb{I}[\mathcal{U}[\mathsf{start}(\mathbf{w}), r] = \mathsf{acc}(\mathbf{w})]$	Algorithm 12
$\mathsf{Pred}^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(a,v,r)$	Claim 4.10	returns $P_{a,v}(r) = L_a((G_k(\tau))_{\leq l_1} \circ r_{>l_2}) \oplus b$	N/A

D.2 Codebase

Catalytic Subroutine 1: $\mathsf{DetWalk}^{\mathbf{w}}(x, k)$ (Theorem 2.7)

$v \leftarrow 0^s, a \leftarrow 0$;
for $t = 1 \dots k$ **do**
  Call $M_{\rightarrow}(x, v, a)$ and store the result in v and a;
  if $a = 0$ *and* v *has the form* $b1^{s(n)-1}$ **then**
    `//` $M_{\rightarrow}(x, v, a)$ `found answer` $b = L(x)$ `and reset` **w**
    Return b;
  end
end
`// No answer found after` k `steps`
Return (v, a);

Catalytic Subroutine 2: $\mathsf{DetRev}^{\mathbf{w}}(x, v, a)$ (Theorem 2.7)

$v' \leftarrow v, a' \leftarrow a$;
while $(v, a) \neq (0^s, 0)$ **do**
  Call $M_{\leftarrow}(x, v', a')$ and store the result in v' and a'.;
  $t \leftarrow t + 1$;
end
return t;

Catalytic Subroutine 3: $\mathsf{RandRev}^{\mathbf{w}'}(x, v', r)$ (Theorem 2.10)

```
// Throughout the algorithm, we use w_CUR to represent the current
   configuration of the catalytic tape.
Initialize v ← v', k ← |r| + 1;
while true do
    if (w_CUR, v) = start(w_CUR) and k = 1 then
        // Start node found
        Return 1;
    end
    if k > 1 and InDegree^{w_CUR}(v, r_{k-1}) > 0 then
        // iteratively search the first child of (w_CUR, v, k)
        v ← InEdge^{w_CUR}(v, 1, r_{k-1});
        k ← k − 1;
    else
        // Reach a leaf that is not of form (·, 0, 1), find the next node
        while k ≤ |r| do
            j ← Rank^{w_CUR}(v, r_k);
            // Remember the rank of the current node
            v ← RandWalk^{w_CUR}(x, v, r_k), k ← k + 1;
            // Walk to its parent
            if InDegree^{w_CUR}(x, v, r_k) ≠ j then
                // We have not visited all children of (w_CUR, v, k)
                v ← InEdge^{w_CUR}(x, v, j + 1) and restart from line 2;
            end
        end
        // all nodes visited, returned to the root
        Return 0;
    end
end
```

Catalytic Subroutine 4: $\mathsf{EQ}^{\mathbf{w}}(x, r, r')$ (Lemma 2.11)

```
for i = 1 ... s + 2^s do
    Run RandWalk^w(x, 0^s, r), end with catalytic tape w_r and return of v_r, where
      (w_r, v_r) := G[start(w), r];
    Set b ← the i-th bit of (w_r, v_r) and run RandRev^{w_r}(x, v_r, r);
    Run RandWalk^w(x, 0^s, r'), end with catalytic tape w_{r'} and return of v_{r'}, where
      (w'_r, v'_r) := G[start(w), r];
    Set b' ← the i-th bit of (w_{r'}, v_{r'}) and run RandRev^{w_{r'}}(x, v_{r'}, r');
    if b ≠ b' then
        // Mismatch between ith bits, so (w_r, v_r) ≠ (w_{r'}, v_{r'})
        Return 0;
    end
end
// Every bit matches, so (w_r, v_r) ≠ (w_{r'}, v_{r'})
Return 1;
```

Catalytic Subroutine 5: $\mathsf{DetComp}^{\mathbf{w}}(1^B, x)$ (Theorem 3.1)

$k \leftarrow 0, \mathbf{w}'_0 \leftarrow \mathbf{w}$;
while $k < B$ **do**
  // Invariant: catalytic tape has the form $\mathbf{w}'_k \circ 0^k$
  Parse $\mathbf{w}'_k$ as $\mathbf{m} \circ i \circ z_k$ for $|\mathbf{m}| = 2^s$, $|i| = s + 2$;
  Run $\mathsf{DetWalk}^{\mathbf{m}}(x, i)$;
  if DetWalk *returns* $L(x)$ **then**
    Call $\mathsf{DetDecomp}^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ and then return $L(x)$;
  end
  else
    $(v', a') \leftarrow \mathsf{DetWalk}^{\mathbf{m}}(x, i)$;
    // At this point, the catalytic tape has the form $\mathbf{m}' \circ i \circ z_k \circ 0^k$.
    Set the catalytic tape to $(\mathbf{w}'_{k+1} := \mathbf{m}' \circ (v' \circ a') \circ z_k) \circ 0^{k+1}$;
  end
  $k \leftarrow k + 1$;
end
Return $\perp$;

Catalytic Subroutine 6: $\mathsf{DetDecomp}^{\mathbf{w}'_B \circ 0^B}(1^B, x)$ (Theorem 3.1)

$k \leftarrow B$;
while $k > 0$ **do**
  // Invariant: catalytic tape has the form $\mathbf{w}'_k \circ 0^k$.
  Parse $\mathbf{w}'_k$ as $\mathbf{m} \circ (v \circ a) \circ z_k$ where $|\mathbf{m}| = 2^s, |v \circ a| = s + 1$;
  $i \leftarrow \mathsf{DetRev}^{\mathbf{m}}(x, v, a)$;
  // At this point, the catalytic tape has the form $\mathbf{m}' \circ (v \circ a) \circ z_k \circ 0^k$.
  Set the catalytic tape to $(\mathbf{w}'_{k-1} := \mathbf{m}' \circ i \circ z_k) \circ 0^{k+1}$. $k \leftarrow k - 1$;
end

Catalytic Subroutine 7: $\mathsf{Size}^{\mathbf{w}}(x, Y)$ (Theorem 4.4)

Initialize $i \leftarrow 0$;
for $k = 1 \ldots m\ell$ **do**
  Set $first \leftarrow 1$;
  for $k' = 1 \ldots k$ **do**
    if $\mathsf{EQ}^{\mathbf{w}}(x, z_k, z_{k'})$ **then**
      // $\mathsf{First}(z_k)$ is not true
      Set $first \leftarrow 0$;
    end
  end
  if $first = 1$ **then**
    // $\mathsf{First}(z_k)$ is true
    Increment i;
  end
end
Return i;

Catalytic Subroutine 8: $\mathsf{RandComp}^{\mathbf{w}}(x, i, Y)$ (Theorem 4.4)

Initialize $i' \leftarrow 0$;
for $k = 1 \dots m\ell$ **do**
  Set $first \leftarrow 1$;
  for $k' = 1 \dots k$ **do**
    if $\mathsf{EQ}^{\mathbf{w}}(x, z_k, z_{k'})$ **then**
      // $\mathsf{First}(z_k)$ is not true
      Set $first \leftarrow 0$;
    end
  end
  if $first = 1$ **then**
    // $\mathsf{First}(z_k)$ is true
    Increment i';
    if $i' = i$ **then**
      Run $\mathsf{RandWalk}^{\mathbf{w}}(x, 0^s, z_k)$ to set the catalytic tape to $\mathbf{w}_k$ and return v_k;
    end
  end
end

Catalytic Subroutine 9: $\mathsf{RandDecomp}^{\mathbf{w}'}(x, v', Y)$ (Theorem 4.4)

Initialize $i' \leftarrow 0$;
for $k = 1 \dots m\ell$ **do**
  Run $\mathsf{RandRev}^{\mathbf{w}'}(x, v', z_k)$;
  if $\mathsf{RandRev}$ *accepts* **then**
    // z_k is the first string such that $\mathsf{RandWalk}^{\mathbf{w}}(x, 0, z_k) = (\mathbf{w}', v')$
    Break from the loop;
  end
end
// z_k is the first string to reach $(\mathbf{w}', v')$, so $\mathsf{Size}^{\mathbf{w}}(x, \{z_1 \dots z_k\}) = f((\mathbf{w}', v'))$
Run $\mathsf{Size}^{\mathbf{w}}(x, \{z_1 \dots z_k\})$ and return output i;

Catalytic Subroutine 10: $\mathsf{Pigeon}^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{2S}}(x)$ (Claim 4.8)

$flag_i \leftarrow 0, flag_y \leftarrow 0$;
for $i = 1 \ldots B$ **do**
  Set $flag_i \leftarrow 0$;
  for *prefix* $y_<$ *of* y *in* Y_i **do**
    // prefix generated via $\mathrm{NW}^{\mathbf{m}_i}(\mathbf{U})$
    Set $flag_y \leftarrow 0$;
    for *prefix* $z_<$ *of* z *in* $Y_{<i}$ **do**
      // prefix generated via $\mathrm{NW}^{\mathbf{m}_k}(\mathbf{U})$ for $k \in [i-1]$
      if $\mathsf{EQ}^{\mathbf{w}}(x, y_<, z_<)$ **then**
        // Found $\mathcal{G}[\mathsf{start}(\mathbf{w}), y_<]$
        Set $flag_y \leftarrow 1$;
      end
    end
    if $flag_y = 0$ **then**
      // No $z_<$ such that $\mathcal{G}[\mathsf{start}(\mathbf{w}), y_<] = \mathcal{G}[\mathsf{start}(\mathbf{w}), y_<]$
      $flag_i \leftarrow 1$;
    end
  end
  if $flag_i = 0$ **then**
    // Every $y_<$ has a $z_<$ such that $\mathcal{G}[\mathsf{start}(\mathbf{w}), y_<] = \mathcal{G}[\mathsf{start}(\mathbf{w}), y_<]$
    Return i;
  end
end

Catalytic Subroutine 11: $A_1^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r)$ (Claim 4.9)

// $\mathcal{U} = \mathcal{Y}_{<i}$.
for *prefix* $r_<$ *of* r **do**
  Set $flag_r \leftarrow 0$;
  for *prefix* $y_<$ *of* y *in* $Y_{<i}$ **do**
    // prefix generated via $\mathrm{NW}^{\mathbf{m}_k}(\mathbf{U})$ for $k \in [i-1]$
    if $\mathsf{EQ}^{\mathbf{w}}(x, r_<, y_<)$ **then**
      // $\mathcal{G}[\mathsf{start}(\mathbf{w}), r_<]$ is inside $\mathcal{U}$
      $flag_r \leftarrow 1$;
    end
  end
  if $flag_r = 0$ **then**
    // $\mathcal{G}[\mathsf{start}(\mathbf{w}), r_<]$ does not match any state in $\mathcal{U}$
    Return 1;
  end
end
// $\mathcal{G}[\mathsf{start}(\mathbf{w}), r]$ has not left $\mathcal{U}$
Return 0;

Catalytic Subroutine 12: $A_2^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r)$ (Claim 4.9)

// $\mathcal{U} = \mathcal{Y}_{<i}$.
Run $A_1^{\mathbf{w}\circ\mathbf{m}_1\circ\ldots\circ\mathbf{m}_{i-1}}(r)$;
if A_1 *accepts* **then**
    // $\mathcal{U}[\mathsf{start}(\mathbf{w}), r] = \perp$
    Return 0;
end
Run $\mathsf{RandWalk}^{\mathbf{w}}(x, 0, r)$ and end in state $(\mathbf{w}', v')$;
if $(\mathbf{w}', v') = \mathsf{acc}(\mathbf{w})$ **then**
    // $\mathcal{U}[\mathsf{start}(\mathbf{w}), r] = \mathsf{acc}(\mathbf{w})$
    Return 1;
end
else
    // $\mathcal{U}[\mathsf{start}(\mathbf{w}), r] \neq \mathsf{acc}(\mathbf{w})$
    Return 0;
end

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il