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

James Cook
falsifian@falsifian.org
Unaffiliated
Toronto, Canada

Jiatu Li*
jiatuli@mit.edu
MIT
Cambridge, United States

Ian Mertz†
ian.mertz@warwick.ac.uk
University of Warwick
Coventry, United Kingdom

Edward Pyne‡
epyne@mit.edu
MIT
Cambridge, United States

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 MIT Akamai Fellowship and the National Science Foundation under Grant CCF-2127597.
†Supported by Royal Society University Research Fellowship URF R1 191059.
‡Supported by a Jane Street Graduate Research Fellowship.

STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718112

CCS Concepts

• **Theory of computation** → **Complexity classes**; **Pseudorandomness and derandomization**.

Keywords

Catalytic Computation, Derandomization, Complexity

ACM Reference Format:
James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. 2025. The Structure of Catalytic Space: Capturing Randomness and Time via Compression. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3717823.3718112

1 Introduction

How useful is access to a full hard drive? The *catalytic logspace* (**CL**) model, introduced by Buhrman, Cleve, Koucký, Loff, and Speelman [2], 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 [9, 13, 18, 23] 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, [2] 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 [1, 3, 5–8, 10, 14, 15, 22, 26] (see surveys of Koucký [21] and Mertz [24] 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 [10], 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 **BPL** = **L**) has been highly fruitful, resulting in the state of the art result of [16, 28] that randomized space s can be simulated deterministically in space $s^{3/2-o(1)}$. In the catalytic setting, however, derandomization — i.e. **CBPL** = **CL**, posed as an open question in Mertz [24] — was only known assuming strong circuit lower bounds [10].

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 **PSPACE** = **BPPSPACE** is known whereas **BPL** = **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 **BPL** = **L** and **ZPP** = **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**, Buhrman et al. [2] showed that $\mathbf{CL} \subseteq \mathbf{ZPP}$; thus $\mathbf{CL} \subseteq \mathbf{P}$ under the widely-believed (uniform) assumption that **ZPP** = **P**. However, putting aside the long-standing intractability of derandomizing **ZPP**, even finding such a derandomization does not necessarily 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 $\mathbf{CLP} \subseteq \mathbf{CL} \cap \mathbf{P}$ is immediate, the converse was not known. The work of [10] showed that **CL** = **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 time and catalytic space *individually* admit algorithms which are efficient with respect to both *simultaneously*.

Theorem 1.2.

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

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

Corollary 1.3.

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

One can view this result in a positive sense, and as a barrier. On the positive side, a proof of $\mathbf{CL} \subseteq \mathbf{P}$ 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 the collapse of two subclasses **CL** and **CLP** of **ZPP**.

Corollary 1.4.

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

Towards proving **CL** = **CLP**. Corollary 1.4 can be strengthened to show that **CL** = **CLP** follows from the derandomization of a syntactic subclass **LOSSY** of **ZPP** that has been studied by several recent works [4, 17, 19, 20, 22].

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 $D(C(x)) \neq x$.

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

Theorem 1.6.

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

Together with Corollary 1.3, we show that **LOSSY** = **P** is sufficient to prove **CL** = **CLP**. This strengthens Corollary 1.4 as $\mathbf{LOSSY} \subseteq \mathbf{ZPP}$, and it is not known whether **LOSSY** = **ZPP**, and constitutes the first improvement on the assumptions required to prove $\mathbf{CL} \subseteq \mathbf{P}$.[1]

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,[2] 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 1.7.

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

While Theorem 1.1 can be thought of as bounding the power of randomness in catalytic computing, the forward direction of Theorem 1.7 gives a highly nontrivial *use* for randomness in the time-bounded catalytic model.

Theorem 1.7 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, as a corollary we unconditionally resolve the **CL** vs **CLP** question in the *randomized* setting.

[1] For instance, Korten [20] prove that under uniform space-time trade-off lower bounds, we can achieve a "scaled-up" version of **LOSSY** = **P**: languages that are $2^{O(n)}$-time reducible to LossyCode can be solved in deterministic $2^{O(n)}$-time. Applying our result, we obtain that $\mathbf{CSPACE}[n] \subseteq \mathbf{DTIME}[2^{O(n)}]$ from the same assumption.

[2] Where on input x, we either compute $L(x)$ or return a special symbol $\perp$, and we return $\perp$ with probability at most 1/3 for every input and starting tape.

Corollary 1.8.

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

By Corollary 1.4, derandomizing **ZPP** would also imply derandomization on both sides of Corollary 1.8, which 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** [11, 12, 22, 24, 26]. 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** [11, 12, 26].

To explain the relevance of compress–or–random to the problem of ensuring (deterministic) catalytic algorithms have fast *runtime*, recall that [2] 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, which is required for the compress–or–random paradigm.

1.4.1 Compress–or–Compute: Compression from High Runtime. To explore our first idea, we will consider Theorem 1.2; our goal is to show, given a language L decidable by both a catalytic logspace machine and a polynomial time machine, how to use the compress–or–random framework to decide L in poly-time bounded **CL**. Our first main insight in this paper — in essence the only tool needed for this result — is a new way of compressing the tape when simulating the **CL** machine M.

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 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 think of our new machine as having a larger starting tape of 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[3]. 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 via 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, since the machine must reset the catalytic tape no matter the sequence of random bits.

Naïvely, 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

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

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

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

[3]Being able to efficiently reverse the computation in order to decompress requires some additional subtlety; see Section 3 for details.

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 naïvely "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: 1) the configuration graph $\mathcal{R}(\mathbf{w})$ is actually small; or 2) our collection of random walks Y does a bad job exploring it. In addition, we must specify where the set of walks Y actually comes from.

To deal with all of these issues, we create the strings Y using an instantiation of the *Nisan-Wigderson generator* [25] developed by Doron, Pyne, and Tell [11]. We denote the generator as

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

where $f \in \{0,1\}^{\text{poly}(n)}$ 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 [11], 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.

What if $\mathsf{NW}^{\mathbf{m}}$ does fool D? Ideally, we would have used the following distinguisher:

$$D^*(r) := \mathbb{I}\left[M \text{ accepts on randomness } r \right]. \tag{1}$$

With this choice of D, if $\mathsf{NW}^{\mathbf{m}}$ fools D, it immediately follows that we can use $\mathsf{NW}^{\mathbf{m}}(\mathbf{U})$ as the random inputs for M and get the right answer by majority vote. However, our compression argument runs into trouble with this distinguisher because we cannot compute the expected value of D^* over the true uniform distribution $\mathbf{U}$ — to wit, this is our entire goal. We will instead use two distinguishers, both easy to test given access to Y, and show that fooling both is sufficient to approximate $D^*(\mathbf{U})$.

Our first test is whether $\mathsf{NW}^{\mathbf{w}}$ does a good job exploring the graph $\mathcal{R}(\mathbf{w})$ by seeing how much is left out of $\mathcal{Y}$:

$$D_{\mathsf{e}}(r) := \mathbb{I}\left[M \text{ leaves } \mathcal{Y} \text{ when reading random bits } r \right], \tag{2}$$

Note that we have $D_{\mathsf{e}}(\mathsf{NW}^{\mathbf{m}}(\mathbf{U})) = 0$, so if $\mathbb{E}[D_{\mathsf{e}}(\mathbf{U})] \geq 1/10$ 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.

The second distinguisher tests whether $\mathsf{NW}^{\mathbf{m}}$ gives similar answers to random walks, but unlike D^* we only care about the "tractable" case when D_{e} is fooled:

$$D_{\mathsf{acc}}(r) := \mathbb{I}[M \text{ does not leave } \mathcal{Y} \text{ and accepts when reading random bits}]. \tag{3}$$

If $\mathsf{NW}^{\mathbf{m}}$ fools both D_{e} and D_{acc}, it can be used to simulate M.

In the case that $\mathsf{NW}^{\mathbf{m}}$ fails to fool one of the distingishers $D = D_{\mathsf{e}}$ or $D = D_{\mathsf{acc}}$, we show how to implement the transformation — that is, from D to a small circuit for f — within catalytic logspace. We build off recent works studying related questions [11, 27], while incorporating new ideas. There are two required steps in this transformation. The first is transforming each distinguisher D into a previous bit predictor[4] P for $\mathsf{NW}^{\mathbf{m}}$, i.e. a function satisfying

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

By Yao's Lemma [29], we have that there exists a predictor for $\mathsf{NW}^{\mathbf{m}}$ with the following form:

$$P(r_{>}) = D(z \circ r_{>}) \oplus b \tag{4}$$

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 [11]: restricting the first bits of D_{e} or D_{acc} to z simply corresponds to starting the random walk at a new location in $\mathcal{Y}$.[5] (This is the reason we used two distinguishers $D_{\mathsf{e}}, D_{\mathsf{acc}}$: the observation does not apply to the "ideal" distinguisher D^* in Equation (1) as M may leave $\mathcal{Y}$ on some randomness r.) As there are only $\text{poly}(n)$ 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 [11, 22], 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 [11].[6] Otherwise, we must have that NW does a good job exploring the configuration graph.

Technical Issue: Compression Destroys the Graph. 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

[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 [22].

[5] More formally, for both D_{e} and D_{acc}, we also need to introduce a new node $\partial\mathcal{Y}$ in the configuration graph to denote the case where M leaves $\mathcal{Y}$ (see the full version for details); this suffices as both distinguishers reject immediately when leaving $\mathcal{Y}$.

[6] For this step, we can replace the Nisan-Widgerson generator of [11] with a compression algorithm utilizing previous bit predictors implicit in Korten's proof of the **prBPP**-hardness of R-Lossy Code [20] (see the full version for details). 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 [11] allows us to compress a truth table of size n^c to n bits, which is indeed an overkill.)

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

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

If this holds, and we use $\mathcal{Y} := \bigcup_{j \neq i} \mathcal{Y}_j$ when defining the distinguishers $D_{\mathrm{e}}, D_{\mathrm{acc}}$, then we can describe the corresponding predictors in Equation (4) using outputs of all PRGs *except* G_i, which will be the only one we attempt to compress, and hence we never lose access to $\mathcal{Y}$.

We remark that [11] did not have to deal with this complication for their proof of **BPL** $\subseteq$ **CL**, as there the graph was always present on the read-only input tape, whereas here we have only implicit access.

Putting the Cases Together. We now present one step (with mild simplifications) of our final algorithm. Let M be the randomized catalytic machine deciding L. 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

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

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 in the configuration graph of M, 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 the walks Y.
(2) If Y explores fewer than $2n^c$ states, there is some j such that G_j exclusively reaches states already reached by the other generators. Then let $\mathcal{Y} = \bigcup_{j \neq i} \mathcal{Y}_i$ be that set of states, and consider the distinguishers D_{e} (see Equation (2)) and D_{acc} (see Equation (3)) based on $\mathcal{Y}$. From these distinguishers, we 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 $\mathcal{Y}$ does a good job of covering $\mathcal{R}(\mathbf{w})$ (otherwise D_{e} is a good distinguisher) and G_j does a good job of simulating M within $\mathcal{Y}$ (otherwise D_{acc} is a good distinguisher), so we can use the output of G_j to derandomize M.

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 **CL** $\subseteq$ **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. **CZPL** $\cap$ **P** = **CZPLP**, then the question is resolved by Theorem 1.7.

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 **CNL** $\subseteq$ **ZPP**, and Buhrman, Koucký, Loff, and Speelman [3] 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. We defer the proof of Theorem 1.1 to the full version.

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 $L(x) := \mathbb{I}[x \in L]$.

For a graph $G = (V, E)$, we use $\mathrm{Vertices}(G) := V$ to denote the set of vertices of G.

2.2 Complexity Classes for Catalytic Computation

Our basic starting point is the notion of a catalytic machine, as defined by [2]:

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 $\mathbf{w}$.

For technical reason, we introduce a weaker definition of catalytic subroutines, which shares the same syntactic setup of catalytic machines but does not necessarily reset its catalytic tape to the initial configuration after the computation. More formally:

Definition 2.2. A *catalytic subroutine* is defined as a standard Turing machine — 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.

That is, a catalytic machine M is a catalytic subroutine that for any input x and any initial configuration $\mathbf{w}$ of the catalytic tape of M, $M^{\mathbf{w}}(x)$ terminates and resets the catalytic tape back to its initial configuration $\mathbf{w}$.

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 ([10])**.** 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, if 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:

- **CSPACE** $[s]$ is the class of languages decidable by catalytic Turing machines using workspace $O(s)$ and catalytic space $2^{O(s)}$.
- **CZPSPACE** $[s]$ and **CBPSPACE** $[s]$[7] are the classes of languages decidable by randomized catalytic Turing machines using workspace $O(s)$, catalytic space $2^{O(s)}$, and access to random bits, with zero-sided and two-sided error, respectively.
- **CTISP** $[t, s]$ is the class of languages decidable by catalytic Turing machines using time $O(t)$, workspace $O(s)$, and catalytic space $2^{O(s)}$.
- **CZPTISP** $[t, s]$ and **CBPTISP** $[t, s]$ are the classes of languages decidable by randomized catalytic Turing machines using time $O(t)$, workspace $O(s)$, catalytic space $2^{O(s)}$, and access to 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:

- **CL** := **CSPACE** $[\log n]$
- **CBPL** := **CBPSPACE** $[\log n]$
- **CLP** := **CTISP** $[\mathrm{poly}(n), \log n]$ (also called **CSC**^{1} [10])
- **CZPLP** := **CZPTISP** $[\mathrm{poly}(n), \log n]$

Note that while some works consider the more general case of **CSPACE** $[s, c]$, where the catalytic tape may take some variable length c different from 2^s (see, e.g., [1, 26]), 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 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 without loss of generality 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 without loss of generality 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

[7] While all published works on the subject of randomized catalytic space [10, 11, 24, 26] put C before e.g. BP in **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.

[8] 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.

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.

Theorem 2.7. *For every catalytic machine M deciding a language L using workspace $s := s(n) \geq \log n$ with configuration graph $\mathcal{G}$, there exist catalytic subroutines* DetWalk, 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 the tuple* (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 the full version; it is adapted from a previous proof due to Dulek [12] (see also Datta, Gupta, Jain, Sharma, and Tewari [10]). 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. 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.[9]
- 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 $\Gamma_b(\mathbf{w}, v) := (\mathbf{w}_b, v_b)$.

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.,

$$\sigma_0 := \sigma, \sigma_1 := \Gamma_{r_1}(\sigma_0), \ldots, \sigma_\ell := \Gamma_{r_\ell}(\sigma_{\ell-1}). \tag{5}$$

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 (5).

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* RandWalk, 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, the subroutine* $\mathsf{RandWalk}^{\mathbf{w}}(x, v, r)$ *only requires one-way access to* r*.*
- *If there is a catalytic tape configuration* $\mathbf{w}$ *such that we have* $\mathcal{G}[\mathsf{start}(\mathbf{w}), r] = (\mathbf{w}', v')$*,* $\mathsf{RandRev}^{\mathbf{w}'}(x, v', r)$ *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 the full version.

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 $r, r' \in \{0,1\}^*$.

[9] 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.

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 the routine* $\mathsf{EQ}^{\mathbf{w}}(x, r, r')$ *accepts if and only if*

$$\mathcal{G}_x[\mathrm{start}(\mathbf{w}), r] = \mathcal{G}_x[\mathrm{start}(\mathbf{w}), r'].$$

Proof. Let

$$(\mathbf{w}_r, v_r) := \mathcal{G}[\mathrm{start}(\mathbf{w}), r] \text{ and } (\mathbf{w}_{r'}, v_{r'}) := \mathcal{G}[\mathrm{start}(\mathbf{w}), r'];$$

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 $\ell := |(\mathbf{w}_r, v_r)|(= |(\mathbf{w}_{r'}, v_{r'})|)$. For each $i \in [\ell]$, 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 $(\mathbf{w}_r, v_r) := \mathcal{G}[\mathrm{start}(\mathbf{w}), r]$. 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 $(\mathbf{w}_{r'}, v_{r'}) := \mathcal{G}[\mathrm{start}(\mathbf{w}), r']$. 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 $b \neq b'$.

The algorithm accepts if it passes the test for any $i \in [\ell]$ (see the full version for the pseudocode).

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: $\mathbf{CLP} = \mathbf{CL} \cap \mathbf{P}$ and $\mathbf{CL} \subseteq \mathbf{CZPLP}$. Finally, we show that the reduction implies a "certified derandomization" for **CL**.

3.1 CL Reduces to Lossy Code

Our main compression algorithm for $\mathbf{CSPACE}[s]$ is as follows:

Theorem 3.1. *Let* $s := s(n) \geq \log n$ *be space constructible. For every* $L \in \mathbf{CSPACE}[s]$*, 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 DetComp 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 $(k+1)$-th 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 $\mathsf{DetDecomp}^{\mathbf{w}'_k \circ 0^k}(1^k, x)$ so that by the invariant we will reset the catalytic tape back 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

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

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 the full version 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 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 our algorithms 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 the full version 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 [20].

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 DetComp′ and DetDecomp′ such that:

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

Our reduction from L to 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 $\mathrm{DetComp}'(x, \cdot) : \{0,1\}^S \to \{0,1\}^{S-1}$ and $\mathrm{DetDecomp}'(x, \cdot) : \{0,1\}^{S-1} \to \{0,1\}^S$.

Let $\mathbf{w}^*$ be a solution to the LossyCode instance

$$(\mathrm{DetComp}'(x, \cdot), \mathrm{DetDecomp}'(x, \cdot)),$$

i.e.,

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

By the correctness of DetComp and DetDecomp (see Theorem 3.1), we know that $\mathrm{DetComp}^{\mathbf{w}^*}(1, x)$ outputs $L(x)$. We can then simulate $\mathrm{DetComp}^{\mathbf{w}^*}(1, x)$ 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 DetComp of Theorem 3.1 for M with input $(x, 1^{2^{(k+1)s} \cdot t^{k+1}})$.

- If 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 returns $L(x)$.
- 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 DetDecomp with input $(x, 1^{2^{(k+1)s} \cdot t^{k+1}})$. By the correctness of DetComp and DetDecomp, the catalytic tape will be reset, and we can decide whether $x \in L$.

Recall that both DetComp and 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 $L \in \mathbf{CTISP}\left[2^{O(s)} \cdot t^{O(1)}, s + \log t\right]$. In particular, $\mathbf{CLP} = \mathbf{CL} \cap \mathbf{P}$ holds if we take $t(n) = n$ and $s(n) = \log n$. □

From Theorem 3.2 we immediately obtain multiple corollaries.

Corollary 1.3.

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

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

Corollary 1.4.

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

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

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 1.7. Our only change will be to no longer assume that we are dealing with a language in **P**, but rather in **ZPP**. Buhrman, Cleve, Koucký, Loff, and Speelman [2] showed that in fact such a containment holds without any further assumptions:

Theorem 3.3 ([2]). *For all space constructible functions $s := s(n) \geq \log n$,*

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

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

This is sufficient to prove the forward direction of Theorem 1.7.

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

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

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

Proof. Let $L \in \mathbf{CSPACE}[s]$ and M be a catalytic $O(s)$-space machine that decides L. Thus by Theorem 3.3 we know that $L \in \mathbf{ZPTIME}\left[2^{O(s)}\right]$. 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 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 outputs $L(x)$.
- 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 probability at most $1/3$. This concludes the correctness of the algorithm.

It is clear that 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 $L \in \mathbf{CZPTISP}\left[2^{O(s)}, s\right]$. In particular, this implies $\mathbf{CL} \subseteq \mathbf{CZPLP}$ if we take $s(n) = \log n$. □

Acknowledgements

Jiatu Li and Edward Pyne thank Oliver Korten and Roei Tell for bringing the link 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

[1] Sagar Bisoyi, Krishnamoorthy Dinesh, and Jayalal Sarma. 2022. On pure space vs catalytic space. *Theor. Comput. Sci.* 921 (2022), 112–126. doi:10.1016/J.TCS.2022.04.005

[2] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. 2014. Computing with a full memory: catalytic space. In *Proc. 46 Annual ACM Symposium on Theory of Computing (STOC)*. 857–866. doi:10.1145/2591796.2591874

[3] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. 2018. Catalytic Space: Non-determinism and Hierarchy. *Theory Comput. Syst.* (2018). doi:10.1007/S00224-017-9784-7

[4] Lijie Chen, Roei Tell, and Ryan Williams. 2023. Derandomization vs Refutation: A Unified Framework for Characterizing Derandomization. In *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. doi:10.1109/FOCS57990.2023.00062 To appear.

[5] James Cook and Ian Mertz. 2020. Catalytic approaches to the tree evaluation problem. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*. ACM, 752–760. doi:10.1145/3357713.3384316

[6] James Cook and Ian Mertz. 2021. Encodings and the Tree Evaluation Problem. *Electron. Colloquium Comput. Complex.* TR21-054 (2021). ECCC:TR21-054 https://eccc.weizmann.ac.il/report/2021/054

[7] James Cook and Ian Mertz. 2022. Trading Time and Space in Catalytic Branching Programs. In *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA (LIPIcs, Vol. 234)*, Shachar Lovett (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:21. doi:10.4230/LIPICS.CCC.2022.8

[8] James Cook and Ian Mertz. 2024. Tree Evaluation is in Space O(log n · log log n). In *Proccedings of the 56nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2024*. ACM, 1268–1278. doi:10.1145/3618260.3649664

[9] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. 2012. Pebbles and Branching Programs for Tree Evaluation. *ACM Trans. Comput. Theory* 3, 2 (2012), 4:1–4:43. doi:10.1145/2077336.2077337

[10] Samir Datta, Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. 2020. Randomized and Symmetric Catalytic Computation. In *Computer Science - Theory and Applications - 15th International Computer Science Symposium in Russia, CSR 2020*. doi:10.1007/978-3-030-50026-9_15

[11] Dean Doron, Edward Pyne, and Roei Tell. 2024. 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)*. doi:10.1145/3618260.3649772

[12] Yfke Dulek. 2015. Catalytic space: on reversibility and multiple-access randomness. private communication.

[13] Jeff Edmonds, Venkatesh Medabalimi, and Toniann Pitassi. 2018. Hardness of Function Composition for Semantic Read once Branching Programs. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA (LIPIcs, Vol. 102)*, Rocco A. Servedio (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:22. doi:10.4230/LIPICS.CCC.2018.15

[14] Marten Folkertsma, Ian Mertz, Florian Speelman, and Quinten Tupker. 2025. Fully Characterizing Lossy Catalytic Computation. In *Proc. 16 Conference on Innovations in Theoretical Computer Science (ITCS) (LIPIcs, Vol. 325)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 50:1–50:13. doi:10.4230/LIPICS.ITCS.2025.50

[15] Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. 2019. Unambiguous Catalytic Computation. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019 (LIPIcs, Vol. 150)*. 16:1–16:13. doi:10.4230/LIPICS.FSTTCS.2019.16

[16] William M. Hoza. 2021. Better Pseudodistributions and Derandomization for Space-Bounded Computation. In *Proceedings of the 25th International Conference on Randomization and Computation (RANDOM)*. 28:1–28:23. doi:10.4230/LIPICS.APPROX/RANDOM.2021.28

[17] Rahul Ilango, Jiatu Li, and R. Ryan Williams. [2023] ©2023. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*. 1076–1089. doi:10.1145/3564246.3585187

[18] Kazuo Iwama and Atsuki Nagao. 2019. Read-Once Branching Programs for Tree Evaluation Problems. *ACM Trans. Comput. Theory* 11, 1 (2019), 5:1–5:12. doi:10.1145/3282433

[19] Oliver Korten. 2021. The Hardest Explicit Construction. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 433–444. doi:10.1109/FOCS52979.2021.00051

[20] Oliver Korten. 2022. Derandomization from time-space tradeoffs. In *Proc. 37th Annual IEEE Conference on Computational Complexity (CCC)*. doi:10.4230/LIPICS.CCC.2022.37

[21] Michal Koucký. 2016. Catalytic computation. *Bull. EATCS* 118 (2016). http://eatcs.org/beatcs/index.php/beatcs/article/view/400

[22] Jiatu Li, Edward Pyne, and Roei Tell. 2024. Distinguishing, Predicting, and Certifying: On the Long Reach of Partial Notions of Pseudorandomness. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 1–13. doi:10.1109/FOCS61266.2024.00095

[23] David Liu. 2013. Pebbling Arguments for Tree Evaluation. *CoRR* abs/1311.0293 (2013). arXiv:1311.0293 http://arxiv.org/abs/1311.0293

[24] Ian Mertz. 2023. Reusing Space: Techniques and Open Problems. *Bulletin of EATCS* 141, 3 (2023). http://eatcs.org/beatcs/index.php/beatcs/article/view/780

[25] Noam Nisan and Avi Wigderson. 1994. Hardness vs. randomness. *Journal of Computer and System Sciences* 49, 2 (1994), 149–167. doi:10.1016/S0022-0000(05)80043-1

[26] Edward Pyne. 2024. Derandomizing Logspace with a Small Shared Hard Drive. In *39th Computational Complexity Conference, CCC 2024 (LIPIcs, Vol. 300)*. 4:1–4:20. doi:10.4230/LIPICS.CCC.2024.4

[27] Edward Pyne, Ran Raz, and Wei Zhan. 2023. Certified Hardness vs. Randomness for Log-Space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. doi:10.1109/FOCS57990.2023.00061

[28] Michael E. Saks and Shiyu Zhou. 1999. $\mathbf{BP_HSPACE}[S] \subseteq \mathbf{DSPACE}[S^{3/2}]$. *Journal of Computer and System Sciences* 58, 2 (1999), 376–403. doi:10.1006/JCSS.1998.1616

[29] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *Proc. 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 162–167. doi:10.1109/SFCS.1986.25

Received 2024-11-04; accepted 2025-02-01