Electronic Colloquium on Computational Complexity, Report No. 16 (2025)

Another way to show BPL ⊆ CL and BPL ⊆ P

James Cook

February 9, 2025

We present a new technique for using catalytic space to simulate space-bounded randomized algorithms. Allocate one bit on the catalytic tape for each configuration of a randomized machine. Simulate the machine several times. Each time it requests a random bit, use the bit from the catalytic tape corresponding to its current configuration, and then flip the bit on the tape. Because the bit is flipped each time, every configuration of the randomized machine will receive a 0 in the same number of simulations as it receives a 1, up to a difference of 1. It follows that the simulation visits each configuration approximately the same number of times as it would if it used true randomness. Finally, the catalytic tape can be restored by re-running the simulations with a small change that reverses their effect. This yields that BPL ⊆ CL and BPL ⊆ P, both of which were already known.

In fact, the technique has so far given no new results. However, we believe it merits further study, because it is simple, and because it is different from all previous catalytic techniques, which, broadly speaking, fall into two categories: either attempt to compress the catalytic tape (deriving something useful when it is incompressible) or treat the tape as registers for a predetermined sequence of mathematical operations. Our technique is different from both, suggesting an opportunity for finding new catalytic algorithms, or for combining techniques to get stronger results.

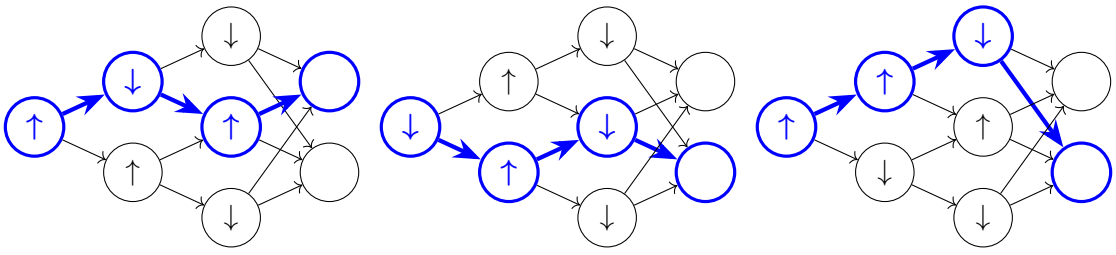

Figure 1: A simulation of a randomized computation, using one bit of catalytic space (represented as ↑, ↓) per configuration to alternate between random choices.

ISSN 1433-8092

1 Introduction

1.1 Catalytic space

In the catalytic space model, an algorithm has two tapes to use as memory: a smaller ordinary tape, and a larger "catalytic" tape which starts out filled with arbitrary data. The algorithm may freely write to and read from both tapes, but when it finishes, the catalytic tape's original content must be restored. Often the working tape has size $O(\log n)$ and the catalytic tape has size $\text{poly}(n)$, defining the decision class CL. Buhrman, Cleve, Koucký, Loff and Speelman [1] initiated the study of this model with their surprising result that $\mathsf{TC}^1 \subseteq \mathsf{CL}$, from which it follows for example that NL and BPL are in CL.

Broadly speaking, two ways to take advantage of catalytic space have been explored previously: compression-based techniques are able to make use of incompressible catalytic tapes (as randomness, for example); and algebra-based techniques treat the tape as an array of registers which they use to execute a "straight-line program", a predetermined sequence of mathematical operations. The original $\mathsf{TC}^1 \subseteq \mathsf{CL}$ result is an example of the algebraic approach. The compression-based approach began with a "compress-or-random" argument that $\mathsf{BPL} \subseteq \mathsf{CL}$ (Mertz [4] gives a sketch of it). Doron, Pyne and Tell [3] showed this can be combined with Nisan and Wigderson's pseudorandom generator [6] to give a new proof that $\mathsf{BPL} \subseteq \mathsf{CL}$; Pyne [7] showed that much less than $\text{poly}(n)$ catalytic space is required; and Cook, Li, Mertz and Pyne [2] used it to derandomize the class CL itself.

1.2 Our technique

We present a third technique distinct from the compression-based and algebraic approaches. It can be summarized as follows. For a full description, see Section 3.

Take a randomized Turing machine M which we want to simulate and an input $x \in \{0,1\}^n$. We will distinguish between the case that M accepts with probability $\geq \frac{2}{3}$, and that it accepts with probability $\leq \frac{1}{3}$.

Let N be the number of possible configurations of M — for example, if M is limited to $O(\log n)$ space, then $N = \text{poly}(n)$. Allocate one bit of the catalytic tape for each possible configuration of M, for a total of N bits of catalytic space.

Run $6N + 1$ simulations of M, using the catalytic tape as a source of randomness in the following way. Each time M requests a random bit, move the catalytic tape head to the index corresponding to M's current configuration. Supply M with whatever bit is currently written there, and then overwrite it with the opposite bit. In this way, as M reaches that same configuration over the course of the $6N + 1$ simulations, an alternating sequence of random bits $0, 1, 0, 1, \ldots$ or $1, 0, 1, 0, \ldots$ is supplied. Figure 1 shows an example of this process, with the configurations of M laid out as nodes in a graph, and the bits of the catalytic tape drawn directly on the corresponding nodes as $\uparrow$ or $\downarrow$.

Now we can argue that the number of visits to each configuration approximately equals the expected number of visits if we had supplied M with truly random bits. A common

way to prove this kind of result is with a "local consistency check" — see for example Nisan's Lemma 2.6 [5]. The general idea is that if every configuration of the machine is given a pseudorandom bit of 0 approximately as many times as it is given a 1, then each configuration is visited approximately the right number of times. Our version of this argument appears in the proof of Lemma 3. By a careful analysis we show that the number of visits to each state is within N of its expected value, regardless of the number of simulations. Therefore, after $6N + 1$ simulations, if M's accept probability was $\geq \frac{2}{3}$, more than $3N$ of our simulations will accept, and if it was $\leq \frac{1}{3}$, at most $3N$ simulations will accept. So, a majority vote suffices.

Finally, we must restore the catalytic tape. This is done by running the "reverse" of our $6N + 1$ simulations. We do not literally run the machine M backward; rather, we run it forward as before, but slightly change the way we supply its random bits, so that each "reverse" simulation exactly undoes the effect of one normal simulation. For details, see Algorithm 2 and the proof of Corollary 4.

This proves the following theorem. It is restated more precisely in Section 3.

Theorem 1 (informal)**.** *Any language in* $\mathsf{BPSPACE}[s(n)]$ *can be computed catalytically in time* $2^{O(s(n))}$*, working space* $O(s(n))$*, and one bit of catalytic space for every possible configuration of the* $\mathsf{BPSPACE}[s(n)]$ *machine.*

Corollary 1. $\mathsf{BPL} \subseteq \mathsf{CL}$.

Corollary 2. $\mathsf{BPL} \subseteq \mathsf{P}$.

1.3 Summary of Contributions

- We introduce a simple new technique which can be used in catalytic algorithms.
- We use it to re-prove $\mathsf{BPL} \subseteq \mathsf{CL}$ and $\mathsf{BPL} \subseteq \mathsf{P}$, and more generally that languages in $\mathsf{BPSPACE}[s(n)]$ can be decided in space $O(s(n))$, catalytic space $2^{O(s(n))}$, and time $2^{O(s(n))}$.
- Since the technique is simple, we are able to analyze its performance somewhat precisely. If N is the number of confurations of the randomized machine being simulated, our algorithm uses exactly N bits of catalytic space, and requires no more than $6N + 1$ simulations.

2 Preliminaries

We denote $\mathbf{N} = \{0, 1, 2, \dots\}$ the natural numbers, and for $n \in \mathbf{N}$ we denote $[n] = \{0, 1, \dots, n-1\}$ the natural numbers less than n.

0^s denotes an s-bit string of zeroes.

For $a, b \in \{0, 1\}$, define $a \oplus b = (a + b) \bmod 2$ and $\neg a = 1 - a$.

Definition 1. A function f is *space constructible* if $f(n)$ can be computed in space $O(f(n))$.

2.1 Randomized and Catalytic Turing Machines

Our goal is to simulate randomized Turing machines with catalytic ones, so we define both here. We assume familiarity with Turing machines generally, but specify details here so that we may simulate randomized Turing machines and count their configurations precisely. In particular:

- All tapes (input, working, catalytic) use the alphabet $\{0, 1\}$.
- A randomized Turing machine consumes one bit of randomness at every step of its computation, which is made directly available to its transition function.

Definition 2 (Randomized Turing machine, BPSPACE, BPL)**.** A randomized Turing machine is a tuple $(Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$, consisting of: a number $Q \in \mathbf{N}$ of states (we may consider the states themselves to be $[Q] = \{0, 1, \ldots, Q-1\}$); distinct states $q_0, q_{\text{accept}}, q_{\text{reject}} \in [Q]$; and a *transition function* $\delta : [Q] \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \{0,1\}^3 \to [Q] \times \{L, R\}^2$. The transition function receives the current state, the symbols on the input and work tapes, and a random bit, and produces a new state and directions (**L**eft or **R**ight) to move the input and work tape heads.

For a function $s : \mathbf{N} \to \mathbf{N}$, a language L is in $\mathsf{BPSPACE}[s(n)]$ if there is a randomized Turing machine which on input $x \in \{0,1\}^n$ always halts, never moves its work tape head beyond the first $s(n)$ cells, and correctly decides $x \in L$ with probability at least $2/3$; we say *M decides L with bounded error probability.*

A language is in BPL if it is in $\mathsf{BPSPACE}[s(n)]$ for some $s(n) = O(\log n)$.

For some discussion of the choices made in the definition of BPSPACE, see Nisan [5].

Definition 3 (Configuration of a space-bounded Turing machine)**.** A *configuration* of a (randomized) Turing machine with state count Q, input size $n \in \mathbf{N}$ and space limit $s \in \mathbf{N}$ is a tuple $g = (\text{state}(g), \text{input\_head}(g), \text{work\_head}(g), \text{work\_tape}(g)) \in [Q] \times [n] \times [s] \times \{0,1\}^s$: the state, the positions of the input and work tape heads, and the contents of the work tape.

Note that there are $Qns2^s$ possible configurations.

Our simulation algorithm is *catalytic*, meaning it makes use of a tape which may be decsribed as *already full*: the tape begins with some arbitrary content which must be restored at the end of the computation.

Definition 4 (Catalytic Turing machine, CL, CTISP)**.** A catalytic Turing machine $(Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$ is described by the same parts as a randomized Turing machine (Definition 2), except that the transition function is $\delta : [Q] \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \{0,1\}^3 \to [Q] \times \{L, R\}^3$; it receives the current state and the symbols on three tapes (input, working and catalytic) and moves all three tape heads.

We say M *decides* a language $L \subseteq \{0,1\}^*$ in working space $s(n)$ and catalytic space $c(n)$ if for every $x \in \{0,1\}^n$ and every $\tau \in \{0,1\}^{c(n)}$, if the catalytic tape is initialized to τ, then M correctly decides whether $x \in L$, restores τ to the catalytic tape, and never

moves its work and catalytic tape heads beyond the first $s(n)$ or $c(n)$ cells of its work and catalytic tapes, respectively.

If $s(n) = O(\log n)$ and $c(n) = \text{poly}(n)$, we say $L \in \mathsf{CL}$. If M always runs for at most $t(n)$ steps and $c(n) = 2^{O(s(n))}$, we say $L \in \mathsf{CTISP}[t(n), s(n)]$.

Space-bounded catalytic Turing machines were proposed by Buhrman, Cleve, Koucký, Loff and Speelman [1]. The class $\mathsf{CTISP}[t(n), s(n)]$ which limites both space and time was introduced by Cook, Li, Mertz and Pyne [2].

3 Simulating BPSPACE

Theorem 1. $\mathsf{BPSPACE}[s(n)] \subseteq \mathsf{CTISP}[2^{O(s(n))}, O(s(n))]$ *for every space-constructible function*[1] $s(n) \geq \log n$.

Specifically, if M is a randomized Turing machine deciding a language L in space $s(n)$ with bounded error probability, then there is a catalytic Turing machine M' which decides L and uses $O(s(n))$ working space, uses exactly one bit of catalytic space for each possible configuration of M (amounting to $Qn \cdot s(n) \cdot 2^{s(n)}$ bits, where Q is the size of M's state machine), and has runtime polynomial in the runtime of M.

The catalytic machine M' is described by SIMULATE (Algorithm 1) and its subroutine SIMULATE_ONCE (Algorithm 2). To prove Theorem 1, it suffices to prove two things for every $x \in \{0,1\}^n$: in Section 3.1, we show that SIMULATE$(M, x, s(n))$ gives the correct output, and in Section 3.2, we show that it restores the catalytic tape at the end of the computation. Section 3.3 ties the proof together.

3.1 The output is correct

To evaluate how well SIMULATE simulates M, we will compare the number of times SIMULATE_ONCE visits each configuration to the probability a true random run of M would reach that configuration. We will argue that the next random bit $r \in \{0,1\}$ splits its time fairly between being 0 or 1 when the STEP function is evaluated on line 13, from which it will follow that our simulation is sufficiently accurate.

Throughout this section, fix a randomized machine M and input $x \in \{0,1\}^n$. Let $N = Qn \cdot s(n) \cdot 2^{s(n)}$ be the number of configurations of M, and fix an initial content of the catalytic tape. Let $K = 6Qn \cdot s(n) \cdot 2^{s(n)} + 1$ as in SIMULATE.

The following definition establishes some notation to help reason about the accuracy of the simulation. It is illustrated in Figure 2.

Definition 5 (visit probability p_g, visit count v_g, error e_g, transition count v_g^r, transition error e_g^r)**.** For a configuration g, let p_g be the probability that the randomized machine M reaches configuration g during its computation on input x.

[1] If $s(n) = o(\log n)$, the result still holds, except the working space becomes $O(\log n)$ and the catalytic space and runtime become $\text{poly}(n)$.

Algorithm 1: SIMULATE(M, x, s). Parameters: randomized Turing machine $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$, input $x \in \{0,1\}^n$, space limit $s \in \mathbf{N}$. ($s = s(n)$ is length of work tape M is permitted to use.)

```
1  Let K = 6Qns2^s + 1.
2  n_accept ← 0
   /* Forward phase:  estimate accept probability                    */
3  for i ← 1 to K do
4      n_accept ← n_accept + SIMULATE_ONCE(M, x, s, forward)
5  end
   /* Reverse phase:  reset the catalytic tape                       */
6  for i ← 1 to K do
7      SIMULATE_ONCE(M, x, s, reverse)
8  end
9  if n_accept ≥ (1/2)K then
10     accept
11 else
12     reject
13 end
```

Let v_g be the number of times SIMULATE_ONCE visits configuration g during the forward phase of SIMULATE (lines 3–5). (A "visit" to the configuration stored in the variable g occurs whenever SIMULATE_ONCE evaluates the while loop condition on line 4.)

Define the *error* $e_g = |v_g - Kp_g|$.

For $r \in \{0,1\}$, let v_g^r be the number of those visits for which the variable r had the given value on line 13 of SIMULATE_ONCE (so $v_g = v_g^0 + v_g^1$ unless state$(g) \in \{q_{\text{accept}}, q_{\text{reject}}\}$).

Define the *transition error* $e_g^r = |v_g^r - \frac{K}{2}p_g|$.

Lemma 1. *For every configuration g and $r \in \{0,1\}$, $e_g^r \le \frac{1}{2} + \frac{1}{2}e_g$.*

Proof. The values of the bit r alternate on successive visits to the configuration g, since the corresponding bit of the catalytic tape is flipped (SIMULATE_ONCE line 8) each time. Therefore $|v_g^0 - v_g^1| \le 1$. Since $v_g = v_g^0 + v_g^1$ it follows that $|v_g^r - \frac{1}{2}v_g| \le \frac{1}{2}$, so

$$e_g^r = |v_g^r - \tfrac{K}{2}p_g| \le |v_g^r - \tfrac{1}{2}v_g| + |\tfrac{1}{2}v_g - \tfrac{K}{2}p_g| \le \tfrac{1}{2} + \tfrac{1}{2}e_g \qquad \square$$

Lemma 2. *For a configuration g_*, let $(g_1, r_1), \ldots, (g_m, r_m)$ be all of the possible (configuration, next random bit) pairs which cause g_* to be the next configuration reached. Then $e_{g_*} \le \sum_{i=1}^m e_{g_i}^{r_i}$.*

Proof. Using the facts that $v_{g_*} = \sum_{i=1}^m v_{g_i}^{r_i}$ and $p_{g_*} = \sum_{i=1}^m \frac{1}{2}p_{g_i}$, we have:

$$e_{g_*} = |v_{g_*} - Kp_{g_*}| = \left|\sum_{i=1}^m (v_{g_i}^{r_i} - \tfrac{K}{2}p_{g_i})\right| \le \sum_{i=1}^m |v_{g_i}^{r_i} - \tfrac{K}{2}p_{g_i}| = \sum_{i=1}^m e_{g_i}^{r_i} \qquad \square$$

Algorithm 2: SIMULATE_ONCE(M, x, s, m). Parameters: randomized Turing machine $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$, input $x \in \{0,1\}^n$, space limit $s \in \mathbf{N}$, mode $m \in \{\textbf{forward}, \textbf{reverse}\}$. Returns 0 or 1. (This algorithm modifies the catalytic tape, so it is not a catalytic algorithm. However, the changes are reversible, so it can be used as a subroutine in a catalytic algorithm.)

Notation: $w[i]$ is the i-th cell of the catalytic tape.
Notation: STEP(M, x, g, r) is the new configuration of M after one step of computation starting in configuration g on input x, with randomness $r \in \{0,1\}$.
$g \leftarrow (q_0, 0, 0, 0^s)$
while state$(g) \notin \{q_{\text{accept}}, q_{\text{reject}}\}$ **do**
    `/* Encode the current configuration` g `as an integer` $i \in [Qns2^s]$`. */`
    $i \leftarrow \text{state}(g) \cdot ns2^s + \text{input\_head}(g) \cdot s2^s + \text{work\_head}(g) \cdot 2^s + \text{work\_tape}(g)$
    `/* Get the next random bit from cell` i `of the catalytic tape and flip the bit on the tape. */`
    if $m =$ **forward then**
        $r \leftarrow w[i]$
        $w[i] \leftarrow \neg w[i]$
    else `/*` $m =$ **reverse** `*/`
        $w[i] \leftarrow \neg w[i]$
        $r \leftarrow w[i]$
    end
    $g \leftarrow \text{STEP}(M, x, g, r)$
end
return 1 *if* state$(g) = q_{\text{accept}}$ *or* 0 *if* state$(g) = q_{\text{reject}}$

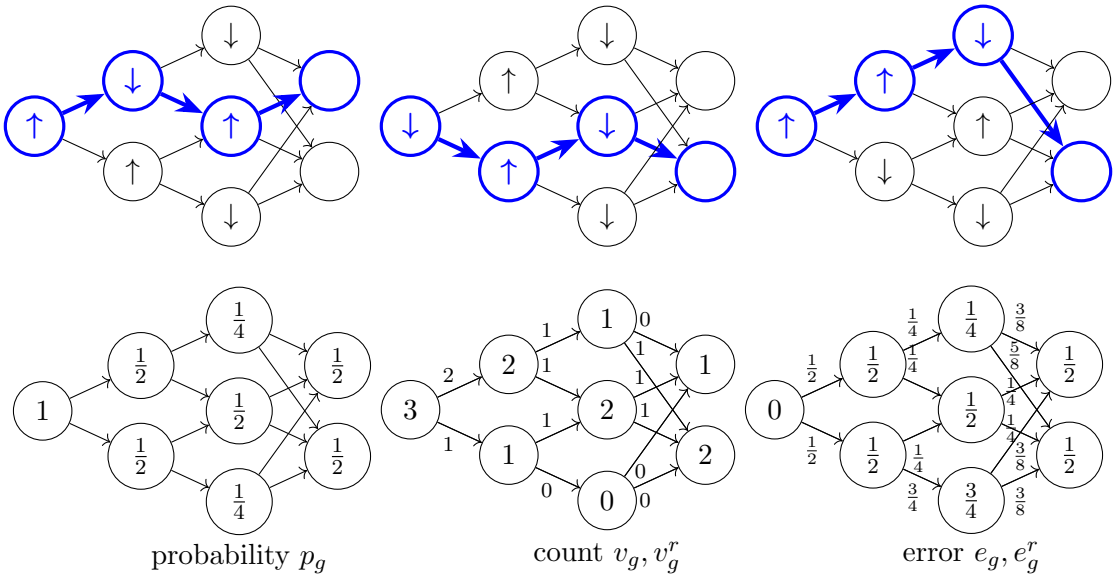

Figure 2: An illustration of Definition 5. Top row: an example simulation, copied from Figure 1 for reference. Bottom left: visit probabilities p_g. Bottom centre: visit counts v_g and transition counts v_g^r specific to this simulation. Bottom right: errors specific to this simulation, taking $K = 3$: $e_g = |v_g - 3p_g|, e_g^r = |v_g^r - \frac{3}{2}p_g|$.

Lemma 3. *Let p_{accept} be the probability M accepts on input x. When* SIMULATE *reaches line 9, $|n_{\text{accept}} - Kp_{\text{accept}}| \leq N$.*

Proof. We prove this from Lemmas 1 and 2 using an induction argument.

Let $G = (V, E)$ be the *configuration graph* of M on input x, defined as follows. The nodes $g \in V$ are the configurations of M. If state$(g) \in \{q_{\text{accept}}, q_{\text{reject}}\}$, then g is a sink. Otherwise, it has two outgoing edges in E corresponding to the configurations reached if the next random bit is 0 or 1. (Recall that we fixed the input x throughout this section, so only the random bits remain undetermined in the course of M's computation.)

Let $G' = (V', E')$ be the subgraph of nodes in G that are reachable from the starting configuration g_0, and let $N' \leq N$ be the number of such nodes.

Note that G' has no cycles, since by assumption M always halts.

(At this point, it would be convenient if G' were arranged in layers, so that each node in layer i had edges only to layer $i + 1$. We could convert M to such a layered machine by adding a clock, but this would increase the number of possible configurations. Instead, we go to slightly more effort to avoid this inefficient conversion.)

Let $H \subseteq V'$ be the sink nodes of G': that is, configurations $g \in V'$ where state$(g) \in \{q_{\text{accept}}, q_{\text{reject}}\}$.

Let $g_0, g_1, \ldots, g_{N'}$ be a topological order on the nodes of G' such that the sink nodes appear at the end of the order. That is, for any edge $g \to g'$ in G', g appears before g' in the order, and $H = \{g_{N'-|H|+1}, \ldots, g_{N'}\}$.

For $i \in \{0, 1, \ldots, N' - |H|\}$, consider the cut of G' with nodes $g_0, \ldots, g_i$ on the left and

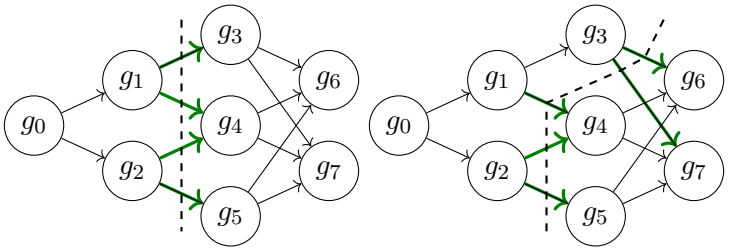

Figure 3: Two steps in the induction argument in the proof of Lemma 3. On the left, $i = 2$ and on the right, $i = 3$. In each case, F_i consists of the edges that cross the dashed line, and σ_i is the sum of the errors on those edges.

$g_{i+1}, \ldots, g_{N'}$ on the right. We are interested in the total error over all the transitions which cross each such cut.

Let $F_i \subseteq V' \times \{0, 1\}$ be the set of transitions which cross the cut. That is, a pair (g, r) is in F_i if g is on the left of the cut and reading random bit r moves the computation to a configuration on the right of the cut. See Figure 3.

Let $\sigma_i = \sum_{(g,r)\in F_i} e_g^r$, recalling from Definition 5 that $e_g^r = |v_g^r - \frac{K}{2} p_g|$.

By induction, we can show that $\sigma_i \leq i + 1$.

Base case: $\sigma_0 = e_{g_0}^0 + e_{g_0}^1$. We know $v_{g_0} = 1$ and $p_{g_0} = 1$, so $e_{g_0} = 0$ and so by Lemma 1, $\sigma_0 \leq 1$.

Induction step: Fix $i \in [N' - 1]$ and assume $\sigma_i \leq i + 1$. Let $(g_{j_1}, r_1), \ldots, (g_{j_m}, r_j)$ be the transitions leading to configuration g_i. Those are the transitions that contribute to σ_i but not σ_{i+1}, so we have $\sigma_{i+1} = \sigma_i + e_{g_i}^0 + e_{g_i}^1 - \sum_{k=1}^m e_{g_{j_k}}^{r_k}$. Applying Lemmas 1 and 2, we have

$$\sigma_{i+1} \leq \sigma_i + 1 + e_{g_i} - \sum_{k=1}^{m} e_{g_{j_k}}^{r_k} \leq \sigma_i + 1 \leq i + 2$$

completing the induction.

In particular, $\sigma_{N'-|H|} \leq N' - |H| + 1$. The corresponding set of transitions $F_{N'-|H|}$ are exactly the transitions leading to (reachable) halting configurations of M. Therefore, by Lemma 2, the total error over all halting configurations of M is at most

$$\sum_{g\in H} e_g \leq \sum_{(g',r)\in F_{N'-|H|}} e_{g'}^r = \sigma_{N'-|H|} \leq N' - |H| + 1$$

Let $A \subseteq H$ be the accepting configurations, and let p_{accept} be the probability that machine M accepts. Noting that on line 9 of SIMULATE, n_{accept} equals the number of visits to configurations in A, we have

$$|n_{\text{accept}} - Kp_{\text{accept}}| = \left|\sum_{g\in A} v_g - Kp_g\right| \leq \sum_{g\in A} |v_g - Kp_g| \leq \sum_{g\in H} e_g \leq N' - |H| + 1 \leq N \quad \square$$

Corollary 3. *If M accepts with probability at least $2/3$, then* SIMULATE *accepts, and similarly it rejects if M rejects with probability at least $2/3$.*

Proof. If $p_{\text{accept}} \geq 2/3$, then by Lemma 3, $n_{\text{accept}} \geq \frac{2}{3}K - N > \frac{1}{2}K$, and similarly if $p_{\text{accept}} \leq 1/3$, then $n_{\text{accept}} < \frac{1}{2}K$. □

3.2 The catalytic tape is restored.

SIMULATE restores its catalytic tape when it finishes. To see this, the following is enough:

Lemma 4. *Running* SIMULATE_ONCE$(M, x, s, \textbf{forward})$ *then* SIMULATE_ONCE$(M, x, s,$ **reverse**$)$ *leaves the catalytic tape unchanged.*

Proof. It is enough to show both calls to SIMULATE_ONCE visit the same sequence of configurations g, since then each bit of the catalytic tape is flipped an even number of times. (We say SIMULATE_ONCE "visits" the configuration stored in variable g each time the loop condition on line 4 is evaluated.)

Loosely speaking, this is true because every time SIMULATE_ONCE$(M, x, s, \textbf{reverse})$ decides which random bit r to feed to M, it first (line 10) undoes the change made by SIMULATE_ONCE$(M, x, s, \textbf{forward})$, and so ends up reading the same bit from the catalytic tape.

This argument relies on the fact that *a single run of* SIMULATE_ONCE *never modifies the same cell $w[i]$ of the catalytic tape more than once*, which follows from the fact that M always halts: if a cell could be modified twice, that would imply a cycle in M's configuration graph.

To make this all a bit more precise, let $g_0, \ldots, g_t$ be the configurations visited by SIMULATE_ONCE$(M, x, s, \textbf{forward})$, and $g'_0, \ldots, g'_{t'}$ be the configurations visited by the subsequent call to SIMULATE_ONCE$(M, x, s, \textbf{reverse})$. We show by induction that $g_i = g'_i$ for each i, so that in particular the main loop ends with the same final configuration in each case and so $t = t'$.

To begin with, $g_0 = g'_0 = (q_0, 0, 0, 0^s)$. Now assume $g_i = g'_i$. If $\text{state}(g_i) \in \{q_{\text{accept}}, q_{\text{reject}}\}$, both subroutine calls halt and we are finished. Otherwise, we must show the same bit r is read from $w[i]$ on the catalytic tape both times. Since neither subroutine call made any other changes to $w[i]$, when the second subroutine call flips $w[i]$ on line 10, it exactly cancels out the only change made by the first subroutine call on line 8, and so the same bit r is recovered, and so the same next step is taken: $g_{i+1} = g'_{i+1}$. □

Corollary 4. SIMULATE *leaves its catalytic tape unchanged.*

Proof. By induction on K, we can see that K calls to SIMULATE_ONCE$(M, x, s, \textbf{forward})$ followed by K calls to SIMULATE_ONCE$(M, x, s, \textbf{reverse})$ has no net effect on the tape. For $K = 0$ this is clear. Lemma 4 provides the induction step. That is, $K + 1$ calls to each can be decomposed as (1) K calls to SIMULATE_ONCE$(M, x, s, \textbf{forward})$, then (2) a single call to each, which by Lemma 4 has no net effect, then (3) K calls to SIMULATE_ONCE$(M, x, s, \textbf{reverse})$. Since (2) has no effect, we are left with K calls each. □

3.3 Proof of Theorem 1

On input x, the catalytic machine M' runs SIMULATE$(M, x, s(n))$. (Here, we have used the fact that s is space-constructible.) Corollary 3 shows the output is correct, and Corollary 4 shows M' restores its catalytic tape.

M' uses $O(s(n))$ working space to hold the loop variable i of SIMULATE and the variables g, i, r of SIMULATE_ONCE, and SIMULATE_ONCE requires the catalytic tape to have one bit for every configuration of M. The runtime is the time needed for $O(N) = 2^{O(s(n))}$ simulations of M by SIMULATE_ONCE, each taking time $2^{O(s(n))}$. □

Acknowledgements

The author thanks Ian Mertz for helpful discussions, and Michal Koucký for pointing out that this technique is also a new way to show $\mathsf{BPL} \subseteq \mathsf{P}$ (and not just $\mathsf{BPL} \subseteq \mathsf{CL}$).

References

[1] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 857–866, New York, NY, USA, 2014. Association for Computing Machinery.

[2] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. *Electronic Colloquium on Computational Complexity: ECCC*, 2024.

[3] Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for BPL=L that uses properties of BPL. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 2039–2049, New York, NY, USA, 2024. Association for Computing Machinery.

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

[5] Noam Nisan. On read once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993.

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

[7] Edward Pyne. Derandomizing Logspace with a Small Shared Hard Drive. In Rahul Santhanam, editor, *39th Computational Complexity Conference (CCC 2024)*, volume 300 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Source code for this document

This document was built using LaTeX and a Perl script to generate TikZ diagrams. The source code is available at `https://www.falsifian.org/a/1KwN`.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il