Catalytic space: non-determinism and hierarchy

Harry Buhrman[1], Michal Koucký[2], Bruno Loff[2], and Florian Speelman[3]

1 **CWI and University of Amsterdam** — h.buhrman@cwi.nl
2 **Charles University, Prague** — koucky@iuuk.mff.cuni.cz, bruno.loff@gmail.com
3 **CWI, Amsterdam** — f.speelman@cwi.nl

Abstract

Catalytic computation, defined by Buhrman, Cleve, Koucký, Loff and Speelman (STOC 2014), is a space-bounded computation where in addition to our working memory we have an exponentially larger auxiliary memory which is full; the auxiliary memory may be used throughout the computation, but it must be restored to its initial content by the end of the computation.

Motivated by the surprising power of this model, we set out to study the non-deterministic version of catalytic computation. We establish that non-deterministic catalytic log-space is contained in ZPP, which is the same bound known for its deterministic counterpart, and we prove that non-deterministic catalytic space is closed under complement (under a standard derandomization assumption). Furthermore, we establish hierarchy theorems for non-deterministic and deterministic catalytic computation.

1 Introduction

Buhrman et al. [3] define the notion of *catalytic computation*, a space-bounded model of computation in which the usual Turing machine has, in addition to its work tape, access to a large auxiliary memory which is full. The auxiliary memory can be used during the computation, but its starting contents must be restored by the end of the computation. The space usage that is counted is the amount of work space s used; the auxiliary memory is for free. In a reasonable setting, the auxiliary memory is of size at most 2^s. One can think of the auxiliary memory as a hard disk full of data. The catch with the auxiliary memory is that it may contain arbitrary content, possibly incompressible, which has to be preserved in some way during the computation. It is not obvious whether such auxiliary memory can be useful at all. Buhrman et al. show that, surprisingly, there is a non-trivial way of using the full memory; that it is possible to compute in work space $O(\log n)$ (*catalytic log-space,* CL) functions not known to be computable in the usual logarithmic space (*log-space,* L) without the auxiliary memory. Indeed, all of TC^1, which includes NL and LOGCFL, is contained in CL.

This motivated us to explore further: What other problems can be solved in catalytic log-space? Buhrman et al. show $\mathsf{CL} \subseteq \mathsf{ZPP}$, so CL is unlikely to contain the whole of PSPACE (even though this is the case relative to some oracle). The fact that $\mathsf{NL} \subseteq \mathsf{CL}$ suggests an obvious question: what about non-deterministic catalytic log-space? Could it be that non-deterministic computation equipped with auxiliary tape has the same power as deterministic catalytic computation? Non-deterministic catalytic computation could possibly allow us to identify further problems that can benefit from having full memory. The previous work also raises a host of further question about the catalytic model such as: Is there a space hierarchy? Does some kind of Savitch's theorem hold for catalytic log-space? Is non-deterministic catalytic space closed under complement? *etc.* This paper aims to shed light on some of these questions.

In this paper we show that non-deterministic catalytic space is closed under complement under a widely accepted derandomization assumption. We also establish hierarchy theorems for catalytic computation in the deterministic and non-deterministic settings. For our non-deterministic catalytic log-space we can also establish the same ZPP upper bound that was known for CL. Hence there seems to be a closeness between determinism and non-determinism for catalytic computation. Despite that we are unable to establish an equivalent of Savitch's theorem. This remains an intriguing open problem.

We prove the closure under complement using the inductive counting technique of Immerman and Szelepcsényi [4, 9]. However, we had to overcome several difficulties. One challenge is that we might be faced with an exponential-size graph of reachable configurations. We show how to use a pseudorandom generator to avoid such a situation. Another issue is that for inductive counting we need to be able to remember and reason about different configurations. However, the full description of a configuration is exponentially bigger than our work space, so we cannot possibly store it in full. This is one of the hurdles that prevents us from carrying out Savitch's algorithm for catalytic computation. For the inductive counting we resolve this issue by using fingerprints for various configurations.

Our hierarchy theorems are proven in the setting of computation with advice. The catalytic model is a semantic restriction. It is an easy exercise to show that it is algorithmically undecidable whether a machine will restore the full memory on every input to its original content. For semantic models of computation, like bounded-error randomized computation, the only hierarchy theorems that we know of are in the setting with advice. The reason is that essentially all known hierarchy theorems are proven by diagonalization, which requires the ability to enumerate exactly all machines of a given type. We do not know any such enumeration for catalytic machines so we have to settle for the weaker result. The advice is used only to tell the diagonalizing machine whether it is safe to diagonalize against a particular machine. The hierarchy theorems follow from the work of Kinne and van Melkebeek, and van Melkebeek and Pervyshev [7, 10]. For some space bounds we provide more accurate separations that were not explicitly calculated before.

The layout of the paper is as follows. Section 2 contains some preliminaries. In Section 3 we define non-deterministic catalytic computation, and prove that the corresponding log-space class CNL is contained in ZPP. Section 4 is devoted to proving that CNL is closed under complement, and in Section 5 we show hierarchy theorems for catalytic computation.

2 Preliminaries

We assume the reader is familiar with basic computational complexity; a good reference is [2]. The complexity class L denotes the problems solvable in log-space, while PSPACE is the class of those problems that can be solved using a polynomial amount of space. The class NL contains the problems that can be solved *non-deterministically* in log-space, and LOGCFL is the class of problems that are log-space many-one reducible to context-free languages.

The problems in ZPP (zero-error probabilistic polynomial time) are the ones computable by a probabilistic Turing machine that halts in expected polynomial time, while always outputting the correct answer for any input.

We mention the circuit class TC^1, which is the class of boolean functions computable by circuits of depth $O(\log n)$ by AND gates, OR gates and MAJ gates, all with unbounded fan-in — a MAJ gate outputs 1 if and only if most of its input bits are 1. We use $\mathsf{SIZE}(s)$ to denote the class of problems that can be solved by circuits of size s.

The formal definition of catalytic computation [3] is the following:

▶ **Definition 1.** Let $\mathcal{M}$ be a deterministic Turing machine with four tapes: one input and one output tape, one work-tape, and one *auxiliary tape* (or *aux-tape*).

$\mathcal{M}$ is said to be a *catalytic Turing machine* using workspace $s(n)$ and auxiliary space $s_a(n)$ if for all inputs $x \in \{0,1\}^n$ and auxiliary tape contents $w \in \{0,1\}^{s_a(n)}$, the following three properties hold.

1. **Space bound.** The machine $\mathcal{M}(x, w)$ uses space $s(n)$ on its work tape and space $s_a(n)$ on its auxiliary tape.
2. **Catalytic condition.** $\mathcal{M}(x, w)$ halts with w on its auxiliary tape.
3. **Consistency.** The outcome of the computation $\mathcal{M}(x, w)$ is consistent among all initial aux-tape contents w.[1]

From this we obtain an analogue of the usual space-bounded complexity classes:

▶ **Definition 2.** $\mathsf{CSPACE}(s(n), s_a(n))$ is the class of decision problems solvable by a catalytic Turing machine using workspace $s(n)$ and auxiliary space $s_a(n)$. The notational shorthand $\mathsf{CSPACE}(s(n))$ is defined as $\mathsf{CSPACE}(s(n), 2^{s(n)})$. The class CL is $\mathsf{CSPACE}(O(\log n))$.

In the paper [3], it was shown that, surprisingly, CL can make a non-trivial use of the auxiliary tape. Indeed, the paper shows that $\mathsf{TC}^1 \subseteq \mathsf{CL}$, but it is generally believed that $\mathsf{TC}^1 \not\subseteq \mathsf{L}$.

In this paper we will first prove a space-hierarchy theorem for catalytic computations. This hierarchy theorem holds for catalytic Turing machines with an advice string.

We define advice added to a catalytic computation in the same way as in the recent line of research that proves hierarchies for certain classes of semantic models, see for example [10, 7]. In our case that means that a computation needs to satisfy the catalytic condition and consistency properties on the *correct* advice, and is allowed to (for example) fail to restore the contents of the aux-tape for other values of the advice. This notion of advice is a variation on the one defined by Karp and Lipton [6], who required that the machine model was robust under all possible values of the advice string. Proving the same hierarchy theorem using the Karp–Lipton definition would be harder, and would indeed imply a hierarchy theorem that also holds without any advice [7].

We will prove an analogue of the Immerman–Szelepcsényi theorem. The definition of the non-deterministic version of CL, denoted CNL, will be left for Section 3. Then $\mathsf{CNL} = \mathsf{coCNL}$ will hold under the same assumption as the following standard derandomization result, whose proof is now standard.[2]

▶ **Lemma 3.** *If there exists a constant $\varepsilon > 0$ such that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\varepsilon n})$ then for all constants c there exists a constant c' and a function $G : \{0,1\}^{c' \log n} \to \{0,1\}^n$ such that for any circuit $\mathcal{C}$ of size n^c*

$$\left| \Pr_{r \in \{0,1\}^n}[\mathcal{C}(r) = 1] - \Pr_{s \in \{0,1\}^{c' \log n}}[\mathcal{C}(G(s)) = 1] \right| < \frac{1}{n}$$

and G is computable in space logarithmic in n.

[1] What this means depends on what we are trying to do. For instance, when solving a decision problem, $\mathcal{M}(x, w)$ should either accept for all choices of w — in which case we say $\mathcal{M}$ accepts x — or it rejects for all possible w — $\mathcal{M}$ rejects x.

[2] For instance, the pseudo-random generator of [5] has the right properties. Or see Appendix C of [8] and Theorem 19 of [1].

We will also need a hash family with nice properties. Its proof of existence is a simple exercise, and we include it in the appendix.

▶ **Lemma 4.** *For every n, there exists a family of hash functions $\{h_k\}_{k=1}^{n^3}$, with each h_k a function $\{0,1\}^n \to \{0,1\}^{4\log n}$, such that the following properties hold. Firstly, h_k is computable in space $O(\log n)$ for every k, and secondly, for every set $S \subset \{0,1\}^n$ with $|S| \le n$ there is a hash function in the family that is injective on S.*

Remarks on notation.

For two binary strings x, y of equal length, we use $x \oplus y$ for the bitwise XOR of x and y. The function log always stands for the logarithm of base 2. For simplicity, all Turing machines are assumed to use a binary alphabet — all definitions and proofs would easily generalize to larger alphabet sizes, at the cost of introducing notational clutter.

3 Non-deterministic catalytic computation

The model for catalytic computation is defined in terms of deterministic Turing machines. This gives rise to the question: What would the power of a non-deterministic version of CL be? In this section we extend the definitions of catalytic-space computation to the non-deterministic case, and prove basic results about this model.

▶ **Definition 5.** Let $\mathcal{M}$ be a non-deterministic Turing machine with four tapes: one input and one output tape, one work-tape, and one *auxiliary tape*.

Let $x \in \{0,1\}^n$ be an input, and $w \in \{0,1\}^{s_a(n)}$ be the initial contents of the auxiliary tape. We say that $\mathcal{M}(x,w)$ accepts x if there exists a sequence of nondeterministic choices that makes the machine accept. If for all possible sequences of nondeterministic choices $\mathcal{M}(x,w)$ does not accept, the machine rejects x.

Then $\mathcal{M}$ is said to be a *catalytic non-deterministic Turing machine* using workspace $s(n)$ and auxiliary space $s_a(n)$ if for all inputs, the following three properties hold.

1. **Space bound.** The machine $\mathcal{M}(x,w)$ uses space $s(n)$ on its work tape and space $s_a(n)$ on its auxiliary tape.
2. **Catalytic condition.** $\mathcal{M}(x,w)$ halts with w on its auxiliary tape, irrespective of its nondeterministic choices.
3. **Consistency.** The outcome of the computation $\mathcal{M}(x,w)$ is consistent among all initial aux-tape contents w. This means that for any given input x, $M(x,w)$ should always accept, or always reject, regardless of w; *however*: the specific nondeterministic choices that make $\mathcal{M}(x,w)$ go one way or the other may depend on w.

▶ **Definition 6.** CNSPACE$(s(n), s_a(n))$ is the class of decision problems solvable by a catalytic Turing machine using workspace $s(n)$ and auxiliary space $s_a(n)$, and CNSPACE$(s(n))$ is defined as CNSPACE$(s(n), 2^{s(n)})$. The class CNL is CNSPACE$(O(\log n))$.

We now have an analogue of non-deterministic space-bounded complexity. For convenience, in the appendix we present an equivalent definition of CNL with all the conditions unfolded. We also discuss possible alternatives to the definition above, and why they should be ruled out.

In [3], we proved that CL $\subseteq$ ZPP; we now generalize this to CNL $\subseteq$ ZPP.

▶ **Definition 7.** Define the directed acyclic graph $\mathcal{G}_{\mathcal{M},x,w}$ to be the configuration graph of a catalytic non-deterministic Turing machine $\mathcal{M}$ on input x and auxiliary tape starting contents w. That is, $\mathcal{G}_{\mathcal{M},x,w}$ has a node for every configuration which is reachable by non-deterministic choices when executing $\mathcal{M}(x,w)$.

We will use $|\mathcal{G}_{\mathcal{M},x,w}|$ to denote the number of nodes of the configuration graph.

▶ **Lemma 8.** *Let $\mathcal{M}$ be a non-deterministic catalytic machine using space $c\log n$ and let $c' = 2c+2$. Then for all x*

$$\mathop{\mathbb{E}}_{w\in_R\{0,1\}^{n^c}}\left[|\mathcal{G}_{\mathcal{M},x,w}|\right] \le O(n^{c'})\,.$$

Proof. Notice that, for any given $x \in \{0,1\}^n$, and for different auxiliary tape contents w, w', the set of configurations in $\mathcal{G}_{\mathcal{M},x,w}$ and in $\mathcal{G}_{\mathcal{M},x,w'}$ have to be disjoint. For the sake of contradiction, consider a configuration q that is reachable both by $\mathcal{M}(x,w)$ and by $\mathcal{M}(x,w')$. Then any halting configuration reachable by q will have the wrong contents on its auxiliary tape for either the computation that started with w or with w'.

The number of bits needed to describe a configuration of $\mathcal{M}$, excluding the contents of the input tape, is bounded by

$$c\log n + n^c + \log n^c + \log n + \log\left(c\log n\right) + O(1) \le (2c+2)\log n + n^c + O(1),$$

where we do include the encoding of the location of the tape heads and the internal state of the Turing Machine. Therefore the total number of reachable configurations, counted over all possible starting auxiliary tape contents, is at most

$$\sum_{w\in\{0,1\}^{n^c}} |\mathcal{G}_{\mathcal{M},x,w}| \le 2^{c'\log n + n^c + O(1)} = O(n^{c'})2^{n^c}$$

And thus:

$$\frac{1}{2^{n^c}} \sum_{w\in\{0,1\}^{n^c}} |\mathcal{G}_{\mathcal{M},x,w}| = \mathop{\mathbb{E}}_{w\in_R\{0,1\}^{n^c}}\left[|G_{\mathcal{M},x,w}|\right] \le O(n^{c'})\,. \qquad ◀$$

Now suppose we have CNL machine $\mathcal{M}$, and let $x \in \{0,1\}^n$ be the input string. Consider an algorithm which flips a random string w and searches $\mathcal{G}_{\mathcal{M},x,w}$ for a path from the initial configuration to an accepting configuration. This takes time polynomial in $|G_{\mathcal{M},x,w}|$. By Lemma 8 this graph is polynomial-sized in expectation, and therefore this procedure finishes in expected polynomial time. Thus we obtain:

▶ **Corollary 9.** CNL $\subseteq$ ZPP.

4 An analog of the Immerman–Szelepcsényi theorem

This section is devoted to proving that CNL is closed under complement. Our proof strategy is based on the inductive-counting argument to prove the Immerman–Szelepcsényi theorem. In order for the proof to work for catalytic computation, we will need a couple of new ideas.

Suppose we are given a CNL machine $\mathcal{M}$, and wish to construct a CNL-machine $\mathcal{M}'$ to compute the complement $\mathcal{M}$, via an inductive-counting argument on the configuration graph of $\mathcal{M}$.

First of all, notice that whenever $\mathcal{M}'$ wishes to simulate a run of $\mathcal{M}$, it must necessarily use its own aux-tape to simulate the aux-tape of $\mathcal{M}$, because it is the only read-write tape that is big enough.

Now, for some w (initial contents of the aux-tape), $\mathcal{M}$ may visit exponentially many configurations. Then the inductive counting would be impossible to do with only logarithmic space. So the first idea is to use the pseudo-random generator G of Lemma 3 to avoid such *bad* w, by using the binary XOR $w \oplus G(s)$ for different seeds s. Lemma 10 below explains why this works.

Notice also that we must be careful that $\mathcal{M}'$, when simulating a run of $\mathcal{M}$, can always restore the initial contents of its aux-tape. We can make sure this happens correctly by using the catalytic condition applied to $\mathcal{M}$: whenever we need to restore the initial contents of the aux-tape, it will be enough to run the simulation of $\mathcal{M}$ to an arbitrary halting configuration.

Finally, recall that the inductive-counting argument involves storing and comparing configurations of $\mathcal{M}$; but the configurations of $\mathcal{M}$ include the aux-tape, and are too big for $\mathcal{M}'$ to store on its work tape. So the second idea is to use the family of hash functions of Lemma 4, and do inductive-counting by storing and comparing *the hashes* of configurations instead.

Putting the whole thing together, however, is rather delicate, because our pseudo-random generator will still give us *bad seeds* — meaning $w \oplus G(s)$ might visit too many configurations. Furthermore, even if we pick a good seed, we may still happen to pick a *bad hash function* — meaning a hash function which is not collision-free on the set of reachable configurations. So the algorithm needs to be able to handle bad seeds and bad hash functions.

It will happen that a bad seed may lead us to falsely certifying that the accepting configuration is unreachable, when in fact it is reachable. This is solved simply by trying all seeds and doing a majority vote.

For good seeds, the number of reachable configurations is bounded by $c = n^{O(1)}$, but it may still happen that the hash collisions of a bad hash function will lead us to falsely believe that there are fewer reachable configurations than the actual number (that c is smaller than it actually is) — because configurations with the same hash are only counted once. But fortunately, good hash functions will give us the correct c, and bad hash functions will always give us a smaller value. So we overcome this problem by remembering, for all hash function we try, the largest claimed number of reachable configurations — this will be the true c.

Let us start by showing how to avoid bad w's.

▶ **Lemma 10.** *Assume the derandomization condition of Lemma 3, and let G be as given therein. Let $\mathcal{M}$ be a non-deterministic catalytic Turing machine using workspace $c \log n$. Then, for every input x and aux-tape contents w, at least half of the seeds $s \in \{0,1\}^{O(\log n)}$ will cause the non-deterministic computation $\mathcal{M}(x, G(s) \oplus w)$ to reach at most n^{2c+3} many different configurations.*

Proof. Let $\mathcal{M}$ be a CNL machine using workspace $c \log n$ and auxiliary space n^c. Let $x \in \{0,1\}^n$, $w \in \{0,1\}^{n^c}$ be given.

Let $C_{x,w}$ be a boolean circuit which, on input $r \in \{0,1\}^{n^c}$, does a breadth-first traversal of $\mathcal{G}_{\mathcal{M},x,r\oplus w}$[3], starting on the initial configuration, until either:

i. More than n^{2c+3} nodes have been found, in which case it outputs 0; or
ii. The graph has been fully traversed, in which case it outputs 1.

[3] Recall that $\mathcal{G}_{\mathcal{M},x,r\oplus w}$ is the configuration graph of $\mathcal{M}$, for input x and aux-tape contents given by the bit-wise XOR of r and w.

The size of $C_{x,w}$ can be bounded by a polynomial, say n^d. The circuit $C_{x,w}$ outputs 1 on input r if and only if $|\mathcal{G}_{\mathcal{M},x,r\oplus w}| \le n^{2c+3}$. Therefore, for large enough n, for all $x \in \{0,1\}^n$ and all $w \in \{0,1\}^{n^c}$,

$$\begin{aligned}\Pr_{r\in_R\{0,1\}^{n^c}}[\mathcal{C}_{x,w}(r)=0] &= \Pr_{r\in_R\{0,1\}^{n^c}}\left[|\mathcal{G}_{\mathcal{M},x,r\oplus w}| \ge n^{2c+3}\right] \\ &= \Pr_{r\in_R\{0,1\}^{n^c}}\left[|\mathcal{G}_{\mathcal{M},x,r}| \ge n^{2c+3}\right] \\ &\le \frac{1}{n^{2c+3}} \mathop{\mathbb{E}}_{r\in_R\{0,1\}^{n^c}}\left[|\mathcal{G}_{\mathcal{M},x,r}|\right] \\ &\le O\left(\frac{1}{n}\right).\end{aligned}$$

Here we have used the fact that, for a fixed w, r and $r\oplus w$ are equidistributed. The last inequality follows from Markov's inequality and Lemma 8.

Now Lemma 3 provides us with a log-space computable function $G : \{0,1\}^{O(\log n)} \to \{0,1\}^{n^c}$ such that, for all $x \in \{0,1\}^n$ and $w \in \{0,1\}^{n^c}$,

$$\left|\Pr_{r\in\{0,1\}^{n^c}}[\mathcal{C}_{x,w}(r)=0] - \Pr_{s\in\{0,1\}^{O(\log n)}}[\mathcal{C}_{x,w}(G(s))=0]\right| \le \frac{1}{n}.$$

In particular, for all sufficiently large n we get the rough bound:

$$\Pr_{s\in\{0,1\}^{O(\log n)}}[\mathcal{C}_w(x,G(s))=0] \le \frac{1}{n} + O\left(\frac{1}{n}\right) < \frac{1}{2}.$$

Therefore, for any x and w, at least half of the seeds s will ensure that the configuration graph $\mathcal{G}_{\mathcal{M},x,G(s)\oplus w}$ has at most n^{2c+3} nodes. ◀

Our goal is now to use an inductive counting argument on $\mathcal{G}_{\mathcal{M},x,G(s)\oplus w}$. Like we mentioned earlier, inductive counting requires us to write down configurations in the work tape, but the tape is not big enough. To circumvent this, we will instead write down the *hash values* of the configurations, via the hash family of Lemma 4. The proof below puts it all together.

▶ **Theorem 11** (Immerman–Szelepcsényi for catalytic computation)**.** *If there exists a constant* $\varepsilon > 0$ *such that* $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\varepsilon n})$ *then* $\mathsf{CNL} = \mathsf{coCNL}$.

Proof. Let $\mathcal{M}$ be a nondeterministic Turing machine that uses $d\log n$ work space, and has an auxiliary tape of size n^d. We wish to construct a nondeterministic catalytic Turing machine $\mathcal{M}'$, using workspace $O(\log n)$, such that for any n and any input $x \in \{0,1\}^n$ our computation accepts x if $\mathcal{M}$ rejects x, and vice-versa.

Without loss of generality, assume that for any given $w \in \{0,1\}^{n^d}$, $\mathcal{M}(x,w)$ has a unique accepting configuration acc_w. Let $start_w$ be the initial configuration of $\mathcal{M}(x,w)$ and let $e = 2d+3$.

By the consistency property, either there exists a path from $start_w$ to acc_w for all w, or it is impossible to reach acc_w from $start_w$, for any w. We prove Theorem 11 by describing a way of certifying that there exists no path between $start_w$ and acc_w in $\mathcal{G}_{\mathcal{M},x,w}$.

Fix some input x, and let w' denote the initial contents of the aux-tape of $\mathcal{M}'$. By Lemma 10, we know that for at least half of the possible seeds $s \in \{0,1\}^{O(\log n)}$, we have

$$|\mathcal{G}_{\mathcal{M},x,G(s)\oplus w'}| \le n^e. \tag{1}$$

If (1) holds, we say s is a *good seed.*

Lemma 4 gives us a family of hash functions $\{h_k\}_{k=1}^{n^{3e}}$, with the property that, for every good seed s, there is at least one hash function in the family which is one-to-one on the nodes of $\mathcal{G}_{\mathcal{M},x,w}$.

In page 9, we give the pseudo-code for $\mathcal{M}'$'s algorithm. Let us now do a guided reading of this code. We begin by breaking the code into three sections, for the lines 2–6, 7–26, and 27–32.

In lines 2–6, we initialize a variable N to 0 (line 2), cycle through every seed s (line 3), XOR the contents of the aux-tape with $G(s)$ (line 4), and initialize two variables g and ℓ to 0 (lines 5 and 6).

Then, in lines 7–26, we have an inner loop that cycles through every hash function (line 7). Below we will prove:

Property I If the seed s is good, then **(I.a)** some sequence of non-deterministic bits will cause the inner loop to exit normally at line 27, with the promise that $g = |\mathcal{G}_{\mathcal{M},x,w}|$, and that h_ℓ is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$; and **(I.b)** any sequence of non-deterministic bits that fails this promise will exit the inner loop by jumping directly to line 30.

At line 27, we use the value of g and ℓ we have obtained to try and certify that acc_w is not reachable. If we succeed to do so, we increment N (line 28). Below we will also prove:

Property II If the seed s is good, $g = |\mathcal{G}_{\mathcal{M},x,w}|$, and h_ℓ is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, then some sequence of non-deterministic bits will cause us to successfully certify that acc_w is not reachable if and only if $M(x, w)$ rejects.[4]

Before we move on to the next seed, we first restore the initial contents of the aux-tape, by once again XORing them with $G(s)$ (line 30).

Finally, the procedure accepts if and only if $N > S/2$ in line 32. Let us prove that, assuming Properties I and II, the procedure accepts if and only if $M(x, w)$ rejects. Lemma 10 ensures that more than half the seeds are good, and hence:

1. If $M(x, w)$ rejects: Property I ensures that, for each good seed s, some non-deterministic guess will cause us to reach line 27 with $g = |\mathcal{G}_{\mathcal{M},x,w}|$ and h_ℓ one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$; then Property II ensures that some further guess will result in N being incremented; hence some overall non-deterministic guess will give $N > S/2$, and the procedure will accept in line 32.
2. If $M(x, w)$ accepts: Property I ensures that, for each good seed s, if we reach line 27, then $g = |\mathcal{G}_{\mathcal{M},x,w}|$ and h_ℓ one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, and thus, by Property II, N will not be incremented in line 28. If some non-deterministic guess fails to get us to line 27, then Property I tells us that the execution jumped directly to line 30, so N was again not incremented. Because no good seed will ever cause N to be incremented, $N < S/2$ and the procedure rejects in line 32.

So all we need to do is prove properties I and II. We first need to specify the canReach and cannotReach subroutines. Their correctness is easy to see from the description and pseudo-code.

[4] But if s, g or h_ℓ are not as assumed, we might get a false-positive, claiming that acc_w is not reachable when in fact it is.

Algorithm 1 Pseudo-code for $\mathcal{M}'$.

Here G is the log-space PRG of Lemma 3, S is the number of seeds, $M = n^e$ stands for the maximum number of configurations allowed in the configuration graph, and H is the size of the hash family given by Lemma 4. The aux-tape is represented by a variable w, whose initial value is w'. The lines that use non-determinism are marked with a $(*)$.

```
1:  procedure coCNL-Simulation(Input $x$, Aux-Tape $w \leftarrow w'$)
2:      $N \leftarrow 0$
3:      for $s = 0 \dots S$ do
4:          $w \leftarrow G(s) \oplus w$
5:          $g \leftarrow 0$
6:          $\ell \leftarrow 0$
7:          for $k = 1 \dots H$ do
8:              $c \leftarrow 1$
9:              for $i = 1 \dots M$ do
10:                 $c' \leftarrow 0$
11:                 for $v = 0 \dots M$ do
12:                     if canReach$(v, i, h_k)$ then                          ▷ $(*)$
13:                         $c' \leftarrow c' + 1$
14:                     else if cannotReach$(v, i, c, h_k)$ then               ▷ $(*)$
15:                         Do nothing
16:                     else
17:                         Jump to line 30
18:                     end if
19:                 end for
20:                 $c \leftarrow c'$
21:             end for
22:             if $c > g$ then
23:                 $g \leftarrow c$
24:                 $\ell \leftarrow k$
25:             end if
26:         end for
27:         if cannotReach$(h_\ell(acc_w), M + 1, g, h_\ell)$ then             ▷ $(*)$
28:             $N \leftarrow N + 1$
29:         end if
30:         $w \leftarrow G(s) \oplus w$
31:     end for
32:     Accept if $N > S/2$, and Reject otherwise
33: end procedure
```

The canReach(v, i, h_k) subroutine (see page 10) checks whether there is a node w in $\mathcal{G}_{\mathcal{M},xw}$, reachable within i steps, with $h_k(w) = v$.

Behavior of the canReach subroutine. If such a w exists, then some non-deterministic guess will cause the procedure to return TRUE, and, otherwise, every non-deterministic guess will return FALSE.

canReach non-deterministically works as follows: we guess a length $L \le i$, and simulate $\mathcal{M}$ for L steps. After this, we hash the configuration $\mathcal{M}$ is currently in, and compare it to v. We will then return TRUE if and only if the two hashes are the same, but before we return, we finish the simulation of $\mathcal{M}$ until we reach a halting state, in order to restore the contents of the aux-tape.

Algorithm 2 The canReach subroutine.

The subroutine to check that a node hashing to v is reachable in at most i steps, given some hash function h_k.

```
1: procedure canReach($v$, $i$, $h_k$)
2:     $z \leftarrow 0$                                   ▷ Workspace and internal state of simulated machine
3:     Non-deterministically guess $L \leq i$                                   ▷ (*)
4:     Simulate $\mathcal{M}(x, w)$ using $z$ as workspace for $L$ steps          ▷ (*)
5:     if $h_k(z, w') = v$ then
6:         $r \leftarrow$ TRUE
7:     else
8:         $r \leftarrow$ FALSE
9:     end if
10:    Continue simulation of $\mathcal{M}(x, w)$ using $z$ and reach any halting state
11:    return $r$
12: end procedure
```

The cannotReach(v, i, c, h_k) subroutine (see page 11) checks that there is no node in $\mathcal{G}_{\mathcal{M},x,w}$ hashing to v and reachable within i steps, as long as c and h_k fulfill the promise that there are exactly c nodes in $\mathcal{G}_{\mathcal{M},x,w}$ that are reachable within $i-1$ steps, and that h_k is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$.

Behavior of the cannotReach subroutine. If the hash v is unreachable within i steps and the given c, h_k obey the promise, then some non-deterministic guess will cause the procedure to return TRUE. If v *is* reachable and c, h_k obey the promise, every guess will return FALSE. *Furthermore*, if the hash v is unreachable within i steps, and c is smaller than the number of nodes in $\mathcal{G}_{\mathcal{M},x,w}$ that are reachable within $i-1$ steps, then there is a non-deterministic guess that causes the procedure to return TRUE, even if h_k is not one-to-one.

The cannotReach subroutine visits c different nodes of $\mathcal{G}_{\mathcal{M},x,w}$ in order of ascending hash value, and for each of them checks that none of their neighbors hash to v. Since a single step of a computation only makes a local change, it is possible to remember this step and revert it afterward to continue with the next neighbor. If one of the neighbors hash to v or if a wrong non-deterministic guess has been made somewhere, we restore the aux-tape and return FALSE. Otherwise finish the simulation of $\mathcal{M}$ until a halting configuration is reached, to restore the orginal value of w. If we have visited c distinct nodes without finding v as a neighbor, then we return TRUE.

Property II follows easily from the correctness of the cannotReach subroutine: indeed, if $M(x, w)$ rejects, then acc_w is not reachable, and hence with the promise made on g and h_ℓ, some guess will cause cannotReach$(h_\ell(acc_w), M+1, g, h_\ell)$ to return TRUE.

We now complete the proof of the theorem by proving Property I. Let us focus on the k-loop (lines 7–26) which goes through every hash function h_k. For each h_k a value c is computed (see lines 8, 10, 13 and 20).

It might happen that the k-loop is aborted (in line 17), but if this never happens, then c will be compared to g (line 22), so that by the time the k-loop terminates, g will hold the maximum c produced for any value of k (line 23), and ℓ will hold the first value of k which produced this maximum (line 24).

Algorithm 3 The CANNOTREACH subroutine.

The subroutine checking that a node hashing to v is *not* reachable within i steps, for hash function h_k, when given c, the number of nodes reachable in $i-1$ steps.

```
 1: procedure CANNOTREACH($v$, $i$, $c$, $h_k$)
 2:     $h' \leftarrow -1$                                   ▷ Hash of previously seen node
 3:     for $j = 1 \dots c$ do
 4:         $z \leftarrow 0$                    ▷ Workspace and internal state of simulated machine
 5:         Non-deterministically guess $L \leq i-1$                              ▷ (*)
 6:         Simulate $\mathcal{M}(x,w)$ using $z$ as workspace for $L$ steps            ▷ (*)
 7:         if $h_k(z,w) \leq h'$ then                     ▷ Visited the nodes in wrong order
 8:             Simulate $\mathcal{M}(x,w)$ using $z$ and reach any halting state
 9:             return FALSE
10:         end if
11:         $h' \leftarrow h_k(z,w)$
12:         while there are unvisited neighbours do
13:             Step $\mathcal{M}(x,w)$ with workspace $z$ into a neighbour configuration
14:             if $h_k(z,w') = v$ then                       ▷ $v$ is reachable in $i$ steps
15:                 Simulate $\mathcal{M}(x,w)$ using $z$ and reach any halting state
16:                 return FALSE
17:             end if
18:             Revert simulation with one step back
19:         end while
20:         Continue simulation of $\mathcal{M}(x,w)$ using $z$ and reach any halting state
21:     end for
22:     return TRUE
23: end procedure
```

Now we make the following two claims:

(i) If s is good, and h_k is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, the i-loop (lines 9–21) will either abort, or set $c = |\mathcal{G}_{\mathcal{M},x,w}|$. Furthermore, some non-deterministic choice within the i-loop will not abort.

(ii) If s is good, but h_k is not one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, the i-loop will either abort, or set c to a value strictly smaller than $|\mathcal{G}_{\mathcal{M},x,w}|$. As above, some non-deterministic choice within the i-loop will not abort.

From these, it follows that if s is good, then for every k there is a non-deterministic guess which does not abort, and using any such non-aborting guess, g will be set to $|\mathcal{G}_{\mathcal{M},x,w}|$, and ℓ will be the smallest k for which h_k is one-to-one. This gives us Property I.

Let us prove claim (i). Suppose that h_k is one-to-one, and that the i-loop does not abort. Then we may prove inductively that in every iteration of the i-loop, c is the number of nodes in $\mathcal{G}_{\mathcal{M},x,w}$ reachable by $M(x,w)$ within $i-1$ steps. Now, c, h_k satisfy the promise required by CANNOTREACH, and hence, for any non-aborting guess, the v-loop will set c' to the number of nodes in $\mathcal{G}_{\mathcal{M},x,w}$ reachable within i steps; this value is then copied to c (line 20) for the next iteration of the i-loop. When the i-loop ends, c has been set to the number of nodes reachable within M steps, which is exactly $|\mathcal{G}_{\mathcal{M},x,w}|$. The fact that there always exists such a non-aborting guess follows from the behavior of the CANREACH procedure, and from the behavior of the CANNOTREACH procedure in the case when c, h_k fulfill the promise.

To prove claim (ii), notice that the value of c' is incremented in line 13, and is thus bounded by the the size of image $h_k(\mathcal{G}_{\mathcal{M},x,w})$. So if h_k is not one-to-one, c' will always be strictly less than $|\mathcal{G}_{\mathcal{M},x,w}|$. On the other hand, it is always possible to find a non-deterministic guess which does not abort, even when h_k is not one-to-one. Whenever hash v is reachable in i steps, we can take the guess which makes canReach in line 12 return TRUE; when hash v is not reachable in i steps, we know from the behavior of cannotReach, that we can find a guess that makes cannotReach return true, provided that the argument c given to cannotReach in iteration i is not more than the number of nodes reachable within $i-1$ steps. This follows from the fact that, in iteration $i-1$, c' is bounded by the number of such nodes (because it is incremented only conditional on canReach of line 12. ◀

5 Hierarchies for Catalytic Computation

In this section we prove space-hierarchy theorems for deterministic and non-deterministic catalytic computation. Hierarchy theorems are usually proven using diagonalization. Since catalytic computation is a semantic model we do not know how to use diagonalization directly. Similarly to other semantic models (such as bounded-error randomized computation) we have to settle for hierarchy theorems with advice. This advice is used to tell the diagonalizing machine which machines can be safely simulated and diagonalized against, and which should not be simulated (so that the diagonalizing machine remains in the model).

The hierarchy theorem can be proven using the technique of Van Melkebeek and Pervyshev [10], which are sophisticated variations of [11]. Separations for certain space bounds follow directly from previous results on generic hierarchy theorems for semantic models of computation [7, 10]. For some ranges of parameters we provide a direct proof, mainly the calculations justifying the correctness of the bounds. Due to space constraints, the proof is left for the appendix. The theorem is:

▶ **Theorem 12.** *Let $a \geq 1$ be an integer and $s'(n)$ and $s(n)$ be space-constructible functions. There is a function in* CNSPACE$(s(n))/1$ *that is not in* CNSPACE$(s'(n))/a$*, and there is a function in* CSPACE$(s(n))/1$ *that is not in* CSPACE$(s'(n))/a$ *if any of the following is satisfied:*

1. *$s'(n) = O(\log n)$ and $s(n) = \omega(\log n)$.*
2. *$s'(n) = O(\log^{k'} n)$ and $s(n) = \Omega(2^{(\log\log n)^{k'}})$, for some constant $k' > 1$.*
3. *$s'(n) = O(n^{k'})$ and $s(n) = \Omega(n^k)$, where $0 < k' < k/2$ and $k' < 1/(1+a)$.*
4. *$s'(n) = O(n^{k'})$ and $s(n) = \Omega(n^k)$, where $k, k' > 0$ are such that $k \geq 2a$ and $k \geq \lceil 4^{ak'^2} \rceil$.*

The following corollary follows by using a padding argument (see [10], §4.4).

▶ **Corollary 13.** *Let $a \geq 1$ be an integer and $k > k'$ be positive reals. Then there is a function in $CNSPACE(n^k)/a$ that is not in $CNSPACE(n^{k'})/a$.*

References

1 E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, June 2006.

2 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

3 H. Buhrman, R. Cleve, M. Koucký, B. Loff, and F. Speelman. Computing with a full memory: Catalytic space. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 857–866, New York, NY, USA, 2014. ACM.

4 N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

5 R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 220–229, New York, NY, USA, 1997. ACM.

6 R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–209, 1982.

7 J. Kinne and D. van Melkebeek. Space hierarchy results for randomized and other semantic models. *Computational Complexity*, 19(3):423–475, 2010.

8 A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

9 R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

10 D. van Melkebeek and K. Pervyshev. A generic time hierarchy with one bit of advice. *Computational Complexity*, 16(2):139–179, 2007.

11 S. Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327 – 333, 1983.

A Appendix

A.1 Existence of hash family

▶ Theorem (Chinese Remainder Theorem). Let $p_1, \dots, p_m$ be a list of relatively prime integers. Any positive integer x is uniquely specified by the list of remainders $a_1 = x \mod p_1, a_2 = x \mod p_2, \dots, a_m = x \mod p_m$, provided that $x < \prod_{i=1}^{m} p_i$.

Proof of Lemma 4. For a natural number k, let p_k be the k-th prime number. For every $k = 1, \dots, n^3$ define the hash function $h_k(x) = x \mod p_k$. We will show that for any set $S \subset \{0,1\}^n$ of size n, there exists a number $k^* \in \{1, \dots, n^3\}$ such that the function h_{k^*} is injective on S. Here we interpret binary strings as natural numbers in the usual way, and hence we can upper bound any element of S by 2^n.

For all $x, y \in S$, where $x \neq y$, define $B_{x,y} = \{p_k \mid x \bmod p_k = y \bmod p_k, 1 \leq k \leq n^3\}$ to be the set of primes for which x and y hash to the same value. Then $B = \bigcup_{x,y \in S, x \neq y} B_{x,y}$ is the set of all primes which give a hash collision on the set S.

For any pair x, y it now holds that $|B_{x,y}| \leq n$. Indeed, assume for a contradiction that the set contains a subset of $n+1$ primes for which x and y have the same remainders. Noting that the product of these $n+1$ primes is at least 2^n, larger than both x and y, and that prime numbers are relatively prime, we find an immediate contradiction with the Chinese Remainder Theorem.

We can bound the number of primes that give a collision by

$$|B| \leq \sum_{x,y \in S, x \neq y} |B_{x,y}| \leq \binom{n}{2} n\,,$$

which is strictly less than n^3 for $n > 1$. Therefore there exists a prime p_k with $1 \leq k \leq n^3$ such that $p_k \notin B$, and therefore $x \bmod p_k$ is unique for all $x \in S$.

Left is to show this algorithm can be executed in logarithmic space. First note that using the prime number theorem we can (imprecisely) bound $p_k \leq n^4$, for $1 \leq k \leq n^3$. Since every number $p \leq n^4$ we try as modulus can be stored using $4 \log n$ bits, checking primality is also readily seen to be in space $O(\log n)$, just by checking all possible factors. To hash a value $x \in \{0,1\}^n$ we can, for example, sum $2^i \bmod p$ for all i such that $x_i = 1$. The value $2^i \bmod p$ can easily be computed in space $O(\log n)$ by repeated multiplication by 2, i.e., a bit shift, followed by subtraction of p whenever the intermediate value becomes too large. ◀

A.2 CNL definition, equivalent to Definition 6

▶ **Definition 14.** A decision problem L is in CNL if there exists a constant c and a deterministic Turing machine $\mathcal{M}$, with a read-only input tape, a uni-directional certificate tape, work tape of size $c \log n$ and an auxiliary tape of size n^c, such that for all n-bit strings x and for all $w \in \{0,1\}^{n^c}$ it holds that

$$x \in L \iff \exists u \in \{0,1\}^{2^{n^c}}\ M(x,u,w) \text{ accepts}$$

and

$$\forall u \in \{0,1\}^{2^{n^c}}\ M(x,u,w) \text{ halts with } w \text{ on its aux-tape.}$$

The string u represents the contents of the uni-directional certificate tape, and w is the starting contents of the auxiliary tape.

A.3 Alternatives to our definition of CNSPACE, and why they should be ruled out.

There are multiple possible ways to add non-determinism to a catalytic Turing machine. For instance, we require the machine to restore the contents of the auxiliary tape for any given sequence of non-deterministic bits; but at a first glance, it seems we could make this requirement only for those non-deterministic guesses which result in accepting states. However, defining the model in this way is less natural for several reasons. For one, we can not run two machines sequentially and accept if one of them accepts: if one of the two machines would reject, the whole computation needs to reject, because the auxiliary tape may have been irreversibly changed; so the class would not be closed under union. This would also prevent amplification of success probability in a probabilistic class defined using such machines. Philosophically speaking, having a catalytic machine which 'sometimes' destroys all data it is guaranteed to preserve, seems to go against the spirit of the model.

Another possible variation would be to require that the accepting sequence of non-deterministic choices is independent of the initial contents of the auxiliary tape, which would give a weaker model. Indeed, this would not look very strange in a certificate definition, effectively requiring that there exists a read-once certificate, independent of the initial contents of the aux-tape, which can be verified by a deterministic log-space catalytic Turing machine. Even so, when describing the model with non-deterministic Turing machines it seems unnatural to have this restriction. Furthermore, the model is weaker, so if we expect to make some use of non-determinism, it should be easier if we define it in the current way. Hence we have also ruled out this alternative definition.

A.4 Proof of the hierarchy theorem

Proof of Theorem 12. The first part is immediate from Kinne and Van Melkebeek [7] as catalytic computation satisfies the requirements on a *reasonable semantic model* and allows *complementation with linear-exponential overhead.*

Now we prove the third part using the technique of Van Melkebeek and Pervyshev [10]. Fix a small enough $\epsilon > 0$ and let's consider the case when $s'(n) = n^{k'+\epsilon}$ and $s(n) = n^k$. Let M_i be an enumeration of possibly catalytic machines working in space $s'(n)$ with catalytic tape of size $2^{s'(n)}$. Assume without loss of generality that each machine appears infinitely often in this enumeration. We will construct a machine M and an advice sequence $\{b_n\}_{n>0}$ so that M/b_n behaves catalytically on inputs of length n and uses space at most $s(n)$ and catalytic space $2^{s(n)}$. No machine M_i will accept the same language as $M/\{b_n\}$ regardless of its a-bit advice.

The proof diagonalizes against all machines M_i with all possible advice sequences. We define a sequence of integers n_i and n_i^* as follows:

$$n_0^* = a,\ n_i = n_{i-1}^* + 1, \text{ and } n_i^* = n_i^{1+an_i}.$$

We will diagonalize against M_i with all possible advices on input of length between n_i and n_i^*. Let $m_i = \log n_i$ and for $j = 0, \dots, m_i$ define

$$n_{i,j} = n_i \cdot (n_i^a)^{2^j}.$$

For $w \in \{0,1\}^{a(m_i-j-1)}$ and $z \in \{0,1\}^a$ define $n_{i,j,wz} = n_{i,j} + \overline{wz}$, where $\overline{wz}$ is the integer represented by wz in binary. For $y \in \{0,1\}^{n_{i,j,wz}}$ define the function

$$f(y) = yz0^{n_{i,j+1,w} - n_{i,j,wz} - a}.$$

Since all $n_{i,j,wz}$ are distinct, this is a well defined partial function. We are ready to define the machine M which takes $\{b_n\}_{n>0}$ as its advice sequence.

1. On input x of length n do:
2. If $b_n = 0$ then REJECT.
3. If $n = n_{i,j,wz}$ for some i, j, wz, where $j \leq m_i$, $|w| = a(m_i - j - 1)$ and $|z| = a$ then (nondeterministically) simulate M_i with advice z on input $f(x)$ and ACCEPT iff M_i accepts, and REJECT iff M_i rejects.
4. If $n = n_i^*$ then find y such that $f(f(\cdots f(y)\cdots)) = x$, where f is applied m_i-times. If no such y is found (such a y is a prefix of x) then REJECT. Let z be the first a bits of y. Using Savitch's algorithm decide whether M_i with advice z accepts y. If it accepts, REJECT, otherwise ACCEPT.

This defines the behavior of machine M. The advice $\{b_n\}_{n>0}$ is defined to be 1 of inputs of length $n_{i,j,wz}$ if and only if on all inputs of length $n_{i,j+1,w}$ machine M_i with advice z behaves in a correct catalytic manner (hence it is safe to simulate).

Assuming that machine M can perform the simulations in the designated space, it is easy to verify that it behaves catalytically and it diagonalizes against all machines M_i and all their possible advice sequences infinitely often.

So we only need to argue about the used space. Let M_i with advice sequence $\{z_n\}_{n>0}$ be a catalytic machine using work space $s'(n)$ and catalytic space $2^{s'(n)}$. On inputs of length $n_{i,j,wz_{n_{i,j+1,w}}}$, M will simulate M_i on inputs of length $n_{i,j+1,w}$ with advice $z_{n_{i,j+1,w}}$. By the choice of small enough ϵ, for all large enough n_i

$$\begin{aligned} s'(n_{i,j+1,w}) &\leq 2n_i^{(1+2^{j+1}a)(k'+\epsilon)} \\ &\leq n_i^{(1+2^j a)k}/n_i^{k/2} \quad \leq \frac{s(n_{i,j,wz})}{n_i^{k/2}}\,. \end{aligned}$$

Hence, M can successfully simulate M_i on these input lengths using its work space and the catalytic space. It remains to verify that the space necessary for Savitch's algorithm on inputs of length n_i^* will fit into our work space. Savitch's algorithm for machine M_i on input y will require space at most $O((\log|y| + \log s'(|y|) + 2s'(|y|) + 2^{s'(|y|)})^2)$, which is less than $2^{3s'(|y|)}$ for y (resp. n_i) large enough. The length of y is at most $2n_i^{1+a}$. Thus

$$s'(|y|) \leq 2n_i^{(1+a)(k'+\epsilon)} < 2n_i$$

and

$$2^{3s'(|y|)} \leq 2^{6n_i} \leq s(n_i^*),$$

for n_i large enough.

To prove the second part one uses the same argument as above but verifies that the space needed by M for the simulations fits into its space bounds:

$$\begin{aligned} s'(n_{i,j+1,w}) &\leq \left(\log 2n_i^{(1+2^{j+1}a)}\right)^{k'} \\ &= \left(1 + (1+2^{j+1}a)\cdot\log n_i\right)^{k'} \\ &\leq o\Big(2^{\log^{k'}\left((1+2^j a)\cdot\log n_i\right)}\Big) \\ &= o\Big(2^{\log^{k'}\log n_i^{(1+2^j a)}}\Big) \\ &= o(s(n_{i,j,wz})). \end{aligned}$$

Similarly,

$$2^{3s'(2n_i^{1+a})} \leq o(s(n_i^*))\,.$$

For the fourth part we set the parameters exactly like Van Melkebeek and Pervyshev [10, 7]: a constant $d = \max(2a, \lceil 4^{ak'^2} \rceil)$, $n_i^* = n_i^{n_i^d}$ and $n_{i,j} = n_i^{d^j}$. With these parameters there is sufficient space for M to simulate M_i's. ◀