Linear Advice for Randomized Auxiliary Pushdown Computation

Shiva Kintali*

September 8, 2026

Abstract

We give a deterministic polynomial-time auxiliary pushdown simulation, with logarithmic work space and linear advice, of one-sided-error randomized auxiliary pushdown computation with fresh random bits. For a fixed input, a smaller-child-first verification of first-pop computations gives an unambiguous syntactic read-once branching program of size $2^{O(\log^2 n)}$. We prove a Fourier-restriction generator for these programs, with seed length $O(\log^4 n)$ in the required regime and logarithmic-space indexed evaluation. A constant-density expander hitting walk then gives $O(n)$ advice. The branching-program representation is used only in the error analysis; the final simulator retains the source machine's single pushdown store.

1 Introduction

Randomness can simplify a computation while making its deterministic simulation difficult to implement with the same memory resources. For logarithmic-space machines, the amount of information retained between successive random choices is small. An auxiliary pushdown automaton (AuxPDA) adds a different resource: a stack whose contents can be much larger than the logarithmic work tape. The stack supports nested computation, but its information is accessible only in last-in-first-out order. We ask whether one-sided randomness can be removed while retaining logarithmic work space, polynomial time, and this single-stack discipline, if the simulator is allowed a short string of nonuniform advice.

Writing n for the input length, we prove

$$\mathrm{RLogCFL} \subseteq \mathrm{LogDCFL}/O(n),$$

with the precise statement in Theorem 2.3. Here the source uses fresh independent unbiased random bits, logarithmic work space, one stack, and a polynomial time bound on every computation. It accepts each yes-input with probability at least $1/2$ and never accepts a no-input; there are no additional nondeterministic choices. The simulator is deterministic, uses logarithmic work space and one pushdown store, and runs in polynomial time. Its advice has length $O(n)$, depends only on the input length, and is available on a two-way read-only tape. Definitions 2.1 and 2.2 specify these conventions. We use RLogCFL only for that explicitly defined fresh-coin machine model; no equivalence with other possible randomized formulations of LogCFL is assumed. In particular, the bounds must hold on every input–advice pair, with resources measured in its total length, although correctness is required only for the selected length-dependent advice. All asymptotic estimates below concern $n \geq 2$; finitely many shorter inputs can be handled separately.

*Email: shiva.kintali@gmail.com Homepage: shivakintali.github.io

The objective is not to store a long successful random computation as advice. A source may use polynomially many random bits, and even a polylogarithmic seed may be too long to fit on the logarithmic work tape. We therefore separate three requirements: a compact representation suitable for pseudorandomness analysis; a generator whose individual output bits can be recovered without using the stack or storing the seed; and a length-dependent hitting sequence that the simulator can traverse without storing its vertices. This separation also distinguishes the proof from an attempt to construct or deterministically evaluate every object used only in its analysis.

1.1 Background and related work

Auxiliary pushdown computation. The model and its relation to time-bounded computation originate in Cook's work [6]. Sudborough's characterization connects polynomial-time logarithmic-work-space nondeterministic AuxPDAs with LogCFL and deterministic AuxPDAs with LogDCFL, where these classes are defined by logspace many-one reductions to context-free and deterministic context-free languages, respectively [21]; see Allender and Lange [2, Propositions 1.1 and 1.2]. Cook's deterministic context-free language simulation, together with closure under these reductions, gives LogDCFL $\subseteq \mathrm{SC}^2$, where SC^2 denotes deterministic polynomial time and $O(\log^2 n)$ space [7]. Niedermeier and Rossmanith developed unambiguous AuxPDA and semi-unbounded-circuit simulations [17]. These works supply the structural setting for distinguishing a small surface configuration from the potentially large stack below it.

Space-bounded randomness and linear advice. Nisan constructed a pseudorandom generator for space-bounded computation [18], and separately showed that bounded-error randomized logarithmic-space computations of polynomial duration admit deterministic polynomial-time, $O(\log^2 n)$-space simulation [19]. The communication-based viewpoint was developed further by Impagliazzo, Nisan, and Wigderson [15]. Fortnow and Klivans start with Nisan's generator: for a polynomial-time logarithmic-space source, its seed has $O(\log^2 n)$ bits and its output is accessible one bit at a time in logarithmic space when the seed is on a two-way read-only tape [10, Section 3, Theorem 7]. They obtain $\mathrm{RL} \subseteq \mathrm{L}/O(n)$ by combining this access property with an advice-efficient expander walk. The final fixing of a successful random choice for all inputs of one length uses the same union-bound probabilistic method as Adleman's nonuniform simulation [1].

Nisan-style generalization to AuxPDAs. Venkateswaran explicitly generalizes the recursive hash-based generator and hash-mixing analysis to probabilistic auxiliary pushdown computations [22]. Section 5.2, Definition 12 of the revised manuscript [22] gives the generator, adapted to balanced realizability decompositions, and the paper proves $\mathrm{BP_H LOGCFL} \subseteq \mathrm{SC}^2$ for its bounded-error probabilistic AuxPDA class [22, Sections 2.2–2.3 and Theorem 9]. This prior uniform bound also applies to our source class: two independent trials, accepting if either accepts, raise completeness from $1/2$ to at least $3/4$ while retaining zero acceptance probability on no-inputs and the same asymptotic resource bounds. Our theorem concerns a different resource tradeoff: it preserves logarithmic work space and a single stack, but permits linear advice. It is not an improvement of that *uniform* SC^2 bound. Neither Nisan's generator nor Venkateswaran's generalization is a logical dependency of the proof below. We do not rule out an alternative proof using the latter; such an argument would have to match its realizability-based guarantee to the indexed-access and single-stack requirements here.

One-way randomness for stack machines. David, Nguyen, Papakonstantinou, and Sidiropoulos study logspace stack machines whose random tape is read in a bounded number of passes [8]. For

work space $O(s)$ and one pass over a $2^{O(s)}$-bit random tape, their Theorem 3 shows that even the two-sided unbounded-error model has exactly deterministic $2^{O(s)}$-time power. At $s = O(\log n)$ this gives a uniform polynomial-time upper bound. Our fresh-coin source is a polynomial-time one-pass special case, but the deterministic simulation in that result is an ordinary time-bounded Turing-machine simulation: it does not give a polynomial-time deterministic AuxPDA with logarithmic work space and one store, with or without linear length-only advice. Thus it is complementary to, rather than a substitute for, the resource-preserving containment proved here.

Stack simulations. Bogdanov, Papakonstantinou, and Wan give a quasipolynomial-width nondeterministic branching-program simulation of stack machines that preserves the number of reads and has unique accepting witnesses [3, Section 4, Lemma 4 and Appendix A]. Their pseudorandomness results include linear-stretch generators for machines making a constant number of passes over their randomness. They also explicitly extend their PRG analysis to unique-witness nondeterministic programs. Their representation is for each fixed input; it is not itself a length-only advice statement. Our compiler is a self-contained, clocked first-pop implementation of this read-preserving unique-witness approach in the strict polynomial-time setting. We do not claim that the existence of such a representation, or the use of unambiguity for stack simulation, originates here.

Restrictions, Fourier analysis, and adaptive order. Haramaty, Lee, and Viola developed the bounded-independence-plus-noise approach to fooling product tests [13]. Forbes and Kelley used this framework to obtain polylogarithmic-seed generators for read-once branching programs with an unknown fixed variable order [9]. Chen, Lyu, Tal, and Wu showed that the Forbes–Kelley construction also fools deterministic adaptive-order read-once programs [4]: the next queried coordinate may depend on the current state. Their Definition 6 and Theorem 7 concern deterministic programs that query every coordinate exactly once. Our analysis starts from their Fourier-cut argument [4, Section 4], but the analytical programs here may also make nondeterministic choices, have epsilon transitions, and omit coordinates. Their stated adaptive-order theorem is therefore not invoked as a theorem for this larger program model.

Expander walks and arithmetic. The advice construction uses Gabber–Galil expansion [11] and the low-space residue-replay method of Gutfreund and Viola [12], as credited in the revised Fortnow–Klivans paper [10]. Canonical conversion from Chinese remainders and indexed binary arithmetic come from the standard logspace-uniform arithmetic results [5, 14]. The spectral hitting estimate uses the discrete Cheeger inequality of Jerrum–Sinclair and Lawler–Sokal, in the form recorded by Levin and Peres [16, Theorem 13.10]. These are inherited ingredients, not new expander or arithmetic constructions.

1.2 Contributions

Our main contribution is the stated single-stack linear-advice simulation, obtained by combining the following extensions and resource bounds. We distinguish what is proved here beyond the cited theorem statements from the constructions and techniques from which it is derived.

Extension of the Fourier argument to unambiguous programs. Lemmas 4.2–4.6 and Theorem 4.7 extend the adaptive-order analysis of [4] to acyclic unambiguous syntactic read-once programs. Here unambiguity means at most one accepting path consistent with each assignment; syntactic read-once means that no coordinate repeats along any start-to-accept graph path, whether

or not the edge labels on that path are mutually consistent. The essential additional observation is that, after deleting vertices outside all start-to-accept paths, each prefix and suffix path-count function is itself Boolean on every assignment, and the two functions depend on disjoint sets of input coordinates. Merely knowing that their product is Boolean would not provide the separate L_2 and L_∞ bounds used in the restriction argument. It permits the Fourier cut and second-moment argument to survive nondeterminism. The proof explicitly preserves path multiplicities under restrictions and accounts for paths that do not query every coordinate. The extension is in the *analysis and model*; the iterative restriction construction is derived from Forbes–Kelley [9], not introduced as a new generator design.

Uniform access in the required resource regime. For a logspace-computable output length $1 \le R(n) \le n^{O(1)}$ and the common size bound $S(n) \le 2^{C\lceil \log(n+2)\rceil^2}$ for a fixed integer C, Lemma 4.8 gives error at most $1/8$, seed length $s(n) = O(\log^4 n)$, and polynomial-time $O(\log n)$-space recovery of any requested output bit from a two-way read-only seed, without touching a stack. The generator depends on the source input only through its length; it is independent of both the input word and the analytical program. The field-polynomial limited-independence sampler is standard; the contribution here is its explicit parameter choice and indexed implementation for this simulation. A generic explicit-PRG guarantee allowing space proportional to the whole seed would not suffice. Nor is this a shorter-seed claim than Nisan's bound for ordinary logarithmic-space machines: the tested program class and its size are different.

Advice amplification with a live stack. Lemma 5.2 formulates the Fortnow–Klivans mechanism for a deterministic AuxPDA consumer and obtains $O(n + s(n))$ advice. The additional work is to compose a stack-free seed provider with a consumer whose stack may already be in use, restore an empty simulated store between trials, and ensure polynomial time and logarithmic work on every input–advice pair, including malformed advice. Together with the common length-dependent generator, this yields the main containment. The expander, residue replay, Chinese-remainder algorithms, and probabilistic fixing argument are derived from the earlier works cited above; the single-stack composition and its use in this containment are the extension supplied here.

The read-once compiler is a rederivation and normalization of the unique-witness stack-simulation approach of [3]; its positive-span potential and smaller-child-first traversal make the required time and read-once invariants explicit.

1.3 Overview of our proof strategy

Step 1: expose a unique read-once certificate. Normalize the source to $T = T(n) = n^{O(1)}$ logical steps, with step t consuming coin r_t. Inserting ignored fresh coins and padding after termination preserve its acceptance probability. A boundary control records the state, work tape, head positions, and clock, but no stack symbols; the current top symbol is carried separately in the descriptor. A first-pop descriptor (p, a, q) asks whether the computation starting in control p with stack $\mathsf{a}\beta$ (top at the left) first exposes the suffix β in control q. This is not merely the deletion of the original symbol a: all symbols replacing it must also be removed. The answer is independent of the protected suffix.

The descriptor's interval $[t(p), t(q))$ has positive span. A two-symbol replacement splits the interval remaining after the parent's entry transition into two positive, disjoint consecutive child intervals. For a fixed coin word, at most one intermediate control makes both child descriptors succeed (Proposition 3.1). The verifier processes the smaller child first, saves the larger one, and

later continues with that saved child without retaining the parent or a return value. The originating-parent spans of simultaneously pending frames form a sequence in which each span is less than half its predecessor, so only $O(\log T)$ pending descriptors, each of $O(\log n)$ bits, are stored. At every loop boundary, the sum of the current and pending spans decreases by one after any iteration that passes its local checks. Hence every verifier branch has at most T query iterations; a failed local check rejects immediately. The active intervals remain pairwise disjoint and exclude every already queried time, which prevents repeated coin queries even on rejecting branches. Thus the verifier has polynomial time and $O(\log^2 n)$ work. Compiling its complete, clocked configurations gives an acyclic unambiguous syntactic read-once program of size $S = 2^{O(\log^2 n)}$ on $R = T$ coordinates (Lemma 3.2). Write $P_x : \{0,1\}^R \to \{0,1\}$ for its acceptance function. This representation is used only in the analysis.

Step 2: fool the analytical program, then run the source. Delete vertices not on any start-to-accept graph path; if none remain, $P_x = 0$ and the fooling claim is immediate. For a retained query vertex v, let F_v count consistent start-to-v prefixes, excluding the query at v, and let Q_v count consistent v-to-accept suffixes, including that query. Syntactic read-once and unambiguity make both functions Boolean on all assignments, and the coordinate sets on which they may depend are disjoint (Lemma 4.2). For an integer $k \geq 1$, let H_v be the homogeneous degree-k Fourier projection of F_v, with respect to uniform measure on $\{0,1\}^R$, and set

$$G_v(r) = \frac{Q_v(r) - Q_v(r \oplus e_{i(v)})}{2},$$

where $i(v)$ is the queried coordinate and $e_{i(v)}$ flips that bit. Thus G_v retains precisely those Fourier coefficients of Q_v whose characters contain $i(v)$. Lemma 4.3 gives the exact identity

$$P_x = (P_x) * \leq k + \sum *v \text{ query} H_v G_v.$$

Here $(P_x)_{\leq k}$ is the Fourier projection onto degrees at most k. Take mutually independent R-bit random vectors $D_1, W_1, \ldots, D_\ell, W_\ell$ and define

$$Z_0 = 0^R, \qquad Z_j = D_j \oplus (W_j \wedge Z_{j-1}) \quad (1 \leq j \leq \ell),$$

where the coordinates of each D_j are $2(k+1)$-wise independent and those of each W_j are $(k+1)$-wise independent; in particular, every mask coordinate is unbiased. Here $\oplus$ and $\wedge$ act coordinatewise. A homogeneous second-moment calculation, closure under restrictions, and a hybrid argument give

$$|\mathbb{E}P_x(Z_\ell) - \mathbb{E}P_x(U_R)| \leq \ell S\, 2^{-k/2} + R\, 2^{-\ell},$$

where U_R is uniform on $\{0,1\}^R$ (Theorem 4.7). Lemma 4.8 chooses $\ell = O(\log n)$ and $k = O(\log^2 n)$ so that each error term is at most $1/16$. Sampling the masks by field polynomials uses $s = 3\ell(k+1)\lceil \log(R+1) \rceil = O(\log^4 n)$ seed bits. The stack-free accessor then lets the deterministic consumer run the *normalized source*, using the same generator for every length-n input. It never constructs or directly simulates the analytical program. A yes-input retains acceptance probability at least $1/2 - 1/8 = 3/8$; every seed rejects on a no-input because every full source coin word does.

Step 3: fix one short hitting walk for each length. Use the symmetric lazy constant-degree expander of Lemma 5.2 on $\mathbb{Z}_m^2$, where m is a product of small distinct primes with $\log m = O(s+1)$ and $m^2 \geq 16 \cdot 2^s$. Map a vertex (u_1, u_2), in canonical coordinates, to the low s bits of $u_1 + m u_2$. The image of a uniform vertex has total variation distance at most $1/64$ from a uniform seed, so each

yes-input's accepting vertex set has density at least $3/8 - 1/64 = 23/64$. A stationary walk with $L = C_{\mathrm{w}}(n+1)$ transitions and $L+1$ visited vertices, for a sufficiently large constant C_{w}, misses any one such set with probability less than 2^{-n-1}. A union bound over at most 2^n yes-inputs shows that some single walk hits every accepting set. The trials along this walk are correlated; independence is not needed.

Advise the start's residues, the prime list, m, encoding headers, and the walk's edge choices. Each prime has $O(\log n)$ bits, while $\sum_i \log \mathfrak{p}_i = \log m = O(s+1)$; consequently the prime records, including their residues and self-delimiting encodings, use $O(s+1)$ bits in total. The complete advice uses $O(n+s)$ bits. To obtain a seed bit at a visited vertex, replay the edge choices modulo one prime at a time and use indexed Chinese-remainder conversion and binary arithmetic. These routines leave the consumer's stack untouched. Lemma 5.1 justifies their composition, and Lemma 5.2 handles stack resets and resource guards on arbitrary advice. Finally $s = O(\log^4 n) = O(n)$ after enlarging the constant to cover the finitely many small lengths, so the advice is linear while the simulation retains polynomial time, logarithmic work, and one stack.

1.4 Organization of the paper

Section 2 defines the models and states the main theorem formally. Section 3 proves the clocked first-pop representation. Section 4 establishes the unambiguous Fourier identity, the generator, and indexed evaluation. Section 5 proves the virtual-tape composition lemma and advice-efficient amplification. Section 6 combines these ingredients to prove Theorem 2.3.

2 Models and the main result

We use $\mathbb{N} = \{0, 1, 2, \ldots\}$, and all logarithms are to base two. Write $n = |x|$ for the source input length. Logarithmic work space means $O(\log(n+2))$ cells over a fixed finite alphabet; for $n \geq 2$ we abbreviate this as $O(\log n)$. For each integer $R \geq 0$, put $I_R = \{i \in \mathbb{N} : i < R\}$ and let U_R be uniform on $\{0,1\}^R$. Thus $I_0 = \varnothing$, $\{0,1\}^0$ consists of the empty word, and U_0 is its point mass. Bit indices start at zero. Machine descriptions and constants are fixed for the language under consideration, independently of the input length, input word, and random choices.

Definition 2.1 (Auxiliary pushdown machines and randomness)**.** An auxiliary pushdown automaton (AuxPDA) has finite control, one two-way endmarked read-only input tape, one work tape, and one pushdown store over fixed finite alphabets. Initially the work tape is blank and the store is empty, apart from a fixed bottom marker if one is used. The store is accessible only at its top; a transition may replace the top symbol by a word of machine-dependent bounded length. Empty-store status is distinguishable, and a transition may push onto an empty store. If a bottom marker is used, "empty" refers to the store containing only that marker, which is never popped. Work space counts visited work-tape cells, not the height of the store. Every transition, including stack operations and deterministic bookkeeping, is charged to time.

A probabilistic AuxPDA has, in each configuration, either a deterministic transition or an instruction requesting one fresh independent unbiased bit. Whether a bit is requested is determined by the current configuration before that bit is generated. In a coin-requesting configuration, the current configuration together with the bit determines the transition; there are no additional existential or universal choices. Equivalently, every infinite stream of independent uniform bits determines a unique computation, whose requested bits are consumed in order. Consumed bits are not freely reread from that stream. The machine may retain their values in finite control, on its work tape, or on its stack, subject to the respective storage and access rules.

A language $\mathcal{L} \subseteq \{0,1\}^*$ belongs to RLogCFL if there exist a fixed probabilistic AuxPDA M, a constant $c_M > 0$, and a fixed integer-valued polynomial bound $p_M(n) \geq 1$ such that, for every input x and every infinite coin stream, M uses at most $c_M \lceil \log(n+2) \rceil$ work-tape cells and halts within $p_M(n)$ transitions in either an accepting or a rejecting state. Writing $M(x) = 1$ for acceptance and $M(x) = 0$ for rejection, require

$$x \notin \mathcal{L} \implies \mathbb{P}[M(x) = 1] = 0, \qquad x \in \mathcal{L} \implies \mathbb{P}[M(x) = 1] \geq \tfrac{1}{2}.$$

The probabilities are over the source's random choices, for each fixed input x.

The simultaneous work and time bounds hold on rejecting computations as well as accepting ones. They are worst-case bounds, not expected-resource bounds. To make the one-sided condition explicit, fix x and put $n = |x|$. Because at most one bit is requested per transition, no computation requests more than $p_M(n)$ bits. Supply a word $r \in \{0,1\}^{p_M(n)}$ in order as coins are requested, and write $M(x;r) \in \{0,1\}$ for the resulting deterministic outcome. Unused trailing bits are ignored. For uniform r this is exactly the original experiment, including when the decision to request a bit depends on previously seen bits. Hence

$$\mathbb{P}[M(x) = 1] = 2^{-p_M(n)} \sum_{r \in \{0,1\}^{p_M(n)}} M(x;r).$$

Each summand is zero or one. Consequently, the probability is zero if and only if every word $r \in \{0,1\}^{p_M(n)}$ rejects. This pointwise statement, rather than a probabilistic interpretation of certificate guesses, is used for soundness later.

Define LogDCFL as the class of languages logspace many-one reducible to a deterministic context-free language. We use its standard characterization by deterministic AuxPDAs that halt on every input in polynomial time with logarithmic work space; see [21] and [2, Proposition 1.2] for the characterization and its original attributions. Such machines have the storage conventions above, but no random choices. The reduction and the deterministic AuxPDA are uniform: each is one fixed machine, not a family chosen separately for each input length.

Definition 2.2 (Advice)**.** Fix the binary pairing

$$\langle x, y \rangle = 1^{|x|}0xy, \qquad |\langle x, y \rangle| = 2|x| + 1 + |y|.$$

For $\varphi : \mathbb{N} \to \mathbb{N}$, a language $\mathcal{L} \subseteq \{0,1\}^*$ belongs to LogDCFL/$O(\varphi(n))$ if there exist one language $\mathcal{B} \in$ LogDCFL, a constant $c > 0$, and binary strings $(\alpha_n)_{n \in \mathbb{N}}$ such that $|\alpha_n| \leq c\,\varphi(n)$ for every n and, for every $x \in \{0,1\}^*$,

$$x \in \mathcal{L} \quad \iff \quad \langle x, \alpha_{|x|} \rangle \in \mathcal{B}.$$

The advice α_n depends only on n; neither the advice sequence nor the length bound φ is required to be computable. Without loss of generality, $\mathcal{B}$ rejects strings that are not valid pair encodings. Write $N = |\langle x, y \rangle| = 2|x| + 1 + |y|$ for the total encoded length. Since $\mathcal{B} \in$ LogDCFL, its decider halts within a fixed polynomial in N, using $O(\log(N+2))$ work space and one stack, on *every* input, including valid pairs with incorrect advice. Only the selected advice must give the correct answer for $\mathcal{L}$.

The pairing is injective and has logspace encoding, decoding, and validity testing. Given a candidate word, count its k leading ones. It is valid exactly when the next symbol is zero and at least k further bits follow; the next k bits are then x, and the remaining suffix is y. The necessary counters use $O(\log(N+2))$ work. Thus the requirement that $\mathcal{B}$ reject invalid encodings is without

loss of generality: a deterministic AuxPDA can perform this stack-free precheck, reset its input head and scratch work, and then run the original decider. This preserves polynomial time, logarithmic work, and one pushdown store. In particular, $\langle \epsilon, \epsilon \rangle$ is the one-bit word 0.

This convention is equivalent to supplying x and the advice on separate two-way endmarked read-only tapes. To simulate separate tapes from one valid encoded input, first locate the two component intervals, store the simulated head positions, and obtain each requested symbol by moving the physical head to the corresponding interval position. Conversely, given separate tapes for x and y, maintain a head position on the virtual word $1^{|x|}0xy$. Its unary-prefix and delimiter symbols are computed from a counter for $|x|$, while symbols in the two data intervals are obtained by scanning the appropriate physical tape. In both directions only a fixed number of positions and counters are stored, using $O(\log(N+2))$ work. The conversion leaves the pushdown store untouched and incurs only polynomial time overhead. On selected advice of length $O(n)$, the encoded length is $N = O(n+1)$, so these bounds become $O(\log(n+2))$ work and polynomial time in $n+2$. Resource bounds on arbitrary advice remain measured in N, not merely in n.

Theorem 2.3. *Under Definitions 2.1 and 2.2,*

$$\mathrm{RLogCFL} \subseteq \mathrm{LogDCFL}/O(n).$$

The proof is given in Section 6, using the read-once representation of Lemma 3.2, the common indexed generator of Lemma 4.8, and the single-stack composition and advice bounds of Lemmas 5.1 and 5.2. In particular, the theorem is a claim about the explicitly defined source model, not about machines with unrestricted rereading of external random bits or without a worst-case polynomial clock. For the bound $\varphi(n) = n$, the advice for input length zero is empty; the answer on the empty input, and on any other finitely many exceptional lengths, can be hardwired into the finite control of the fixed decider.

3 A read-once representation of the source coins

This section represents the source's acceptance function by a small unambiguous nondeterministic branching program. The program is used only as an analytical test for the generator in Section 4; it is not the deterministic simulator. Read-preserving stack-machine simulations with unique witnesses appear in [3, Section 4, Lemma 4 and Appendix A]. We give a self-contained first-pop construction for the worst-case polynomial-time model of Definition 2.1. In particular, we prove the read bound on all verifier branches, including branches with incorrect guesses. Throughout this section $n \geq 2$; the finitely many shorter inputs are handled as in Section 2.

3.1 Clocked normal form

Fix a source machine M_0 and an input x of length n. Stack words are written with the top at the left. Write M for its normalization, constructed uniformly from M_0. There is a fixed polynomially bounded, logspace-computable integer $T = T(n) \geq 1$ and logical boundary times $0, \ldots, T$. Boundary t is the complete nonstack configuration immediately before logical transition t; boundary $t+1$ is reached when that transition's final stack action has been performed. A transition at time $t < T$ consumes coin r_t, possibly ignoring it, and replaces the top symbol by a word of length zero, one, or two. All other work between boundaries is deterministic and leaves the stack unchanged. The replacement is the last stack action of a logical transition, so the newly exposed suffix cannot influence its exit control.

Local implementation. Simulate each ordinary source transition using the old top symbol and, when the source requests one, the fresh bit assigned to that transition. Carry out work-tape and head updates before the stack replacement. The decision whether the source requests a bit is made before that bit is read. A replacement $\mathsf{a} \mapsto \omega_1 \cdots \omega_j$, with $j \geq 1$, can be split into $\mathsf{a} \mapsto \omega_j$, followed by pushes of $\omega_{j-1}, \ldots, \omega_1$. The selected bounded-length word and the current substep fit in finite control. The source transition's bit is used only to select that transition; subsequent substeps consume ignored bits. A pop is already an allowed replacement. Thus a source transition takes a bounded number of logical steps. Maintain a binary logical clock in a separate logarithmic work region. Clock updates take physical microsteps; unit-time binary arithmetic is not assumed. All scratch work and clock updates occur before the last stack action. Reading the old top is allowed during this work; reading the newly exposed top is deferred to the next logical step. Logical time counts replacements, whereas physical time counts all these microsteps. Every logical step, whether stack-changing or stack-preserving, consumes its allocated coin; the coin is ignored except on the first substep of a source transition that requests a bit. Thus M requests exactly one coin per logical step and none between boundaries, so $M(x; r)$ is defined for $r \in \{0,1\}^T$.

Initialization and canonical exits. Begin at time zero with a fresh protected marker # as the entire physical stack. This marker is distinct from the source's stack alphabet. During source simulation it represents the source's empty store and is never popped. A push on an empty source store is implemented by pushing above #; an empty-store operation that does not push leaves # in place. If M_0 starts with its own bottom marker, an initial ignored-coin step pushes that symbol above #. After the source halts, retain its answer in finite control, clear the stack above #, clear non-clock work, and reset heads to canonical boundary positions. Pad until time $T-1$, then pop # using the ignored coin r_{T-1} into a canonical accepting or rejecting control at time T.

Choose T to dominate initialization, the source simulation, cleanup, and at least one final step, uniformly over length-n inputs and all source choices. Bounded-length replacements and the source clock give a polynomial bound on stack height, hence on cleanup time. After enlarging fixed integer constants, take $T = C_T(n+2)^{d_T}$ for fixed positive integers C_T, d_T. This is logspace computable and has $O(\log n)$ bits. All physical routines have polynomial cost. Before the final pop, scratch data and head positions are canonical and the next clock value is prepared; no suffix inspection or further bookkeeping follows that pop. The two exits differ only in their answer state. The normalized machine is one fixed machine determined by M_0. Its source-work, clock, and scratch regions together use $O(\log(n+2))$ work cells. Since it executes exactly $T(n)$ logical transitions and each intervening physical routine has polynomial cost, its worst-case physical running time is polynomial in $n+2$ on every coin word.

Preservation of randomness. Whether the coin at logical time t is used by the source is determined by x and the prefix $r_0, \ldots, r_{t-1}$, not by r_t. Conditional on any such prefix, r_t is uniform and independent of the previous choices. Induction on the source requests therefore gives exactly the original distribution of source transitions:

$$\mathbb{P}_{r \sim U_T}[M(x; r) = 1] = \mathbb{P}[M_0(x) = 1].$$

Pointwise, every normalized coin word supplies a possible finite sequence of source choices, so rejection for every source sequence implies rejection for every normalized word. No fixed-coordinate padding identity is asserted: the positions of ignored coins can depend on the execution. All subsequent pointwise statements concern the normalized M.

Boundary controls and local rules. A boundary control p records finite state, input-head position, all work data and work-head positions, normalization phase, and clock $t(p)$, but no buried stack symbols. It does not automatically record the newly exposed top after a pop. Values explicitly saved earlier by the source remain part of its work data, as permitted in Definition 2.1. Let $\mathcal{C}_n$ be the polynomial-size set of syntactically legal boundary controls, each with one fixed-length canonical encoding of $O(\log n)$ bits. Legality means that fields, head positions, phases, and clock values are in their prescribed ranges, not that the control is reachable. Unused encoding bits are fixed to zero. Membership in this set is decidable in logarithmic work space. The fixed alphabets and the $O(\log n)$ encoding bound give $|\mathcal{C}_n| = n^{O(1)}$, uniformly over inputs of length n.

Local routines are defined also on hypothetical controls: fixed work and time guards send any invalid local operation to a rejecting phase without inspecting a stack suffix. The guards do not alter actual source simulations. For fixed x, $t(p) < T$, old top a, and coin $\xi \in \{0,1\}$, the transition

$$\tau_x(p, \mathsf{a}, \xi) = (p', \omega), \qquad |\omega| \le 2, \qquad t(p') = t(p) + 1$$

is deterministic and computable with logarithmic work in polynomial time, uniformly over all syntactically legal arguments. The output control p' is again syntactically legal, so $p' \in \mathcal{C}_n$. The rejecting phase performs stack-preserving length-one replacements and also advances the clock, so the rule is total before time T. It depends on a, not on the suffix below it. No transition is needed at time T.

3.2 First-pop computations

Recall that stack words are written with the top at the left. A descriptor $\delta = (p, \mathsf{a}, q)$, with $p, q \in \mathcal{C}_n$ and stack symbol a, has

$$I(\delta) = [t(p), t(q)), \qquad |I(\delta)| = t(q) - t(p) > 0.$$

Intervals here are sets of integer logical times. The descriptor succeeds on $r \in \{0,1\}^T$ if the computation starting in boundary control p with stack $\mathsf{a}\beta$ first exposes the suffix β in boundary control q. More explicitly, run the local rules until the stack first has height $|\beta|$, or until time T if this never happens. Success means that such a boundary exists and the first one is exactly q; if height $|\beta|$ is not reached by time T, the descriptor fails. Popping a alone is not sufficient if it has been replaced by new symbols: all symbols above β must disappear.

Before this boundary the stack has form $\gamma\beta$ with $\gamma \neq \varepsilon$, so only the top of γ is inspected. Coupling two runs with different suffixes gives the same control and the same word γ at each boundary until γ becomes empty. At that last pop the exit control is chosen before the suffix can be read. Thus the answer is independent of β, including when β is empty. This defines $\Gamma_{p,\mathsf{a},q}(r) \in \{0,1\}$ for hypothetical controls and stacks as well as reachable ones: its value is one exactly on success and zero otherwise. Its value depends only on the coin coordinates in $I(\delta)$. Indeed, simulation through boundary $t(q)$ determines whether the suffix was first exposed there; if it was exposed earlier or is not exposed at that boundary, no later coin can make q the first exposed-suffix boundary.

Proposition 3.1 (Canonical first-pop decomposition)**.** *Let $\delta = (p, \mathsf{a}, q)$ be a legal positive-span descriptor and put $\tau_x(p, \mathsf{a}, r_{t(p)}) = (p', \omega)$.*

(i) If ω is empty, the answer is $\mathbf{1}_{\{p'=q\}}$; equality implies $|I(\delta)| = 1$.

(ii) If $\omega = \mathsf{b}$, the answer is $\Gamma_{p',\mathsf{b},q}(r)$ when $t(p') < t(q)$, and zero otherwise.

(iii) If $\omega = \mathsf{bc}$, the answer is

$$\bigvee_{\varsigma \in \mathcal{C}_n:\ t(p')<t(\varsigma)<t(q)} \big(\Gamma_{p',\mathsf{b},\varsigma}(r) \wedge \Gamma_{\varsigma,\mathsf{c},q}(r)\big). \tag{1}$$

For fixed r, at most one summand in (1) is true.

Every successful descriptor has exactly one successful decomposition tree. Each node is labeled by a descriptor and its prescribed local transition; binary children are ordered chronologically, as in (1). The node entry times are a bijection onto $I(\delta)$, so the tree has exactly $|I(\delta)|$ nodes.

Proof. In case (i), the first transition exposes β in control p'. Equality $p' = q$ includes the clock field, so it forces $|I(\delta)| = 1$. In case (ii), the remaining task first exposes β; this takes positive time and is exactly the indicated child predicate. In case (iii), the remaining task first exposes $\mathsf{c}\beta$ and then, from the control ς reached there, first exposes β. Since every replacement changes the height by at most one, the stack, starting at height $|\beta| + 2$, reaches height $|\beta|$ only after first reaching height $|\beta| + 1$, and by the coupling argument it then equals $\mathsf{c}\beta$. The first boundary exposing $\mathsf{c}\beta$ determines a unique ς in any successful deterministic execution. Both child spans are positive, and their disjoint consecutive intervals partition $[t(p) + 1, t(q))$.

Conversely, successful children concatenate to a successful parent: the first child does not inspect $\mathsf{c}\beta$, and the second starts in its prescribed complete nonstack control ς. Two successful choices of ς would give different first-pop boundaries in the same deterministic computation, which is impossible. This proves both directions of the recurrence. Each child has strictly smaller positive span, so induction on $|I(\delta)|$ gives existence and uniqueness of the whole successful tree. A leaf contributes its single entry time. For a unary or binary node, its entry time together with the child intervals is a disjoint partition of its own interval, proving the node-time bijection by the same induction. An empty disjunction in (1) has value zero. □

The root is $(p_{\text{init}}, \#, q_{\text{acc}})$, of interval $[0, T)$, where $\#$ is the protected marker and q_{acc} the canonical accepting exit. The marker is popped by the transition at time $T - 1$ and never earlier, so on every normalized run the empty stack is first exposed at boundary T; thus this descriptor succeeds exactly when $M(x; r)$ accepts. Its initial control includes any initialization phase.

3.3 Smaller-child-first verification

Lemma 3.2 (Read-once compiler)**.** *For each fixed length-n input x, the function $r \mapsto M(x; r)$ has an acyclic unambiguous syntactic read-once nondeterministic branching program of size $2^{O(\log^2 n)}$ on $T(n)$ coin coordinates. The constant in the exponent depends only on M, not on x. Epsilon edges are allowed. The terminology is formalized in Definition 4.1 below; syntactic read-once is a graph-path property, not a promise about consistent accepting paths alone. Neither the program nor its accepting-path subgraph must be constructed by the final simulator.*

Proof. The verifier stores one current descriptor and a LIFO list of pending descriptors, initially the root and the empty list. This is an ordinary nondeterministic work-tape verifier with query access to r, not an AuxPDA restricted to logarithmic work space. Its entire pending list is charged to work space and will be encoded in program vertices. An iteration is:

1. Check legality and $0 \le t(p) < t(q) \le T$. Query $r_{t(p)}$ once and compute $\tau_x(p, \mathsf{a}, r_{t(p)}) = (p', \omega)$.
2. In case (i) of Proposition 3.1, reject unless $p' = q$. On success, accept if the pending list is empty; otherwise remove its last descriptor and make it current.

3. In case (ii), reject unless $t(p') < t(q)$; otherwise make the child (p', b, q) current.

4. In case (iii), guess each bit of ς's fixed-length canonical encoding once, with one binary choice per position. Reject invalid encodings, and reject unless $t(p') < t(\varsigma) < t(q)$. The children are $(p', \mathsf{b}, \varsigma)$ and $(\varsigma, \mathsf{c}, q)$; append the one of larger span to the pending list and make the other current, taking the chronologically first child $(p', \mathsf{b}, \varsigma)$ in a tie.

No other recursion stack is retained. On completion of the smaller child, the saved larger child is removed and tail-called; its parent needs no return value. All choices except the canonical guess of ς are deterministic. Evaluating the right child first when it is smaller checks two fully specified predicates in a different order; it does not assert that the source executes in that order. Their conjunction is unchanged. The guessed ς is checked as the left child's exit and the right child's entry.

Space. A pending frame is created only when the verifier starts the smaller child of a binary node; the frame stores the larger child. It remains live throughout the smaller child's subtree and is removed before the stored larger child is processed. Thus coexisting frames arise only from nested descents through smaller children. If a binary parent has span Δ, the smaller child has span at most $(\Delta - 1)/2 < \Delta/2$. Consequently, if $\Lambda \geq 1$ frames coexist and $\Delta_1, \ldots, \Delta_\Lambda$ are their parent spans from oldest to newest, then

$$\Delta_1 \leq T, \qquad \Delta_{j+1} < \Delta_j/2, \qquad \Delta_\Lambda \geq 3.$$

The final inequality holds because a binary node has two positive-span children. It follows that $T \geq \Delta_1 \geq 2^{\Lambda-1}\Delta_\Lambda \geq 3 \cdot 2^{\Lambda-1} > 2^\Lambda$, hence $\Lambda \leq \lfloor \log T \rfloor$. The case $T = 1$ has no pending frame. The *saved larger-child spans* need not halve: a span-101 parent can have children of spans $49, 51$, and its span-49 child can have children of spans $1, 47$. The two saved spans are then $51, 47$, whereas their parent spans are $101, 49$. Each descriptor, list index, and local temporary computation uses $O(\log n)$ bits. Including the current descriptor, total verifier space is $O(\log n(1 + \log T)) = O(\log^2 n)$.

Time. At a loop boundary, let $\mathcal{P}$ be the current pending list and put

$$\Psi = |I(\text{current})| + \sum_{\delta \in \mathcal{P}} |I(\delta)|.$$

Initially $\Psi = T$. Each successful local expansion decreases Ψ by one: the child spans sum to the parent span minus one, and a valid leaf has span one. Moving a pending descriptor to current changes no total span. After j completed iterations that have not halted, $\Psi = T - j \geq 1$; a possible final failed check is therefore still among at most T iterations. For accepting termination define the remaining potential to be zero. Comparisons, fixed-length guesses, list operations, and evaluation of τ_x all have uniform polynomial physical cost. This yields a fixed polynomial bound on the verifier's physical steps on every branch, including unrealizable guesses, not just on its number of queries.

Syntactic read-once. At every loop boundary, current and pending intervals are pairwise disjoint, and all previously queried times are distinct and outside their union. The next query is the current descriptor's entry time and is therefore new. A successful expansion removes that time and partitions the remaining interval among its positive-span children. A failed check halts, without any additional query. Induction proves the invariant even if each queried label is chosen freely along a graph path; neither consistency with a previously fixed word nor realizability of guessed controls is used. It holds on rejecting paths as well. Only the explicit coin-read instruction is a query. Bookkeeping may

depend on a bit already stored in the verifier's configuration, but does not query that coordinate again and gives epsilon transitions. On an accepting branch the potential reaches zero, so all T coordinates have been queried exactly once, possibly out of order.

Correctness and unambiguity. For fixed r, an accepting verifier path certifies every leaf and every local production of its tree. Proposition 3.1 then gives success of the root by induction from leaves to root. Conversely, the root's successful tree supplies a valid guess at every binary node, and the prescribed traversal verifies every node and accepts. This proves acceptance exactly on source-accepting coin words. Such a word has one successful tree, one traversal order, and one encoding of each intermediate control. Hence it has exactly one accepting verifier path. Nondeterministic guesses are not given a probability distribution and cause no loss in acceptance probability of the represented Boolean function.

Configuration graph. Layer the verifier's configuration graph by its physical step number, bounded by the polynomial just proved. This adds an $O(\log n)$-bit graph coordinate; it does not assume a unit-cost counter increment inside the verifier. Use complete verifier configurations as vertices, together with this layer index, including the current descriptor, the entire pending list, scratch work, stored query answers, all head positions, and the instruction pointer. Keep vertices reachable from the initial configuration by syntactic transitions, allowing either label at a query. Every nonhalting edge advances the layer index, so the graph is acyclic. A query instruction has one edge labeled 0 and one labeled 1; all deterministic internal steps and binary guess choices occur at epsilon vertices. Thus a query vertex has no additional epsilon choice. Even when both query labels have the same effect, a fixed r selects exactly one of these differently labeled edges. Distinct syntactic choices are retained as distinct edges.

Every graph path from the start is a verifier execution: its legal continuations depend only on the complete stored configuration. Conversely, every verifier execution determines one graph path. Merging equal layered configurations therefore does not identify edge sequences, and preserves both the preceding read-once invariant and the number of accepting paths for each r. Add one fresh accepting sink, with exactly one epsilon edge to it from each accepting halting configuration and no outgoing edges. Rejecting configurations have no outgoing edges. This does not introduce a cycle, a repeated query, or an additional accepting choice. The configurations have $O(\log^2 n)$ bits in total, so their number, and hence the program size, is $2^{O(\log^2 n)}$. The constants and the physical time bound are uniform over x of length n. Reachability restriction here is a definition of the analytical graph, not an operation demanded of the final simulator. □

4 A generator for unambiguous read-once programs

We analyze the restriction recurrence of Forbes–Kelley [9] using the adaptive-order Fourier method of [4, Section 4]. The latter paper's Definition 6 concerns deterministic programs reading every coordinate exactly once. Here the analytical programs may be nondeterministic, have epsilon edges, and omit coordinates. We therefore prove the required separation, Fourier identity, and restriction closure for this precise class instead of invoking the deterministic theorem for it. The generator depends only on numerical size and error bounds, not on the individual program.

4.1 The program model and prefix–suffix separation

Definition 4.1. For an integer $R \geq 1$, a branching program on R bits is a finite acyclic directed multigraph with a designated start vertex and a designated accepting sink. Every vertex is either a query vertex or an epsilon vertex; the accepting sink is an epsilon vertex with no outgoing edges. The start may equal that sink. A query vertex v has an index $i(v) \in I_R$ and outgoing edges labeled in $\{0, 1\}$. An epsilon vertex has only unlabeled outgoing edges. Several edges may have the same label, and parallel edges are distinct. A path is consistent with r when each query-edge label equals the corresponding bit. Paths are distinguished by their edge sequences, including parallel-edge choices. The length-zero path from a vertex to itself is counted once. Acceptance means existence of a consistent start-to-accept path; a maximal path ending anywhere else rejects.

The program is *unambiguous* if each input has at most one consistent accepting path. It is *syntactic read-once* if every start-to-accept graph path queries each index at most once, without assuming consistency. Size is the number of vertices. Constant functions and empty accepting-path subgraphs are permitted.

For integers $R, S \geq 1$, let $\mathcal{U}(R, S)$ denote the programs of Definition 4.1 that are *both unambiguous and syntactic read-once* and have at most S vertices. All program claims below concern this class. Fix a program in $\mathcal{U}(R, S)$ with Boolean acceptance function P. Analytically delete vertices not on a start-to-accept graph path. This preserves P and does not increase size; it is not an operation required of the generator. If no accepting path remains, $P = 0$ and all error bounds are immediate. Otherwise retain the induced subgraph on the remaining vertices. Every retained prefix extends to a retained accepting path, and every retained suffix has a retained prefix. All subsequent vertex and path notation refers to this subgraph.

For a retained v, let $\mathrm{Pre}(v)$ be the union of indices queried on start-to-v paths, excluding the query at v. Let $\mathrm{Post}(v)$ be the union on v-to-accept paths, including the query at v if present. Let $F_v(r)$ and $Q_v(r)$ count consistent prefixes and accepting suffixes, respectively. A length-zero prefix or suffix counts once when applicable. A query at v belongs to the suffix, not the prefix. In particular, $i(v) \in \mathrm{Post}(v)$ for a retained query vertex.

Lemma 4.2. *For each retained v, $\mathrm{Pre}(v) \cap \mathrm{Post}(v) = \varnothing$. Both F_v and Q_v are Boolean. They depend only on coordinates in $\mathrm{Pre}(v)$ and $\mathrm{Post}(v)$, respectively.*

Proof. Any prefix and suffix concatenate to a graph path: a shared vertex other than v would create a directed cycle. An index in both unions would therefore create a start-to-accept path querying it twice.

Suppose two prefixes are consistent with one assignment. Choose any retained suffix. Since v is retained, it extends to a start-to-accept path and hence, by syntactic read-once, has internally consistent labels. Its coordinates are disjoint from all prefix coordinates. Satisfy its labels while retaining the assignment on $\mathrm{Pre}(v)$, giving two accepting paths on one input. This contradicts unambiguity. The symmetric argument uses any one retained prefix to exclude two compatible suffixes: preserve their assignment on $\mathrm{Post}(v)$ and set the disjoint prefix coordinates to satisfy a fixed prefix. Thus $F_v(r), Q_v(r) \in \{0, 1\}$ on every input; the bound holds for each factor separately, not merely for the product $F_v Q_v$. Coordinate dependence follows from the definitions. □

4.2 An exact Fourier identity

For $A \subseteq I_R$, write $\chi_A(r) = (-1)^{\sum_{i \in A} r_i}$. For real f on $\{0, 1\}^R$,

$$\widehat{f}(A) = \mathbb{E}[f(U_R)\chi_A(U_R)], \qquad \|f\| * 2^2 = \mathbb{E}[f(U_R)^2] = \sum_A \widehat{f}(A)^2.$$

The characters are an orthonormal basis, so $f = \sum_{A \subseteq I_R} \widehat{f}(A)\chi_A$. All coefficient sums run over subsets of I_R, and all coefficients and L_2 norms use the uniform probability measure. Set $\|f\|_\infty = \max_{r \in \{0,1\}^R} |f(r)|$. Let e_i denote the bit vector that is one exactly at coordinate i; $\oplus$ is coordinatewise addition modulo two. For an integer $k \geq 1$ and a query vertex v, define

$$H_v = \sum *|A| = k\widehat{F}_v(A)\chi_A, \qquad G_v(r) = \frac{Q_v(r) - Q_v(r \oplus e_{i(v)})}{2}. \tag{2}$$

The two factors depend on disjoint sets of *input coordinates*: $\mathrm{Pre}(v)$ and $\mathrm{Post}(v)$. This does not mean disjoint supports as functions on the cube. Parseval and Lemma 4.2 give

$$\|H_v\| * 2 \leq 1, \qquad \|G_v\| * \infty \leq \tfrac{1}{2}. \tag{3}$$

Indeed $\|H_v\|_2 \leq \|F_v\|_2 \leq 1$, and G_v is half the difference of two Boolean values. The identity $\chi_A(r \oplus e_i) = (-1)^{\mathbf{1}_{\{i \in A\}}}\chi_A(r)$ gives

$$\widehat{G}_v(A) = \mathbf{1}_{\{i(v) \in A\}}\widehat{Q}_v(A).$$

Thus every nonzero character of G_v contains $i(v)$, which is outside $\mathrm{Pre}(v)$ by Lemma 4.2. When $k > R$, the defining sum for H_v is empty and equals zero.

Let V_{q} denote the set of query vertices in the retained subgraph.

Lemma 4.3 (Fourier cut identity)**.** *For the fixed program in $\mathcal{U}(R, S)$ and every integer $k \geq 1$,*

$$P = \sum_{|A| \leq k} \widehat{P}(A)\chi_A + \sum_{v \in V_{\mathrm{q}}} H_v G_v. \tag{4}$$

Proof. For an accepting graph path ψ, let J_ψ be its query set and $\psi(i) \in \{0, 1\}$ its required label at i. Its literal cube is

$$\mathbf{1}_\psi(r) = \prod_{i \in J_\psi} \frac{1 + (-1)^{\psi(i)}\chi_{\{i\}}(r)}{2}.$$

The empty product is one. Each accepting path queries its coordinates without repetition, so $\mathbf{1}_\psi$ is its consistency indicator. Unambiguity gives $P = \sum_\psi \mathbf{1}_\psi$ pointwise, with paths counted with their edge multiplicities. The coefficient of $\mathbf{1}_\psi$ at A is zero unless $A \subseteq J_\psi$, and otherwise equals $2^{-|J_\psi|}(-1)^{\sum_{i \in A} \psi(i)}$.

Fix $|A| > k$. Each path with $A \subseteq J_\psi$ has a unique vertex v querying the $(k+1)$-st index of A in path order. It satisfies

$$|A \cap \mathrm{Pre}(v)| = k, \qquad i(v) \in A, \qquad A \subseteq \mathrm{Pre}(v) \cup \mathrm{Post}(v). \tag{5}$$

Indeed the k earlier indices belong to $\mathrm{Pre}(v)$. An additional element of $A \cap \mathrm{Pre}(v)$ would occur at or after v on this particular path, since all of A is queried, and would also belong to $\mathrm{Post}(v)$, contradicting Lemma 4.2.

Let $V_A \subseteq V_{\mathrm{q}}$ be the set of vertices satisfying the three conditions in (5). At each $v \in V_A$, concatenation bijects prefix–suffix pairs with full paths through v, retaining edge multiplicities. Since F_v and Q_v depend on the disjoint sets $\mathrm{Pre}(v)$ and $\mathrm{Post}(v)$, orthogonality gives

$$\widehat{F_v Q_v}(A) = \begin{cases} \widehat{F}_v(A \cap \mathrm{Pre}(v))\,\widehat{Q}_v(A \cap \mathrm{Post}(v)), & A \subseteq \mathrm{Pre}(v) \cup \mathrm{Post}(v), \\ 0, & A \not\subseteq \mathrm{Pre}(v) \cup \mathrm{Post}(v). \end{cases}$$

In the second case, averaging a character on a coordinate outside $\mathrm{Pre}(v) \cup \mathrm{Post}(v)$ gives zero. In the first, the averages over $\mathrm{Pre}(v)$ and $\mathrm{Post}(v)$ factor. This justifies the coverage condition in (5), even for paths that omit variables. Expanding the coefficient product at $A \cap \mathrm{Pre}(v)$ and $A \cap \mathrm{Post}(v)$ into path contributions gives exactly the full paths through v querying all of A whose designated cut is v: a path omitting any member of A has zero coefficient at A, and on a path through v querying all of A, every index of $A \cap \mathrm{Pre}(v)$ is queried before v, since an index queried at or after v lies in $\mathrm{Post}(v)$. Thus $|A \cap \mathrm{Pre}(v)| = k$ and $i(v) \in A$ say precisely that v queries the $(k+1)$-st index of A on that path. Summing gives

$$\widehat{P}(A) = \sum_{v \in V_A} \widehat{F}_v(A \cap \mathrm{Pre}(v))\, \widehat{Q}_v(A \cap \mathrm{Post}(v)). \tag{6}$$

These are exactly the coefficients at A of $\sum_{v \in V_q} H_v G_v$. Indeed, H_v depends only on $\mathrm{Pre}(v)$ and G_v only on $\mathrm{Post}(v)$, so the coefficient of $H_v G_v$ at A is $\widehat{H}_v(A \cap \mathrm{Pre}(v))\, \widehat{G}_v(A \cap \mathrm{Post}(v))$ when $A \subseteq \mathrm{Pre}(v) \cup \mathrm{Post}(v)$ and zero otherwise; by (2) and $\widehat{G}_v(A) = \mathbf{1}_{\{i(v) \in A\}} \widehat{Q}_v(A)$, it is nonzero only when $v \in V_A$, and then equals the corresponding summand in (6). The products have no coefficients of degree at most k: their coordinate sets are disjoint, H_v is homogeneous of degree k, and every character of G_v contains $i(v) \notin \mathrm{Pre}(v)$. Comparing all coefficients proves (4). This also covers $k \geq R$, when there are no coefficients of degree greater than k. □

Remark 4.4. The last condition in (5) cannot be dropped for programs omitting coordinates. Consider $P(r_0, r_1, r_2) = r_0 r_1$, querying r_0 first and r_1 only when $r_0 = 1$. Take $k = 1$ and $A = \{0, 1, 2\}$. At the second query v, $\mathrm{Pre}(v) = \{0\}$, $\mathrm{Post}(v) = \{1\}$, $F_v = r_0$, and $Q_v = r_1$. Hence $\widehat{F}_v(\{0\}) = \widehat{Q}_v(\{1\}) = -1/2$. Omitting the coverage condition from (6) would give $(-1/2)(-1/2) = 1/4$, whereas $\widehat{P}(A) = 0$ because P does not depend on r_2. Unambiguity is essential to the path-count argument: two parallel epsilon edges from start to accept give Boolean acceptance one but accepting-path count two. Such a program is outside $\mathcal{U}(R, S)$.

4.3 One restriction and its iteration

A distribution on R bits is d-wise independent, for an integer $d \geq 1$, if every collection of at most $\min(d, R)$ distinct coordinates is uniform. Fix an integer $k \geq 1$. Let D be $2(k+1)$-wise independent and W be $(k+1)$-wise independent. Take D, W, U_R mutually independent, and set

$$Y = D \oplus (W \wedge U_R),$$

with coordinatewise operations. The proof below uses only $2k$-wise independence of D and k-wise independence of W; the extra unit is harmless slack that fixes the coefficient counts in Theorem 4.7.

Lemma 4.5 (One restriction)**.** *For a program in $\mathcal{U}(R, S)$ with acceptance function P,*

$$|\mathbb{E}P(Y) - \mathbb{E}P(U_R)| \leq S 2^{-k/2}. \tag{7}$$

Proof. For a homogeneous degree-k function H,

$$\mathbb{E}_{U_R} H(Y) = \sum_{|A|=k} \widehat{H}(A) \chi_A(D) \mathbf{1}_{\{\forall i \in A,\ W_i = 0\}}.$$

This conditional expectation holds with D, W fixed: averaging $\chi_A(W \wedge U_R)$ leaves one exactly when every W_i on A is zero. In the square, unequal sets A, B average to zero over D, because

$0 < |A \triangle B| \le 2k$. Each diagonal mask event has probability 2^{-k}. Mutual independence permits these separate averages, so

$$\mathbb{E}_{D,W}\big(\mathbb{E}_{U_R} H(Y)\big)^2 = 2^{-k} \sum_{|A|=k} \widehat{H}(A)^2.$$

Cauchy–Schwarz and Parseval now give

$$\mathbb{E}_{D,W} \left|\mathbb{E}_{U_R} H(Y)\right| \le \Big(\mathbb{E}_{D,W}(\mathbb{E}_{U_R} H(Y))^2\Big)^{1/2} = 2^{-k/2}\|H\|_2. \tag{8}$$

When $k > R$, $H = 0$ and the identity still holds.

In (4), each nonconstant character of degree at most k has zero expectation on Y, already by averaging over the independent D, which is at least k-wise independent. The constant term is $\widehat{P}(\varnothing) = \mathbb{E}P(U_R)$. For fixed D, W, the U_R-coordinates used by $H_v(Y)$ and $G_v(Y)$ are disjoint. Consequently the conditional expectation of their product factors; unconditional independence is neither asserted nor needed. Thus

$$|\mathbb{E}_{D,W,U_R} H_v(Y) G_v(Y)| \le \mathbb{E}_{D,W} \left|\mathbb{E}_{U_R} H_v(Y)\right| \left|\mathbb{E}_{U_R} G_v(Y)\right| \le \tfrac{1}{2}\, 2^{-k/2}\|H_v\|_2 \le 2^{-k/2}$$

by (3) and (8). Summing at most S terms proves the assertion. □

Lemma 4.6 (Closure under restrictions)**.** *For a program in $\mathcal{U}(R,S)$ and fixed $\mathbf{d}, \mathbf{w} \in \{0,1\}^R$, the function $r \mapsto P(\mathbf{d} \oplus (\mathbf{w} \wedge r))$ has a program in $\mathcal{U}(R,S)$ with no more vertices than the original program.*

Proof. If $\mathbf{w}_i = 0$, retain only edges labeled $\mathbf{d}_i$ at queries of i, erase their labels, and make these vertices epsilon vertices, possibly with no outgoing edges; such dead ends reject, as permitted by Definition 4.1. If $\mathbf{w}_i = 1$, flip the labels there when $\mathbf{d}_i = 1$. The underlying directed graph is a subgraph of the original acyclic graph and cannot acquire a repeated query. Two accepting paths for r would give two original paths for $\mathbf{d} \oplus (\mathbf{w} \wedge r)$. Conversely, each original path consistent with that substituted word gives one retained path consistent with r. This proves equality of the computed functions, not just preservation of unambiguity. Distinct paths and parallel edges remain distinct; no epsilon elimination or merging is performed. Analytical trimming may follow. □

Theorem 4.7 (Generator)**.** *Let $R, S \ge 1$ be integers and $0 < \varepsilon < 1$. Set*

$$\ell = \lceil \log(2R/\varepsilon) \rceil, \qquad k = \lceil 2\log(2S\ell/\varepsilon) \rceil, \qquad a = \lceil \log(R+1) \rceil. \tag{9}$$

There is a single generator $\mathcal{G}_{R,S,\varepsilon} : \{0,1\}^s \to \{0,1\}^R$ that ε-fools every program in $\mathcal{U}(R,S)$, with seed length

$$s = 3\ell(k+1)a = O\big(\log(2R/\varepsilon)\log((2S/\varepsilon)\log(2R/\varepsilon))\log(R+1)\big)\,. \tag{10}$$

Precisely, for every acceptance function P in this class, $|\mathbb{E}P(\mathcal{G}_{R,S,\varepsilon}(U_s)) - \mathbb{E}P(U_R)| \le \varepsilon$. The generator may depend on R, S, ε, but not on P.

Proof. Take mutually independent masks $D_1, W_1, \ldots, D_\ell, W_\ell$, with D_j being $2(k+1)$-wise and W_j being $(k+1)$-wise independent as fixed before Lemma 4.5, and put $\mathcal{R}_j(r) = D_j \oplus (W_j \wedge r)$. Define

$$Z_0 = 0^R, \qquad Z_j = \mathcal{R}_j(Z_{j-1}) \quad (1 \le j \le \ell), \tag{11}$$

and put $\mathcal{G}_{R,S,\varepsilon}(z) = Z_\ell$, where z is the concatenated seed of all masks described below. For $0 \le j \le \ell$, set

$$\Theta_j = \mathbb{E}P(\mathcal{R}_\ell \circ \cdots \circ \mathcal{R}_{\ell-j+1}(U_R)).$$

The expectation is over all displayed masks and an independent U_R; use the empty composition for $j = 0$, so $\Theta_0 = \mathbb{E}P(U_R)$. For $0 \le j < \ell$, fixing the j outside restrictions leaves a test in the same class, by j applications of Lemma 4.6. The next restriction is independent of them and U_R, so $|\Theta_{j+1} - \Theta_j| \le S2^{-k/2}$ by Lemma 4.5, after averaging the conditional bound over those outside masks. Consequently, $|\Theta_\ell - \Theta_0| \le \ell S2^{-k/2}$. Now couple the innermost U_R in Θ_ℓ to 0^R, keeping the restrictions fixed. For coordinate i, the XOR difference $\eta_{j,i}$ between the two coupled iterates satisfies $\eta_{0,i} = U_{R,i}$ and $\eta_{j,i} = W_{j,i} \wedge \eta_{j-1,i}$. Thus coordinate i can change only if $W_{j,i} = 1$ for every j; each $W_{j,i}$ is uniform because W_j is at least 1-wise independent, so this event has probability $2^{-\ell}$ by independence between rounds. No independence between different coordinates is required for this step. A union bound and the Boolean range of P give

$$|\mathbb{E}P(Z_\ell) - \mathbb{E}P(U_R)| \le \ell S2^{-k/2} + R2^{-\ell} \le \varepsilon. \tag{12}$$

The parameter choices give $R2^{-\ell} \le \varepsilon/2$ and $2^{-k/2} \le \varepsilon/(2S\ell)$, proving the last inequality.

Use R distinct points of $\mathbb{F}_{2^a}$ to sample the masks exactly. There are enough points because $2^a \ge R+1$. For d-wise independent bits, choose d independent uniform field coefficients and evaluate the polynomial of degree *at most* $d-1$ at these points. Project each value using a fixed nonzero $\mathbb{F}_2$-linear functional. For any $h \le \min(d, R)$ points, the first h coefficient columns are an invertible Vandermonde matrix, so the evaluations are independent uniform field elements: condition on the remaining coefficients and invert that matrix on the first h. This argument is valid even if $d > R$ or $d > 2^a$. A nonzero linear functional onto $\mathbb{F}_2$ has equally sized fibers, so the projected bits are independent and uniform. Leading coefficients are allowed to be zero; requiring degree exactly $d-1$ would change this sampling distribution.

Use disjoint coefficient seeds for every mask and round. The number of field coefficients is $\ell(2(k+1)+(k+1))$, giving the exact seed length in (10), even if some seed coordinates are redundant. Fix the field representation, ordered evaluation points, and projection independently of P; then the construction is a deterministic function of its seed. The asymptotic estimate follows from (9); all its logarithms are positive throughout the stated parameter range. □

4.4 Uniform indexed evaluation

Lemma 4.8. *Let R be a fixed integer-valued function satisfying $1 \le R(n) \le n^{O(1)}$ for $n \ge 2$, whose binary value can be computed from 1^n in logarithmic space. Fix a positive integer C. Put $b(n) = C\lceil \log(n+2)\rceil^2$. There is a common generator $\mathcal{G}_n : \{0,1\}^{s(n)} \to \{0,1\}^{R(n)}$ for all programs in $\mathcal{U}(R(n), 2^{b(n)})$, with error at most $1/8$, seed length $s(n) = O(\log^4 n)$, and the following uniform property: given x, a seed z of length $s(n)$ on a two-way read-only tape, and an output index $i \in I_{R(n)}$, the bit $\mathcal{G}_n(z)_i$ is computable in polynomial time and logarithmic work space, without a pushdown store. Here $n = |x|$. The generator depends on x only through n. Its evaluator is one fixed deterministic machine for the chosen R and C; it is not supplied with a branching program.*

Proof. Write $R = R(n)$. Use

$$\ell = \lceil \log(16R)\rceil, \quad k = 2\big(b(n) + \lceil \log(16\ell)\rceil\big), \quad a = \lceil \log(R+1)\rceil.$$

For every $S \le 2^{b(n)}$,

$$\ell S2^{-k/2} \le \ell\, 2^{-\lceil \log(16\ell)\rceil} \le 1/16, \qquad R2^{-\ell} \le 1/16.$$

Thus the proof of Theorem 4.7 applies verbatim with these ℓ, k, a: it uses only $k \ge 1$, $2^a \ge R+1$, and the bound in (12), whose two terms are now each at most $1/16$, giving error at most $1/8$. These integer parameters are computed from fixed bounds without storing the quasipolynomial number

S. They satisfy $\ell = O(\log n)$, $k = O(\log^2 n)$, $a = O(\log n)$, and $s = 3\ell(k+1)a = O(\log^4 n)$. No real-number logarithm computation is needed: for a positive integer ν, $\lceil \log \nu \rceil$ is the binary length of $\nu - 1$, with length zero for zero. These parameters and all seed offsets are logspace computable.

A canonical field. Use the polynomial basis of $\mathbb{F}_2[X]/(g)$ for a monic irreducible polynomial g of degree a. Such a polynomial exists over $\mathbb{F}_2$ for every $a \geq 1$. Identify a monic degree-a candidate $X^a + \sum_{h<a} g_h X^h$ with the integer $\sum_{h<a} g_h 2^h$ and take the irreducible candidate with the smallest integer: enumerate candidates in increasing order and test division by every monic polynomial of degrees $1, \ldots, \lfloor a/2 \rfloor$. A reducible polynomial has a factor in that range, so the test is necessary and sufficient. There are 2^a candidates and fewer than $2^{\lfloor a/2 \rfloor + 1}$ divisors to test per candidate. Binary polynomial long division uses $O(a)$ bits and polynomial time in a; the entire search therefore takes at most $2^{3a/2+O(1)}\,\mathrm{poly}(a)$ time and $O(a)$ work, polynomial in n since $2^a \leq 2(R+1) = n^{O(1)}$. For $a = 1$ the divisor range is empty, as required.

Represent point i by the residue class of $\sum_{h=0}^{a-1} i_h X^h$, where i_h is its h-th binary bit, i_0 being the least significant, and project field values to their constant basis coefficient. This is a nonzero linear functional, and the points $0 \leq i < R$ are distinct.

Seed layout and indexed output. Order the seed by rounds $j = 1, \ldots, \ell$, with the $2(k+1)$ coefficients of D_j followed by the $k+1$ coefficients of W_j. Within each polynomial, list coefficients in increasing degree, each as a bits in increasing basis degree. Coefficient h of D_j starts at offset $3(j-1)(k+1)a + ha$ for $0 \leq h < 2(k+1)$, and coefficient h of W_j starts at $3(j-1)(k+1)a + 2(k+1)a + ha$ for $0 \leq h < k+1$. All offsets are zero-based.

Evaluate the two field values by Horner's rule in decreasing coefficient order, reading coefficients from these indexed seed locations, and write $D_j(i), W_j(i) \in \{0,1\}$ for their constant basis coefficients; these are the projected mask bits $D_{j,i}$ and $W_{j,i}$ of Theorem 4.7. Keep a one-bit accumulator initially zero and update it by $c \leftarrow D_j(i) \oplus (W_j(i) \wedge c)$ for $j = 1, \ldots, \ell$. Induction on j shows that the accumulator equals coordinate i of Z_j, so the final bit is $\mathcal{G}_n(z)_i$.

Field addition is bitwise exclusive-or. Multiplication followed by reduction modulo g can use polynomial long multiplication and division, with $O(a)$ work and polynomial time in a. A constant number of field registers (including g and the point), coefficient and round indices, and seed-head positions suffice. Including input-length and parameter-computation counters, the total work space is $O(\log n + a + \log(k+1) + \log(\ell+1) + \log(s+1)) = O(\log n)$. The seed is never copied to work storage. Each coefficient can be obtained by scanning the read-only seed with an offset counter; there are $3\ell(k+1)$ coefficient reads per requested output bit, and the seed has length s, so these scans have polynomial cost in n. Together with the field search, this proves the required physical time bound without using a stack.

Indices may be represented by exactly a bits. Incorrect seed lengths and out-of-range or incorrectly encoded indices can be rejected using bounded scans and counters before evaluation. The evaluator obtains $R(n)$ by simulating its prescribed logspace transducer on the virtual unary input 1^n, generated from the input-length counter; it never uses the bits of x for this purpose. All construction choices depend on n, R, and C, not on the bits of x or on the analytical program. No accepting-path trimming, Fourier coefficient computation, or seed enumeration is performed by this evaluator. □

5 Advice-efficient amplification with one stack

This section uses the arithmetic-walk method of Gutfreund–Viola [12] and Fortnow–Klivans [10, Section 4]. The graph acts on seeds, not on configurations of the consumer. Consequently, the argument requires a stack-free seed accessor, but no bound on the number of possible consumer stacks. We give the constant-density hitting argument and the one-stack implementation explicitly; the expander and logarithmic-space arithmetic are established ingredients, not new constructions.

Lemma 5.1 (Virtual read-only tapes)**.** *Fix a common size parameter $n \geq 2$. A consumer runs in polynomial time in n, uses $O(\log n)$ work space and at most one pushdown store, and has a fixed number of physical read-only input tapes of polynomial length. Suppose it also reads an endmarked virtual string* w *of polynomial length. Its length is supplied or computable in logarithmic space. A fixed deterministic provider, given an index and any fixed parameters of* w *on read-only tapes or in $O(\log n)$ work space, returns the indexed bit of* w *in polynomial time and $O(\log n)$ work, without accessing the store. Then the virtual tape can be eliminated with polynomial time, $O(\log n)$ work and the same store bound.*

The provider must specify the same string on every request during one consumer run, including repeated requests for a bit. The same conclusion holds for a fixed number of such tapes and a fixed depth of nested providers satisfying these conditions. In particular, a deterministic consumer remains deterministic.

Proof. Keep the consumer's work tape, finite control and head positions in a logarithmic simulation region. A virtual head position, including endmarkers, requires $O(\log n)$ bits. At a virtual read, suspend the consumer and run the provider from its initial configuration in a separate logarithmic region; then return its one-bit answer. Endmarkers are supplied directly from the virtual length and head position. Restore any shared physical read-only heads by scanning to their saved positions. No part of the consumer's store is copied, read or modified during a provider call. Induction on the consumer's steps proves an exact simulation, since every virtual read returns the specified symbol.

If the consumer makes at most n^{c_1} requests and each provider call, including all restoration scans, costs at most n^{c_2} steps, their contribution is at most $n^{c_1+c_2}$ after enlarging fixed exponents if necessary. At fixed nesting depth, only a constant number of logarithmic regions and saved head positions coexist; multiplying a fixed number of polynomial bounds still gives polynomial time. No claim is made for a nesting depth that grows with n. □

Lemma 5.2 (Linear-advice amplification)**.** *Let $s : \mathbb{N} \to \mathbb{N}$ satisfy $s(n) \leq (n+2)^{d_0}$ for some fixed integer $d_0 \geq 1$. Suppose a fixed deterministic AuxPDA $V(x, z)$, on every input x of length n and every seed z of length $s(n)$, halts within a fixed polynomial in $n + 2$, uses $O(\log(n + 2))$ work space and at most one store, and has two-way endmarked read-only seed access. If $\mathcal{L} \subseteq \{0,1\}^*$ satisfies*

$$x \notin \mathcal{L} \implies V(x,z) = 0 \text{ for every } z \in \{0,1\}^{s(n)}, \qquad x \in \mathcal{L} \implies \mathbb{P}_{z \sim U_{s(n)}}[V(x,z) = 1] \geq \tfrac{3}{8},$$

then $\mathcal{L} \in \mathrm{LogDCFL}/O(n + s(n))$. The function s need not be computable: its value will be advised.

Proof. Handle the finitely many inputs with $n < 2$ by hardwiring their membership and using empty advice. In the rest of the proof $n \geq 2$.

A modulus with short residue fields. Put $s = s(n)$ and $s_0 = \lceil s/2 \rceil + 2$. Take the first primes $\mathfrak{p}_1, \ldots, \mathfrak{p}_\theta$, stopping at the first product $m = \prod_{i=1}^{\theta} \mathfrak{p}_i \geq 2^{s_0}$. Such a product exists, and $\theta \leq s_0$ because the product of the first s_0 primes is at least 2^{s_0}. In particular $m^2 \geq 2^{2s_0} \geq 16 \cdot 2^s$. The

standard bound $\mathfrak{p}_\theta = O(\theta \log(\theta+1))$ from [20, Theorem 3 and its corollary], absorbing the finitely many small indices into the constant, and minimality of θ give

$$\log m < s_0 + \log \mathfrak{p}_\theta, \qquad \log m = O(s+1), \qquad \mathfrak{p}_\theta \le (s+2)^{O(1)}, \qquad \log \mathfrak{p}_\theta = O(\log n). \tag{13}$$

Here minimality says $m/\mathfrak{p}_\theta < 2^{s_0}$. Only existence and size bounds are required at this stage; the primes and binary m will be advised. The choice also covers $s = 0$, when the seed is empty.

The expander and its transition operator. On $\Omega = \mathbb{Z}_m^2$, label four forward permutations by ports $0, 1, 2, 3$, respectively:

$$(u_1, u_2) \mapsto (u_1, u_1 + u_2), \quad (u_1, u_1 + u_2 + 1), \quad (u_1 + u_2, u_2), \quad (u_1 + u_2 + 1, u_2).$$

Ports $4, 5, 6, 7$ are their respective inverses:

$$(u_1, u_2) \mapsto (u_1, u_2 - u_1), \quad (u_1, u_2 - u_1 - 1), \quad (u_1 - u_2, u_2), \quad (u_1 - u_2 - 1, u_2).$$

All operations are modulo m; ports $8, \dots, 15$ are identities. Coincident endpoints retain distinct ports. Choosing a port uniformly defines a symmetric stochastic matrix K; the uniform distribution π on Ω is stationary.

Gabber–Galil expansion [11], in the form stated in [10, Definition 5 and Theorem 10], supplies an absolute $0 < \eta \le 1$: for $\Xi \subseteq \Omega$ with $0 < |\Xi| \le m^2/2$, the union of Ξ and its images under the four forward maps has size at least $(1+\eta)|\Xi|$. Thus at least $\eta|\Xi|$ distinct vertices outside Ξ lie in that union. For each such vertex, choose one witness consisting of $w \in \Xi$ and a forward port mapping w to it. Witness pairs chosen for distinct outside vertices are distinct, because a fixed pair has only one image. Hence at least $\eta|\Xi|$ directed ports leave Ξ. The conductance of K therefore satisfies

$$\Phi(K) = \min_{0 < \pi(\Xi) \le 1/2} \frac{\sum_{w \in \Xi,\, w' \notin \Xi} \pi(w) K(w, w')}{\pi(\Xi)} \ge \frac{\eta}{16}.$$

In particular the graph is connected. The discrete Cheeger inequality in [16, Theorem 13.10], applicable since K is symmetric and hence reversible, gives

$$1 - \lambda_2(K) \ge \frac{\Phi(K)^2}{2} \ge \frac{\eta^2}{512}.$$

Moreover, $K = (I + K_0)/2$, where K_0 averages the eight forward and inverse ports. Since K_0 is symmetric and stochastic, its eigenvalues lie in $[-1, 1]$, so those of K are nonnegative. We may therefore fix $\lambda = 1 - \eta^2/512 < 1$ such that all eigenvalues of K on the subspace orthogonal to constants lie in $[0, \lambda]$, independently of m.

Mapping vertices to seeds. For canonical $0 \le u_1, u_2 < m$, let $\sigma(u_1, u_2)$ be the low s bits of $u_1 + m u_2$, in increasing bit-index order, padding higher bit indices with zeros if necessary. The empty string is used when $s = 0$. The integers $u_1 + m u_2$ bijectively range over $\{0, \dots, m^2 - 1\}$. Write $m^2 = \lfloor m^2/2^s \rfloor 2^s + \zeta$ with $0 \le \zeta < 2^s$. Exactly ζ seeds have $\lfloor m^2/2^s \rfloor + 1$ preimages and the other $2^s - \zeta$ seeds have $\lfloor m^2/2^s \rfloor$. Let U_Ω be uniform on Ω. Summing the absolute deviations of the seed probabilities from 2^{-s} and dividing by two gives

$$\mathrm{TV}(\sigma(U_\Omega), U_s) = \frac{\zeta(2^s - \zeta)}{2^s m^2} \le \frac{2^s}{4m^2} \le \frac{1}{64}. \tag{14}$$

The middle inequality uses $\zeta(2^s - \zeta) \le 2^{2s}/4$. Total variation bounds the difference of the probabilities of any event, so for each yes-input the set $\mathcal{A}_x = \{w \in \Omega : V(x, \sigma(w)) = 1\}$ has density at least $3/8 - 1/64 = 23/64 > 1/4$.

One walk for every input of a length. Equip the real functions on Ω with the inner product $\langle f, f' \rangle = m^{-2} \sum_{w \in \Omega} f(w) f'(w)$ and its norm $\| \cdot \|_2$. For a fixed yes-input put $\bar{\mathcal{A}}_x = \Omega \setminus \mathcal{A}_x$, and let Π be multiplication by $\mathbf{1}_{\bar{\mathcal{A}}_x}$, an orthogonal projection. For f supported on $\bar{\mathcal{A}}_x$, decompose $f = \langle f, 1 \rangle 1 + f_\perp$. The spectral bound and Cauchy–Schwarz, using $\langle f, 1 \rangle = \langle f, \mathbf{1}_{\bar{\mathcal{A}}_x} \rangle$ and $\|\mathbf{1}_{\bar{\mathcal{A}}_x}\|_2^2 = \pi(\bar{\mathcal{A}}_x) \le 41/64 < 3/4$, give

$$\langle f, Kf \rangle \le \lambda \|f\|_2^2 + (1-\lambda)\langle f, 1 \rangle^2 \le (\lambda + (1-\lambda)\tfrac{3}{4}) \|f\|_2^2.$$

Put $\rho = \lambda + (1-\lambda)3/4 < 1$. The operator $\Pi K \Pi$ is positive semidefinite; for arbitrary f' the preceding inequality applied to $\Pi f'$ gives $0 \le \langle f', \Pi K \Pi f' \rangle \le \rho \|\Pi f'\|_2^2 \le \rho \|f'\|_2^2$. Consequently its operator norm is at most ρ.

Choose w_0 uniformly on Ω and then choose L independent uniform ports, independently of w_0, to obtain a stationary walk $w_0, \ldots, w_L$. Expanding the matrix product over these trajectories gives

$$\mathbb{P}[w_0, \ldots, w_L \in \bar{\mathcal{A}} * x] = \langle \mathbf{1} * \bar{\mathcal{A}} * x, (\Pi K \Pi)^L \mathbf{1} * \bar{\mathcal{A}} * x \rangle \le \rho^L \|\mathbf{1} * \bar{\mathcal{A}}_x\|_2^2 \le \rho^L. \tag{15}$$

This also holds for $L = 0$, with the zeroth power equal to I. Fix an integer constant C_{w} with $\rho^{C_{\mathrm{w}}} < 1/4$ and set $L = C_{\mathrm{w}}(n+1)$. No attempt is made to optimize this constant. Then $\rho^L < 4^{-(n+1)} < 2^{-n-1}$. A union bound over the at most 2^n yes-inputs of length n shows that the probability of missing even one of their sets is less than $1/2$. Thus there is one starting vertex and one port sequence hitting every $\mathcal{A}_x$ simultaneously. If there are no yes-inputs, any walk suffices. Independence of successive consumer outcomes is neither assumed nor needed.

A concrete advice encoding and its length. For a positive integer ν with binary length $\mathrm{len}(\nu)$, encode it by $0^{\mathrm{len}(\nu)-1}\,\mathrm{bin}(\nu)$, of length $2\,\mathrm{len}(\nu) - 1$, where $\mathrm{bin}(\nu)$ is most-significant-bit first. The first 1 terminates the zero prefix and begins the binary payload; the zero-prefix length determines the payload length, so the code is self-delimiting. Encode a nonnegative integer ν by applying this code to $\nu + 1$.

Let $\mathrm{len}(m) = \lfloor \log m \rfloor + 1$ be the binary length of m. Advise nonnegative-integer headers $s, \theta, \mathrm{len}(m)$, then the $\mathrm{len}(m)$ bits of binary m, most-significant-bit first with leading bit 1, then θ records. Record i consists of the positive-integer code for $\mathfrak{p}_i$ and the residues modulo $\mathfrak{p}_i$ of the two coordinates of the starting vertex w_0, each in exactly $\lceil \log \mathfrak{p}_i \rceil$ bits, most-significant-bit first. Finally advise exactly $L = C_{\mathrm{w}}(n+1)$ four-bit port labels, in walk order. The fixed machine knows C_{w}, an absolute constant determined by η, so a walk-length header is unnecessary. The record lengths allow all fields to be found by rescanning; no table of their positions is stored.

Each record has length $O(\log \mathfrak{p}_i)$, and $\sum_i \log \mathfrak{p}_i = \log m$. The three headers cost $O(\log(n+2))$, since their values are polynomially bounded in n. Hence

$$|\alpha_n| = O\left(n + \log m + \sum_{i=1}^{\theta} \log \mathfrak{p}_i + \log(n+2) \right) = O(n+s). \tag{16}$$

In particular, no $O(\log n)$ allowance is charged separately to every prime regardless of its actual size. The existence argument samples a uniform mathematical vertex, not a uniform binary advice encoding. CRT supplies exactly one legal pair of residue lists for every selected starting vertex.

Indexed residue replay. To recover the two coordinates of $w_j = (u_{1,j}, u_{2,j})$ modulo $\mathfrak{p}_i$, read the two advised initial residues and replay ports $1, \ldots, j$ of the advised walk modulo $\mathfrak{p}_i$ (the ports themselves have labels in $\{0, \ldots, 15\}$). Since $\mathfrak{p}_i$ divides m, reduction from $\mathbb{Z}_m$ to $\mathbb{Z}_{\mathfrak{p}_i}$ commutes with every displayed affine map and its inverse. Induction on j proves that the resulting residues

are exact, including at $j = 0$. Two residues, the small modulus $\mathfrak{p}_i$, indices and scan positions require $O(\log n)$ bits by (13). Each replay has polynomial time cost. This is the Gutfreund–Viola method [12], as used in [10, Lemma 11].

Canonical CRT and seed-bit recovery. We use the standard logarithmic-space arithmetic interface: given distinct polynomially bounded primes and legal residues on read-only input of polynomial length, return a specified bit of the unique represented integer in $[0, \prod_i \mathfrak{p}_i)$. The indexed CRT statement is [10, Theorem 12]; see also [5, Theorems 1.3, 1.5 and 3.3] and the uniform arithmetic results of [14]. Binary addition and multiplication likewise have logarithmic-space indexed output. Here the input size for these routines is polynomial in n, not the numerical value of m.

For completeness, the advised binary m also permits the following direct implementation of the required CRT interface. For one coordinate with residues ϱ_i, put

$$M_i = m/\mathfrak{p}_i, \qquad \mu_i = \varrho_i \, (M_i \bmod \mathfrak{p}_i)^{-1} \bmod \mathfrak{p}_i, \qquad \mathcal{S} = \sum_{i=1}^{\theta} \mu_i M_i.$$

On the selected advice the primes are distinct and $m = \prod_i \mathfrak{p}_i$, so all inverses exist. Each $0 \le \mu_i < \mathfrak{p}_i$, whence $0 \le \mathcal{S} < \theta m$. CRT implies that the canonical coordinate is

$$\mathcal{X} = \mathcal{S} - \kappa m, \qquad \kappa = \lfloor \mathcal{S}/m \rfloor \in \{0, \dots, \theta - 1\}.$$

The integer κ, unlike $\mathcal{X}$, fits on the logarithmic work tape.

Here are explicit bit operations; their operands of polynomial bit length are virtual read-only strings. To obtain a bit of M_i, scan binary m from most to least significant bit, performing long division with a remainder less than $\mathfrak{p}_i$. Its remainder and scan index use $O(\log n)$ bits. The quotient bit produced at scan position ι, counting from the most significant bit of m with $0 \le \iota < \mathrm{len}(m)$, is bit $\mathrm{len}(m) - 1 - \iota$ of M_i. A complete scan also verifies that the final division remainder is zero. Streaming the quotient bits modulo $\mathfrak{p}_i$ gives $M_i \bmod \mathfrak{p}_i$. Enumerating the nonzero residues modulo $\mathfrak{p}_i$ finds its inverse; then μ_i is a small modular product, with ϱ_i supplied by replay.

For multiplication of two operands of at most ϖ bits, compute columns from bit zero through the requested bit. Each column adds at most ϖ bit products to the previous carry, outputs the parity and halves the result to get the next carry. The carry is at most ϖ, so it needs only $O(\log(\varpi + 1))$ bits. For summing θ operands, the analogous incoming carry is at most $\theta - 1$. All operand bits are regenerated when needed; bits beyond prescribed operand lengths are zero. Thus a bit of $\mathcal{S}$ is recoverable without storing its terms.

To determine κ, compare $\mathcal{S}$ successively with $0m, 1m, \dots, \theta m$, retaining the last index whose product is at most $\mathcal{S}$. Scan from high to low bits for each comparison. A common length of $\mathrm{len}(m) + \lceil \log(\theta + 1) \rceil + 1$ bits suffices, and $\mathcal{S} < \theta m$ ensures a first strictly larger product by index θ. Subtract κm from $\mathcal{S}$ through the requested position, using a single borrow bit, to recover a bit of $\mathcal{X}$. Finally recover bits of the two coordinates $u_{1,j}, u_{2,j}$ of w_j, and apply the same multiplication and addition procedures to $u_{1,j} + m u_{2,j}$, retaining positions $0, \dots, s - 1$; bits beyond the length of $u_{1,j} + m u_{2,j}$ are zero, as prescribed for σ.

All loops have polynomial bounds in n: $\mathrm{len}(m), \theta, s, \mathfrak{p}_\theta, L$ are polynomially bounded, and every index, carry, small modulus and residue uses $O(\log n)$ bits. The dependency depth of these bit routines is fixed: seed arithmetic calls coordinate recovery, which calls comparisons and sums of products, which call small-modulus division and residue replay. Column computations are iterative: lower columns needed for a requested bit are recomputed sequentially with one carry, not stored or recursively nested. Thus a fixed number of logarithmic regions suffice, and a fixed composition of

polynomial running times remains polynomial by Lemma 5.1. Every seed request returns the same bit for fixed j and advice, and endmarkers are determined by s. No entire vertex, CRT integer or seed is stored, and none of these routines accesses the pushdown store.

The consumer, resets and correctness. Before the first trial, push a protected simulation-bottom marker onto the empty store. For $j = 0, \ldots, L$, simulate $V(x, \sigma(w_j))$ from its prescribed initial configuration, accepting if any trial accepts. Lemma 5.1 supplies the seed tape even if V repeatedly rereads bits while using its store. The trial number j remains fixed during that run. At rejection, clear the simulated store down to the protected marker, blank the consumer's work region, and restore its initial control and head positions. The simulator distinguishes the protected marker from all symbols of V and interprets it as V's empty store; a simulated pop cannot remove the protected marker. Any initial bottom symbol of V is re-created above it. This uses a fixed enlarged alphabet, not a second store.

A fixed bound on replacement length and the polynomial trial time bound imply polynomial store height, so resets take polynomial time. There are $L + 1 = O(n)$ trials. By the fixed-depth accounting above and Lemma 5.1, the seed-bit providers, scans and resets therefore give polynomial total time, $O(\log n)$ work and one store. For the selected advice, each trial is an actual run on a seed of length $s(n)$. Every no-input rejects by the pointwise soundness hypothesis; every yes-input accepts by the common hitting walk.

A total decider on arbitrary input–advice pairs. It remains to define one genuine LogDCFL pair language, rather than a simulation promised well-formed advice. Parse $\langle x, y\rangle$ as in Definition 2.2, rejecting invalid pair encodings, and put $N = 2n + 1 + |y|$. The decider does not try to compute $s(n)$, nor to check that y is the selected advice. Instead choose fixed polynomial caps in $n + 2$, using the bounds above, for header values, prime counts, the binary-modulus length, virtual-tape lengths and all decoded loop bounds.

Before decoding a header or prime, bound its self-delimiting prefix length by a fixed multiple of $\log(n + 2)$; reject a truncated or excessive code. After decoding a prime require $2 \le \mathfrak{p}_i \le (n + 2)^{c_0}$ for a fixed sufficiently large c_0, test primality by trial division, and check pairwise distinctness by rescanning records. Check residue ranges, $\theta \ge 1$, $\mathrm{len}(m) \ge 1$, a canonical positive $\mathrm{len}(m)$-bit modulus, and the exact record and port counts with no trailing bits. These checks need only logarithmic counters and small registers. The long modulus remains read-only.

The direct bit routines above are used with their prescribed length bounds. Before using a quotient M_i, require division of m by $\mathfrak{p}_i$ to have zero remainder; also require the displayed inverse modulo $\mathfrak{p}_i$ to exist. A failed precondition causes rejection. It is unnecessary to verify $m = \prod_i \mathfrak{p}_i$, that the primes are the first θ primes, or that the walk is good: correctness for $\mathcal{L}$ is required only on the selected advice. On other data all routines are still deterministic and explicitly bounded, whether or not their outputs represent CRT coordinates. If any comparison, subtraction or index operation fails its stated range conditions, reject.

Finally enforce V's fixed logarithmic work bound and polynomial transition clock, since an arbitrary header may specify a seed length other than $s(n)$; enforce fixed logarithmic work regions for the providers as well. Arithmetic scans and bit routines have polynomial bounds in $n + 2$, and complete input rescans cost $O(N)$. A fixed polynomial clock in $N + 2$ may additionally guard the whole decider. No guard triggers on the selected advice, after choosing the fixed constants large enough. On every pair these rules give polynomial time in $N + 2$ and $O(\log(N + 2))$ work, with one store. The resulting pair language is in LogDCFL, and (16) proves the lemma. □

6 Proof of the containment

We now verify the hypotheses of the preceding lemmas in order. The branching program is used only to analyze the generator; the actual deterministic simulator executes the normalized source machine.

Proof of Theorem 2.3. Fix $\mathcal{L} \in$ RLogCFL and a source machine M_0, together with its fixed work and time bounds, witnessing Definition 2.1. Apply the normalization of Section 3.1. This gives one fixed normalized machine M and fixed positive integers C_T, d_T such that its number of logical transitions is $T(n) = C_T(n+2)^{d_T}$. In particular, T is positive, polynomially bounded, and logspace computable from 1^n. For now let $n \geq 2$. For every $x \in \{0,1\}^n$,

$$\mathbb{P}_{r \sim U_{T(n)}}[M(x;r) = 1] = \mathbb{P}[M_0(x) = 1].$$

Moreover, if $x \notin \mathcal{L}$, then $M(x;r) = 0$ for every $r \in \{0,1\}^{T(n)}$, by the pointwise soundness observation in Section 2 and its preservation under normalization.

One generator for an entire input length. Set $R(n) = T(n)$. Lemma 3.2 supplies, for every $x \in \{0,1\}^n$, an acyclic unambiguous syntactic read-once program P_x whose Boolean acceptance function is

$$P_x(r) = M(x;r) \qquad (r \in \{0,1\}^{R(n)}).$$

The constant hidden in the compiler's size bound is independent of x. Consequently, one may choose a positive integer C, depending only on the fixed machine M (and hence only on M_0), such that for every $n \geq 2$, every $x \in \{0,1\}^n$, and the corresponding program,

$$\log |P_x| \leq b(n) := C\lceil \log(n+2) \rceil^2.$$

Indeed, the asymptotic bound supplies such an inequality beyond a fixed threshold, and increasing C covers the finitely many remaining lengths uniformly over x. Thus $P_x \in \mathcal{U}(R(n), 2^{b(n)})$ in the notation of Section 4; that class permits the compiler's epsilon edges and does not require that every coordinate be queried, so no property beyond Lemma 3.2 is needed.

For $n \geq 2$,

$$1 \leq R(n) = T(n) = C_T(n+2)^{d_T} \leq C_T 2^{d_T} n^{d_T}.$$

Its binary value is logspace computable from 1^n, so the fixed function R and the fixed integer C satisfy every hypothesis of Lemma 4.8. That lemma gives one generator

$$\mathcal{G}_n : \{0,1\}^{s(n)} \longrightarrow \{0,1\}^{R(n)}, \qquad s(n) = O(\log^4 n),$$

common to every P_x at length n, with

$$\left| \mathbb{P}_{z \sim U_{s(n)}}[P_x(\mathcal{G}_n(z)) = 1] - \mathbb{P}_{r \sim U_{R(n)}}[P_x(r) = 1] \right| \leq \tfrac{1}{8}.$$

Only n, the fixed function R, and the fixed constant C determine the generator, its seed length, and its indexed evaluator. In particular, neither x nor P_x is part of the generator's description. The explicit integer formulas in the proof of Lemma 4.8 make both $s(n)$ and all seed offsets logspace computable. The evaluator need not construct or store the quasipolynomial bound $2^{b(n)}$.

The uniform one-stack seed consumer. Define the acceptance function, for seeds of the prescribed length,

$$V(x,z) = M(x;\mathcal{G}_n(z)), \qquad n = |x|, \quad z \in \{0,1\}^{s(n)}.$$

One fixed deterministic AuxPDA implements this function. It first computes n, $R(n)$, and $s(n)$ with logarithmic counters and rejects if the seed tape does not have exactly $s(n)$ bits. On a correctly sized seed, it runs M chronologically. At logical transition $t \in I_{R(n)}$, suspend its work configuration and heads and obtain bit $\mathcal{G}_n(z)_t$ from the stack-free indexed evaluator of Lemma 4.8. Return that bit as the transition's coin and resume M. The normalization consumes one bit at every logical transition, including those that ignore it. Provider microsteps are not source logical transitions and do not advance the suspended source clock. All transitions are now determined: the source has no choices other than its coins, and the indexed evaluator is deterministic.

For fixed x, z, induction on the logical transitions shows that this procedure has exactly the same source configurations as M on the fixed coin word $\mathcal{G}_n(z)$. This statement is pointwise in z; no independence or freshness of the generated bits is asserted. The evaluator's finite-field work and seed-tape scans use no pushdown store and leave M's store unchanged during each call.

Apply Lemma 5.1 with M as the consumer, the coin word $\mathcal{G}_n(z) \in \{0,1\}^{R(n)}$ as the virtual string, and the indexed evaluator of Lemma 4.8, reading the physical seed tape, as the stack-free provider. The virtual length $R(n)$ is logspace computable. Both the physical seed and the virtual coin tape have polynomial length, and every head position has $O(\log n)$ bits. There are exactly $R(n) = n^{O(1)}$ virtual coin requests. Each provider call has polynomial physical cost, as does the underlying normalized simulation. The resulting machine V consequently runs in polynomial time on every seed of the prescribed length, uses $O(\log n)$ work, and has only the source's one store. Both M and the indexed evaluator are fixed uniform machines, so V is one fixed deterministic machine, not a machine chosen for each input or each length. It neither constructs nor evaluates the analytical branching program. The preliminary length check and all computations before the source simulation are stack-free.

The hypotheses of amplification. For $x \in \mathcal{L}$, the pointwise equality $P_x = M(x;\cdot)$ and the generator error bound give

$$\begin{aligned}\mathbb{P}_{z\sim U_{s(n)}}[V(x,z)=1] &= \mathbb{P}_{z\sim U_{s(n)}}[P_x(\mathcal{G}_n(z))=1] \\ &\geq \mathbb{P}_{r\sim U_{R(n)}}[P_x(r)=1] - \tfrac{1}{8} \\ &= \mathbb{P}[M_0(x)=1] - \tfrac{1}{8} \geq \tfrac{3}{8}.\end{aligned}$$

For $x \notin \mathcal{L}$, every word in $\{0,1\}^{R(n)}$ is rejected by $M(x;\cdot)$. Since $\mathcal{G}_n(z)$ is such a word for every seed,

$$V(x,z) = 0 \qquad \text{for every } z \in \{0,1\}^{s(n)}.$$

This zero-error assertion follows from pointwise soundness, not from subtracting the generator's error estimate. The success density is with respect to uniform seeds; it does not count verifier guesses or require uniformity of the generated coin word.

Since $s(n) = O(\log^4 n)$, one can choose a fixed integer $d_0 \geq 1$ such that $s(n) \leq (n+2)^{d_0}$ for every $n \geq 2$: first choose an exponent valid beyond the asymptotic threshold, and then increase it, if necessary, to cover the finitely many remaining lengths. Extend s to $n = 0, 1$ by setting $s(n) = 0$, and extend the single machine V by hardwiring the membership bits of the three words of those lengths. Thus $V(x,\epsilon) = 1$ exactly when $x \in \mathcal{L}$ for $|x| < 2$. The time, work, store, completeness, and

pointwise-soundness hypotheses of Lemma 5.2 now hold for every input length. The lemma yields one pair language $\mathcal{B} \in \mathrm{LogDCFL}$ and length-dependent advice strings α_n of length $O(n + s(n))$ with

$$x \in \mathcal{L} \iff \langle x, \alpha_{|x|} \rangle \in \mathcal{B}.$$

The advice is shared by all inputs of a length; its computation is not required. The total-machine guards in Lemma 5.2 ensure the stated resources for $\mathcal{B}$ on every input–advice pair, not merely on the selected advice.

Linear advice and the final resource measure. Since $(\log n)^4/n \to 0$, the bound $s(n) = O(\log^4 n)$ implies that there is a constant c_s with $s(n) \le c_s n$ for every $n \ge 2$. Hence $|\alpha_n| = O(n + s(n)) = O(n)$ on those lengths. Choose $\alpha_0 = \alpha_1 = \epsilon$; membership there was hardwired, and these choices satisfy the literal bounds $|\alpha_0| \le c \cdot 0$ and $|\alpha_1| \le c \cdot 1$ in Definition 2.2. Increasing one global constant c if necessary therefore gives $|\alpha_n| \le cn$ for every $n \in \mathbb{N}$. It follows that

$$\mathcal{L} \in \mathrm{LogDCFL}/O(n).$$

On selected advice the encoded pair length is $N = 2n + 1 + |\alpha_n| = O(n + 1)$, so $\mathcal{B}$ uses polynomial time in $n + 2$, $O(\log(n + 2))$ work and one store. Its bounds on arbitrary advice remain polynomial time in $N + 2$ and $O(\log(N + 2))$ work, exactly as required by Definition 2.2. □

Remark 6.1 (Scope). The proof uses unambiguity to obtain an exact Boolean representation for pseudorandomness analysis. It does not assert deterministic AuxPDA evaluation of arbitrary unambiguous branching programs, nor does it sample first-pop certificates. A unique certificate need not be found with noticeable probability by random guessing; no such claim is needed because the simulator executes M itself.

The source model uses fresh independent bits as in Definition 2.1. For a fixed seed, the generator's output is a fixed word; under a uniform seed, its coordinates may be correlated. This word is consumed in the normalized source's chronological order. Seeds and advice are available on two-way read-only tapes, and only indexed bits of larger virtual objects are recovered. The conclusion is the stated nonuniform containment; it does not give a uniform derandomization without advice or a method for computing the selected advice.

Use of AI: The author proved the theorems without using AI, and used AI (Claude, ChatGPT) to verify his proofs and find the missing gaps. The author fixed those missing gaps and iteratively verified and expanded the proofs with AI. The author is solely responsible for the correctness of the proofs.

References

[1] L. M. Adleman. Two theorems on random polynomial time. In *FOCS 1978*, pages 75–83, 1978. https://doi.org/10.1109/SFCS.1978.37.

[2] E. Allender and K.-J. Lange. Symmetry coincides with nondeterminism for time-bounded auxiliary pushdown automata. *Theory of Computing*, 10(8):199–215, 2014. https://doi.org/10.4086/toc.2014.v010a008.

[3] A. Bogdanov, P. A. Papakonstantinou, and A. Wan. Pseudorandomness for linear length branching programs and stack machines. In *APPROX/RANDOM 2012*, LNCS 7408, pages

447–458, 2012. https://doi.org/10.1007/978-3-642-32512-0_38. Extended version: https://andrejb.net/pubs/branching.pdf. The section, lemma and appendix numbers cited here refer to the extended version.

[4] L. Chen, X. Lyu, A. Tal, and H. Wu. New PRGs for unbounded-width/adaptive-order read-once branching programs. In *ICALP 2023*, LIPIcs 261, Article 39, 2023. https://doi.org/10.4230/LIPIcs.ICALP.2023.39.

[5] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *RAIRO–Theoretical Informatics and Applications*, 35(3):259–275, 2001. https://www.numdam.org/item/ITA_2001__35_3_259_0.pdf.

[6] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, 1971. https://doi.org/10.1145/321623.321625.

[7] S. A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *STOC 1979*, pages 338–345, 1979. https://doi.org/10.1145/800135.804426.

[8] M. David, P. Nguyen, P. A. Papakonstantinou, and A. Sidiropoulos. Computationally limited randomness. In *Innovations in Computer Science (ICS 2011)*, pages 522–536, Tsinghua University Press, 2011. https://www.cs.toronto.edu/~pnguyen/studies/ics_camera_ready.pdf.

[9] M. A. Forbes and Z. Kelley. Pseudorandom generators for read-once branching programs, in any order. In *FOCS 2018*, pages 946–955, 2018. https://doi.org/10.1109/FOCS.2018.00093; arXiv:1808.06265.

[10] L. Fortnow and A. R. Klivans. Linear advice for randomized logarithmic space. ECCC TR05-042, revision 1, May 2005. https://eccc.weizmann.ac.il/report/2005/042/revision/1/download/. Conference version: *STACS 2006*, LNCS 3884, pages 469–476. https://doi.org/10.1007/11672142_38. The theorem numbers cited here refer to the ECCC revision.

[11] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981. https://doi.org/10.1016/0022-0000(81)90040-490040-4).

[12] D. Gutfreund and E. Viola. Fooling parity tests with parity gates. In *APPROX/RANDOM 2004*, LNCS 3122, pages 381–392, 2004. https://doi.org/10.1007/978-3-540-27821-4_34; ECCC TR04-088.

[13] E. Haramaty, C. H. Lee, and E. Viola. Bounded independence plus noise fools products. In *CCC 2017*, LIPIcs 79, Article 14, 2017. https://doi.org/10.4230/LIPIcs.CCC.2017.14. Journal version: *SIAM Journal on Computing*, 47(2):493–523, 2018.

[14] W. Hesse, E. Allender, and D. A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. https://doi.org/10.1016/S0022-0000(02)00025-900025-9).

[15] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *STOC 1994*, pages 356–364, 1994. https://doi.org/10.1145/195058.195190.

[16] D. A. Levin and Y. Peres, with contributions by E. L. Wilmer. *Markov Chains and Mixing Times*, second edition. American Mathematical Society, 2017. https://pages.uoregon.edu/dlevin/MARKOV/mcmt2e.pdf.

[17] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, 1995. https://doi.org/10.1006/inco.1995.1064.

[18] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. https://doi.org/10.1007/BF01305237.

[19] N. Nisan. RL $\subseteq$ SC. In *STOC 1992*, pages 619–623, 1992. Journal version: *Computational Complexity*, 4(1):1–11, 1994. https://doi.org/10.1007/BF01205052.

[20] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64–94, 1962. https://doi.org/10.1215/ijm/1255631807.

[21] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978. https://doi.org/10.1145/322077.322083.

[22] H. Venkateswaran. Derandomization of probabilistic auxiliary pushdown automata classes. Georgia Tech College of Computing Technical Report GT-CS-09-06, revised and corrected manuscript, 19 March 2009. https://repository.gatech.edu/bitstreams/5b6cba24-69a3-4639-8f1d-b3f187a1f343/download. Preliminary version: *CCC 2006*, pages 355–370. https://doi.org/10.1109/CCC.2006.16. The section and definition numbers cited here refer to the revised manuscript.