Pseudorandomness for linear length branching programs and stack machines

Andrej Bogdanov* Periklis Papakonstantinou†§ Andrew Wan‡§

Abstract

We show the existence of an explicit pseudorandom generator G of linear stretch such that for every constant k, the output of G is pseudorandom against:

- *Oblivious* branching programs over alphabet $\{0,1\}$ of length kn and size $2^{O(n/\log n)}$ on inputs of size n.
- *Non-oblivious* branching programs over alphabet Σ of length kn, provided the size of Σ is a power of 2 and sufficiently large in terms of k.
- The well-studied model of logarithmic space randomized Turing Machines (over alphabet $\{0,1\}$) extended with an unbounded stack that make k passes over their randomness.

The construction of the pseudorandom generator G is the same as in our previous work (FOCS 2011). The results rely on a stronger analysis of the construction. For the last result, we give a length-efficient simulation of such stack machines by non-deterministic branching programs (over large alphabet) whose accepting computations have a unique witness.

*andrejb@cse.cuhk.edu.hk. Department of Computer Science and Engineering and Institute for Theoretical Science and Communications, Chinese University of Hong Kong. Work partially supported by RGC GRF grant CUHK410309.

†papakon@tsinghua.edu.cn. Institute for Theoretical Computer Science at IIIS, Tsinghua University.

‡andrew@tsinghua.edu.cn. Institute for Theoretical Computer Science at IIIS, Tsinghua University.

§This work was supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, the National Natural Science Foundation of China Grant 61050110452, 61150110163, 61033001, 61061130540, 61073174.

1 Introduction

We consider the problem of constructing an explicit pseudorandom distribution for branching programs of bounded width. A branching program with input symbols from the alphabet Σ, is a directed acyclic graph with a unique start vertex, where every non-sink vertex is labeled by one of n variables and has $|\Sigma|$ outgoing arcs, each labelled with $\sigma \in \Sigma$, and each sink vertex is labeled by an output value "accept" or "reject." The branching program computes a Boolean function over n variables in the natural way: it begins at the start vertex, reads the value of the variable at that vertex, and follows the corresponding arc to the next vertex. When it reaches a sink vertex, it halts and outputs the corresponding label.

Fix an alphabet Σ. A family of distributions $\mathrm{P}\colon \Sigma^{s(n)} \to \Sigma^n$ is *pseudorandom* with seed length $s(n) < n$, and bias $\epsilon(n)$ for a class of functions $\mathcal{F}$ if for every $f \in \mathcal{F}$ in n inputs,

$$\left|\mathbf{E}_{\mathrm{P}}[f(\mathrm{P})] - \mathbf{E}_{\mathrm{U}}[f(\mathrm{U})]\right| \leq \epsilon(n).$$

where U is the uniform distribution on n symbols.

The problem of constructing explicit, unconditionally pseudorandom distributions for various models of computation has been met with the most success for two types of models, the first being small-depth computation [AB84, AW85, Nis91, Bra10]. The second type is space-bounded computation, for which branching programs play an important role: the computation of a randomized Turing Machine that uses n random bits and space S can be modeled as a width 2^S branching program, where the inputs to the program are the n random bits. The pseudorandom generators constructed by Nisan [Nis92] and by Impagliazzo et al. [INW94] use seed length $O(\log^2 n)$ to fool fixed input-order, poly(n) width, read-once branching programs.

Pseudorandom generators for space-bounded algorithms take advantage of the limited communication that can occur between parts of the computation, and are typically based on the following principle: a space-bounded algorithm records a small amount of information between stages of its computation, so randomness may be *reused* from one stage to the next without substantially altering performance.

However, in the constructions mentioned, the ability to recycle randomness relies not only on limited communication between the computation stages, but also on the nature of its access to the randomness. The random bits cannot be accessed too often and the order in which they are accessed must be known in advance. A natural goal is to construct distributions that remain pseudorandom without these access restrictions.

Recent work [BPW11] makes some progress towards removing these restrictions, giving the first pseudorandom generator (with linear stretch) for read-once branching programs under any ordering of the inputs. However, the access to the randomness is still restricted: the branching program is read-once and *oblivious*, i.e., it reads bits in an order independent of their values.

One motivation for our work comes from the problem of derandomizing log-space stack machines (see Section 4) which make a bounded number of sequential passes over their randomness. These machines were proposed by David et al. [DNPS11] as a model of randomized polynomial time with limited access to randomness.[1] Without the random tape access restriction, randomized stack machines characterize randomized polynomial time [Coo71]. If they are allowed one pass over the randomness, however, such machines can be simulated deterministically. David et al. suggest studying what happens between these two extreme cases.

[1] They are also known as auxiliary pushdown automata, see the full version of [DP10] for terminology.

1.1 Results

In this work we show that the distribution in [BPW11] (with different parameters) is pseudorandom even for bounded-width branching programs that have linear length. In other words, the input symbols may be accessed adaptively and arbitrarily many times, provided that the total number of accesses is $O(n)$.

Theorem 1. *For every $k > 1$ there exist constants ρ, γ, λ and and an explicit pseudorandom distribution family $\mathrm{P} : \Sigma^{(1-\rho)n} \to \Sigma^n$, where $\Sigma = \{0,1\}^\lambda$ so that for every n,*

$$\left|\mathbf{E}_{\mathrm{P}}[F(\mathrm{P})] - \mathbf{E}_{\mathrm{U}}[F(\mathrm{U})]\right| = 2^{-\Omega(n)}$$

for every length kn, width $2^{\gamma n}$ branching program $F\colon \Sigma^n \to \{0,1\}$ over n inputs.

Here the constants ρ, γ, λ are inverse exponential in k; see the end of Section 3.2 for the precise dependence on k.

For oblivious branching programs, we obtain a stronger form of the theorem in which $\Sigma = \{0,1\}$. This theorem is stated and proved as Theorem 2 in Section 3.1.

As an example application, consider the problem of identity testing for linear-size arithmetic formulas (see [KI04]). Let f be a linear-size arithmetic formula on inputs of length n coming from some subset S of a field $\mathbb{F}$. Such a formula can be computed[2] by a boolean oblivious branching program of linear length and width $|\mathbb{F}|^{O(\log n)}$. The Schwarz-Zippel lemma says that if f is nonzero, then $f(\mathrm{U})$ takes value zero with probability at most $\deg(f)/|S|$. By Theorem 2, $f(\mathrm{P})$ takes value zero with probability at most $\deg(f)/|S| - 2^{-\Omega(n)}$, as long as $|\mathbb{F}| \ll 2^{n/(\log n)^2}$.

Our main application of Theorem 1 is in fooling (non-uniform) randomized Turing Machines over alphabet $\{0,1\}$ extended with an unbounded stack, henceforth called *stack machines*, which make a constant number of passes over their randomness tape. As mentioned previously, randomized log-space stack-machines characterize probabilistic polynomial time. David et al. [DNPS11] showed that pseudorandom generators that fool polynomial size circuits of depth $d(n) = \Omega(\log n)$ also fool stack machines that make $2^{O(d(n))}$ passes over their randomness. It is conceivable that one can derandomize stack machines that make a sub-polynomial (and in particular constant) number of passes over the randomness without the full derandomization of BPNC^1.

In Section 4 we show that our pseudorandom distribution fools stack machines that make k sequential passes over their input. This in particular implies that we can replace the random tape of a randomized stack machine (restricted to make k passes over its randomness – and unrestricted in every other tape) with our distribution. Here k is the same constant as in Theorem 1. Previously, no nontrivial simulation was known even for $k = 2$.

1.2 Techniques

Fooling branching programs In order to construct pseudorandomness that can be accessed in arbitrary order, the approach in [BPW11] addresses the issue of limited communication in the following way. Consider the computation as occurring in two halves, where only a bit of information (however, it may be computed in an arbitrary fashion) is remembered from each half. The distribution P constructed in [BPW11] was shown to satisfy the following property:

> For every pair of Boolean functions $f, g : \{0,1\}^{n/2} \to \{0,1\}$ and every equipartition (I, J) of $[n]$, the joint distribution $(f(\mathrm{P}|_I), g(\mathrm{P}|_J))$ is close (in statistical distance) to the distribution $(f(U|_I), g(U|_J))$.

[2] Lemma 1 of [BPW11] shows this for boolean formulas. The extension to larger domains is straightforward.

The distribution output by the base generator of the expander-based construction from [INW94] satisfies the above property for any fixed equipartition such as $\{1, \cdots, n/2\} \cup \{n/2+1, \cdots, n\}$ (but not all at the same time).

The distribution from [BPW11] has the advantage that it is pseudorandom for every equipartition and hence will accomodate access to the inputs under every ordering. In fact, it was observed in [BPW11], without proof, that the distribution remains pseudorandom for any f and g which depend on at most $(1 - \Omega(1))n$ of the input bits. We prove this more general lemma (Lemma 1) in Section 2. In the lemma, inputs to f and g can now be shared, so one might expect that the distribution will remain pseudorandom with multiple accesses.

Now consider an oblivious branching program of length kn. We split the computation into t stages, for some large enough t that will be set later. The result of the computation can then be stated as a sum over w^t products of t Boolean functions, each over nk/t variables. We do not argue that the outputs of these functions look independent; instead, we show in Section 3.1 that each summand can always be rewritten as a pair of functions (f, g), where f and g each depend on at most $(1 - \Omega(1))n$ bits, and then apply Lemma 1.

A more complicated argument is required if the branching program is non-oblivious; under the previous decomposition, a single stage of the computation may depend on all n input symbols. In fact, in this case we do not know how to construct a pseudorandom distribution with symbols from $\{0, 1\}$. However, we can achieve this over any sufficiently large (in terms of k) alphabet Σ, where $|\Sigma|$ is a power of 2. Achieving this over $\{0, 1\}$ is a very interesting open question (interesting even for any non-trivial stretch; e.g. stretching $0.99n$ bits to n). In Lemma 3, we show how to rearrange the paths of the branching program so that the combinatorial argument in Section 3.1 can still be used. Thus, we can express any branching program as a short sum (the size of the summation is substantially larger than in the oblivious case) over pairs of functions that fulfill the conditions of Lemma 1.

In fact, the decompositions we obtain for (oblivious) branching programs are implicit in work of Beame, et al. [BJS01]. That work gives similar decompositions for branching programs in order to prove lower bounds using communication complexity arguments. Accordingly, they decompose a branching program as a disjunction of function pairs, and the conditions on the function pairs are stronger. Our application requires the summation instead of the disjunction; however, the proofs are essentially the same, and we include them here for completeness and simplification. We remark that further decompositions that yielded stronger lower bounds were given in subsequent work [Ajt99, BSSV03], but, to our knowledge, these are not relevant to the constructions here.

Fooling Stack Machines We show that for every constant λ, a log-space stack machine over alphabet $\{0, 1\}$ that makes $k(n)$ passes over its randomness can be simulated by a family of *nondeterministic* branching programs over alphabet $\{0, 1\}^\lambda$ of size $2^{O((\log n)^2)}$ and length $nk(n)$. Moreover, the branching programs can be designed to have unique witnesses; namely, for every accepting input there is exactly one accepting computation path. We observe that our proof of Theorem 1 easily extends to nondeterministic branching programs with unique witnesses, and we conclude that our distribution P is pseudorandom for the corresponding stack machines.

A log-space stack machine computes a polynomial time predicate but it may run in time $2^{n^{O(1)}}$. In [DNPS11] it is shown that given a stack machine that makes $k(n)$ passes over its randomness, for a given input x, there is an advice string and a stack machine that computes the same predicate, runs in time $k(n) \cdot \mathsf{poly}(n)$, and preserves the number of passes over the random tape. Such stack machines can be simulated by small space computations [All89, BCD$^+$89, Ruz80]. Niedermeier and Rossmanith [NR95] give a variant of this simulation that preserves the number of witnesses. However, these simulations fail to preserve the number of accesses to the input, even when the stack machines are equipped with an index tape to access the memory.

We show that with a non-trivial modification to [Ruz80], a randomized stack machine that makes $k(n)$ many passes can be simulated by a non-deterministic branching program with a unique witness that preserves the number of accesses to input bits (but not necessarily the order). More specifically, the branching program recursively verifies a kind of a "proof tree" that the computation accepts. For our purposes it is crucial to ensure that the random tape is not accessed more than $nk(n)$ times.

2 Fooling pairs of functions with shared inputs

In this section we give a distribution P over $\{0,1\}^n$ that looks pseudorandom to any pair of functions $f, g\colon \{0,1\}^n \to [-1,1]$ such that f and g depend on at most $(1-\Omega(1))n$ of their inputs. The construction is identical to the one from our previous work [BPW11], with different parameters. Note, however, that later on we will apply this theorem in two different ways, one of which regards distributions over alphabets other than $\{0,1\}$ (and this is essential for obtaining non-trivial stretch). In [BPW11] we proved that the desired pseudorandomness under the additional restriction that f and g each depend on $n/2$ bit inputs which are *disjoint*. We also remarked (without proof) that our analysis can be extended to the more general case, which is needed for the applications in this work. We now give a proof of that statement.

Our pseudorandom distribution The pseudorandom distribution P has the form $\mathrm{P} = Mz + e$, where M is a fixed $n \times m$ for $m = (1-\rho)\cdot n$ matrix over $GF(2)$ such that every subspace spanned by $\alpha \cdot n$ rows has dimension $\alpha \cdot n - r$, and all operations are over $GF(2)$. Here $z \sim \{0,1\}^m$ is a uniformly random seed, and $e \in \{0,1\}^n$ is chosen independently of z from an ϵ-biased distribution. (Recall that e is ϵ-biased if for every $s \in \{0,1\}^n$, $s \neq 0$, $|\mathbf{E}_e[(-1)^{|\langle s,e\rangle|}]| \leq \epsilon$.)

The existence of an explicit matrix M with the desired properties follows from constructions of binary codes with small list size for list-decoding radius bounded away from $1/2$. We now explain this connection. Recall that a linear code C over $\{0,1\}^n$ is (δ, ℓ) *list-decodable* if for every $x \in \{0,1\}^n$, the number of codewords of C within hamming distance δn of x is at most ℓ. A *parity check matrix* M for C is a $GF(2)$ matrix such that $c^T M = 0$ if and only if c is a codeword of C.

It is easily seen (by substituting α for $1/2$) that the proof of Proposition 1 from [BPW11] yields the following more general statement:

Proposition 1. *Let C be a $(\frac{\alpha}{2}, \ell)$ list-decodable code over $\{0,1\}^n$. Let M be the parity check matrix of C. Then every subset of $\alpha \cdot n$ rows of M spans a vector space over $GF(2)$ of dimension at least $\alpha \cdot n - \log_2(2\ell)$.*

Then we have the following fact, which follows from the Johnson bound and standard constructions of asymptotically good binary linear codes; see Theorems 3.1 and 7.1 from [Gur07].

Proposition 2. *For every $\alpha > 0$ there exists $\rho > 0$ and an explicit matrix M of size $n \times (1-\rho)n$ such that every subset of $\alpha \cdot n$ rows spans a vector space over $GF(2)$ of dimension at least $\alpha \cdot n - r$, with $r = 4\log(4n/(1-\alpha))$.*

The main lemma Now, we prove the main lemma that powers our results in Section 3.

Lemma 1. *For every $\alpha > 0$ there exists $\rho > 0$ and an explicit matrix M of size $n \times (1-\rho)\cdot n$ so that for every pair of (possibly intersecting) ordered sets I, J with $|I|, |J| \leq \alpha n$ and for every pair of functions $f : \{0,1\}^{|I|} \to [-1,1], g : \{0,1\}^{|J|} \to [-1,1]$,*

$$\left|\mathbf{E}_{\mathrm{P}}[f(\mathrm{P}|_I)g(\mathrm{P}|_J)] - \mathbf{E}_{\mathrm{U}}[f(\mathrm{U}|_I)g(\mathrm{U}|_J)]\right| \leq 2^r\epsilon$$

where U *is the uniform distribution over* $\{0,1\}^n$, P *is defined as above, and* $x|_I, x|_J$ *denote the projections of* x *on the sets* I *and* J*, respectively, and* $r = 4 \cdot \log \frac{4n}{1-\alpha}$.

In particular, when $g = 1$, $|\mathbf{E}_{\mathrm{P}}[f(\mathrm{P}|_I)] - \mathbf{E}_{\mathrm{U}}[f(\mathrm{U}|_I)]| \leq 2^r\epsilon$, so the pseudorandom distribution also preserves the marginal probabilities of events, within $2^r\epsilon$, over all subsets of size at most $\alpha \cdot n$.

Proof of Lemma 1. Using Fourier decomposition, for any pair of subsets I, J of $[n]$ with $|I|, |J| \leq \alpha n$, we have

$$\begin{aligned}\mathbf{E}_{\mathrm{P}}[f(\mathrm{P}|_I)g(\mathrm{P}|_J)] &= \mathbf{E}_{z,e}[f((Mz+e)|_I)g((Mz+e)|_J)] \\ &= \sum_{S\subseteq I, T\subseteq J} \hat{f}(S)\hat{g}(T)\mathbf{E}_{z,e}[\chi_S(Mz|_I)\chi_S(e|_I)\chi_T(Mz|_J)\chi_T(e|_J)] \qquad (1)\end{aligned}$$

We may view subsets $S \subseteq I$ and $T \subseteq J$ as subsets of $[n]$, so we write $\chi_S(Mz|_I) = \chi_S(Mz)$ and $\chi_T(Mz|_J) = \chi_T(Mz)$, and (1) becomes:

$$\sum_{S\subseteq I, T\subseteq J} \hat{f}(S)\hat{g}(T)\mathbf{E}_z[\chi_S(Mz)\chi_T(Mz)]\mathbf{E}_e[\chi_S(e)\chi_T(e)].$$

We denote by $S\Delta T$ the symmetric difference of S and T viewed as subsets of $[n]$.

We have that $\mathbf{E}_{\mathrm{U}}[f(\mathrm{U}|_I)g(\mathrm{U}|_J)] = \sum_{S\subseteq I\cap J} \hat{f}(S)\hat{g}(S)$ and $|\mathbf{E}_e[\chi_{S\Delta T}(e)]| \leq \epsilon$, therefore

$$\begin{aligned}\left|\mathbf{E}_{\mathrm{P}}[f(\mathrm{P}|_I)g(\mathrm{P}|_J)] - \mathbf{E}_{\mathrm{U}}[f(\mathrm{U}|_I)g(\mathrm{U}|_J)]\right| &= \left|\sum_{\substack{S\subseteq I, T\subseteq J \\ S\Delta T\neq\emptyset}} \hat{f}(S)\hat{g}(T)\mathbf{E}_z[\chi_{S\Delta T}(Mz)]\mathbf{E}_e[\chi_{S\Delta T}(e)]\right| \\ &\leq \sum_{\substack{S\subseteq I, T\subseteq J \\ S\Delta T\neq\emptyset}} \epsilon \cdot |\hat{f}(S)||\hat{g}(T)|\left|\mathbf{E}_z[\chi_{S\Delta T}(Mz)]\right|.\end{aligned}$$

Let G be a bipartite graph over vertices (subsets of I) $\cup$ (subsets of J), with an edge (S,T) present whenever $\mathbf{E}_z[\chi_{S\Delta T}(Mz)] \neq 0$. We will shortly argue that G has maximum degree 2^r. Assuming this, we can upper bound the last expression by

$$\begin{aligned}\epsilon \cdot \sum_{\text{edge } (S,T)} |\hat{f}(S)||\hat{g}(T)| &\leq \epsilon \cdot \sqrt{\sum_{\text{edge } (S,T)} \hat{f}(S)^2}\sqrt{\sum_{\text{edge } (S,T)} \hat{g}(T)^2} \\ &\leq \epsilon \cdot \sqrt{2^r \cdot \sum_{S\subseteq I} \hat{f}(S)^2}\sqrt{2^r \cdot \sum_{T\subseteq J} \hat{g}(T)^2} \\ &\leq \epsilon \cdot 2^r,\end{aligned}$$

where the first line follows from the Cauchy-Schwarz inequality, the second line follows from the fact that G has maximum degree 2^r, and the third line is an application of Parseval's identity.

It remains to argue that G has maximum degree 2^r. We let $s \in \{0,1\}^n, t \in \{0,1\}^n$ be indicator vectors for the sets S and T, respectively, and s and t as vectors in $GF(2)^n$. Then

$$\mathbf{E}_z[\chi_{S\Delta T}(Mz)] = \mathbf{E}[(-1)^{(s+t)^T Mz}] = \begin{cases}1, & \text{if } (s+t)^T M = 0 \\ 0, & \text{otherwise.}\end{cases}$$

Now, $(s+t)^T M = 0$ if and only if $s^T M = t^T M$, where $s^T M$ is zero at least everywhere outside I and similarly for $t^T M$ and J. Since (by assumption) the matrix $M|_I$ has rank at least $\alpha n - r$, for every $t \in \{0,1\}^n$, there can be at most 2^r distinct vectors $s \in \{0,1\}^n$ such that $s^T M = t^T M$. Similarly, for every $s \in \{0,1\}^n$, there can be at most 2^r vectors $t \in \{0,1\}^n$ such that $s^T M = t^T M$. ■

In Sections 3.2 and 4 we will apply the pseudorandom generator to strings of length n over alphabet $\Sigma = \{0,1\}^\lambda$. These can be viewed as strings in $\{0,1\}^{\lambda n}$ in the natural way.

3 Fooling branching programs of linear length

We show that essentially the same generator (modulo the setting of the parameters and the different input alphabets) fools branching programs of linear length. For oblivious branching programs we obtain this for binary $\{0,1\}$ input alphabets (Section 3.1), whereas for arbitrary branching programs we show this for branching programs over (larger) constant size alphabets (Section 3.2).

Let F be a width w, length kn, layered branching program over n inputs; we think of k as an arbitrarily large but fixed constant as n increases. We view the computation of the branching program on an input x as occurring in t stages, where each stage reads kn/t variables. Suppose first that the branching program is oblivious. Then, for every input each stage reads the *same* kn/t variables. In this case, we may write the branching program as a sum over w^t many t-tuples of Boolean functions (as was done in [BPW11] for $k = 1$ and $t = 2$).

More formally, divide the inputs into t sets of layers so that S_1 consists of inputs $\{1, \cdots, kn/t\}$, S_2 of inputs $\{kn/t+1, \cdots, 2kn/t\}$, etc. (if variables reoccur within a set, its size might be smaller). We define functions $f_{i,p,q}(x|_{S_i}) : \{0,1\}^{|S_i|} \to \{0,1\}$ to be indicator functions for the event that the program moves from state p to q when the inputs in S_i are read from x. By definition, we have

$$F(x) = \sum_{\substack{p_1,\cdots,p_t: \\ p_t \in \text{accept}}} f_{1,s,p_1}(x|_{S_1}) f_{2,p_1,p_2}(x|_{S_2}) \cdots f_{t,p_{t-1},p_t}(x|_{S_t}) \tag{2}$$

3.1 Pseudorandomness for oblivious branching programs

We will argue that each of the summands in (2) can be rewritten in terms of two functions, each over at most αn bits. Then, we apply Lemma 1 to show that the output of the generator fools each of these summands. This will give us the following theorem for oblivious branching programs.

Theorem 2. *Let $F : \{0,1\}^n \to \{1,-1\}$ be computable by a width w, length kn oblivious branching program on n inputs. Let P be the pseudorandom distribution. Then*

$$|\mathbf{E}_{\mathrm{P}}[F(\mathrm{P})] - \mathbf{E}_U[F(U)]| \leq w^t \cdot 2^r \epsilon.$$

where $t = 2^{4k}$, $r = 4 \log \frac{4n}{1-\alpha}$, and $\alpha > 1 - \frac{1}{2^{2k}}$.

The proof of Theorem 2 will use the following combinatorial lemma, which shows that we can always find a way to color each stage by one of two colors, so that neither color will contain too many variables. A slightly different version of this lemma was proven in [BJS01]; we include a proof in Appendix B for simplicity and completeness.

Lemma 2. *Fix any $k \in \mathbb{Z}^+$. Let $\{S_1, \cdots, S_t\}$ be a collection of subsets over $[n]$, each of size at most kn/t. Then there exists a partition $(\mathcal{C}, \overline{\mathcal{C}})$ of $\{1, \cdots, t\}$ satisfying:*

$$\left|\bigcup_{i \in \mathcal{C}} S_i\right| \leq \alpha \cdot n \quad \text{and} \quad \left|\bigcup_{i \in \overline{\mathcal{C}}} S_i\right| \leq \alpha \cdot n$$

where $\alpha \geq 1 - \frac{1}{2^k} + \frac{2k}{\sqrt{t}} + \frac{2}{\sqrt{n}}$.

In Appendix B we argue that our lower bound on α is close to optimal.

Proof of Theorem 2. Now, consider the expected bias of the branching program using Equation 2; by linearity of expectation and the triangle inequality, we have:

$$|\mathbf{E}[F(U)]-\mathbf{E}[F(\mathrm{P})]| \leq \sum_{\substack{p_1,\ldots,p_t:\\ p_t\in\text{accept}}} |\mathbf{E}[f_{1,s,p_1}(\mathrm{P}|_{S_1})\cdots f_{t,p_{t-1},p_t}(\mathrm{P}|_{S_t})]-\mathbf{E}[f_{1,s,p_1}(U|_{S_1})\cdots f_{t,p_{t-1},p_t}(U|_{S_t})]|.$$

For each expectation of the summation, we can apply Lemma 2 to rewrite each product as a product of two functions, i.e.,

$$f_{1,s,p_1}(x|_{S_1})\cdots f_{t,p_{t-1},a}(x_{S_t}) = g_1(x|_{\mathcal{S}})g_2(x|_{\overline{\mathcal{S}}}),$$

where both $\mathcal{S} := \left|\bigcup_{i\in\mathcal{C}} S_i\right| \leq \alpha\cdot n$ and $\overline{\mathcal{S}}$ contain at most $\alpha\cdot n$ variables.

Setting $t = 2^{4k}$ in Lemma 2 and applying Lemma 1 with α from Lemma 2, we bound the magnitude of each difference by $2^r\epsilon$. Since there are w^t terms, we obtain

$$|\mathbf{E}[F(U)] - \mathbf{E}[F(\mathrm{P})]| \leq w^t 2^r\cdot\epsilon. \qquad \blacksquare$$

3.2 Arbitrary linear size branching programs over large alphabets

We show how to fool arbitrary branching programs with inputs over alphabet $\Sigma = \{0,1\}^\lambda$, where λ is a sufficiently large constant which depends on the multiplicative constant k in the length of the branching program.

Lemma 3. *Let $P(x) = P_1(x)\wedge\ldots\wedge P_t(x)$, where $P_1,\cdots,P_t\colon \Sigma^n\to\{0,1\}$ are branching programs of length at most kn/t each. Then, there exist collections of boolean functions $\{F_{\mathcal{C},U}\}$ and $\{G_{\mathcal{C},V}\}$, where $\mathcal{C}$ ranges over all partitions of $\{1,\ldots,t\}$ and U,V range over all subsets of $[n]$ of size αn such that*

$$P(x) = \sum_{\substack{\mathcal{C}\subseteq[t],U,V\subseteq[n]\\ |U|=|V|=\alpha n}} F_{\mathcal{C},U}(x)\cdot G_{\mathcal{C},V}(x) \tag{3}$$

and $\alpha \geq 1-\frac{1}{2^k}+\frac{2k}{\sqrt{t}}+\frac{2}{\sqrt{n}}$.

Proof. We can express every P_i as

$$P_i(x) = \sum_{\ell_i\in\mathcal{L}_i} f_{i,\ell_i}(x)$$

where the summation ranges over $\mathcal{L}_i$ which denotes all accepting paths ℓ_i of P_i and $f_{i,\ell_i}(x)$ is the indicator function for the event that the computation of P_i on input x takes path ℓ_i. We can write

$$P(x) = \prod_{i=1}^t P_i(x) = \prod_{i=1}^t \sum_{\ell_i\in\mathcal{L}_i} f_{i,\ell_i}(x) = \sum_{(\ell_1,\ldots,\ell_t)\in\mathcal{L}_1\times\cdots\times\mathcal{L}_t} f_{1,\ell_1}(x)\cdot\ldots\cdot f_{t,\ell_t}(x).$$

By Lemma 2, for every collection $\ell = (\ell_1,\ldots,\ell_t)$ there exists a partition $\mathcal{C}(\ell)$ of $[t]$ and sets $U(\ell)$ and $V(\ell)$, each of size at most αn, such that when $i\in\mathcal{C}$, $f_{i,\ell_i}(x)$ depends only on inputs in $U(\ell)$ and when $i\in\overline{\mathcal{C}}$, $f_{i,\ell_i}(x)$ depends only on inputs in $V(\ell)$. Without loss of generality we will assume that the sizes of $U(\ell)$ and $V(\ell)$ are exactly αn. We set

$$F_{\mathcal{C},U}(x) = \bigvee_{\substack{\ell:\mathcal{C}(\ell)=\mathcal{C}\\ U(\ell)=U}} \bigwedge_{i\in\mathcal{C}} f_{i,\ell_i}(x) \qquad \text{and} \qquad G_{\mathcal{C},V}(x) = \bigvee_{\substack{\ell:\mathcal{C}(\ell)=\mathcal{C}\\ V(\ell)=V}} \bigwedge_{i\in\overline{\mathcal{C}}} f_{i,\ell_i}(x)$$

We now prove the identity (3). If $P(x) = 1$, then there is a unique path $\ell = (\ell_1, \ldots, \ell_t)$ such that $f_{i,\ell_i}(x) = 1$ for all i, and so $F_{\mathcal{C},U}(x)$ and $G_{\mathcal{C},V}(x)$ both take value 1 when and only when $\mathcal{C} = \mathcal{C}(\ell)$, $U = U(\ell)$, and $V = V(\ell)$. Then exactly one term on the right hand side of (3) evaluates to 1.

If $P(x) = 0$, then $P_i(x) = 0$ for some i, so $f_{i,\ell_i}(x) = 0$ for all accepting paths ℓ_i of P_i. This forces $F_{\mathcal{C},U}(x)$ to equal zero when $i \in \mathcal{C}$, and $G_{\mathcal{C},V}(x) = 0$ when $i \in \overline{\mathcal{C}}$. So all terms on the right hand side of (3) evaluate to 0. ■

To prove Theorem 3 below, we will use Lemma 3 to write the branching program as a sum of a limited number of pairs of functions, where each pair satisfies the desired property. We then use Lemma 1 to bound the deviation of each term in this summation when the uniform distribution is replaced by the pseudorandom one.

Theorem 3. *Let $k > 0$ be a constant, and let λ be sufficiently large given k. Let $F : \Sigma^n \to \{0,1\}$ be computable by a branching program on n inputs of width w and length kn. Then*

$$|\mathbf{E}_{\mathrm{P}}[F(\mathrm{P})] - \mathbf{E}_{\mathrm{U}}[F(\mathrm{U})]| \leq \left(\frac{4\lambda n}{1-\alpha}\right)^4 \cdot w^{2^{4k}} \cdot 2^{2H(\alpha)n} \cdot \epsilon,$$

where P *is the pseudorandom distribution over $\{0,1\}^{\lambda n}$, and $\alpha \geq 1 - \frac{1}{2^{2k}}$.*

Proof. Applying the decomposition (2) we write

$$F(x) = \sum_{\substack{p_1,\ldots,p_t: \\ p_t \in \text{accept}}} f_{1,s,p_1}(x|_{S_1}) f_{2,p_1,p_2}(x|_{S_2}) \cdots f_{t,p_{t-1},p_t}(x|_{S_t}) = \sum_{\substack{p_1,\ldots,p_t: \\ p_t \in \text{accept}}} F_{p_1,\ldots,p_t}(x).$$

Here, $f_{i,p,q}$ are all branching programs of length kn/t. By Lemma 3 we have

$$F(x) = \sum_{\substack{p_1,\ldots,p_t: \\ p_t \in \text{accept}}} \sum_{\mathcal{C},U,V} F_{p_1,\ldots,p_t,\mathcal{C},U}(x) \cdot G_{p_1,\ldots,p_t,\mathcal{C},V}(x). \tag{4}$$

where $\mathcal{C}$ ranges over all partitions of $[t]$, U, V range over all subsets of $[n]$ of size αn, and $F_{p_1,\ldots,p_t,\mathcal{C},U} : \Sigma^{\alpha n} \to \{0,1\}$ and $G_{p_1,\ldots,p_t,\mathcal{C},V} : \Sigma^{\alpha n} \to \{0,1\}$ depend only on inputs coming from U and V respectively. Now, let us view $F_{p_1,\ldots,p_t,\mathcal{C},U}, G_{p_1,\ldots,p_t,\mathcal{C},V}$ as functions with domain $\{0,1\}^{\alpha\lambda n}$. Set $t = 2^{4k}$ and $r = 4\log(4\lambda n/(1-\alpha))$. By Lemma 1 for each term in the sum, the difference in expectations under the uniform and pseudorandom distributions is at most $\epsilon 2^r$ in absolute value. Since there are at most w^t choices for $(p_1, \ldots, p_t)$, 2^t choices for $\mathcal{C}$, and $\binom{n}{\alpha n}$ choices for each of U and V, by the triangle inequality we obtain that

$$|\mathbf{E}_{\mathrm{P}}[F(\mathrm{P})] - \mathbf{E}_{\mathrm{U}}[F(\mathrm{U})]| \leq \epsilon 2^r \cdot w^t \cdot 2^t \binom{n}{\alpha n}^2,$$

which yields the desired bound after substituting the values for r and t and the standard bound for binomial coefficients. ■

Parameters We now set the parameters to obtain Theorem 1. We assume the availability of a family small-biased generators over $\{0,1\}^m$ for bias ϵ and seed length $\log(m/\epsilon)^K$ for some constant K constructible in time polynomial in the seed length (see e.g. [AGHP90] for a construction with $K = 2$). We instantiate this construction with parameters $m = \lambda n$ and $\epsilon = 2^{-4n}$ to obtain a seed length of $4Kn + o(n)$.

Set $\alpha = 1 - 2^{-2k}$. By Lemma 1, there exists a constant 2ρ (depending on α) for which the distribution P can be generated efficiently with seed length $(1-2\rho)\lambda n + 4Kn + o(n)$. Setting $\lambda = 5K/\rho$, the seed length is upper bounded by $(1-\rho)\lambda n$ bits, i.e. $(1-\rho)n$ elements of Σ, when n is sufficiently large.

To calculate the bias, we simplify the upper bound in Theorem 3 to $4\lambda^4 n^4 w^{2^{4k}} \cdot 2^{2n} \cdot \epsilon$. When $w \leq 2^{n/2^{4k}}$, this expression is upper bounded by $4\lambda^4 n^4 \cdot 2^{-n} = 2^{-\Omega(n)}$.

4 Stack machines and branching programs

A *stack machine* is a space-bounded Turing Machine extended with a stack. The stack can be accessed only through its top symbol but is otherwise unbounded (see [Coo71] for a precise definition). In the discussion here, the stack machines will be deterministic in the sense that the next state and stack operation are uniquely determined by the current state, the current tape symbols and the symbol on top of the stack.

A *randomized stack machine* is a stack machine with a read-only, polynomially long, two-way random tape. In this section we sketch a proof that randomized stack machines (with random tape alphabet $\{0,1\}$) that make k passes (where k is any constant) over their randomness can be simulated by families of read k-times nondeterministic branching programs over alphabet $\{0,1\}^\lambda$ for every constant λ. Moreover, the branching programs have *unique witnesses*, in the sense that for every input there is at most one accepting computation path. We observe that our proof of Theorem 1, which concerns deterministic branching programs, extends directly to nondeterministic ones with unique witnesses.

A *nondeterministic branching program* is defined in the same way as a branching program, except that there can be any number of transitions out of a node labeled by the same symbol. Such a branching program accepts input r if there is a choice of transitions consistent with the input that lead from its start state to its accept state. We say a nondeterministic branching program has *unique witnesses* if for every input there are zero or one accepting computation paths.

Lemma 4. *Let S be a log-space stack machine with random tape alphabet $\{0,1\}$ that makes k passes over its random tape. Then, for every λ and x there is a nondeterministic branching program B_x with random tape alphabet $\Sigma = \{0,1\}^\lambda$ of width $2^{O_\lambda((\log n)^2)}$ that reads every random input symbol at most k times such that B_x on input r simulates S on input x and randomness r. Moreover, B_x has unique witnesses.*

We now sketch the proof of this lemma. We will first describe the proof without worrying about the unique witness property, then describe some modifications that provide this additional feature.

To begin with, we may assume without loss of generality that the tape of S consists of $\lceil n/\lambda \rceil = m$ symbols taking values in alphabet Σ instead of n symbols from $\{0,1\}$. This can be shown by a straightforward simulation of which we omit the details.

Notice that there is no a priori running time bound on log-space stack machines.[3] However, the time-compression lemma [DNPS11] allows us at the cost of some advice to bound the running time of a stack machine in time proportional to its number of passes over the random tape. By doing a standard timed simulation, we may also assume without loss of generality that the head of the randomness tape moves in an oblivious manner, i.e. independent of the random bits seen in the computation. It will also be convenient to assume without loss of generality that the stack machine accesses its random tape only k many times, i.e. when it moves its head over the random tape. Finally, we may assume that S empties its stack and enters a "canonical configuration" before accepting. We will also assume that λ is a constant depending only on k. We summarize these simplifications in the following claim.

Claim 1. *Let S be a stack machine with space $O(\log n)$ that makes k passes over its random tape. There exists a stack machine M simulating S and a collection of advice strings $\{a_x, x \in \{0,1\}^*\}$ with $|a_x| = \mathrm{poly}(|x|)$ such that on input (x, a_x), where $|x| = n$, M uses space $O(\log n)$, runs in time $k\mathrm{poly}(n)$, makes k passes over its random tape, the motion of the randomness tape head of M is*

[3] It is possible to show that a halting computation takes at most $2^{\mathsf{poly}(n)}$ steps, but super-polynomial running time seems necessary, since if for every log-space stack machine there is a log-space stack machine that works in quasi-polynomial time then $\mathsf{P} = \mathsf{NC}$ (e.g. [Ruz80, BCD$^+$89]).

independent of the contents of this tape, the transitions depend on the random tape symbols only at the time steps where the random tape head moves, and whenever $M(x, a_x)$ accepts its stack and its work tape are empty and its heads point to the last positions of the respective tapes.

For every $x \in \{0,1\}^*$, we now show how to construct a nondeterministic branching program that simulates M on input (x, a_x). The canonical simulation of $M(x, a_x)$ by a branching program is not width-bounded, as the stack may grow indefinitely. The idea is to use nondeterminism in order to keep the portions of the stack that the branching program has to remember small. The main challenge is doing this while preserving the number of accesses to the random tape symbols.

A *surface configuration* of M on input (x, a_x) consists of the head positions on the tapes, the content of the tapes, the state, the top stack symbol (or the symbol $\perp$ if the stack is empty), *and* the time step of the computation.[4] An *incomplete configuration* is a surface configuration without the contents of the randomness tape; we use $c(r)$ to denote incomplete surface configuration c instantiated with randomness r. The surface configurations of $M(x, a_x)$ can be naturally represented by strings of size $O(\log n)$. Let start and accept denote the unique starting and accepting incomplete surface configurations of M.

We call a pair of configurations $(c_0(r), c_1(r))$ *realizable* if starting from configuration $c_0(r)$ the computation reaches $c_1(r)$ at the same stack height during which the machine never pops a symbol below this stack height. It is easy to see that the realizable pairs can be characterized inductively as follows: (1) For every configuration $c(r)$, the pair $(c(r), c(r))$ is realizable; and (2) If $(c_0(r), c(r))$ is realizable, $(c'(r), c'_1(r))$ is realizable, and there exists a stack symbol σ so that $c'(r)$ follows $c(r)$ by pushing σ and $c_1(r)$ follows $c'_1(r)$ by popping σ, then $(c_0(r), c_1(r))$ is realizable. In this case we will say $(c_0(r), c_1(r))$ *breaks into* $(c_0(r), c(r))$ and $(c'(r), c'_1(r))$.

Applying this characterization recursively, we can break every realizable pair into a binary tree. A *realizable pair tree* is a tree whose nodes are labeled by realizable pairs in the following way. The leaves are labeled by pairs of the form $(c(r), c(r))$, and a node $(c_0(r), c_1(r))$ has children $(c_0(r), c(r))$ and $(c'(r), c'_1(r))$ whenever $(c_0(r), c_1(r))$ breaks into $(c_0(r), c(r))$ and $(c'(r), c'_1(r))$. If a computation path of $M(x, a_x)$ reaches $c_1(r)$ from $c_0(r)$ in t steps, then the corresponding realizable pair tree whose root is labeled by $(c_0(r), c_1(r))$ has $t + 1$ nodes.

Let c_0 and c_1 be a pair of incomplete configurations. Consider the following nondeterministic branching program $B_{c_0,c_1}(r)$ that checks if $(c_0(r), c_1(r))$ is realizable: If $c_0 = c_1$, accept. Otherwise, guess three incomplete configurations c, c', c'_1 such that $(c_0(r), c_1(r))$ breaks into $(c_0(r), c(r))$ and $(c'(r), c'_1(r))$. Accept if both $B_{c_0,c}(r)$ and $B_{c',c'_1}(r)$ accept and reject otherwise.

By construction, the branching program $B_{\text{start,accept}}(r)$ accepts if and only if $S(x)$ accepts with randomness r. Moreover, this branching program preserves the access pattern of $S(x)$ to its random tape (recall that we have assumed that there are $\leq k \cdot m$ many configurations where the stack machine accesses its random tape). However, there are two main issues we need to resolve. First, the width of the branching program could be very large because the depth of the recursion, which equals the depth of the realizable pair-tree whose root is labeled by $(\text{start}(r), \text{accept}(r))$, could be very large. We resolve this by traversing the tree in a balanced way. Second, we must ensure that each symbol of the random tape is accessed at most k times. Note that even rejecting paths of the nondeterministic computation must obey this restriction. To resolve this, we ensure that each time step of the stack machine computation is not simulated more than once. To that end, we maintain a set of active time intervals. Because of the depth of the recursion and the invariants of the simulation, we are able to update this set consistently without ever storing more than $O_\lambda(logn)$ intervals. The formal description of the simulation and the details of the analysis, as well as the modification for unique witnesses are given in Appendix A.

[4] Crucially, in Cook's work [Coo71] the surface configuration does not include the time-step.

Acknowledgements

We are grateful to the anonymous referee that pointed out a fundamental flaw in the proof of a previous version of our main theorem.

References

[AB84] M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computations. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 471–474, New York, NY, USA, 1984. ACM.

[AGHP90] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 544–553, 1990.

[Ajt99] M. Ajtai. A non-linear time lower bound for boolean branching programs. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 60–70. IEEE, 1999.

[All89] E. W. Allender. P-uniform circuit complexity. *Journal of the ACM*, 36(4):912–928, October 1989.

[AW85] M. Ajtai and A. Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *26th Annual Symposium on Foundations of Computer Science*, pages 11–19. IEEE, 1985.

[BCD+89] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM J. Comput*, 18(3):559–578, 1989.

[BJS01] P. Beame, TS Jayram, and M. Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, 2001.

[BPW11] A. Bogdanov, P.A. Papakonstantinou, and A. Wan. Pseudorandomness for read-once formulas. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, 2011.

[Bra10] M. Braverman. Polylogarithmic independence fools AC^0 circuits. *Journal of the ACM (JACM)*, 57(5):1–10, 2010.

[BSSV03] P. Beame, M. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *Journal of the ACM (JACM)*, 50(2):154–195, 2003.

[Coo71] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of ACM (JACM)*, 18(1):4–18, 1971.

[DNPS11] M. David, P. Nguyen, P. A. Papakonstantinou, and A. Sidiropoulos. Computationally limited randomness. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 522–536. Tsinghua University Press, 2011.

[DP10] M. David and P. A. Papakonstantinou. Trade-off lower bounds for stack machines. In *IEEE Conference on Computational Complexity (CCC)*, pages 163–171, Boston, USA, 2010.

[Gur07] V. Guruswami. Algorithmic results in list decoding. *Foundations and Trends® in Theoretical Computer Science*, 2(2):107–195, 2007.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, STOC'94 (Montréal, Québec, Canada, May 23-25, 1994)*, pages 356–364, New York, 1994. ACM Press.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NR95] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, May 1995.

[Ruz80] W. L. Ruzzo. Tree-size bounded alternation. *Journal of Computer Systems and Sciences (JCSS)*, 21(2):218–235, 1980.

A Omitted Details of Stack Simulation by Branching Programs

In Section 4 we described a simulation which resulted in a read-k branching program with very large width. Here, we show how to perform the simulation with width $2^{O_\lambda(\log^2 n)}$, with out increasing the number of reads per symbol.

To reduce the width of this branching program we perform an optimization. If the realizable pair tree has t nodes, then it has a subtree of size at least $t/3$ but no more than $2t/3$ (see for example Lemma 1 of [BPW11]). The branching program will guess the label at the root of this subtree, recursively check that (c_0, c_1) is realizable assuming its guess was correct, and verify that its guess was correct. This optimization breaks up the verification of a realizable pair tree of size t into two subproblems of size at most $2t/3$ each, which can be solved recursively.

To implement this recursion, the branching program will need to keep track of its realizable pair guesses G so far. To make sure the resulting branching program performs at most k reads per symbol, we must ensure that every time interval of the simulation is covered at most once. We now give the implementation of this optimized branching program that represents the predicate "There exists a realizable pair tree of size t whose root is labeled by $(c_0(r), c_1(r))$ where all the incomplete pairs in G are assumed to be realizable on input r."

The *time interval* $\mathrm{int}(c_0, c_1)$ of a configuration pair (c_0, c_1) is the interval $[t_0, t_1]$ where t_0 and t_1 are the timestamps of c_0 and c_1, respectively. We use $|\mathrm{int}(c_0, c_1)|$ to denote the length $t_1 - t_0$ of the time interval.

$B_{c_0,c_1,G,t}$: On input r,
1 If $t = 0$, reject. If $c_0 = c_1$ or $(c_0, c_1) \in G$, accept.

2 If G has size more than $K \log n$, reject. Otherwise,
3 Guess five incomplete configurations c, c', c'_1, g_0, g_1 and a number s, $0 \leq s < 2t/3$
such that their time stamps of c_0, c, c', c'_1, c_1 are increasing,
and $\mathrm{int}(g_0, g_1) \subseteq \mathrm{int}(c, c')$ or $\mathrm{int}(g_0, g_1) \subseteq \mathrm{int}(c', c'_1)$,
and $\mathrm{int}(g_0, g_1)$ does not intersect the time interval of any pair in G.
4 Recursively evaluate $B_{c_0,c,G\cup\{(g_0,g_1)\},s}(r)$. If it rejects, reject.
5 If $c'(r)$ does not follow $c(r)$, reject.
6 Recursively evaluate $B_{c',c'_1,G\cup\{(g_0,g_1)\},2t/3-s-1}(r)$. If it rejects, reject.
7 If $c_1(r)$ does not follow $c'_1(r)$, reject.
8 Recursively evaluate $B_{g_0,g_1,\varnothing,2t/3}(r)$ and output its answer.

In line 2, K is a constant whose meaning we explain shortly. Let $B_x(r) = B_{\mathrm{start},\mathrm{accept},\varnothing,t+1}(r)$, where t is an upper bound on the running time of $M(x, a_x)$ where $|x| = n$. It follows from our discussion that if S accepts x on randomness r, then there exists a realizable pair tree of size $t+1$ whose root is labeled by $(\mathrm{start}(r), \mathrm{accept}(r))$. Moreover, there exists a tree of "good guesses" G that reduce the sizes of the trees to be checked at each step by a factor of at least $2/3$. Since the size of the realizable pair tree of $(\mathrm{start}, \mathrm{accept})$ is polynomial in n, it follows that the depth of this tree of good guesses is at most $K \log n$ for a sufficiently large constant K.

If S does not accept x, then no such realizable pair tree of any size exists so B_x does not accept r. Finally, note that by the properties of the realizable pair tree this recursion cannot access the same random tape symbol more than k times; whereas if all of the guesses correctly correspond to a realizable pair tree for an accepting computation then no time interval overlaps. The details of the inductive argument are left to the reader.

It remains to analyze the width and input access of B_x. Assuming G has size $O_\lambda(\log n)$, the tuple (c_0, c_1, G, t) can be described using $O_\lambda((\log n)^2)$ bits. Apart from the recursive calls $B_{c_0,c_1,G,t}$ has $\mathrm{poly}(n)$ nodes; so $B_{c_0,c_1,G,t}$ has size (and therefore width) $2^{O_\lambda((\log n)^2)}$ as desired. For the input access, notice that the symbols of r accessed in the recursive calls of B_x are all disjoint, and so if every symbol of r in the computation $M(x, a_x)$ is accessed at most k times, the same will be true for B_x (although the order of access may change).

Unique witnesses Assume $B_{c_0,c_1,G,t}(r)$ accepts. We describe how to modify $B_{c_0,c_1,G,t}$ to make its accepting computation unique. To do so, is it sufficient to ensure that inductively, there is a unique choice of the guesses c, c', c'_1, g_0, g_1, s that leads to acceptance.

Let us assume that all the guesses in G are realizable. We observe that because our stack machines are deterministic, a realizable pair (c_0, c_1) admits a unique realizable pair tree. Moreover, it can be shown inductively that this tree has exactly $|\mathrm{int}(c_0, c_1)|+1$ nodes. Thus, without sacrificing correctness, we may set $s = |\mathrm{int}(c, c')| + 1$ in line 3. Then there is a unique guess of c' and c'_1 (the unique choice so that (c_0, c_1) breaks into (c_0, c) and (c', c'_1)) that leads to acceptance.

However, there may be multiple choices of (g_0, g_1) that yield accepting computations because the realizable pair tree of c_0, c_1 could have several subtrees of size at most $2t/3$. To remove this source of alternatives, we will force $B_{c_0,c_1,G,t}$ to make a canonical choice of (g_0, g_1): We will require that the subtree rooted at (g_0, g_1) is the largest leftmost subtree of size at most $2t/3$. We enforce this requirement by checking that all subtrees encountered along accepting computations are consistent with this condition.

To carry out these checks, we extend the definition of G to include the maximum allowed size $s_{\max}$ of the realizable pair tree for each of its pairs (g_0, g_1); now the elements of G will have the form $(g_0, g_1, s_{\max})$ where $s_{\max}$ is an integer. We will say G is *inconsistent* with the pair (c_0, c_1) if for some $(g_0, g_1, s_{\max})$ in G, $|\mathrm{int}(g_0, g_1)| + 1 \leq |\mathrm{int}(c_0, c_1)| + 1 \leq s_{\max}$, but c_1 has a smaller time stamp than g_0.

We make the following changes to the above branching program $B_{c_0,c_1,G,t}$. We replace lines 1 and 3 with the following ones:

1 If $t = 0$, reject. If $c_0 = c_1$ or $(c_0, c_1, s) \in G$ for some s, accept.
3 Guess five incomplete configurations c, c', c_1', g_0, g_1 and set $s = |\mathrm{int}(c, c')| + 1$

and lines 4 and 6 with

4 Recursively evaluate $B_{c_0,c,G\cup\{(g_0,g_1,2t/3)\},s}(r)$. If it rejects, reject.
4.1 If G is inconsistent with (c_0, c), reject.
6 Recursively evaluate $B_{c',c_1',G\cup\{(g_0,g_1,2t/3)\},2t/3-s-1}(r)$. If it rejects, reject.
6.1 If G is inconsistent with (c', c_1'), reject.

Lines 4.1 and 6.1 ensure that all the realizable pair trees in G are the leftmost subtrees (of the realizable pair tree of (c_0, c_1) with at most the proscribed size, and so the choices in G that lead to an accepting computation are unique.

B Partition Lemmas

In this section, we give a proof of Lemma 2 and show that it is nearly tight.

Proof. The collection $\mathcal{C}$ can be constructed randomly by including each $i \in [t]$ into $\mathcal{C}$ independently with probability $1/2$. On average, every "variable" $j \in [n]$ appears in at most k of the sets S_i; in fact, we will argue that this is the worst case. Let X_j be the indicator random variable which takes value 1 if the index $j \in [n]$ belongs to some S_i with $i \in \mathcal{C}$. We will bound the expectation and variance of $\sum_{j=1}^n X_j$, the number of variables covered by $\mathcal{C}$.

If index j appears in ℓ_j subsets, we have $\mathbf{E}[X_j] = 1 - 2^{-\ell_j}$ (the probability they are all excluded is exactly $2^{-\ell_j}$). By the arithmetic-geometric mean inequality, we have that

$$\mathbf{E}[\sum_{j=1}^n X_j] = \sum_{j=1}^n \mathbf{E}[X_j] = n - \sum_{j=1}^n 2^{-\ell_j} \leq n - n \cdot 2^{-(1/n)\sum_{j=1}^n \ell_j} = (1 - 2^{-k})n.$$

We proceed to bound the variance so that we may apply Chebyshev's inequality:

$$\begin{aligned}
\sigma^2 = \mathbf{Var}[\sum_{j=1}^n X_j] &= \sum_{j=1}^n \mathbf{Var}[X_j] + \sum_{j\neq\ell} \mathbf{Cov}[X_j, X_\ell] \\
&= \sum_{j=1}^n \mathbf{Var}[X_j] + \sum_{j\neq\ell} \mathbf{E}[X_j X_\ell] - \sum_{j\neq\ell} \mathbf{E}[X_j]\mathbf{E}[X_\ell] \\
&\leq \sum_{j=1}^n \mathbf{Var}[X_j] + t(nk/t)^2 \\
&\leq n + t(nk/t)^2
\end{aligned}$$

The last inequality is a trivial upper bound on the sum of the variances; the second to last inequality holds because $\mathbf{E}[X_j X_\ell] = \mathbf{E}[X_j]\mathbf{E}[X_\ell]$ unless indices j and ℓ occur in the same bucket, and there are at most $t(kn/t)^2$ such pairs (since each bucket has size at most kn/t). Now we have $\sigma \leq (nk/\sqrt{t}) + \sqrt{n}$, so Chebyshev's inequality yields

$$1/4 \geq \Pr[\sum_{j=1}^n X_j > 2\sigma + \mu] \geq \Pr[\sum_{j=1}^n X_j > n(\frac{2k}{\sqrt{t}} + \frac{2}{\sqrt{n}} + 1 - 2^{-k})] = \Pr[\sum_{j=1}^n X_j > \alpha \cdot n].$$

The argument can be repeated for $\overline{\mathcal{C}}$; a union bound gives that over half of the partitions will satisfy $\left|\bigcup_{i\in\mathcal{C}} S_i\right| \leq \alpha \cdot n$ and $\left|\bigcup_{i\in\overline{\mathcal{C}}} S_i\right| \leq \alpha \cdot n$. ■

Here we argue that the partition $(\mathcal{C}, \overline{\mathcal{C}})$ in Lemma 2 in Section 3.1 provides a cover of essentially optimal size. We view this as evidence that the analysis from this work is unlikely to extend to formulas of significantly superlinear size.

Lemma 5. *Suppose $k(1-\alpha) > H(\alpha)+t/n+O(1/n)$. There exists a collection of sets $S_1, \ldots, S_t \subseteq [n]$ such that (1) Every $j \in [n]$ appears in at most k of the S_i and (2) For every partition $(\mathcal{C}, \overline{\mathcal{C}})$, either $|\cup_{i\in\mathcal{C}} S_i| \geq \alpha n$ or $|\cup_{i\notin\mathcal{C}} S_i| \geq \alpha n$ (or both).*

Thus if the conclusion of the lemma is violated, we must have $k(1-\alpha) \leq H(\alpha)+t/n+O(1/n)$. For $t \ll n$ and α, k constant in n, $k \leq H(\alpha)/(1-\alpha) + o(1) \leq -\log(1-\alpha) + 1 + o(1)$, so $\alpha \geq 1 - 2^{-k+1+o(1)}$.

Proof. Make k copies of every variable $j \in [n]$ and place each of these copies independently and equiprobably into a set S_i. Fix any partition $(\mathcal{C}, \overline{\mathcal{C}})$ of $[t]$. Without loss of generality, assume $|\mathcal{C}| \geq t/2$, for otherwise we may work with its complement. The probability that there exists a set $U \subseteq [n]$ of size $(1-\alpha)n$ that is disjoint from $\cup_{i\in\mathcal{C}} S_i$ is at most $\binom{n}{\alpha n} 2^{-k(1-\alpha)n} \leq 2^{-k(1-\alpha)n+H(\alpha)n+O(1)}$. Taking a union bound over all partitions, we obtain that the probability there exists a partition $(\mathcal{C}, \overline{\mathcal{C}})$ condition (2) is at most $2^{t-k(1-\alpha)n+H(\alpha)n+O(1)}$, so conditions (1) and (2) can be satisfied as long as $k(1-\alpha) > H(\alpha) + t/n + O(1/n)$. ■