Electronic Colloquium on Computational Complexity, Report No. 39 (2009)

Polynomial Time with Restricted Use of Randomness*

Matei David Periklis A. Papakonstantinou Anastasios Sidiropoulos

University of Toronto

{matei,papakons,tasoss}@cs.toronto.edu

Abstract

We define a hierarchy of complexity classes that lie between P and RP, yielding a new way of quantifying partial progress towards the derandomization of RP. A standard approach in derandomization is to reduce the number of random bits an algorithm uses. We instead focus on a model of computation that allows us to quantify the extent to which random bits are being used. More specifically, we consider *Stack Machines* (SMs), which are log-space Turing Machines that have access to an unbounded stack, an input tape of length N, and a random tape of length $N^{O(1)}$. We parameterize these machines by allowing at most $r(N) - 1$ reversals on the random tape, thus obtaining the $r(N)$-th level of our hierarchy, denoted by $\mathsf{RPdL}[r]$. It follows by a result of Cook [Coo71] that $\mathsf{RPdL}[1] = \mathsf{P}$, and of Ruzzo [Ruz81] that $\mathsf{RPdL}[\exp(N)] = \mathsf{RP}$. Our main results are the following.

- For every $i \geq 1$, derandomizing $\mathsf{RPdL}[2^{O(\log^i N)}]$ implies the derandomization of RNC^i. Thus, progress towards the P vs RP question along our hierarchy implies also progress towards derandomizing RNC. Perhaps more surprisingly, we also prove a partial converse: Pseurorandom generators (PRGs) for RNC^{i+1} are sufficient to derandomize $\mathsf{RPdL}[2^{O(\log^i N)}]$; i.e. derandomizing using PRGs a class believed to be strictly inside P, we derandomize a class containing P.

  More generally, we introduce *Randomness Compilers*, a model equivalent to Stack Machines. In this model a polynomial time algorithm gets an input x and it outputs a circuit C_x, which takes random inputs. Acceptance of x is determined by the acceptance probability of C_x. When C_x is of polynomial size and depth $O(\log^i N)$ the corresponding class is denoted by $\mathsf{P{+}RNC}^i$, and we show that $\mathsf{RPdL}[2^{O(\log^i N)}] \subseteq \mathsf{P{+}RNC}^i \subseteq \mathsf{RPdL}[2^{O(\log^{i+1} N)}]$.
- We show an unconditional $N^{\Omega(1)}$ lower bound on the number of reversals required by a SM for Polynomial Evaluation. This in particular implies that known Schwartz-Zippel-like algorithms for Polynomial Identity Testing cannot be implemented in the lowest levels of our hierarchy.
- We show that in the 1-st level of our hierarchy, machines with one-sided error are as powerful as machines with two-sided and unbounded error.

Keywords: probabilistic polynomial time, complexity hierarchy, derandomization, polynomial identity testing, communication complexity

*An earlier version of this paper has the title "A Hierarchy between P and RP".

ISSN 1433-8092

1 Introduction

Randomness is central to computer science and its intersection with engineering, and natural sciences. Some of the most intriguing and long-standing open questions in theoretical computer science regard the gap between polynomial time and its probabilistic analogs (e.g. RP, BPP). It is conjectured (e.g. [IW97, KI03, NW88]) that this gap is small or even that randomness is in general not necessary. A standard measure of how far a probabilistic algorithm is from a deterministic one is the *number of random bits* it uses. As such, the quest for derandomization concentrates on reducing this number, for example, using pseudorandom generators.

In this paper, we propose an orthogonal perspective to randomness. Informally, we say that a machine "makes essential use of randomness" if it revisits the same random bit "many times" during its computation. More precisely, we consider a natural model of computation characterizing polynomial time, in which it is meaningful to count the *number of head reversals on the random tape*.

Consider for example a randomized procedure which performs a random walk in a graph. After each transition the algorithm may "forget" the random bits used so far. Contrast this with a randomized procedure where random bits used at one step of the computation are revisited in future steps. Our definition of essential use of randomness allows us to distinguish between these two scenarios.

A characterization of essential use of randomness. Consider a polynomial time TM M which receives its input on a regular (read-write) work tape and its random bits on a separate read-only tape. As explained before, we would like to say that M makes essential use of randomness if it accesses the same random bit many times. There seems to be no clean way to capture this notion in an arbitrary polynomial time TM. Counting head reversals over the random tape is not interesting since M can simply copy all the random bits on its work tape. This simple trick does not work of course for machines with small space. However, since we are interested in arbitrary polynomial-time computation, we need a characterization of P in terms of space-bounded machines. To that end, we consider a characterization due to Cook [Coo71], which shows that a logarithmic-space TM equipped with an unbounded stack, henceforth called a *Stack Machine* (SM), exactly characterizes P.

The above simple observation naturally gives rise to a hierarchy of classes between P and RP. For $r(N) \geq 1$, we denote by $\mathsf{RPdL}[r]$ (which stands for Randomized Pushdown Log-space) the class of languages that can be decided by a SM with an input tape of length N and a random tape of length $N^{O(1)}$, which is allowed at most $r(N) - 1$ reversals over the random tape, and has bounded one-sided error. It follows by a result of Cook [Coo71] that $\mathsf{RPdL}[1] = \mathsf{P}$, and that $\mathsf{RPdL}[\exp(N)] = \mathsf{RP}$. We similarly define the classes $\mathsf{NPdL}[r]$, $\mathsf{coNPdL}[r]$, $\mathsf{coRPdL}[r]$, $\mathsf{BPPdL}[r]$, and $\mathsf{PPdL}[r]$. We have arbitrarily chosen to make our exposition RP-centric. Everything can be restated in terms of other probabilistic polynomial time classes such as BPP.

The stack is indispensable for the study of probabilistic *polynomial time* classes, in the following sense. Logspace TMs without a stack characterize $\mathsf{LogSPACE} \subseteq \mathsf{NC}^2$. By adding a polynomially long (two-way access) random tape to logspace machines we define [Nis93] the class $\mathsf{R}^*\mathsf{LogSPACE} \subseteq \mathsf{RNC}^2$.

1.1 Our results

How easy is it to collapse the first levels? Given the definition of our hierarchy between P and RP, several questions arise. If $\mathsf{P} = \mathsf{RP}$, then $\mathsf{RPdL}[1] = \mathsf{RPdL}[\exp(N)]$. An immediate question is then

How easy is it to prove $\mathsf{RPdL}[1] = \mathsf{RPdL}[r]$, for some $r(N)$?

We give strong evidence that collapsing the first levels of the hierarchy is a non-trivial task, and requires progress towards existing major open questions. In particular, for any $i \geq 1$, derandomizing $\mathsf{RPdL}[2^{O(\log^i N)}]$ implies derandomizing RNC^i. In other words, progress towards the P vs RP question along our hierarchy implies also progress towards the derandomization of RNC. Perhaps more surprisingly, we also prove a partial converse: pseurorandom generators for RNC^{i+1} are sufficient to derandomize $\mathsf{RPdL}[2^{O(\log^i N)}]$. In other words, modulo the derandomization technique (use of PRGs), derandomizing classes believed to lie far inside P, we derandomize the levels of our hierarchy (even the first level contains P).

This follows by an alternative definition of restricted use of randomness, which we show to be equivalent to the one that uses Stack Machines. We consider *Randomness Compilers* to be polynomial time algorithms, which on every input they construct a shallow circuit, which in turn it is evaluated at a random input. More precisely, let P+RNC^i denote the class of languages L decidable in the following sense: a polytime transducer, on input x it computes a probabilistic circuit C_x of constant fan-in, polynomial size and depth $O(\log^i N)$. If $x \in L$, then $\Pr_\rho[C_x(\rho) = 1] \geq \frac{1}{2}$, whereas if $x \notin L$ then $\Pr_\rho[C_x(\rho) = 1] = 0$. We show that P+$\mathsf{RNC}^i \subseteq \mathsf{RPdL}[2^{O(\log^i N)}] \subseteq \mathsf{P}$+$\mathsf{RNC}^{i+1}$ (Section 3). Contrary to a possible interpretation of the notation P+RNC^i, we stress-out that the computed circuit is much stronger than a P-uniform RNC circuit, since in general it depends on the given input.

How easy is it to collapse the last levels? Another immediate question is of course

How easy is to prove $\mathsf{RPdL}[r] = \mathsf{RPdL}[\exp(N)]$, for some $r(N)$?

We also provide evidence that this is also a challenging problem given the current state of the art in derandomization. If such a small $r(N)$ exists, then one should at least be able to show that certain problems in e.g. coRP can be decided in the low-levels of the $\mathsf{coRPdL}[\cdot]$ hierarchy. In particular, we consider the problem of Polynomial Identity Testing[1] (PIT), a prototypical problem in coRP. We observe that almost all currently known Schwartz-Zippel-like algorithms for PIT evaluate a polynomial at a random point. We prove (Section 5) an unconditional $N^{\Omega(1)}$ lower bound on the number of reversals required by a SM for Polynomial Evaluation (PE). This in particular implies that implementing PIT in $\mathsf{RPdL}[N^{o(1)}]$ is a non-trivial task.

Understanding the model. We study some natural variants of our model. In particular, we show that in the 1-st level, a SM that is allowed unbounded (two-sided) error is as powerful as a SM with bounded and one-sided error. In other words, $\mathsf{PPdL}[1] = \mathsf{RPdL}[1] = \mathsf{P}$, which in particular derandomizes $\mathsf{BPPdL}[1]$ (Section 4).

Finally, we consider the case of non-deterministic tape under different settings. A detailed exposition is given in Appendix C.2. Among others, we show that the corresponding hierarchy (namely, $\mathsf{NPdL}[\cdot]$) turns out to be trivial: $\mathsf{NPdL}[1] = \mathsf{P}$, while $\mathsf{NPdL}[2] = \mathsf{NP}$.

1.2 Related Work

Stack Machines. [Coo71] characterizes poly-time computation in terms of Stack Machines (AuxPDAs): log-space bounded TMs equipped with an unbounded stack. Recursive log-space computation is a natural

[1] In the PIT problem, the input is an arithmetic circuit describing a polynomial $P \in \mathbb{F}[x_1, \ldots, x_m]$, where $\mathbb{F}$ is a field, and the question is whether P always evaluates to 0 over $\mathbb{F}^m$ (we are interested in polynomials of degree bigger than the characteristic of the field). Note that in fields of characteristic zero this problem coincides to deciding whether the given arithmetic circuit corresponds to the identically zero polynomial.

concept. Moreover, SMs exhibit quite interesting equivalences to other models of computation. Our work relies on the connections between deterministic (and non-deterministic) logspace and time-bounded SMs, and that of SAC (and NC) circuits with various forms of uniformity [All89, BCD$^+$89, Ruz80, Ruz81, Ven91]. When simultaneously to the space we also bound the time, there is a constructive and direct correspondence (which intermediately goes through ATMs) between SMs and combinatorial circuits. This enable us to blur the distinction between space-time bounded SMs and families of circuits. In a study of NC in presence of strong uniformity (P-uniformity) [All89] studies SMs with restricted input-head moves. This is related to the concept of head-reversals on the random tape considered in our paper. For example, the discussion in [All89] (p.919) about the fact that the natural restriction on the head-moves of a SM translates to something more subtle in case of ATMs, is related to our definition of essential use of randomness using SMs with restricted number of reversals on the random tape.

Derandomization. There is a long line of research in derandomization. Successful stories range from straightforward algorithms e.g. for MaxSAT (e.g. [AB08]) to quite sophisticated ones such as the AKS primality test [AKS04]. There is a plethora of works that aim to derandomize probabilistic polynomial time using certain types of pseudorandom generators. The majority of them relate the problem of derandomization to the problem of proving lower bounds for the average-case complexity of one-way functions [Yao82], or to arbitrary hard functions (cf. [NW88, IW97, STV99, Uma03]). The general theme of this research direction comes under the title "randomness-hardness tradeoffs". These works are in line with our intuition about the circuit complexity of certain problems in E, EXP or NEXP. They provide evidence that derandomization of classes such as BPP might be possible, and simultaneously they give evidence about the conceptual difficulty of achieving such a goal.

Polynomial Identity Testing. Kabanets and Impagliazzo [KI03] show that even the task of derandomizing a particular problem in coRP, namely PIT, is essentially the same as proving superpolynomial lower bounds on the arithmetic circuit complexity of NEXP. Hence, derandomizing PIT is a challenging task. Our results give evidence for the limitations of Schwartz-Zippel-like algorithms (e.g. [Zip79, Sch80]) for derandomizing PIT. There are also algorithms [CK97, LV98] that use fewer bits than the standard Schwartz-Zippel algorithm. Despite the fact that the number of random bits is smaller, it is still the case that these algorithms evaluate the input polynomial at a chosen point. Therefore in our framework, their use of randomness is as essential as in the standard test. Yet other algorithms [AB99, ABKPM06, KS01] avoid evaluating the input polynomial directly at a chosen point. Although our lower bound does not apply in this setting, implementing these algorithms in the lower levels of our hierarchy appears to be a difficult task.

1.3 Our Techniques

The technically more involved argument is an unconditional $N^{\Omega(1)}$ lower bound on the number of reversals a SM makes for polynomial evaluation - even when the polynomial is non-uniformly given to the machine. The lower bound follows by a direct-sum type of argument, using a NIH communication complexity reduction to inner product (over any field). This argument uses some ideas from [BHN08], but it is essentially different (see Section 5.2 for a comparison).

The relations between P+RNC and RPdL$[N^{\mathsf{polylog}N}]$ builds upon the work of [All89, BCD$^+$89, Ruz80, Ruz81]. The main tool is the time-compression lemma (Lemma 7), together with theorems from the related work.

The derandomization of $\mathsf{PPdL}[1]$ (a two-sided error class) relies on extending the Dynamic Programming algorithm in [Coo71], so as to count the number of accepting leaves in the computation tree of a non-deterministic SM. Starting from the observation that in the computation tree of depth $\exp(N)$ on every path there are at most polynomially many branches, we count the number of accepting paths.

We also provide some other structural results. These results range from the simple argument showing $\mathsf{NPdL}[2] = \mathsf{NP}$, to arguments that put together structural implications building on the work of [All89, BCD$^+$89, HS08, Ruz80, Ruz81].

1.4 Organization

We use Stack Machines to define randomized hierarchies between P and RP in Section 2. Some preliminary results are given in the same section. In Section 3 we study the relation between Stack Machines and Randomness Compilers, and we show that $\mathsf{RPdL}[N^{\mathsf{polylog}N}] = \mathsf{P+RNC}$. This also suggests that PRGs used for derandomization along RNC imply the derandomization along $\mathsf{RPdL}[N^{\mathsf{polylog}N}]$. In Section 4 we show that for every SM M that makes one scan over its randomness and has two-sided, unbounded error, there is a polynomial-time algorithm deciding the same language as M. In Section 5 we give the unconditional lower bound for Polynomial Evaluation (PE) which implies the lower bound for the family of Schwartz-Zippel-like algorithms. We conclude in Section 6 by outlining a few among the future research directions.

2 Definitions & Preliminaries: Stack Machines and Randomness Compilers

Notation and conventions. We use standard names for complexity classes, e.g. P, NP, BPP, NC, NEXP, $\mathsf{DSPACE}(f(n))$ (see e.g. [AB08]). We denote by N the number of input bits. Let $\mathsf{E} := \cup_{c>0}\mathsf{DTIME}(2^{cN})$, and $\mathsf{EXP} := \cup_{k=1}^{\infty}\mathsf{DTIME}(2^{N^k})$, $\mathsf{QuasiP} := \cup_{i=1}^{\infty}\mathsf{DTIME}(2^{\log^i N})$. We simplify notation by omitting floors and ceilings for integer values. The auxiliary, external read-only tape of SMs is of length polynomial in N. The working memory size is $O(\log N)$ unless mentioned otherwise. If the head on the external tape reverses $r-1$ times then we say that the *number of scans* is r. Let $[n] := \{1, \ldots, n\}$. Let $\alpha \in \{0,1\}^n$, we denote by α_i the i-th coordinate of α. Occasionally, we identify a string $\alpha \in \{0,1\}^n$ by its support $\mathsf{supp}(\alpha) = \{i \ : \ \alpha_i = 1\}$. Let S_m be the set of permutations over $[m]$. $\mathbb{F}$ is used to denote a field and $\oplus$ denotes the addition in $\mathbb{F}$. $\rho \in \{0,1\}^*$ denotes a random (or pseudorandom) string. $I \subseteq \mathbb{Z}^+$ usually denotes an infinite subset of $\mathbb{Z}^+$. For a positive integer i, $\mathsf{SAC}^i \subseteq \mathsf{AC}^i$ is the semi-unbounded restriction of AC^i; i.e. only the OR gates have bounded fan-in, and the negations are to the input level. A family $\{C_m\}_{m\in I}$ of circuits is an NC^i (SAC^i) circuit-family of polysize (semi-unbounded) and depth $O(\log^i N)$ circuits defined for input-lengths in I. U_m denotes the uniform distribution over $\{0,1\}^m$.

Stack Machines. We consider three types of Stack Machines. A SM (AuxPDA) M is a logspace Turing Machine augmented with an unbounded stack. We may also write (r,s)-SM to denote that the machine makes at most $r(N)$ reversals on its input and it has worktapes of size s. All SMs have logarithmic space-bound, unless mentioned otherwise. For a detailed definition of a SM (AuxPDA) and its computation see [Coo71]. We assume that the SMs always halt with an empty stack. Furthermore, in each transition exactly one of the following happens:either a head-move, or a push to the stack or a pop from the stack. Occasionally we allow the SM to do coin flipping, in which case we write *randomized SM*. Central to our study are *Verifier Stack Machines (vSMs)*. These are polytime verifiers with access to a random tape; i.e. a vSM is a SM extended with a polynomially long (read-only) random tape. Finally, we consider *non-uniform*

Stack Machines (nu-SMs) which are SMs extended with a polynomially long tape containing a non-uniform advice. Non-uniform Stack Machines appear for different reasons in Section 3 and Section 5. We make use of the following theorem which is a corollary of [BCD$^+$89, Ruz80, Ruz81].

Theorem 1. *Let $\mathcal{C}$ be the class of languages decided by (deterministic) SMs that work in time $2^{O(\log^i N)}$. Then, $\mathsf{NC}^i \subseteq \mathcal{C} \subseteq \mathsf{SAC}^i \subseteq \mathsf{NC}^{i+1}$, where the circuit classes are (logspace) uniform. The same relation holds for non-uniform SMs and $\mathsf{NC}, \mathsf{SAC}$ circuits.*

Verifier Stack Machines compute arbitrary polytime predicates [Coo71]. Also, every probabilistic polytime TM computes a deterministic predicate $R(x, \rho)$, $\rho \in_R \{0,1\}^{N^k}$ where ρ is the random bits. Hence, it is straightforward to modify the proofs in [Coo71] and to obtain the following fact.

Fact 2. *A verifier Stack Machine with the standard definition of error characterizes the corresponding probabilistic polynomial time class such as $\mathsf{RP}, \mathsf{coRP}, \mathsf{BPP}, \mathsf{PP}$. Furthermore, if the external tape contains non-deterministic bits (equivalently: false-negatives with unbounded error) then we characterize NP.*

Randomized and non-deterministic hierarchies. We consider false-negatives/false-positives/two-sided bounded/unbounded error regimes, corresponding in the usual way to the prefixes N-, coN-, R-, coR-, P- and BP-. Furthermore, we consider bounding the number of reversals $r(N)$ a vSM is allowed on its external tape. Accordingly, $x\mathsf{PdL}[r]$ is the class of languages accepted by vSMs that are allowed at most $r-1$ head reversals on the external tape, operating in error regime x. Thus, e.g., $\mathsf{RPdL}[1]$ is the class of languages accepted by vSMs with one-way (no reversals) access to the external tape and one-sided error. We use the term *non-deterministic hierarchy* to refer to the collection of levels of $\mathsf{NPdL}[\cdot]$. Similarly, when the error condition is clear from the context we use the term *randomized hierarchy*.

It is easy to check that the following classes are closed under logspace (or weaker) transformations: $\mathsf{RPdL}[c]$, $c \in \mathbb{Z}^+$, $\mathsf{RPdL}[\mathsf{constant}] := \bigcup_{k=1}^{\infty} \mathsf{RPdL}[k]$, $\mathsf{RPdL}[O(\log N)] := \bigcup_{c>0} \mathsf{RPdL}[c \log N]$, and $\mathsf{RPdL}[\mathsf{poly}] := \bigcup_{c>0} \mathsf{RPdL}[N^c]$.

$\mathsf{P{+}RNC}^i$: polytime Randomness Compilers. $\mathsf{P{+}RNC}^i$ denotes the class of languages L decidable as follows: there exists a polynomial time TM M which on every input x, M computes a probabilistic circuit C_x of constant fan-in, polynomial size and depth $O(\log^i N)$. If $x \in L$, then $\Pr_\rho[C_x(\rho) = 1] \geq \frac{1}{2}$, whereas if $x \notin L$ then $\Pr_\rho[C_x(\rho) = 1] = 0$. We refer to this model of computation as *$\mathsf{P{+}RNC}$ model*, and to the polytime transducer M as *Randomness Compiler*.

Variants of our model. In the parameterization on the number of reversals over the random tape, the "reversals resource" is a worst-case resource; similar to the resource of running time in the definition of RP. In Section 3 we show that vSMs have the same power as the P+RNC model. The model of vSMs allows more flexibility in defining natural variants of the model, whereas it is not obvious how to do something similar for the P+RNC model. For example, for randomized computation one may want to define $\mathsf{RPdL}^{\mathbb{E}}[r(N)]$ as the class of problems decided by vSMs where the expected number of scans over the randomness is $r(N)$. In Appendix C.1 we show that $\mathsf{RPdL}[\mathsf{poly}] = \mathsf{RPdL}^{\mathbb{E}}[\mathsf{poly}]$; thus, simplifying the task of giving SM algorithms to show containment in $\mathsf{RPdL}[\mathsf{poly}]$.

Another variant of the vSM model is when the auxiliary tape contains non-deterministic bits. In this case the situation becomes technically simple, and we show that one reversal (two scans) is sufficient to characterize NP.

Theorem 3. *The non-deterministic hierarchy collapses to level* 2*. More specifically,* $\mathsf{NPdL}[2] = \mathsf{NP}$ *and also* $\mathsf{NPdL}[1] = \mathsf{P}$*.*

Among others, this theorem indicates that the randomized case is non-trivial even for the second level $\mathsf{RPdL}[2]$. Discussion and further results about variants of our model are given in Appendix C.2.

3 $\mathsf{RPdL}[N^{\mathsf{polylog}N}]$ and P+RNC

We have introduced two models to define restricted use of randomness. The first model (characterization of RPdL) is a transition system on which we make explicit the concept of essential use of randomness, by counting random bits accesses. In the second model (characterization of P+RNC) we compile all the needed randomness in a shallow circuit. It turns out that the two models are equivalent. Intuitively, this says that the syntactically defined model of verifier Stack Machines is able to express computations which are simultaneously (i) arbitrary deterministic polytime and (ii) when it comes to randomness much weaker than polytime.

Theorem 4. *Let* $i \in \mathbb{Z}^+$*. Then,* $\mathsf{P+RNC}^i \subseteq \mathsf{RPdL}[2^{O(\log^i N)}] \subseteq \mathsf{P+RNC}^{i+1}$.

The proofs of the statements of this section appear in Appendix D.

An immediate consequence of Theorem 4 is that $\mathsf{P} \cup \mathsf{RNC}^i \subseteq \mathsf{RPdL}[2^{O(\log^i N)}]$. Thus, derandomization along the $\mathsf{RPdL}[\cdot]$ implies derandomization along RNC (even P-uniform RNC). More importantly, using PRGs to derandomize RNC (which is believed to be deeply inside P) implies derandomization of classes that contain P. That is, such a derandomization of RNC implies progress in a concrete sense towards the derandomization of RP. As mentioned, everything stated for RP and RNC holds for other probabilistic classes, e.g. BPP and BPNC.

The derandomization of the quasi-polynomial levels of $\mathsf{RPdL}[\cdot]$ using weak PRGs (Theorem 6 - see below) is a corollary of Theorem 4.

ϵ-biased pseudorandom generators (PRGs). Our PRGs definitions aim to a more qualitative view than the usual. Several parameters have been fixed to reduce clutter. For example, we focus on PRGs that stretch polylogarithmic bits to polynomial. These parameters can be adjusted in the standard way to generalize our results. All distinguishers are non-uniform circuits.

Definition 5. Let $0 < \epsilon < 1$, $k \geq 1$. Let $G : \{0,1\}^* \to \{0,1\}^*$ be a function, such that $G(z)$ is computable in time $2^{O(|z|^{1/k})}$. We say that G is an $(\mathsf{NC}^i, k, \epsilon)$*-pseudorandom generator* if for every non-uniform NC^i circuit-family $\mathcal{C}$ and for sufficiently large $|z| := n$, $z \in \{0,1\}^*$, where $|G(z)| = 2^{|z|^{1/k}} := m$

$$|\Pr_{z \in_R \{0,1\}^n}[C_m(G(z)) = 1] - \Pr_{\rho \in U_m}[C_m(\rho) = 1]| \leq \epsilon$$

where $C_m \in \mathcal{C}$ has m input bits.

Theorem 6. *Let* $k, i \geq 1$*. If there exists a* $(\mathsf{NC}^{i+1}, k, \frac{1}{7})$*-PRG, then* $\mathsf{RPdL}[2^{O(\log^i N)}] \subseteq \mathsf{DTIME}(2^{O(\log^k N)})$.

Hence, if there exists a k for all i's then $\mathsf{RPdL}[N^{\mathsf{polylog}N}] \subseteq \mathsf{DTIME}(2^{O(\log^k N)})$, whereas if k is a function of i then $\mathsf{RPdL}[N^{\mathsf{polylog}N}] \subseteq \mathsf{QuasiP}$. We remark that a slightly different argument proves that the PRG is sufficient to be secure against SAC^i circuits, instead of NC^{i+1}.

The proof of Theorem 4 relies on Lemma 7, which we show following [All89]. The proof is based on the fact that the computation between two successive head moves on the input can be compressed to

be polynomially-long, given some small (polysize) advice which depends on the input length. At first, the feasibility of this "compression" may be seen as counter-intuitive since the computation between two successive head-moves depends on the content of the stack, which in turn it depends on the given input (not only its length).

Lemma 7 (Time-compression lemma)**.** *Let M be a deterministic nu-SM which makes $r(N)$ reversals over its input. Then, there exists a deterministic nu-SM M' which makes $r(N)$ reversals over its input, it works in time $O(r(N)N^{O(1)})$, and it decides the same language as M.*

In practice, the non-uniformity in Theorem 6 seems to be sufficient, since most constructions of PRGs are against non-uniform distinguishers. A minor modification in Cook's construction [Coo71] shows that given the advice for M (or if M is uniform), then the non-uniform advice in the proof of Lemma 7 can be replaced with a polytime computable advice. It turns out that one cannot do better than this.

Lemma 8. *Given the advice for M, then the non-uniform advice in the proof of Lemma 7 cannot be computed in logspace uniform NC, unless PSPACE = EXP.*

4 $\mathsf{PPdL}[1] = \mathsf{P}$

It can be shown along the lines of the proof of Theorem 3, that $\mathsf{PPdL}[2] = \mathsf{PP}$. In this section we derandomize the first level of the randomized hierarchy with two-sided and unbounded error.

Theorem 9. *$\mathsf{PPdL}[1] \subseteq \mathsf{P}$.*

An immediate corollary is $\mathsf{BPPdL}[1] = \mathsf{PPdL}[1] = \mathsf{P}$. The description of the algorithm and its proof of correctness are given in Appendix B.

Outline of the algorithm - comparison with [Coo71]. We look at the computation of a vSM M on an input x as a tree of depth $\exp(N)$ of configurations, where every branching node corresponds to tossing a coin. In the *nondeterministic (false-negatives unbounded)* error regime considered in [Cook71], to decide whether M accepts x, one has to determine whether there is a path from the initial to the accepting configuration.

We define a *surface configuration* C of M on w to consist of (0) the state of M, (1) the location of the input head; (2) the full configuration of the work tapes (head positions and tape contents); (3) the top stack symbol; and (4) the location of the external tape head (denoted by $h(C)$) and the symbol it is scanning.

Cook defines the notion of "realizable pair of surface configurations" to be a pair (C_1, C_2) such that there exists a computation path that takes M from C_1 to C_2, with the stack level being the same in both, and never going below that level between them [2]. The key in Cook's proof is showing how existing realizable pairs can be combined to yield new ones, until, eventually, all realizable pairs are found. The number of distinct surface configurations is polynomial and thus this takes polytime. At the end, to decide whether M accepts x, we check if the surface of the initial configuration and that of the accepting configuration are realizable.

In the *two-sided unbounded* error regime considered in here, one way to decide if M accepts x is to compute the total probability of all strings that take M from the initial configuration to the accepting one. We simplify things by including the external head position in (full and surface) configurations, so that all strings

[2] More precisely, this a path that takes M from some (full) configuration $\overline{C_1}$ with surface C_1 to some (full) configuration $\overline{C_2}$ with surface C_2. This path cannot depend on the stack contents in $\overline{C_1}$ (except for the top stack symbol) because they are not accessed.

that take M from one (surface) configuration to another have the same length. Then, we only have to *count* the number of strings leading to the accepting configuration. Not unexpectedly, we are only able to perform this counting when M tosses at most polynomially many coins, in contrast to the nondeterministic regime [Coo71] where the proof works for exponentially many tosses. We consider pairs of surface configurations (C_1, C_2), and we now *count exactly* the number $\alpha(C_1, C_2)$ of strings that take M from C_1 to C_2, with stack level the same, and never going below that level in-between them. As compared to Cook, the rule for computing the values $\alpha(\cdot, \cdot)$ for new pairs from previously computed ones is more involved.

5 The Schwartz-Zippel test cannot show PIT $\in$ coRPdL$[N^c]$

The only known derandomization for coRPdL$[\cdot]$ is that RPdL$[1]$ = coRPdL$[1]$ = P, since RPdL$[1] \subseteq$ NPdL$[1]$ and P is closed under complement. We show that algorithms which evaluate even a fixed polynomial on an arbitrary point cannot show containment of PIT in coRPdL$[N^c]$, for constant $c > 0$.

Translations of a family of Probabilistic Turing Machines (PTMs) to vSMs. The technically more involved part of our contribution is an unconditional[3] lower bound for a popular family of probabilistic polynomial time algorithms for PIT. These algorithms evaluate the input polynomial on a random point; we refer to this family as SZ-algorithms (where SZ stands for Schwartz-Zippel). Our lower bound refers to these algorithms when realized as vSMs. Note that if we allow arbitrary translations of PTMs to vSMs and P = RP then no non-trivial lower bound is possible: we could always map a PTM to a vSM that does not access its randomness. To that end, we consider a natural restriction on such translations. Informally, the dependence on Polynomial Evaluation (PE) of the PTM is preserved. More formally, consider PTMs which at some point in their execution make a call to an oracle which evaluates the input polynomial at a random point. We allow both the PTM and the oracle to be implemented *arbitrarily* as a vSM, with the only restriction that the same oracle call be made. The term arbitrary *PE-preserving implementation as a vSM* of a SZ-algorithm corresponds to the collection of translations as mentioned above.

The lower bound. Every SZ-algorithm is realized as a vSM. Our lower bound on the number of reversals on the random tape focuses on the (arbitrarily implemented) evaluation procedure of the algorithm. By definition of the randomized hierarchy we parametrize on the worst-case number of reversals. Hence, the lower bound for the SZ-family follows directly by showing a lower bound for *deterministic* SMs with two input tapes: one contains the input polynomial p and the other the point ρ on which we evaluate the polynomial. We count reversals only on the tape which contains the point ρ. We emphasize that since we are interested in the worst-case number of reversals to prove the lower bound it suffices to adversarially choose the point ρ. The proof of Theorem 10 follows from Corollary 15, since the inner product of two vectors is the evaluation of the quadratic polynomial induced by the inner product.

Theorem 10. *Let $\mathcal{A}$ be a PTM which in every execution evaluates at least once the input polynomial p at a random point ρ, presented in terms of its coordinates on the random tape. Then, every PE-preserving implementation of $\mathcal{A}$ as a vSM makes at least N^c reversals on the random tape, for some explicit constant $0 < c < 1$, where N is the input length.*

[3] Recall that the input in PE is an arithmetic circuit. This problem is easily shown to be hard for P. Hence, conditionally, if P $\neq$ NC it is easy to see that SZ-algorithms cannot show containment in coRPdL$[N^{\text{polylog}\,N}]$. Note, that proving unconditionally any such superpolynomial lower bound would separate LogSPACE from P (in fact LOGDCFL from P).

Although no known Schwartz-Zippel-like algorithm evaluates the polynomial approximately, our lower bound can be generalized to hold even for algorithms that err with error bounded away from 1, when the input points are chosen uniformly at random.

5.1 Definitions related to the Communication Complexity lower bound

Communication Complexity. We consider the standard Number-In-Hand (NIH) model [KN97], where each of the p players gets an n-bit input part. Communication is done via a shared blackboard. Deterministic protocols are always correct and randomized protocols are correct with probability at least $2/3$ where the randomness is public. For a function $f : \left(\{0,1\}^n\right)^p \to \{0,1\}$ we denote by $R(f)$ the minimum cost of the best randomized communication protocol for f, over all inputs and random strings.

We define the functions pSETINT and $\text{IP}^{\mathbb{F}}$. Let $\text{pSETINT}_{p,n} : (\{0,1\}^n)^p \to \{0,1\}$ be the *p-player n-bit promise intersection* function defined as follows. On input $(x_1, \ldots, x_p)$, $x_i \in \{0,1\}^n$ the promise is that either the sets $\mathsf{supp}(x_i)$ are mutually disjoint or their intersection is a singleton set. For a promised input $x = (x_1, \ldots, x_p)$ define $\text{pSETINT}_{p,n}(x) = 1$ iff $\bigcap_{i=1}^{p} \mathsf{supp}(x_i) = \{a\}$, $a \in [n]$. Let $\text{IP}^{\mathbb{F}}_{2,n} : (\mathbb{F}^n)^2 \to \{0,1\}$ be the *2-player n-ary inner product* function over field $\mathbb{F}$, defined by $\text{IP}^{\mathbb{F}}_{2,n}(x_1, x_2) = \sum_{i=1}^{n} x_{1,i}x_{2,i}$. By [CKS03], $R(\text{pSETINT}_{p,n}) = \Omega(\frac{n}{p \log p})$. By a simple reduction from $\text{pSETINT}_{2,n}$, $R(\text{IP}^{\mathbb{F}}_{2,n}) = \Omega(n)$.

Our lower bound is based on a direct sum type of argument for NIH communication complexity for set disjointness and generalized inner product. Using its stack, a SM can solve efficiently the straightforwardly defined direct sum version of the problems. To that end, we apply permutations on the inputs.

Definition 11. Let $X = \{0,1\}^n$. Consider a *base function* $f : X^p \to \{0,1\}$, $p \geq 2$. In the communication complexity model player I gets x_i, from an input $(x_1, \ldots, x_p) \in X^p$. Let $\Phi = (\phi_1, \ldots, \phi_p)$ be a sequence of permutations. The (pmn)-bit function $f^{\vee,\Phi} : (X^m)^p \to \{0,1\}$ is defined on input $x = (x_1, \ldots, x_p) \in X^{mp}$, where $x_i = (x_{i,1}, \ldots, x_{i,m}) \in X^m$ as follows: $f^{\vee,\Phi}(x) = \bigvee_{j=1}^{m} f(x_{[j],\Phi})$, where $x_{[j],\Phi} = (x_{1,\phi_1(j)}, \ldots, x_{p,\phi_p(j)})$. When no confusion arises instead of $x_{[j],\Phi}$ we write $x_{[j]}$. When f is the generalized inner product then $f^{\oplus,\Phi}(x) = \bigoplus_{j=1}^{m} f(x_{[j],\Phi})$, is also a lifting of the given inner product problem, which by definition is a bigger (permuted) inner product instance.

Notation: Let $n \geq 1$ and let $X = \{0,1\}^n$. Let $m \geq 1$ and $p \geq 2$. Let $f = \text{pSETINT}_{p,n} : X^p \to \{0,1\}$. Let $g = \text{IP}^{\mathbb{F}}_{2,n}$. Let $\Phi = (\phi_1, \ldots, \phi_p) \in (S_m)^p$ be the sequence of p permutations on $[m]$ that have small sortedness (see Appendix A). Recall Definition 11 of $f^{\vee,\Phi}$ and of $g^{\oplus,\Phi}$. Let $r, s : \mathbb{N} \to \mathbb{N}$ be functions. Let $d \leq O(p(r(mnp))^2/\sqrt{m})$. In what follows the choice of r and m will be such that $p(r(mnp))^2 = o(\sqrt{m})$.

5.2 A Streaming Perspective and A Comparison to [BHN08]

The lower bound is a corollary of a technical result which can be independently interpreted from the perspective of *streaming*. A detailed treatment of Section 5 is given in Appendix A.

The streaming model of computation was introduced by [AMS99] in an attempt to model the huge real-world differences in access times to internal memory (e.g., RAM) and external memory (e.g., hard drives). In this model, the computation device is a space bounded TM that is allowed a bounded number of reversals on the input tape. An (r, s)-Stack Machine can be naturally seen as an extension of the original streaming model, in which the TM is given access to an unbounded stack.

One of the most important problems studied in the context of streaming is that of computing the frequency moments of a data stream. The *k-th frequency moment* F_k of a sequence $\alpha = (a_1, \ldots, a_M)$, $a_i \in [R]$

is $F_k(\alpha) = \sum_{j \in |R|} (f_j)^k$, where $f_j = |\{i \ : \ a_i = j\}|$. [4] Positive results for approximating this problem are given by [AMS99, IW05, BGKS06], and negative results are given by [AMS99, SS02, BYJKS04, CKS03].

[GS05] introduced another extension of the streaming model of computation, called an (r, s, t)-read/write stream algorithm: a TM with t external tapes on which the heads can reverse a total of r times, and internal tapes of total size s. [BHN08], building on [BJR07, GS05, GHS06], prove that an $(o(\log n), s, O(1))$-r/w stream algorithm that approximates the k-th frequency moment of a data stream requires $s = n^{1-4/k-\delta}$ for any $k > 4$ and $\delta > 0$.

Our lower bound proof is inspired by [BHN08], but the heart of our argument is entirely new. Below, we address how r/w stream algorithms compare with SMs, and the novelty in our proof.

Comparison of computational models. In Lemma 12 (the proof is given in Appendix A.5) we formally show that under a widely-believed complexity assumption[5], the unboundness when accessing the stack makes our model stronger than the one considered in [BHN08], for some languages in P.

Lemma 12 (The power of one-way SMs)**.** *Let $m \in \mathbb{Z}^+$. There exists $L \in \mathsf{P}$ such that (i) there is a SM deciding L with one scan over the input, and (ii) every TM M with a logspace work tape (on which we don't count reversals) and a constant number of tapes on which we count reversals, M cannot decide L with $\log^m N$ reversals, unless $\mathsf{E} \subseteq \mathsf{PSPACE}$.*

This lemma refers to the (uniform) models of interest. We derive the lower bound (Theorem 13) using communication complexity, which means that inherently this lower bound applies even to non-uniform machines. Hence, Lemma 12 leaves open the possibility that lower bounds on the non-uniform version of our model follow by lower bounds on the non-uniform version of $(r, s, 2)$-r/w stream algorithms. However, it is impossible to prove *polynomial* lower bounds - as we do in this paper for SMs - for $(r, s, 2)$-r/w stream algorithms [6].

Comparison of proof techniques. The *general setup* of both proofs is the same. We assume we have an efficient r/w-stream algorithm/SM M (henceforth, referred to as "machine") that solves a "permuted" product of several instances of a base function f. Using this machine, we construct an efficient communication protocol for f as follows. Given an input instance to the protocol, referred to as the "real" instance, the players construct many "decoy" instances, they combine them together with the real one, and they simulate the machine on this extended input. The fact that M solves a permuted product of instances to f is used in order to argue that M must dedicate its resources (external tapes in [BHN08], stack in our case) to solving only a fraction of all instances.

The *heart* of both arguments is an "algorithmic" part, which says that if M does not dedicate its resources to solving the real instance, then there exists an efficient communication protocol simulating M. The difference between our proof and the one in [BHN08] is the very heart of the argument. [BHN08] gives an efficient simulation of an (r, s, t)-r/w stream algorithm, whereas we give an entirely new simulation of an (r, s)-SM.

A subtle issue, which makes our simulation somehow unexpected, is that 2 stacks have the full power of an unrestricted TM. In contrast, the simulation in [BHN08] works for any number of tapes.

[4] We consider this problem for $M = R^{\Theta(1)}$. For the computational/streaming problems we are interested in approximating $F_k(\alpha)$ where α is presented in the input by listing the elements of α, each of which is encoded using $\Theta(\log R)$ bits; i.e. the input length is also $N = R^{\Theta(1)}$.

[5] It is straightforward to show unconditionally that $\mathsf{E} \neq \mathsf{PSPACE}$. It is usually thought that these classes are incomparable.

[6] Because an $(O(\log n), s, 2)$-r/w stream algorithm can *sort*, which is enough to compute frequency moments exactly [GHS06].

Finally, note that we show a polynomial lower bound on the number of reversals on the worktape, whereas [BHN08] can only show a logarithmic lower bound.

5.3 The Statement of the Technical Result

Theorem 13 (Main Reduction)**.** *Assume there exists a randomized (r, s)-Stack Machine M for $f^{\vee,\Phi}$ with error at most δ. Then, there exists a randomized protocol P for f, with cost $O(p^2 \cdot (r(mnp))^2 \cdot s(mnp))$ and error at most $\delta + d(1-\delta)$. Furthermore, the same holds when we replace f by g, and $f^{\vee,\Phi}$ by $g^{\oplus,\Phi}$.*

Corollary 14. *Let $k > 5$ and $\epsilon \geq 0$ be constants. There exists a constant $\beta > 0$ such that any (r, s)-SM A_N computing an $(1+\epsilon)$ approximation of F_k with constant error requires $r = \omega(N^\beta)$ or $s = \omega(N^\beta)$.*

Corollary 15. *Let $\mathbb{F}$ be any field, let $p = 2$, and let $m = n^{8/6}$. Then, any (r, s)-SM A_N computing $\left(\mathrm{IP}_{2,n}^{\mathbb{F}}\right)^{\oplus,\Phi}$ with constant error requires $r = \omega(N^{1/8})$ or $s = \omega(N^{1/8})$.*

6 Conclusions and Future Work

This paper initiates the study of randomness based on the number of accesses to the random bits as opposed to the amount of random bits a polytime algorithm uses. A meaningful definition becomes possible by extending SMs with a polynomially-long auxiliary tape. Parametrizing on the number of head-reversals on the random tape we define hierarchies lying in-between polynomial time and probabilistic polynomial time. When one considers the auxiliary resource to consist of non-deterministic or non-uniform bits one can define similar hierarchies.

We also introduce an alternative, equivalent definition for the use of randomness, where a polynomial time algorithm on a given input compiles all the randomness it needs in a polynomial size, swallow circuit. One implication is that derandomization (using PRGs) of the RNC hierarchy, which is believed to be a small fraction of P, implies derandomization for classes that lie much higher than RNC. Another consequence is that this definition provides an alternative perspective, which may find use in devising algorithms in the lower levels of our hierarchy. The extensive literature in parallel computation can be potentially used towards this goal.

The derandomization for one-sided error classes is an immediate corollary of the Dynamic Programming algorithm given in [Coo71]. A non-trivial modification of this algorithm is used to derandomize the first level of two-sided error probabilistic polynomial time, even with unbounded error. Other than this the resolution of the fundamental complexity questions motivating this paper remains widely open. In [Ven06] a pseudorandom generator is given against polynomial time vSMs, with one-way access over the randomness. It is conceivable that this generator may be secure against more than one scans over the randomness. We have recently obtained [DPS09] such a result for logspace TMs without a stack, by investigating properties of Nisan's PRG [Nis92] against logspace, and one-scan over the randomness.

Our main lower bound refers to a particular use of the random stream, showing that every algorithm which evaluates the given polynomial on the provided random point witnesses containment of PIT away from P, with respect to our parameterization of head-reversals. Although this result does not have immediate structural consequences, it does provide grounds for future research according to the proposed framework. Proving unconditional lower bounds for time and space resources is in general a far-to-reach goal. On the other hand, we see that there is a certain success in proving such lower bounds on head reversals. One may speculate that this opens the possibility the proof techniques introduced in this paper to find application in the derandomization of the randomized hierarchy.

This paper gives rise to a number of questions (even in relation to areas such as Cryptography, randomness extractors etc) not possible to be listed here in some comprehensive form. We conclude by mentioning five of them which at this stage seem to be more relevant to our study.

- *Derandomization of the first few levels of the $\mathsf{RPdL}[\cdot]$ hierarchy.* In particular, derandomizing even the second level of the hierarchy is interesting. It seems that new tools have to be developed towards this direction.
- *Derandomize the first few level of $\mathsf{BPPdL}[\cdot]$.* Note that in the two-sided error case it would also be interesting to show containment in NP.
- *Investigate structural relations among levels of the hierarchies.* For example, is it true that $\mathsf{BPPdL}[k] \subseteq \mathsf{RPdL}[k']$ for some $k' > k$? Another possibility would be to prove theorems of the form e.g. "if $\mathsf{P} \neq \mathsf{RP}$ then $\mathsf{RPdL}[\mathsf{exp}] \neq \mathsf{RPdL}[r(N)]$". That is, to derandomize RP it is sufficient to derandomize only the first $r(N)$ many levels of the randomized hierarchy.
- *Complexity-theoretic questions for variants of our model.* For example, we have shown that constantly-many reversals on the non-deterministic tape of a logspace TM cannot get us outside NL. We also know that polynomially many reversals are sufficient to characterize NP. Thus it seems interesting to check whether for superlogarithmic number of reversals we remain inside P. Another question is to study what happens when the auxiliary tape of the SM is bigger than polynomial. It is easy to see that if the random tape is exponentially long tape we characterize PSPACE. What happens with a subexponentially-long tape? For example, is there any relation to the levels of the polynomial hierarchy? How does the number of reversals become relevant in such a case?
- *Devise PIT randomized algorithms with reduced number of reversals*. Instead of trying to reduce the number of random bits an algorithm for Polynomial Identity Testing uses, design algorithms that use as few reversals over the randomness as possible. Equivalently (using randomness compilers), devise randomized algorithms where their "deterministic part" is arbitrary polytime and their "randomized part" is parallelizable.

The proposed approach to randomness creates potential to bring together tools developed in many excellent works in the areas of derandomization, communication complexity, streaming and structural complexity related to AuxPDAs (SMs). Devising new tools based on amalgamating results from these areas seems to be our best bet for the moment.

Acknowledgements

We would like to thank Phuong Nguyen for his extensive collaboration in the beginnings of this research several years ago. Our thanks to Mark Braverman whose remarks initiated the study of the relation between RPdL and $\mathsf{P+RNC}$. We would also like to thank Siavosh Benabbas, Stephen Cook, Faith Ellen, Toniann Pitassi and Charles Rackoff for many helpful discussions.

References

[AB99] M. Agrawal and S. Biswas. Primality and identity testing via Chinese remaindering. *J. ACM*, 50(4):429–443 (electronic), 2003 (preliminary version in FOCS 1999).

[AB08] S. Arora and B. Barak. *Complexity Theory: A Modern Approach*. (draft), 2008.

[ABKPM06] E. Allender, P. Burgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. In *CCC 2006*, volume 21, 2006.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[All89] E. W. Allender. P-uniform circuit complexity. *J. Assoc. Comput. Mach.*, 36(4):912–928, 1989.

[AMS99] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[BCD$^+$89] A. Borodin, S. A. Cook, P. Dymond, L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SICOMP: SIAM Journal on Computing*, 18, 1989.

[BGKS06] L. Bhuvanagiri, S. Ganguly, D. Kesh, and C. Saha. Simpler algorithm for estimating frequency moments of data streams. In *SODA 2006*, pages 708–713, New York, 2006. ACM.

[BHN08] P. Beame and D.-T. Huynh-Ngoc. On the value of multiple read/write streams for approximating frequency moments. In *FOCS 2008*, 2008.

[BJR07] P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *STOC 2007*, 2007.

[BYJKS04] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. System Sci.*, 68(4):702–732, 2004.

[CK97] Z.-Z. Chen and M.-Y. Kao. Reducing randomness via irrational numbers. *SIAM J. Comput.*, 29(4):1247–1256 (electronic), 2000 (preliminary version in STOC 1997).

[CKS03] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *CCC 2003*, pages 107–117. IEEE Computer Society, 2003.

[Coo71] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. Assoc. Comput. Mach.*, 18:4–18, 1971.

[DPS09] M. David, P. A. Papakonstantinou, and A. Sidiropoulos. On the derandomization of RNC^1. (unpublished manuscript), March 2009.

[ES35] P. Erdös and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

[GHS06] M. Grohe, A. Hernich, and N. Schweikardt. Randomized computations on large data sets: Tight lower bounds. In *PODC06*, 2006.

[GS05] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *PODS 2005*, 2005.

[HS08] A. Hernich and N. Schweikardt. Reversal complexity revisited. *Theoret. Comput. Sci.*, 401(1-3):191–205, 2008.

[IW97] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC 1997*, 1997.

[IW05] P. Indyk and D. P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC 2005*, pages 202–208. ACM, 2005.

[KI03] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complexity*, 13(1-2):1–46, 2004 (preliminary version in STOC 2003).

[KN97] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, New York, NY, USA, 1997.

[KS01] A. R. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC 2001*, pages 216–223 (electronic), New York, 2001. ACM.

[LV98] D. Lewin and S. Vadhan. Checking polynomial identities over any field: towards a derandomization? In *STOC 1998*, pages 438–447. ACM, New York, 1998.

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

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in logspace. *Theoret. Comput. Sci.*, 107(1):135–144, 1993. Structure in complexity theory (Barcelona, 1990).

[NW88] N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. System Sci.*, 49(2):149–167, 1994 (preliminary version in FOCS 1988).

[Ruz80] W. L. Ruzzo. Tree-size bounded alternation. *J. Comput. System Sci.*, 21(2):218–235, 1980.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22(3):365–383, 1981. Special issue dedicated to Michael Machtey.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, 27(4):701–717, 1980.

[SS02] M. Saks and X. Sun. Space lower bounds for distance approximation in the data stream model. In *STOC 2002*, pages 360–369 (electronic), New York, 2002. ACM.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. System Sci.*, 62(2):236–266, 2001 (preliminary version in STOC 1999). Special issue on the Fourteenth Annual IEEE Conference on Computational Complexity (Atlanta, GA, 1999).

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. System Sci.*, 67(2):419–440, 2003. Special issue on STOC2002 (Montreal, QC).

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. System Sci.*, 43(2):380–404, 1991.

[Ven06] H. Venkateswaran. Derandomization of probabilistic auxiliary pushdown automata classes. In *CCC 2006*, volume 21, 2006.

[Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *FOCS 1982*, pages 80–91. IEEE, New York, 1982.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic computation (EUROSAM '79, Internat. Sympos., Marseille, 1979)*, volume 72 of *Lecture Notes in Comput. Sci.*, pages 216–226. Springer, Berlin, 1979.

Appendix

A Streaming Lower Bound

A.1 The family of Schwartz-Zippel algorithms

Our main lower bound is on the number of reversals when computing a permuted version of inner product, a special case of polynomial evaluation. This, in particular rules-out the family of SZ-algorithms, defined in Section 5. Furthermore, our lower bound rules-out a slightly bigger family, which allows us to include the algorithms in [LV98, CK97]. If a SM M:

> (i) evaluates the input polynomial P at a point $X = (X_1, \ldots, X_m) \in \mathbb{F}^m$, and
> (ii) derives the point X from its random tape R by partitioning R into m contiguous pieces $(R_1, \ldots, R_m)$, and arbitrarily, but surjectively, computing X_i from R_i (i.e., X_i may not depend on $R_{i'}$ for $i' \neq i$, and X_i takes all possible values in $\mathbb{F}$ as R_i varies);

then M makes N^ϵ reversals on its random tape, for some $\epsilon > 0$. Thus, such algorithms cannot show that PIT is in a level lower than $\mathsf{coRPdL}[N^\epsilon]$.

A.2 Preliminaries, sketch of main lemmas, and proof of the main theorem

In this section, we prove Theorem 13. We give the intuition and the proof for the case when $f = \textsc{pSetInt}_{p,n}$ and M computes $f^{\vee,\Phi}$.

Overview of the proof. We show that evaluating a polynomial at a given point requires polynomially many reversals on the input tape. For this we rely on the lower bounds for $\textsc{pSetInt}$ and $\mathrm{IP}^{\mathbb{F}}$ (see Section 2). Hence, it suffices to present a protocol that efficiently simulates an (r, s)-Stack Machine.

Our goal is to construct a randomized protocol P which runs on the given input. As an intermediate step we construct a deterministic protocol P' where the players in P they use P' as a routine. In particular, in the description of P the players first transform the given (actual) input into an artificial bigger input using shared randomness and the actual input. Then, they simulate the deterministic protocol P' on this special input they have created. The extension of the given input is such that the output to this input is the same as for the actual one.

The construction of P' is the main bulk of the proof. The intuitive goal is to show that in P', it is possible for the players to perform the simulation of the Stack Machine without communicating the stack content.

1. Identify an obstruction to efficient simulation. This is the definition of a *corrupted instance* (Definition 18). In absence of the obstruction it is possible to simulate the (r, s)-Stack Machine without communicating the stack content.

2. Bound the frequency that this obstruction occurs (Lemma 20).

3. Construct the deterministic protocol P' (Lemma 19). This protocol may abort the simulation when we have a corrupted instance. However, the protocol P' identifies when the situation of a corrupted instance occurs. When we do not have a corrupted instance then the protocol works correctly and it is efficient, in the sense that the players do not have to communicate the stack content during the simulation of the machine.

4. Use Yao's min-max principle to show the existence of a randomized protocol P (Theorem 13). Randomization is mainly used to construct the extension and to avoid (in a probabilistic sense) the situation where we have a corrupted instance.

In Remark 21, we explain what is different in the case when $g = \mathrm{IP}^{\mathbb{F}}_{p,n}$ and M computes $g^{\oplus,\Phi}$. We begin by defining the sortedness of a permutation, an important concept for our construction.

Permutations, Permuted Functions and Frequency Moments. For a permutation $\phi \in S_m$ we define its *sortedness*, denoted by $\mathrm{sortedness}(\phi)$, to be the length of the longest monotone subsequence of $(\phi(1), \ldots, \phi(m))$. The *relative sortedness* of $\phi_1, \phi_2 \in S_m$ is defined as $\mathrm{relsorted}(\phi_1, \phi_2) = \mathrm{sortedness}(\phi_1 \circ \phi_2^{-1})$. In general, for a sequence of permutations $\Phi = (\phi_1, \ldots, \phi_p)$, let $\mathrm{relsorted}(\Phi) = \max_{i \neq j}(\mathrm{relsorted}(\phi_i, \phi_j))$. We are interested in sequences of permutations with small relative sortedness. The relative sortedness of any two permutations is at least $\sqrt{m}$ [ES35] , so the following is optimal up to constants.

Fact 16 (Corollary 2.2 in [BHN08])**.** *Let $p = m^{O(1)}$. Then, there exists a sequence $\Phi \in (S_m)^p$ such that* $\mathrm{relsorted}(\Phi) = O(\sqrt{m})$.

Additional definitions and notation. Since in each transition exactly one of the following happens:either a head-move, or a push to the stack or a pop from the stack, we refer to *move transitions*, *push transitions* and *pop transitions*, respectively. For a field $\mathbb{F}$ and $\mathrm{IP}^{\mathbb{F}}_{2,n}$ we write 0-*inputs* to refer to the subset of $\mathbb{F}^{2n}$ which consists of every element whose inner product is 0. This is not to be confused with the all-0 vector in $\mathbb{F}^{2n}$ which is just one 0-input.

So, given a Stack Machine M for $f^{\vee,\Phi}$, we build a communication protocol for f. Let x be an input to P. Our strategy is as follows: the players in P extend x to an input v to $f^{\vee,\Phi}$ by randomly choosing an instance number $j \in [m]$, randomly choosing $m-1$ 0-inputs $\overline{y} = (y^1, \ldots, y^{m-1})$ to f, embedding x as instance number j and $\overline{y}$ as the other instances in an input $v(j, x, \overline{y})$ to M, and simulating M on input v.

Definition 17. For the fixed Φ, let $j \in [m]$, $x \in X^p$ and $\overline{y} = (y^1, \ldots, y^{m-1}) \in (X^p)^{m-1}$. Recalling Definition 11, we define $v = v(j, x, \overline{y}) \in (X^m)^p$ to be the string satisfying: $v_{[j]} = x$; $v_{[j']} = y^{j'}$ for $1 \leq j' < j$; and $v_{[j']} = y^{j'-1}$ for $j < j' \leq m$.

Since $\overline{y}$ consists of 0-inputs to f, it's clear that $f^{\vee,\Phi}(v(j, x, \overline{y})) = f(x)$. Thus, if the players in P manage to simulate M on $v(j, x, \overline{y})$, they obtain $f(x)$.

Assume M is deterministic, and let Γ denote the sequence of configurations of M on input $v = v(j, x, \overline{y})$. The players in protocol P collectively see most of the tape of M, except for the parts where their respective inputs are embedded into $v(j, x, \overline{y})$. Specifically, player i is the only one who sees the symbols in $v_{i,\phi_i(j)}$, which is x_i. In protocol P, the players will take turns simulating the transitions in Γ one by one, and it will be the job of player i to simulate the transitions when the input head is scanning $v_{i,\phi_i(j)}$. Intuitively, we expect that by using $m > 1$ and the special sequence of permutations Φ, the machine M does not use its stack on instance j, which is the one we really want to solve.

Definition 18. Let $j' \in [m]$ and let $v' \in (X^m)^p$. [7] Let M' be a deterministic (r, s)-Stack Machine. Let Γ' be the sequence of configurations of M' on v'. Partition Γ' into r contiguous "scans" according to the movement of the input head. We say that *instance j' is corrupted in input v' on machine M'* if the following holds. There exist scans $l_1 \leq l_2 \in [r]$, there exist $i_1 \neq i_2 \in [p]$, and configurations $\gamma_1, \gamma_2 \in \Gamma'$ such that:

[7] In this definition, j' and v' are *not* necessarily related by $v' = v(j', x, \overline{y})$ for some $x, \overline{y}$.

(i) for $a \in [2]$, γ_a belongs to scan l_a and the input head in γ_a is scanning a symbol from $v'_{i_a,\phi_{i_a}(j')}$; (ii) the transition out of γ_1 is a push transition (see Section 2); (iii) the transition out of γ_2 is a pop transition; (iv) the stack height is the same in γ_1 and in the configuration following γ_2; and (v) the stack height is strictly higher in between them. Let $\mathrm{BAD}(M', v') \subseteq [m]$ denote the set of instances that are corrupted in v' on M'.

Intuitively, according to this definition, instance j is corrupted in input $v(j, x, \overline{y})$ on Stack Machine M if M ever pushes a symbol on the stack in a transition that is to be simulated by player i_1, and later pops that same symbol in a transition that is to be simulated by another player $i_2 \neq i_1$. We claim that indeed, when j is *not* corrupted in $v(j, x, \overline{y})$ on M, the protocol P can efficiently simulate M on $v(j, x, \overline{y})$.

Lemma 19. *Let M' be a deterministic (r, s)-Stack Machine for $f^{\vee,\Phi}$. Let $j \in [m]$ and let $\overline{y} \in (X^p)^{m-1}$. There exists a deterministic communication protocol $P' = P'(M', j, \overline{y})$ such that, on input $x \in X^p$: (i) if $j \in \mathrm{BAD}(M', v(j, x, \overline{y}))$, then P' outputs "fail"; (ii) otherwise, P' outputs $M'(v(j, x, \overline{y}))$; and (iii) the cost of P' is $O(p^2 \cdot (r(mnp))^2 \cdot s(mnp))$.*

We defer the proof of this lemma to Appendix A.4. This lemma is the main part of the reduction. It is not hard to see how it is possible to efficiently detect whether j is corrupted in v. Assuming that j is not corrupted it is possible to efficiently simulate the Stack Machine; i.e. without communicating the entire stack content. Let us give an example so as to develop some intuition. Although the machine computation described in this example is among the simpler cases when it comes to efficient simulation by the players, this example does illustrate some of the main issues.

Example for a special case of the efficient simulation. Consider the case of two players. Both players know the 0-inputs. We refer to the part of the 0-inputs as public input zone. If a stack symbol was pushed to the stack during the simulation of M' when the head was over a public zone then this stack symbol (for the corresponding time step and stack height) is a public stack symbol. This way we define the concept of public stack zone. Similarly, we define the private zones for Player-i. The symbols in the stack, have value (i.e. the actual symbol) and in addition we label them according to which zone the input head was when they were pushed in the stack. This distinction is necessary, because during the simulation, for a particular stack height a player may know the label of the input-zone associated with the stack symbol, but not the actual symbol.

Consider the situation depicted on Figure 1 and suppose that the simulation has reached time t_1. On the right we depict the stack content which was created by M by computing on the input and by possibly making a few scans over the tape between time 0 and t_1. Furthermore, assume the invariant that up to time t_1 each player knows: (i) the position (stack height) of every public and private stack-symbol, (ii) the symbol for each public stack symbol, (iii) Player-i knows her private stack symbols and (iv) the top stack symbol of the private stack zone which belongs to the other player. Player-1 will carry the simulation at this point since the input head enters her private zone. She can do the simulation up to time t_2 where the head exits her private zone. As the simulation proceeds she pushes to the stack her private stack symbols and when she pops stack symbols these are either private to her or public. At time t_2 it is not a good idea to send to Player-2 the stack height and the input-head position. The reason is that later on and as long as the head is over the public zone, the popped stack symbols enter at time t_3 the private zone of Player-1 (and Player-2 does not know these stack symbols). Thus, Player-1 continues to simulate after t_2 and up to the lowest level l_4 of the stack with the input head over the public zone. At this point and since she knows this is the lowest level, she sends to Player-2 the stack height, the top stack symbol, and the position of the input head. Since this is the lowest stack height and the input head is over the publicly known zone both Players can independently reconstruct the public stack zone from time t_4 all the way to time t_5, where Player-2 takes

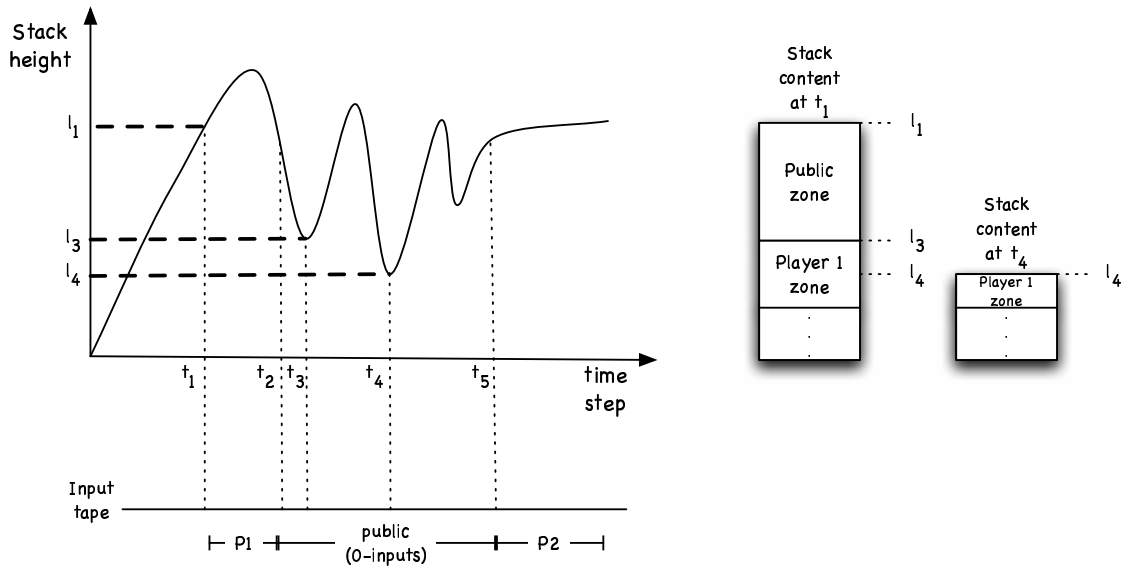

Figure 1: On the upper part we depict how the stack height changes as a function of the simulation step of M' on the given input. On the right side of this upper part we depict the stack content at two different times. On the lower part we depict the input and which part of the input is known only to player 1, only to player 2 and which is known to both (public).

over the simulation. This completes the description of the example. □

We would like to make sure that in protocol P, the probability that instance j is corrupted on $v(j, x, \overline{y})$ is small. By construction, of the m inputs to f embedded in $v(j, x, \overline{y})$, $m - 1$ of them are always 0-inputs. If, on the one hand, $f(x) = 1$, then instance j will be, intuitively, probabilistically distinguishable, so it is plausible that M dedicates the use of its stack to solving this instance, causing j to be corrupted in $v(j, x, \overline{y})$. However, note that the protocol P' from Lemma 19 can detect that j is corrupted in $v(j, x, \overline{y})$. So, in this case, the protocol P will simply output 1. If, on the other hand, $f(x) = 0$, then $v(j, x, \overline{y})$ consists of m 0-inputs to f. In the following Lemma (proof deferred in Appendix A.4), we use the property of the sequence of permutations Φ to give an upper bound on the number of instances that are corrupted in a fixed input v'.

Lemma 20. *Let M' be a deterministic (r, s)-Stack Machine for $f^{\vee,\Phi}$ and let $v' \in (X^m)^p$. Then, we have $|\mathrm{BAD}(M', v')| \leq O(p \cdot r(mnp) \cdot \log(r(mnp)) \cdot \sqrt{m})$.*

In the case $f(x) = 0$, if we were to show that the choice of j *is statistically independent from* $v(j, x, \overline{y})$, the Lemma above would give us a bound for the probability that j is corrupted in $v(j, x, \overline{y})$. A subtle point is that we cannot immediately prove this for *every* 0-input x. We overcome this by proving the statement distributionally, when x and the 0-inputs in $\overline{y}$ come from the same distribution. [8] To implement this intuition, we use Yao's min-max principle that connects the randomized communication complexity of a function with its distributional complexity.

[8] As explained in [BHN08], another way would be to observe that for the special case of $f = \textsc{pSetInt}$, any 0-input can be transformed into any other 0-input by a column permutation, and then modifying the protocol P to apply such a random permutation to x itself.

Proof of Theorem 13 (for the case when $f = \text{pSETINT}_{p,n}$). For every q, let M_q denote the deterministic (r, s)-Stack Machine obtained by running M with random string ρ.

Let D be any distribution on the space X^p of inputs to f. Below, we give a randomized protocol P_D for f with cost and error as described in Theorem 13, but with error computed over the choice of the input x from D *and* of the random coins ρ. By standard arguments, we can fix ρ to obtain a deterministic protocol with the same error, but only over the choice of x from D. Since such a protocol exists for every distribution D, by Yao's min-max principle (e.g., Theorem 3.20 in [KN97]), the conclusion of Theorem 13 follows.

If D has support only on 0-inputs or only on 1-inputs, P_D is trivial. Otherwise, let $D_0 = D$ conditioned on $f(x) = 0$.

Consider the following protocol P_D:

On input $x \in X^p$, where player i gets $x_i \in X$, and shared random string ρ:
Publicly draw j uniformly from $[m]$
publicly draw $\overline{y} = (y^1, \ldots, y^{m-1})$ from D_0^{m-1}
publicly draw q uniformly
Run $P' = P'(M_q, j, \overline{y})$ (from Lemma 19) on input x, simulating $M_q(v(x, j, \overline{y}))$
If P' outputs "fail", then P_D outputs 1. Else, P_D outputs the same value as P'

The cost of P_D is equal to the cost of P' which is $O(p^2(r(mnp))^2 s(mnp))$. To argue about the error in P_D, define the following events:

$A = A(x, \rho)$: P_D is correct on input x with coins ρ
$B = B(q, v)$: M is correct on input v with coins q and
$C = C(j, q, v)$: $j \notin \text{BAD}(M_q, v)$.

By Lemma 19, the simulation in P' fails if and only if $\overline{C}$. Let $\alpha = \Pr_x[f(x) = 0]$. We write $\Pr[A] = \alpha \cdot \Pr[A|f(x) = 0] + (1 - \alpha) \cdot \Pr[A|f(x) = 1]$.

For the right term, consider the conditioning $f(x) = 1$. Then, P_D is correct if either the simulation in P' fails (and P_D correctly outputs 1), or the simulation in P' does not fail and M is correct, thus, $\overline{C} \vee (C \wedge B) \Rightarrow A$, in particular $B \Rightarrow A$, so $\Pr[A|f(x) = 1] \geq \Pr[B|f(x) = 1]$. The event $f(x) = 1$ is independent of the choice of q, and by the correctness condition of M, $\forall v, \Pr_Q[B(Q, v)] \geq (1 - \delta)$. Hence, $\Pr[B|f(x) = 1] = \Pr[B] \geq (1 - \delta)$.

For the left term, consider the conditioning $f(x) = 0$. Then, P_D is correct whenever the simulation in P' works and M is correct, thus $B \wedge C \Rightarrow A$, and $\Pr[A|f(x) = 0] \geq \Pr[B|f(x) = 0] \Pr[C|B, f(x) = 0]$. As before, $\Pr[B|f(x) = 0] \geq (1 - \delta)$. Next, j is clearly statistically independent of x and q. Furthermore, under the conditioning $f(x) = 0$, we claim that j *is statistically independent from* v. To see this, notice how the distribution obtained on pairs (j, v) in P_D is the same as independently choosing m 0-inputs from D_0 and combining them in v, and choosing j uniformly from $[m]$. Thus, in the expression $\Pr[C(j, q, v)|B(q, v), f(x) = 0]$, the conditioning depends on (x, v, q), C itself depends on j, and j is statistically independent from (x, v, q). By Lemma 20, $\forall (q, v), \Pr_J[C(J, q, v)] \geq (1 - d)$. Then, $\Pr[C|B, f(x) = 0] = \Pr[C] \geq (1 - d)$.

Putting everything together gives the claimed error bound, $\Pr[A] \geq 1 - (\delta + d(1 - \delta))$. □

Remark 21. Theorem 13 states that if $g = \text{IP}^{\mathbb{F}}_{2,n}$ and M is a Stack Machine for $g^{\oplus,\Phi}$, we can build a similar protocol P for g. One can check that Definitions 17, 18, Lemmas 19 and 20 do *not* depend on either the "base" function f, or on the "combining" function $\vee$. Furthermore, one can check that even the proof for the case $f = \text{pSETINT}_{p,n}$ goes though if we replace f by g, and $f^{\vee,\Phi}$ by $g^{\oplus,\Phi}$. Note that if we were to write

the proof for the case of g and $g^{\oplus,\Phi}$, we need not bother defining distribution D_0: the players in P_D could simply draw $\overline{y}$ from D^{m-1} (rather than D_0^{m-1}), because they could still compute $g(x)$ from $g^{\oplus,\Phi}(v(j,x,\overline{y}))$ by adding to the latter the outputs of the $m-1$ inputs in $\overline{y}$.

A.3 Proof of Corollaries 14 and 15

To prove Corollary 14, we need the following result. This reduction originates, in a non-permuted form, from [AMS99]. [BHN08] point out that the reduction still works for permuted instances of pSETINT. However, while [BHN08] make use of an additional external tape to make it work, we do not have that luxury when dealing with Stack Machines. We explain how to get the reduction to work with no additional tapes. Intuitively, the idea is that during the simulation we only need to look at a limited portion of the tape. We also slightly improve the dependency between parameters over [BHN08].

Theorem 22. *Let $\epsilon \geq 0$. Let $M = mnp$, with $p \geq (mn)^{\Omega(1)}$.*

1. *Let $k > 1$ and $p > (2\epsilon(2+\epsilon)mn)^{1/k}$. Assume that, for large N, there exists a nonuniform (r,s)-Stack Machine A_N that, given a sequence a of $N^{\Theta(1)}$ elements from $[N^{\Theta(1)}]$, outputs a $(1+\epsilon)$-approximation of $F_k(a)$. Then, for large M, there exists a nonuniform $(r+2, s+O(\log M))$-Stack Machine B_M such that, when given an input $v \in \{0,1\}^M$, B_M outputs $f^{\vee,\Phi}(v)$, where $f = \text{pSETINT}_{p,n}$. Moreover, if A_N is randomized with error at most $\delta < 1/2$, then so is B_M.*

2. *The same holds when $k < 1$ and $p > 2\epsilon(2+\epsilon)mn/(1+\epsilon)^2$.*

Proof of Theorem 22. First, consider the case $k > 1$. Let $v \in \{0,1\}^{mnp}$ be an input to $f^{\vee,\Phi}(v)$.

To begin with, B_M uses its first two reversals to count the number of 1s in v, storing this value as C on $\Theta(\log M)$ bits. If $C < p$, B_M outputs 0, as none of the m instances of $\text{pSETINT}_{p,n}$ inside $f^{\vee,\Phi}$ can contain an all-1 column. Similarly, if $C > mn$, B_M outputs 1. To see this is correct, notice that by the promise, one 0-instance of $\text{pSETINT}_{p,n}$ contains at most n 1s, one in each column. Thus, m 0-instances of $\text{pSETINT}_{p,n}$ contain at most mn 1s. Henceforth, assume $p \leq C \leq mn$.

Let $N = C\log(mn)$. Note that $N \leq mn\log mn \leq mnp = M$ since $p = (mn)^{\Omega(1)}$. Also, $N \geq p\log mn \geq M^{\Omega(1)}$, so A_N exists for large M. The machine B_M simulates the machine A_N on the following N-bit input stream $a = a(v)$: for every $(i,j,k) \in [p] \times [m] \times [n]$ such that $v_{i,j,k} = 1$ (recall the notation from Definition 11), the stream a contains a value $(\phi_i^{-1}(j)-1)n + k \in [mn]$. It is not hard to check the following property of this construction:

> If $f^{\vee,\Phi}(v) = 0$, the values in the stream a are distinct;
> and if $f^{\vee,\Phi}(v) = 1$, the stream a contains p occurrences of some value, and the rest are distinct.

Suppose that B_M can somehow simulate A_N on input a. So, A_N outputs a value y which is an $(1+\epsilon)$-approximation of $F_k(a)$. Note that:

> If $f^{\vee,\Phi}(v) = 0$, then $F_k(a) = C$, so $y \leq (1+\epsilon)C$;
> and if $f^{\vee,\Phi}(v) = 1$, then $F_k(a) = C - p + p^k$, so $y \geq (C + p^k - p)/(1+\epsilon)$.

When p is larger than a constant, $p < p^k/2$. Furthermore, by the assumption in the Theorem, $p^k > 2\epsilon(2+\epsilon)mn \geq 2\epsilon(2+\epsilon)C$. Rewriting, we obtain $(C + p^k - p) > (1+\epsilon)^2 C$, meaning that the ranges of y for $f^{\vee,\Phi}(v) = 0$ and $f^{\vee,\Phi}(v) = 1$ are disjoint. At this point, B_M outputs 1 if $y > (1+\epsilon)C$.

However, note that B_M cannot simply write down a on a tape, so instead, B_M will decode small portions of the stream containing a when they are needed. More precisely, B_M holds in its memory: (i) a triple

$(i,j,k) \in [p] \times [m] \times [n]$ on $\Theta(\log M)$ bits, initialized to $(1,1,1)$, and always containing the player number, the instance number and the bit position of the location where the input head is scanning v; (ii) a buffer D of $\log mn = \Theta(\log M)$ bits containing one element of a; (iii) the space $s(N)$ of the Stack Machine A_N; and (iv) extra $O(1)$ bits. In total, B_M uses space $s(N) + \Theta(\log M) \leq s(M) + \Theta(\log M)$.

Before starting the simulation of A_N, B_M scans its input until it finds the first 1. It then writes in the buffer D the value $(\phi_i^{-1}(j)-1)n+k$. >From now on, B_M simulates A_N, using its stack as the stack of A_N, the reserved $s(N)$-bits as the memory of A_N, and using D as the input tape for A_N. Whenever A_N attempts to move its head left/right of the buffer D, B_M clears the buffer, moves its own input head left/right until it finds the first 1, refills D with the value $(\phi_i^{-1}(j)-1)n+k$, and resumes the simulation. It is easy to see that during this simulation, B_M performs at most as many reversals as A_N. Finally, if A_N is randomized with error $\delta < 1/2$, so is B_M.

For the case $k < 1$, the simulation is the same, but now for correctness we need $C - (p - p^k) < C/(1+\epsilon)^2$. This follows by taking p greater than a constant, so that $p^k < p/2$, and using the new condition $p > (2\epsilon(2+\epsilon)mn/(1+\epsilon)^2) \geq 2(1-1/(1+\epsilon)^2)C$. □

Proof of Corollary 14. Let $\beta = (k-5)/(9k+9) > 0$. Assume $r = O(N^\beta)$ and $s = O(N^\beta)$. We will derive a contradiction.

Let $\alpha = (5k-7)/(4k+7)$, and let $m = n^\alpha$. Let $p = (3\epsilon(2+\epsilon)mn)^{1/k} = O(1) \cdot n^{(1+\alpha)(1+1/k)}$ (if $\epsilon = 0$, set $p = 2$). Let $f = \text{pSETINT}_{p,n}$. Let $r'(M) = r(M) + 2$ and let $s'(M) = s(M) + O(\log(M))$. By Theorem 22, there exists an (r', s')-SM B_M for $f^{\vee,\Phi}$ with error $\delta < 1/4$. By Theorem 13, there exists a protocol for $\text{pSETINT}_{p,n}$ with cost $O(p^2(r(mnp))^2 s(mnp))$ and error at most $\delta + d(1-\delta)$ where $d = O(p(r(mnp))^2/\sqrt{m})$. Note that both $r(mnp)$ and $s(mnp)$ are $O(n^{(1+\alpha)(1+1/k)\beta})$. One can check that, with the settings above, $d = o(1)$, so for large n, the error of B_M is at most, say 1/3. Then, by [CKS03], the cost of the protocol should be $\Omega(n/(p \log p))$. However, one can also check that, with the settings above, $p^2(r(mnp))^2 s(mnp) = o(n/(p \log p))$. This is a contradiction. □

Proof of Corollary 15. Similarly to the proof of Corollary 14, we use Theorem 13 to obtain an efficient protocol for $\text{IP}^{\mathbb{F}}_{2,n}$, which requires $\Omega(n)$ communication. □

A.4 Full proofs - omitted from the proof of Theorem 13

Proof of Lemma 19. Let $x \in X^p$ be an input to P'. The players share M', j and $\overline{y}$. The goal of P' is to simulate M' on input $v = v(j, x, \overline{y})$. We say that *an input symbol* from v *is private to player* i if it belongs to $v_{i,\phi_i(j)}$, which is where x_i, the input to player i, is embedded in v. Input symbols that are not private to any player are *public*. Private input symbols are never communicated in P'.

Let Γ be the sequence of configurations of M' on input v. Consider $\gamma \in \Gamma$ and look at the contents of the stack in γ. For every symbol ξ appearing on the stack, we say that *stack symbol* ξ *is private to player* i if during the transition when ξ was pushed on the stack, the input head was scanning an input symbol private to player i. (The same symbol may appear many times on the stack, so we should formally talk about a private stack *level*, rather than *symbol*, but we believe this degree of formalization negatively affects the intuition.) A stack symbol that is not private to any player is *public*.

We say that *player* i *sees a hollow view of the stack in configuration* γ if player i knows: (i) the stack height in γ; (ii) for every symbol on the stack, whether it is public or the player to which it is private (that is, without knowing the symbol itself); (iii) all stack symbols that are public or private to player i; and (iv) for every $i' \neq i$, the top stack symbol in any contiguous zone of symbols private to player i'. Note that the hollow views of different players of the same stack differ. We say that *the players see a hollow view of*

configuration γ if every player knows the state in γ, the input head position in γ, and sees a hollow view of the stack in γ.

We say that *configuration* γ *is input-private to player* i if the input head is scanning an input symbol that is private to player i. We say that *configuration* γ *is stack-private to player* i if the transition out of γ is a pop transition and the top stack symbol in γ is private to player i. Intuitively, player i will be responsible for simulating the transitions out of configurations input- or stack-private to it. Note that there exists a configuration that is input-private and stack-private to two different players if and only if $j \in \mathrm{BAD}(M', v)$. A configuration that is neither input-private nor stack-private is *public*.

In the protocol P', the players will simulate Γ transition by transition, always using hollow views of the configurations along the way. It is not hard to see that: (1) if a player sees a hollow view of the stack in a public configuration γ, then that player can compute a hollow view of the stack in configuration $\mathrm{next}(\gamma)$; (2) as long as $j \notin \mathrm{BAD}(M', v)$, if player i sees a hollow view of the stack in a configuration γ which is input- or stack-private to player i, then player i can compute a hollow view of the stack in $\mathrm{next}(\gamma)$; and (3) if player i sees a hollow view of the stack in a configuration γ that is input-private to player i, it can detect whether γ is stack-private to another player $i' \neq i$.

We now describe the protocol P' inductively. Clearly, all players see a hollow view of the initial configuration, because there is nothing on the stack. Inductively, assume all players see a hollow view of configuration γ. We consider three cases.

Case 1: γ *is public*. By fact (1) above, each player can privately compute a hollow view of the stack in $\mathrm{next}(\gamma)$, so all players now see a hollow view of $\mathrm{next}(\gamma)$.

Case 2: γ *is input-private to player* i. This means the input head is scanning $v_{i,\phi_i(j)}$ in γ. All players recognize this because they know the input head position, so they give control of the simulation to player i. Let γ' be the first configuration following γ in Γ that is no longer input-private to player i. By applying facts (2) and (3) inductively, player i either detects that $v \in \mathrm{BAD}(M', v)$ (in which case the simulation is aborted and protocol P' outputs "fail"), or it can compute hollow views of the stack in every configuration γ'' between, and including, γ and γ'. Player i recognizes γ' is no longer input-private because it knows the input head position in γ'. At this point, player i communicates: (a) the state in γ'; (b) the input head position in γ'; (c) the stack height in γ'; (d) the top stack symbol in γ'; and (e) the lowest stack height L in any configuration γ'' between γ and γ', and the top stack symbol in γ''.

We claim the information communicated is enough for a player $i' \neq i$ to compute a hollow view of the stack in configuration γ'. Specifically, player i' takes the hollow view of the stack from configuration γ, truncates it at height L, and fills it to the height in γ' with symbols private to player i. To compute part (iv) of the hollow view, player i' obtains the top stack symbol in γ' from (d), and the stack symbol at level L from (e). Hence, after this communication, all players see a hollow view of configuration γ'.

Case 3: γ *is stack-private to player* i (and not input-private). That is, the input head is scanning a public symbol, the transition out of γ is a pop transition, and the top stack symbol in γ is private to player i. All players recognize this situation, because they know whether the top stack symbol is private to player i, so they give control of the simulation to player i. Let γ' be the first configuration following γ in Γ that is either input-private (to any player) or stack-private to some player $i' \neq i$. By facts (1) and (2), player i can compute hollow views of the stack for every configuration γ'' between, and including, γ and γ'.

In order to best explain what player i communicates in case 3, we need to step back for a second and look at what we are trying to do: we are trying to design an efficient protocol P', so we want to have as little communication as possible. Going backward from γ, let γ_a be the last configuration which is not input-private. Going forward from γ, let γ_b be the first configuration which is input-private. Let T denote the number of contiguous parts of input-private configurations before γ_a in Γ. During each reversal there are at

most p such parts, one corresponding to every player, and there are at most r reversals, so $T \leq pr$. Looking at the stack in γ_a, observe that if we group together contiguous zones of stack symbols private to the same player, *the number of such contiguous zones is at most* T, the number of contiguous parts of input-private configurations before γ_a. Possibly all of these stack symbols are popped between γ_a and γ_b, but in order to keep the cost of the protocol low, we will devise a way in which *communication occurs in* P' *at most* T *times*, once for every zone of contiguous stack symbols private to some player.

The discussion above suggest that in case 3 it would be a bad idea for player i to fall back to public simulation the moment it encounters a public configuration between γ and γ': possibly that public configuration is closely followed by another stack-private configuration of player i, such that both the latter and γ correspond to the same contiguous zone of stack symbols in γ_a, which would invalidate the communication bound above.

Another idea is to let player i perform the simulation all the way to γ'. This also turns out to be a bad idea, because, in general, stack symbols might have been pushed on the stack between γ and γ', and those should not be private to any player, as the input head is in a public zone. Then, player i would have to actually communicate those symbols in order to give the other players a proper hollow view of γ'.

Instead, we let γ^* be a configuration between, and including, $\mathrm{next}(\gamma)$ and γ', where *the stack height is the lowest possible* (if there are several, any will do). Then, player i simulates only until γ^*, and it communicates items (a)—(d) as before, skipping (e). Again, we claim the information communicated is enough for a player $i' \neq i$ to compute a hollow view of the stack in γ^*. This is simpler to see than in case 2 before, because, by choice of γ^*, it is enough for player i' to truncate its hollow view of the stack in configuration γ to obtain a hollow view of the stack in γ^*.

This completes the description of the protocol. If $j \notin \mathrm{BAD}(M', v)$, one of the players gets to the final configuration and announces the output of the function. If $j \in \mathrm{BAD}(M', v)$, one of the players detects this and outputs "fail".

To obtain the stated communication bound, we first observe that every communication takes $O(s)$ bits. Because there are at most pr contiguous zones of input-private configurations, the communication generated by case 2 is at most $O(prs)$ bits. Finally, by the discussion in case 3, at most pr communications occur during each contiguous zone of non-input-private configurations. There are $pr - 1$ such zones, to a total of $O(p^2r^2s)$ bits. □

Proof of Lemma 20. We say that *instance* $j \in \mathrm{BAD}(M', v')$ *is corrupted by the tuple* (l_1, l_2, i_1, i_2) if the latter is the lexicographically smallest such tuple satisfying the conditions in Definition 18 for instance j. First, we consider the question of how many instances can be corrupted by one 4-tuple. Second, we argue that not all 4-tuples can simultaneously corrupt instances.

Let $J \subseteq [m]$ be k distinct instances that are corrupted by the same 4-tuple (l_1, l_2, i_1, i_2). Recalling Definition 11, rename the instances in J in such a way that $\phi_{i_1}^{-1}(j_1) < \phi_{i_1}^{-1}(j_2) < \cdots < \phi_{i_1}^{-1}(j_k)$. Assume that both l_1 and l_2 are odd. Then, the head of M' scans v_{i_1} during reversal l_1 from left to right, So M' first visits $v_{i_1,\phi_{i_1}^{-1}(j_1)}$, then $v_{i_1,\phi_{i_1}^{-1}(j_2)}$, and so on, up to $v_{i_1,\phi_{i_1}^{-1}(j_k)}$. For $a \in [k]$, let $\gamma_{a,1}, \gamma_{a,2}$ be the configurations mentioned in Definition 18. By this definition, the stack level cannot drop between $\gamma_{a,1}$ and $\gamma_{a,2}$, so then we must have $\gamma_{1,1} \prec \gamma_{2,1} \prec \cdots \prec \gamma_{k,1} \prec \gamma_{k,2} \prec \cdots \prec \gamma_{2,2} \prec \gamma_{1,2}$. Since we assumed l_2 is also odd, the head is also moving left to right in reversal l_2, so we obtain that $\phi_{i_2}^{-1}(j_k) < \cdots < \phi_{i_2}^{-1}(j_2) < \phi_{i_2}^{-1}(j_1)$. Then, $\phi_{i_2}^{-1} \circ \phi_{i_1}(\phi_{i_1}^{-1}(j_1) < \phi_{i_1}^{-1}(j_2) < \cdots < \phi_{i_1}^{-1}(j_k))$ is a size-k monotone decreasing subsequence of $\phi_{i_2}^{-1} \circ \phi_{i_1}$. By Fact 16, $k \leq O(\sqrt{m})$. It is easy to see that the relative parities of l_1 and l_2 change this argument only in that we might obtain a monotone increasing subsequence instead of a monotone decreasing one. Thus, every 4-tuple can corrupt at most $O(\sqrt{m})$ instances.

Next, consider the $r \times r$ matrix A, where $A[l_1, l_2] = 1$ if there exist $i_1, i_2 \in [p]$ such that some instance is corrupted by the tuple (l_1, l_2, i_1, i_2), and $A[l_1, l_2] = 0$ otherwise. Note that $A = 0$ under the main diagonal, because for a tuple to corrupt an instance we must have $l_1 \leq l_2$.

Moreover, for $k \geq 1$, consider the diagonal $l_2 - l_1 = k$, and two entries on this diagonal that are k' cells apart, for $1 \leq k' < k$. We claim that we cannot have $A[l_1, l_1 + k] = 1$ and $A[l_1 + k', l_1 + k + k'] = 1$. Assume this were true. Then, some instance j is corrupted because a symbol is pushed on the stack in scan l_1 and popped in scan $l_1 + k$, and another instance j' is corrupted because a symbol is pushed on the stack in scan $l_1 + k'$ and popped in scan $l_1 + k + k'$. But, with these settings, $l_1 < l_1 + k' < l_1 + k < l_1 + k + k'$, which contradicts the way a stack works.

Hence, for $k \geq 1$, the diagonal $l_2 - l_1 = k$ can contain at most r/k 1s. In total, we see that A contains at most $O(r \log r)$ 1s.

Finally, for fixed $l_1 \leq l_2$, consider the $p \times p$ matrix $B = B_{(l_1, l_2)}$, where $B[i_1, i_2] = 1$ if some instance is corrupted by the tuple (l_1, l_2, i_1, i_2), and $B[i_1, i_2] = 0$ otherwise. First, consider the case when l_1 and l_2 are both odd, so the input head scans v left to right during both scans l_1 and l_2.

We claim that for $k' \geq 1$, we cannot have $B[i_1, i_2] = 1$ and $B[i_1 + k', i_2 + k'] = 1$. Assume this were true. Then, some instance j is corrupted because a symbol is pushed on the stack during scan l_1 of the input part of player i_1 and popped during scan l_2 of the input part of player i_2, and another instance j' is corrupted because a symbol is pushed on the stack during scan l_1 of the input part of player $i_1 + k'$ and popped during scan l_2 of the input part of player $i_2 + k'$. But in scan l_1, the input part of player i_1 is visited before the input part of player $i_1 + k'$, and in scan l_2, the input part of player i_2 is visited before the input part of player $i_2 + k'$. This contradicts the way a stack works.

Hence, on every diagonal $i_2 - i_1 = k$, the matrix B can contain a single 1. In total, B contains at most $O(p)$ 1s. The cases where either l_1 or l_2 or both are even are treated similarly.

Therefore, at most $O(pr \log r)$ 4-tuples can corrupt instances, and each 4-tuple can corrupt at most $O(\sqrt{m})$ instances. Thus, $|\mathrm{BAD}(M', v')| \leq O(pr \log r \sqrt{m})$. □

A.5 The power of one-way SMs

Proof of Lemma 12. Consider the canonical complete language for E under log-lin reductions, $U := \{M\#w\#^{k|w|} \mid M(w) \text{ accepts within time } 2^{(k+1)|w|}\}$. Let the tally set $T_U := \{0^{|x|} \mid x \in U\}$.

Fact 23.

1. *If there exists $L \in (\mathsf{E} - \mathsf{PSPACE})$, then U requires super-polynomial space.*
2. *$T_U \in \mathsf{P}$*
3. *If there exists $L \in (\mathsf{E} - \mathsf{PSPACE})$, then T_U requires super-polylogarithmic space.*

Proof. Suppose $U \in \mathsf{PSPACE}$ and arbitrary $L \in \mathsf{DTIME}(2^{k'n})$, witnessed by a TM M, $\mathcal{L}(M) = L$. An obvious simulation (by considering the projection of U when the machine is M and $k = k'$) shows that L also requires polynomial space. T_U is trivially in P since it is the unary representation of the strings in $U \in \mathsf{E}$. The last bullet follows by the first, and the fact that T_U is the unary representation of U. □

Since T_U is a polytime tally set, it can be decided by an one-way SM (Proposition 2.1 [All89]). Suppose that there is $L \in (\mathsf{E} - \mathsf{PSPACE})$ and that T_U can be decided by a TM, with a logspace working tape (on which we don't count reversals), and a constant number of additional tapes on which we count reversals (and no space bounds). Suppose that the total number of reversals is at most $\log^k N$, for an arbitrary

constant $k \in \mathbb{Z}^+$. By [HS08] (Lemma 4.8), $L \in \mathsf{DSPACE}(r^2(N)\log N) = \mathsf{DSPACE}(\log^{2k+1} N)$, which contradicts Fact 23 (3). □

B $\mathsf{PPdL}[1] = \mathsf{P}$

Proof of Theorem 9. Let M be a vSM with one-way access to its external tape and let w be a string. We assume, without loss of generality, that M always halts with an empty stack, scanning the last symbol on the external tape. By definition of the P- (two-sided unbounded) error regime, M accepts w iff the number of random strings that take M to an accepting configuration is greater than the number of random strings that take it to a rejecting configuration. In what follows, we show that there is a poly-time algorithm that, given M and w, outputs the exact number of random strings that take M to an accepting configuration. Clearly, the existence of such an algorithm places $L(M)$ in P.

Without loss of generality (it costs an increase in the number of states by only a constant factor), we assume that every transition of M is of exactly one of the following types: a *push* transition, a *pop* transition, a (coin) *toss* transition (in which the external tape head moves right), and a *move* transition (in which any of the input tape head and internal tape heads can move).

We define a *surface configuration* C of M on w to consist of (0) the state of M, (1) the location of the input head; (2) the full configuration of the work tapes (head positions and tape contents); (3) the top stack symbol; and (4) the location of the external tape head (denoted by $h(C)$) and the symbol it is scanning. We define a *full configuration* $\overline{C}$ of M on w to consist of everything included in a surface configuration, plus (5) the entire stack content.

Let $\overline{C_1}, \overline{C_2}$ be two full configurations of M on w with $h(\overline{C_1}) \leq h(\overline{C_2})$, and let ρ be a string of size $h(\overline{C_2}) - h(\overline{C_1})$ (when $h(C_1) = h(C_2)$, ρ is the empty string). We say that $(\overline{C_1}, \overline{C_2})$ *is up-realizable along* ρ if, when we plug in ρ on the external tape of M between locations $h(\overline{C_1}) + 1$ and $h(\overline{C_2})$, and we start M in full configuration $\overline{C_1}$, M eventually reaches full configuration $\overline{C_2}$, and the stack level never drops below the level in $\overline{C_1}$. We say that $(\overline{C_1}, \overline{C_2})$ *is realizable along* ρ if, additionally, the stack level is the same in $\overline{C_1}$ and in $\overline{C_2}$. We say that a surface configuration C_1 is *reachable*, if there is a full configuration $\overline{C_1}$ with surface C_1 and a string ρ such that $(\overline{C_0}, \overline{C_1})$ is up-realizable along ρ, where $\overline{C_0}$ is the initial configuration of M.

Informally, the heart of the proof in [Coo71], showing that a nondeterministic SM can be simulated in polynomial time, is that one can compute all realizable pairs of surface configurations, where a pair is realizable (in their sense) if it is realizable (in our sense) along *some* string (in that terminology, a sequence of nondeterministic guesses). In here, to decide whether M accepts w, we compute exactly the number of strings along which each pair is realizable.

More formally, let $\overline{C_1}$ be a full configuration such that its surface C_1 is reachable and let C_2 be a surface configuration. We define $\alpha(\overline{C_1}, C_2)$ to be the number of strings ρ such that there exists a full configuration $\overline{C_2}$ with surface C_2 such that $(\overline{C_1}, \overline{C_2})$ is realizable along ρ. Crucially, observe that a computation path that starts at $\overline{C_1}$ and along which the stack level never drops below that in $\overline{C_1}$ only depends on the surface C_1 of $\overline{C_1}$. Thus, for full configurations $\overline{C_1}, \overline{C_1}'$ with the same surface C_1, and for all surface configurations C_2, we have $\alpha(\overline{C_1}, C_2) = \alpha(\overline{C_1}', C_2)$. We denote this quantity by $\alpha(C_1, C_2)$. [9] Below, we give an algorithm computing all non-zero entries $\alpha(C_1, C_2)$. Having done that, to decide whether M accepts w, we simply sum the entries $\alpha(C_0, C_a)$, where C_0 is the surface of the initial configuration, and C_a ranges over the surfaces of accept configurations, and compare the resulting value with half times the maximum number of random strings.

[9] Note, we only care about the entry $\alpha(C_1, C_2)$ when C_1 is reachable, we leave it undefined otherwise.

Let C be a surface configuration of M. Note that C completely determines the next transition of M. If C is followed by a *move* or a *push* transition, let $\mathrm{next}(C)$ denote the next surface configuration of M (observe that, in this case, C completely determines $\mathrm{next}(C)$). If C is followed by a *toss* transition, slightly overloading notation, let $\mathrm{next}(C)$ denote the set consisting of the two possible surface configurations following C, corresponding to the two possible outcomes of the coin toss (i.e., of the next symbol read on the external tape). If C is followed by a *pop* transition, let $\mathrm{next}(C, x)$ denote the surface configuration following C in which the top stack symbol is now x. The following are easy consequences of our definitions.

Lemma 24. *For all surface configurations C_1, C_2 such that C_1 is reachable:*

- *If $C_1 = C_2$, then $\alpha(C_1, C_2) = 1$.*
- *If C_1 is followed by a* move *transition, then $\alpha(C_1, C_2) = \alpha(\mathrm{next}(C_1), C_2)$.*
- *If C_1 is followed by a* toss *transition, and $\mathrm{next}(C_1) = \{C_3, C_4\}$, then $\alpha(C_1, C_2) = \alpha(C_3, C_2) + \alpha(C_4, C_2)$.*
- *If C_1 is followed by a* push *transition, denoting by x the top stack symbol in C_1,*

$$\alpha(C_1, C_2) = \sum_{C_3 : C_3 \text{ is followed by a pop transition}} \alpha(\mathrm{next}(C_1), C_3) \cdot \alpha(\mathrm{next}(C_3, x), C_2).$$

To compute the entries $\alpha(C_1, C_2)$, we first define a partial order relation on surface configurations, as follows. Informally, $C_1 \prec C_1'$ when some entry of the form $\alpha(C_1, \cdot)$ directly depends on another entry $\alpha(C_1', \cdot)$. Formally, we write $C_1 \prec C_1'$ if (i) C_1 is followed by a *move* transition and $C_1' = \mathrm{next}(C_1)$; (ii) C_1 is followed by a *toss* transition and $C_1' \in \mathrm{next}(C_1)$; and (iii) C_1 is followed by a *push* transition and (iii-a) $C_1' = \mathrm{next}(C_1)$, *or* (iii-b) there exists a surface configuration C_1'' such that $\alpha(\mathrm{next}(C_1), C_1'') > 0$, C_1'' is followed by a *pop* transition, and $C_1' = \mathrm{next}(C_1'', x)$, where x is the top stack symbol in C_1.

Lemma 25. *The relation $\prec$ contains no cycles.*

Proof of Lemma 25. Assume a cycle exists. Let $C_1, \ldots, C_m$ be a sequence of surface configurations such that: $C_1 = C_m$, and for every $1 \leq i < m$, $C_i \prec C_{i+1}$. Observe that whenever $C_i \prec C_{i+1}$, we have $h(C_i) \leq h(C_{i+1})$. Hence, $h(C_1) = h(C_m) = h(C_i)$ for all $i \in [m]$, and so none of the C_i for $i < m$ is followed by a *toss* transition. Furthermore, if C_i is followed by a *pop* transition, we cannot have $C_i \prec C_{i+1}$ because such configurations create no dependencies. Hence, every one of the configurations in the sequence $C_1, \ldots, C_m$ is followed by either a *move* or a *push* transition.

But then, there is a path from a full configuration $\overline{C_1}$ to a full configuration $\overline{C_1}'$ with the same surface $C_1 = C_m$, with the stack level never dropping below that in $\overline{C_1}$. Since this path only depends on the surface C_1 of $\overline{C_1}$, and since $\overline{C_1}'$ has the same surface, this path is infinite. We assumed C_1 is reachable from the initial configuration of M, hence M does not halt along this path, a contradiction. □

To fill in the values in $\alpha(\cdot, \cdot)$, we proceed as follows.

1. for all surface configs C, $\texttt{seen}[C] \leftarrow \texttt{false}$
2. find any surface config C_1 such that $\texttt{seen}[C_1']$ for all C_1' with $C_1 \prec C_1'$
3. for all surface configs C_2
4. compute $\alpha(C_1, C_2)$ using the formulas in Lemma 24
5. endfor
6. $\texttt{seen}[C_1] \leftarrow \texttt{true}$
7. repeat step 2 as long there are unseen surface configurations

The test in step 2 can easily be implemented in the obvious way suggested by the definition of $\prec$. In the "most complex" case (iii-b), one can simply try out all possibilities for C_1'', since there are only polynomially many of them. By Lemma 25, this algorithm does not get stuck at step 2. As an invariant, when line 6 is executed, all non-zero entries in the row $\alpha(C_1, \cdot)$ have been computed. Since there are polynomially many surface configs, the algorithm runs in polynomial time. [10] □

C Variants of the model and omitted structural results

C.1 $\mathsf{RPdL}[\mathrm{poly}] = \mathsf{RPdL}^{\mathbb{E}}[\mathrm{poly}]$

Lemma 26. *$\mathsf{RPdL}[r(N)] \subseteq \mathsf{RPdL}^{\mathbb{E}}[r(N)] \subseteq \mathsf{RPdL}[O(r(N))]$, where $r(N) = N^{O(1)}$.*

Proof of Lemma 26. The first inclusion is trivial. The second is also easy. Let $L \in \mathsf{RPdL}[r(N)]$, and let M be a SM such that for every $x \notin L$, $M(x)$ rejects with probability 1 and for every $x \in L$, $M(x)$ accepts with probability $\frac{1}{2}$. Moreover, the expected number of reversals on every input x is $r(N)$, $|x| = N$. Construct M' which on input x computes as follows.

1. Simulate $M(x)$ for at most $4r(N)$ reversals; count the reversals on the worktape.
2. If more than $4r(N)$ reversals are needed then reject.
3. else do what $M(x)$ does.

Fix an input x. If $x \notin L$ then M' outputs the correct answer with probability 1.

Suppose that $x \in L$ and let R be the random variable that corresponds to the number of reversals. By Markov's inequality we have that $\Pr[R \geq 4r(N)] \leq 1/4$, where $r(N) = \mathbb{E}[R]$ by definition. First, using Markov's inequality we show that $\Pr[M(x) \text{ rejects} | R \leq 4r(N)] \leq \frac{2}{3}$. The details of the calculations are given below. Then, by a different application of Markov's inequality, together with the definition of $\mathsf{RPdL}^{\mathbb{E}}[r(N)]$ we obtain that $\Pr[M'(x) \text{ rejects}] \leq \frac{2}{3} + \frac{1}{4} = \frac{11}{12}$. This probability is amplified to $\leq \frac{1}{2}$ in the standard way, by considering a constant times larger random tape and use its bits for independent repetitions. This also increases the number of reversals by a constant factor.

Here are the details of the calculations. First, we bound $\Pr[M(x) \text{ rejects} | R \leq 4r(N)]$ from above.

$$\begin{aligned}
\Pr[M(x) \text{ rejects}] &= \Pr[R \leq 4r(N)] \Pr[M(x) \text{ rejects} | R \leq 4r(N)] \\
&\quad + \Pr[R > 4r(N)] \Pr[M(x) \text{ rejects} | R > 4r(N)] \\
&\geq (1 - \Pr[R > 4r(N)]) \Pr[M(x) \text{ rejects} | R \leq 4r(N)] \\
&\geq \frac{3}{4} \Pr[M(x) \text{ rejects} | R \leq 4r(N)] \\
&\Rightarrow \Pr[M(x) \text{ rejects} | R \leq 4r(N)] \leq \frac{2}{3}
\end{aligned}$$

Then,

[10] As a side note, observe that the algorithm might compute entries of the form $\alpha(C_1, C_2)$ for unreachable C_1, because we cannot test whether C_1 is reachable. These entries are useless (they are formally undefined), but also harmless, because well-defined entries never depend on undefined ones.

$$\begin{aligned}\Pr[M'(x) \text{ rejects}] &= \Pr[R \leq 4r(N)] \Pr[M'(x) \text{ rejects} | R \leq 4r(N)] \\ &\quad + \Pr[R > 4r(N)] \Pr[M'(x) \text{ rejects} | R > 4r(N)] \\ &= \Pr[R \leq 4r(N)] \Pr[M(x) \text{ rejects} | R \leq 4r(N)] + \Pr[R > 4r(N)] \\ &\leq \Pr[M(x) \text{ rejects} | R \leq 4r(N)] + \Pr[R > 4r(N)] \leq \frac{2}{3} + \frac{1}{4} = \frac{11}{12}\end{aligned}$$

□

C.2 Non-deterministic variants

Proof of Theorem 3. $\mathsf{NPdL}[1] = \mathsf{P}$ follows directly by the nontrivial Theorem 1 p.7 [Coo71]. By the NP-completeness of 3-SAT under many-to-one logspace reductions, it suffices to decide 3-SAT in $\mathsf{NPdL}[2]$, since $\mathsf{NPdL}[2]$ is closed under logspace reductions.

We describe a non-deterministic SM for 3-SAT. Let ϕ be the 3-CNF formula given in the input, with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. Let τ be a truth assignment identified by the string $\langle \tau(x_1), \ldots, \tau(x_n) \rangle$ and let τ^R denote the reversed string. Simultaneously, we: (i) check the truth assignment is a satisfying one; and (ii) check that the external tape is of the form $\underbrace{\langle \tau, \tau^R, \ldots, \tau, \tau^R, \tau, \tau^R \rangle}_{m \text{ times}}$. Associate with the first clause C_1 the first copy of the truth assignment τ, with C_2 the second copy τ^R, and so on up to C_m.

To check (i) we do not use the stack, just the logspace worktape to verify that τ satisfies C_1, τ^R satisfies C_2 and so on. This verification can be done in one scan over the non-deterministic tape.

To check (ii) we use the stack and the worktape as follows. Suppose that the external memory contains the strings $w_1, \ldots w_{n-1}, w_n$. In the first scan the machine uses the stack to verify that $w_1 = w_2^R$, ..., $w_{n-3} = w_{n-2}^R$, and $w_{n-1} = w_n^R$. When the head reverses it uses the stack to verify that $w_{n-1} = w_{n-2}^R$ and so on. Intuitively, we first verify equality between mutually exclusive pairs and then we "link" them by verifying equality among the pairs. □

For Theorem 3 it is essential that the machine has a stack. In case of a regular log-space bounded TM the situation is very different. Recall that a log-space TM with a polynomially-long read-only non-deterministic tape characterizes NP. However, if we bound the number of reversals to be constant then we cannot go outside NL. We denote by $\mathsf{L}^+[r(N)]$ the class of sets decidable by logspace machines with external non-deterministic memory and $r(N)$ scans.

Lemma 27. $\mathsf{L}^+[r(N)] \subseteq \mathsf{NSPACE}(r(N) \log N)$

Proof. We simulate a log-space $r(N)$-scan NTM M with a $O(r(N) \log N)$ space one-scan NTM M'.

Fix an arbitrary branch of the non-deterministic computation; i.e. fix a choice of non-deterministic coins. We partition this branch of the computation according to the scan number. Associate each tape-scan $1, \ldots, r(N)$ with a part of this branch of the non-deterministic computation. here is how M' works. We begin by making $r(N)$-many non-deterministic guesses for the configurations for each part of the computation when the head is over the first symbol of the non-deterministic tape. We keep an additional copy (which we won't modify during the simulation) for each of these $r(N)$ configurations corresponding to head-position over the first symbol. Storing each configuration takes space $O(\log N)$. M' continues simulating M simultaneously (and step-by-step) for the $r(N)$-many parts of the computation. For the odd-numbered parts the

simulation moves forward in time, whereas for the even ones the simulation goes backwards in time (since the head has opposite direction). If the guesses are valid at some point the $r(N)$ parts of the computation are going to meet and form a valid branch of the non-deterministic computation. □

Corollary 28. *Let* $\mathsf{L}^+[\mathsf{constant}] := \bigcup_{k=1}^{\infty} L^+[k]$. *Then,* $\mathsf{NL} = \mathsf{L}^+[\mathsf{constant}]$.

Since with polynomially many reversals we go all the way to NP it is natural to ask what is the first time we capture P. Is it possible to capture P before we capture the whole NP? For instance, is it possible that $\mathsf{P} \subseteq \mathsf{L}^+[N^k]$, for some constant $k > 0$? This is not true, unless $\mathsf{P} \subseteq \mathsf{NSPACE}(N^k \log N) \subseteq \mathsf{DSPACE}(N^{k'})$, for some constant k'.

D Omitted proofs from Section 3

Proof of Theorem 4. $\mathsf{P{+}RNC}^i \subseteq \mathsf{RPdL}[2^{O(\log^i N)}]$ is the easy inclusion. Let $L \in \mathsf{P{+}RNC}^i$ and M be the polytime transducer that on input x computes C_x. Construct a vSM M' as follows: use Cook's algorithm [Coo71] to simulate M so as to obtain the k-th output bit from the description of C_x. Note that each bit is computed without accessing the random tape. M' evaluates C_x on the provided randomness in the usual way (e.g. [Ruz81]) by a depth first search. Note that if the description of the circuit were given through oracle access, the evaluation procedure would have taken time $2^{O(\log^i N)}$. Hence, M' makes at most $2^{O(\log^i N)}$ reversals on the random tape and the accepting probability is the same as that of C_x.

To show $\mathsf{RPdL}[2^{O(\log^i N)}] \subseteq \mathsf{P{+}RNC}^{i+1}$ we rely on the time-compression Lemma 7 and on [Ruz80, Ruz81]. In the version of the time-compression lemma for uniform machines, the advice can be computed in polynomial time. Fix $L \in \mathsf{RPdL}[2^{O(\log^i N)}]$, and let M' be a vSM such that $\mathcal{L}(M') = L$. We construct a polytime M that on input x outputs a circuit C_x with the same accepting probability as M'.

There exists M'' extending M' as follows: (1) M'' has an extra input (read-only) tape which will contain the particular advice described in the proof of Lemma 7. (2) Furthermore, M'' is a modification of M' as described in the proof of Lemma 7. Therefore, on input x given that the extra-tape contains this advice, M'' computes identically to M'. Syntactically, M'' is a SM with three read-only input tapes. When the 3rd tape contains the appropriate advice, M'' is a SM that works in space $O(\log N)$ and in time $2^{O(\log^i N)} N^{O(1)} = 2^{O(\log^i N)}$, $i \geq 1$. We assert the existence of an equivalent ATM M_{ATM} that works in space $O(\log N)$ and in time $O(\log^{i+1} N)$. This is being done by observing that the corresponding equivalences carry through when the only difference is that both the SM and the ATM instead of one they have three input tapes. It is straightforward to verify that all equivalences between SMs and ATMs in the constructions of Theorem 5 part 3 [Ruz81] p.379 (i.e. Theorem 2 [Ruz80] pp. 227-231), and Corollary 3 (c,d,e) [Ruz81] pp. 379-380 are the same when the number of input tapes is a constant bigger than one. Hence, syntactically given the 3-input tape SM M'' we have an ATM M_{ATM} with 3-input tapes that computes identically. The constant description of M_{ATM} it can be hardwired in a (polytime) TM M. Although M_{ATM} and M'' accept the same inputs, we are only interested in the computations where their 3rd tape contains the advice of Lemma 7; in which case M'' computes the same as M'. Intuitively, one can blur the distinction between space-time bounded ATMs and size-depth families of combinatorial circuits, and moreover we observe that given the description of the ATM we can construct efficiently the circuit for the corresponding input length. That is, the description of the polytime M should be evident through the observation that the construction in the proof of Theorem 3 [Ruz81] p.375 is computable in time $2^{O(S(N))} O(T(N)) N^{O(1)} = N^{O(1)}$, where $S(N) = O(\log N)$ and $T(N) = O(\log^{i+1} N)$ is the space and the time of M_{ATM}. For completeness we briefly review this construction below.

1. On input x use (the modified) Cook's algorithm to compute the advice of Lemma 7, which is a function of $N = |x|$.

2. M has hardwired the description of the ATM $M_{\rm ATM}$ and it computes the description of a circuit C_x. In this circuit, both the input x and the advice are hardwired using the constant 0/1 gates of the circuit.

3. The circuit gates are labelled with (α, t), where α is the configuration of the ATM, and t is the time where the output gates has label $(\alpha_{\rm initial}, 0)$, where $\alpha_{\rm initial}$ is the starting configuration. Configurations of type $\forall, \exists$ correspond to gates $\wedge, \vee$, we connect gates (α, t), $(\beta, t+1)$ if α yields β. The only exceptions to this rule is when the time and the space becomes bigger than $T(N), S(N)$ in which case we hardwire the gates to 0, and when we have configurations accessing the input in which case instead of a gate we have an input gate.

Step (1) takes polynomial time. The construction of the circuit in Step 3, also takes polynomial time ($2^{O(S(N))}O(T(N))N^{O(1)}$). □

Proof of Theorem 6. This is a straightforward argument, standard in derandomization. We show the stronger containment $\mathsf{RPdL}[2^{O(\log^i N)}] \subseteq \mathsf{BPPdL}[2^{O(\log^i N)}] \subseteq \mathsf{DTIME}(2^{O(\log^k N)})$. It directly follows that Theorem 4 holds also if we consider BPdL and P+BPNC, instead of RPdL and P+RNC.

Let $L \in \mathsf{BPPdL}[2^{O(\log^i N)}]$. Then, there exists polytime transducer M that on input x outputs a circuit C_x of depth $O(\log^{i+1} N)$, such that if $x \in L$ then $\Pr_\rho[C_x(\rho) = 1] \geq 2/3$, whereas if $x \notin L$ then $\Pr_\rho[C_x(\rho) = 1] \leq 1/3$, and the input to the circuit is of length N^m, for a constant $m > 0$. We construct a deterministic TM $\hat{M}$ that runs in time $O(2^{\log^k N})$. Here is the description of $\hat{M}$: (1) Enumerate all strings of length $O(m^k \log^k N)$. (2) For each such string use the PRG to compute a string ρ of length N^m. (3) Fix the random tape of M to be ρ and simulate M in polytime. (4) Output the majority of the outcome of the simulation.

$\hat{M}$ takes time $2^{O(\log^k N)}$ to enumerate all strings, for each such string it takes polynomial time to compute ρ and then polynomial time to simulate M on (x, ρ).

We claim that for infinitely many x_i's:

> (*) for input x_i, at least a fraction $\frac{2}{3} - \frac{1}{7} = \frac{11}{21} > \frac{1}{2}$ of the pseudorandom strings, M gives the correct answer, for sufficiently large input length N.

Note that the error probability in the definition of P+BPPdL can be amplified to any constant arbitrarily close to $1/2$, and thus (*) suffices to conclude the theorem.

Suppose (*) is not true. Then, we are going to use M to construct a (non-uniform) family of distinguishers D_N, for infinitely many N's (where N takes values among the output length of the PRG). Let $\{x_i\}$ be an infinite family of inputs where (*) is false. Then, by definition C_{x_i} is an appropriate distinguisher with distinguishing probability strictly greater than $\frac{2}{3} - \frac{11}{21} = \frac{1}{7}$. □

Proof of Lemma 7. It suffices to show that the computation between two successive head-moves can be "compressed" to be polynomially long by the use of the non-uniform advice on the tape. This non-uniform advice extends the non-uniform advice already given to the machine. Fix two arbitrary successive head-moves. Partition the computation γ between these head-moves in two phases. Suppose that immediately after reading the input symbol the stack level is at l. In phase 1, γ reaches its lowest stack height l_{min}. Let γ_1 be the computation subsequence of γ from the beginning until we reach the lowest stack level and just before we start going upwards (pushing symbols to the stack). Define γ_2 to be the complement of γ_1 wrt γ. Hence, in γ_2 the computation reaches its final stack height l_{last}. A simple counting argument shows

that for a halting SM, the stack height is polynomial, and therefore the stack height in γ_1 gets decreased polynomially low (from l to l_{min}) and in γ_2 gets increased polynomially high (from l_{min} to l_{last}). We construct M' simulating M using the following non-uniform advice. The advice is a function from every surface configuration to the set of surface configuration together with two special symbols $\{\uparrow, \downarrow\}$. For every surface configuration σ define exactly one of the three pairs:

1. If starting from σ we can return to the same stack level without ever going below the initial stack-level (and without a head-move on the input) then consider the configuration after a maximally-long computation such that when M returns to the same stack-level the surface configuration is σ'. Then, the associated pair is (σ, σ').

2. If starting from σ we move at least one level upwards without ever returning to the initial stack-level (and without moving the head) then the pair is $(\sigma, \uparrow)$.

3. Else, the pair is $(\sigma, \downarrow)$.

Obviously, this is a well-defined function and we say that a surface configuration σ is of type (1), (2) or (3) respectively.

Between two successive head-moves M' simulates M by reading the advice tape and updating its surface configuration appropriately. In case of (1) it updates the worktape, the state and the top stack symbol. In case of (2) and (3) it simulates M for one step.

We refer to a *simulation step* as the computation sequence of M' in which M' reads the non-uniform tape, compares it to the current surface configuration and updates the surface configuration appropriately. In what follows the reader is reminded that γ_1, γ_2 is the computation of M which is simulated by the machine M', and that M' is given the non-uniform advice. We say that a function from the integers is 2-monotonically increasing (decreasing) if it is strictly increasing (decreasing) for two successive integers; i.e. for the function $h : \mathbb{Z}^+ \to \mathbb{Z}^+$, $h(n) \leq h(n+1)$ and $h(n) < h(n+2)$.

Claim 29. *In the simulation of γ_1 the stack height in M' is 2-monotonically decreasing. Hence, this simulation takes at most $2(l - l_{min})$ simulation steps of M'.*

Proof. Consider two successive stack levels $l_1 > l_2 := l_1 - 1$ in γ_1 and consider the first time M' gets to l_1. The current surface configuration σ_1 cannot be of type (2). Suppose that σ_1 is of type (2). Since we are in γ_1 we know that the stack level gets as low as l_{min}. If σ_1 is of type (2) then we know that during γ_1 the stack level will get back to l_1. Hence, σ_1 should instead be of type (1).

Hence, σ_1 is either of type (1) or of type (3). If it is of type (3) there is nothing left to show. Suppose σ_1 is of type (1). The fact that the next surface configuration in the simulation cannot be of the same type (1) follows by the maximality in the definition of type (1). □

Similarly, we show that in the simulation of γ_2 the stack height in M' is 2-monotonically increasing. □

Proof of Lemma 8. In this proof all circuit classes are uniform. Suppose that the non-uniform advice can be computed in $\mathsf{NC} = \mathsf{SAC}$; i.e. let $\hat{M}$ be an SM that computes the advice string (i.e. the i-th bit of the output) in the proof of Lemma 7 in NC. We show that every polytime computable tally language $L \subseteq \{0\}^*$ can be computed in NC. By [All89] (Corollary 6.3 and Corollary 6.7) this implies that $\mathsf{PSPACE} = \mathsf{EXP}$.

Since, $\mathcal{L}(\hat{M}) \in \mathsf{NC}$, by Theorem 1 we know that M works in time $O(2^{\log^{O(1)} N})$.

Fix arbitrary $L \subseteq \{0\}^*$, $L \in \mathsf{P}$. Let M be a SM, such that $\mathcal{L}(M) = L$.

We construct a SM M'', which works in time $O(2^{\log^{O(1)} N})$, such that $\mathcal{L}(M'') = L$; i.e. $L \in \mathsf{NC}$.

Here is the description of M''. In a single tape-scan over 0^N compute N and store it on the work-tape. Note that M'' can simulate M on the given input, without moving its input head (also see Proposition 2.1 [All89]). Since the advice in the proof of Lemma 7 can be computed by $\hat{M}$ following this proof we can construct a uniform M' that decides L and it makes polynomial many calls to an oracle computing the advice bit given its index. By assumption each oracle call can be computed by simulating $\hat{M}$ in time $O(2^{\log^{O(1)} N})$. Therefore, M'' works in time $O(N^{O(1)}2^{\log^{O(1)} N}) = O(2^{\log^{O(1)} N})$. □

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/