Electronic Colloquium on Computational Complexity, Report No. 130 (2017)

Worst-case to Average-case reductions for subclasses of P

Oded Goldreich* Guy N. Rothblum†

August 30, 2017

Abstract

For every polynomial q, we present worst-case to average-case (almost-linear-time) reductions for a class of problems in $\mathcal{P}$ that are widely conjectured not to be solvable in time q. These classes contain, for example, the problems of counting the number of k-cliques in a graph, for any fixed $k \geq 3$. In general, we consider the class of problems that consist of counting the number of local neighborhoods in the input that satisfy some predetermined conditions, where the number of neighborhoods is polynomial, and the neighborhoods as well as the conditions can be specified by small uniform Boolean formulas. Hence, we show an almost-linear-time reduction from solving one such problem in the worst-case to solving some other problem (in the same class) on typical inputs.

Contents

*Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. oded.goldreich@weizmann.ac.il

†Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. rothblum@alum.mit.edu

ISSN 1433-8092

1 Introduction

While most research in the theory of computation refers to worst-case complexity, the importance of average-case complexity is widely recognized (cf., e.g., [10, Sec. 1–10.1] versus [10, Sec. 10.2]). Worst-case to average-case reductions, which allow for bridging the gap between the two theories, are of natural appeal (to say the least). Unfortunately, worst-case to average-case reductions are known only either for "very high" complexity classes, such as $\mathcal{E}$ (see [3] (or [10, Sec. 7.2])), or for "very low" complexity classes, such as $\mathcal{AC}_0$ (cf. [1, 2]). In contrast, presenting a worst-case to average-case reduction for $\mathcal{NP}$ is a well-known open problem, which faces significant obstacles as articulated by Bogdanov and Trevisan [7].

A recent work by Ball, Rosen, Sabin, and Vasudevan [4] initiated the study of worst-case to average-case reductions in the setting of fine-grained complexity. The latter setting focuses on the exact complexity of problems in $\mathcal{P}$ (see, e.g., survey by V. Williams [20]), attempting to classify problems into classes of similar polynomial-time complexity (and distinguishing, say, linear-time from quadratic-time and cubic-time). Needless to say, reductions used in the context of fine-grained complexity must preserve the foregoing classification, and the simplest choice – taken in [4] (and followed here) – is to use almost linear-time reductions.

The pioneering paper of Ball *et al.* [4] shows that there exist (almost linear-time) reductions from the worst-case of several natural problems in $\mathcal{P}$, which are widely believed to be "somewhat hard" (i.e., have super-linear time complexity (in the worst case)), to the average-case of some other problems that are in $\mathcal{P}$. In particular, this is shown for the Orthogonal Vector problem, for the 3-SUM problem, and for the All Pairs Shortest Path problem. Hence, the worst-case complexity of problems that are widely believed to be "somewhat hard" is reduced to the average-case complexity of problems in $\mathcal{P}$. Furthermore, the worst-case complexity of the latter problems matches (approximately) the best known algorithms for the former problems (although the actual worst-case complexity of these problems may in fact be lower).

In this paper we tighten the foregoing result by defining, for each polynomial p, a worst-case complexity class $\mathcal{C}^{(p)}$ that is a subset of $\mathrm{Dtime}(p^{1+o(1)})$, and showing for any problem Π in $\mathcal{C}^{(p)}$ an almost linear-time reduction of Π to the average-case of some problem Π' in $\mathcal{C}^{(p)}$. Loosely speaking, the class $\mathcal{C}^{(p)}$ consists of counting problems that refer to $p(n)$ local conditions regarding the n-bit long input, where each local condition refers to $n^{o(1)}$ bit locations and can be evaluated in $n^{o(1)}$-time. In particular, for any constant $t > 2$ and $p_t(n) = n^t$, the class $\mathcal{C}^{(p_t)}$ contains problems such as t-CLIQUE and t-SUM.

Theorem 1.1 (worst-case to average-case reduction, loosely stated):[1] *For every polynomial p, and every counting problem Π in $\mathcal{C}^{(p)}$, there exists a counting problem Π' in $\mathcal{C}^{(p)}$ and an almost-linear time randomized reduction of solving Π on the worst-case to solving Π' on the average* (i.e., on at least 0.751 fraction of the domain).

Hence, the average-case complexity of Π' is sandwiched between the worst-case complexities of Π and Π', where the worst-case complexities of Π' is at most $p^{1+o(1)}$. This might leave a rather wide gap between the average-case complexity of Π' and its worst-case complexity (if the worst-case of Π is significantly smaller than the worst-case of Π'). We can narrow this gap by taking advantage of the fact that our reduction preserves membership in the class $\mathcal{C}^{(p)}$. Specifically, assuming that the worst-case complexity of Π is n^c and invoking Theorem 1.1 for $O(1/\epsilon)$ times, we obtain a problem

[1] See Theorem 3.1 for a precise statement.

whose average-case complexity is at least $n^{c'}$ and whose worst-case complexity is at most $n^{c'+\epsilon}$, where $c' \in [c, O(1)]$. Hence, the average-case complexity of this problem (which remains in $\mathcal{C}^{(p)}$) is approximately equal to its worst-case complexity.

Corollary 1.2 (average-case approximating worst-case): *Suppose that $\mathcal{C}^{(p)}$ contains a problem of worst-case complexity at least n^c. Then, for every constant $\epsilon > 0$, there exists $c' \in [c, \log_n p(n)]$ such that $\mathcal{C}^{(p)}$ contains a problem of average-case complexity $n^{c'}$ and worst-case complexity at most $n^{c'+\epsilon}$.*

Proof: Starting with $\Pi^{(0)} \in \mathcal{C}^{(p)}$ as in the hypothesis, we consider a sequence of $O(1/\epsilon)$ problems in $\mathcal{C}^{(p)}$ such that the problem $\Pi^{(i)}$ is obtained by applying (the worst-case to average-case reduction of) Theorem 1.1 to problem $\Pi^{(i-1)}$. Recall that the average-case complexity of $\Pi^{(i)}$ is sandwiched between the worst-case complexities of $\Pi^{(i-1)}$ and $\Pi^{(i)}$. Hence, the worst-case complexities of these problems constitute a non-decreasing sequence that is upper bounded by $p(n)^{1+o(1)}$, and the claim follows. ■

Reductions to rare-case. The notion of average-case complexity that underlies the foregoing discussion (see Theorem 1.1) refers to solving the problem on at least 0.76 fraction of the instances. This notion may also be called *typical-case complexity*. A much more relaxed notion, called *rare-case complexity*, refers to solving the problem on a noticeable[2] fraction of the instances (say, on a $n^{-o(1)}$ fraction of the n-bit long instances). Using non-uniform reductions, we establish the following result.

Theorem 1.3 (worst-case to rare-case reduction, loosely stated):[3] *For every polynomial p, and every counting problem Π in $\mathcal{C}^{(p)}$, there exists a counting problem Π' in $\mathcal{C}^{(p)}$ and an almost-linear time* non-uniform *reduction of solving Π on the worst-case to solving Π' on a noticeable fraction of the instances* (e.g., on at least $\exp(-\log^{0.999} n)$ fraction of the n-bit long instances).

We also provide a uniform reduction from the worst-case complexity of the problem $\Pi \in \mathcal{C}^{(p)}$ to the rare-case complexity of some problem in $\mathrm{Dtime}(p')$, where $p'(n) = p(n) \cdot n^{1+o(1)}$. For details, see Theorem 4.4.

Comparison to the known average-case hierarchy theorem. We mention an (unconditional) average-case hierarchy theorem of Goldmann, Grape, and Hastad [9], which is obtained by diagonalization. While that result offers no (almost linear time) worst-case to average-case reduction, it does imply the existence of problems in $\mathcal{P}$ that are hard on the average (at any desired polynomial level), let alone that this result is unconditional. We point out several advantages of the current work (as well as of [4]) over the aforementioned hierarchy theorem.

1. Worst-case to average-case reductions yield hardness results also with respect to probabilistic algorithms, whereas the known hierarchy theorem does not hold for that case. Recall that an honest-to-God hierarchy theorem is not known even for worst-case probabilistic time (cf. [5]).

[2] Here a "noticeable fraction" is the ratio of a linear function over an almost linear function. We stress that this is not the standard definition of this notion (at least not in cryptography).

[3] See Theorem 4.2 for a precise statement. Note that Theorem 4.2 refers to sample-aided reductions, which imply non-uniform reductions.

2. Worst-case to average-case reductions are robust under the presence of auxiliary inputs (or, alternatively, w.r.t non-uniform complexity), whereas the aforementioned hierarchy theorem is not.

3. Our worst-case to average-case reductions (as well as those in [4]) do not depend on huge and unspecified constants.

More importantly, while the standard interpretation of worst-case to average-case reductions is negative (i.e., establishes hardness of the average-case problem based on the worst-case problem), such reductions have also a positive interpretation (which is not offered by results of the type of [9]): They can be used to actually solve a worst-case problem when given a procedure for the average-case problem. Similarly, they may allow to privately compute the value of a function of a secret instance by making queries that are distributed independently of that instance (see, e.g., [6]).

Terminology. For a function $f : \mathbb{N} \to \mathbb{N}$, we use $\exp(f(n))$ to denote $e^{O(f(n))}$. The notion of "almost linear" functions is commonly given a variety of interpretations ranging from saying that a function $f : \mathbb{N} \to \mathbb{N}$ is almost linear if $f(n) = n^{1+o(1)}$ to requiring that $f(n) = \widetilde{O}(n)$. We shall restrict the range of interpretations to the case of $f(n) \leq \exp(f'(n)) \cdot n$, where at the very minimum $f'(n) = \widetilde{O}(\log \log n)$ and at the very maximum $f'(n) = (\log n)/(\log \log n)^{\omega(1)}$. In these cases, we shall consider $\exp(f'(n))$ to be *small*, and $\exp(-f'(n))$ to be *noticeable*.

2 The class of counting problems: The actual definition

We consider a class of counting problems that are solvable in polynomial time. Such a counting problem specifies a polynomial number of local conditions, where each local condition consists of a short sequence of locations (in the input) and a corresponding predicate, and the problem consists of counting the number of local conditions that are satisfied by the input. That is, on input $x \in \{0,1\}^n$, we count the number of sets $S_i^{(n)}$'s such that $\phi_i^{(n)}(x_{S_i}) = 1$, where $S_1^{(n)}, ..., S_{\mathrm{poly}(n)}^{(n)}$ and $\phi_1^{(n)}, ..., \phi_{\mathrm{poly}(n)}^{(n)}$ are efficiently specified by n. (As usual, for a set $S = \{i_1, ..., i_s\} \subseteq [n]$ such that $i_1 < i_2 < \cdots < i_s$ and $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$, we denote $x_S = x_{i_1} x_{i_2} \cdots x_{i_s}$ the projection of x at the coordinates S.)

Specifically, we assume that for some $\ell, m : \mathbb{N} \to \mathbb{N}$ such that $\ell(n) = O(\log n)$, there exists a sub-exponential time algorithm that, on input n, outputs Boolean circuits $\phi_n : \{0,1\}^{\ell(n)+m(n)} \to \{0,1\}$ and $\pi_n : \{0,1\}^{\ell(n)} \to [n]^{m(n)}$ (of bounded fan-in) such that $\phi_i^{(n)}(z) = \phi_n(i,z)$ and $S_i^{(n)} = \pi_n(i)$. (Note that the running time of the foregoing algorithm is upper-bounded by $\exp(|n|^c) = \exp((\log n)^c)$, for some constant $c \in (0,1)$.) In addition, we shall assume that these circuits have bounded depth, which may mean that they have depth at most $(\log n)^c$ for some constant $c \in (0,1)$.

The fact that the time (upper) bound is exponential in the depth (upper) bound is no coincidence. It reflects the fact that one measure of overhead in our results is polynomial in the time bound, whereas another is exponential in the depth bound. Since our reductions may increase the depth of the formulae by a $\mathrm{poly}(\log \log n)$ factor, we consider classes of bounding functions that are closed under multiplication by such a factor. Hence, we call a class of functions *admissible* if it is closed in the foregoing sense, or actually closed under polylogarithmic factors (i.e., $\widetilde{O}(f)$ is in the class if f is in it). In addition, we require that the class contains only non-decreasing functions of sublogarithmic growth rate and is closed under addition of $O(\log \log n)$ terms.

Definition 2.1 (admissible classes of functions): *A class of functions $\mathcal{D}$ is* admissible *if for every $f \in \mathcal{D}$ it holds that*

1. Closure: *The functions f' and f'' such that $f'(n) = \widetilde{O}(f(n))$ and $f''(n) = f(n) + \log\log n$ are in $\mathcal{D}$.*
2. Non-decreasing: $f(n+1) \geq f(n)$.
3. Sublogarithmic growth rate: $f(n) = o(\log n)$.

Examples of admissible classes of functions include $\mathcal{D}_1 = \{f : f(n) \leq \widetilde{O}(\log\log n)\}$, $\mathcal{D}_2 = \{f : f(n) \leq \mathrm{poly}(\log\log n)\}$, $\mathcal{D}_3 = \{f : f(n) \leq O(\log n)^c\}_{c\in(0,1)}$, and $\mathcal{D}_4 = \{f : f(n) \leq \frac{\log n}{(\log\log n)^{\omega(1)}}\}$, where the unspecified constants allow for ignoring finitely many n's. Recall that admissible classes will be used as bounds on the depth of the circuits that define the local patterns. For a fixed admissible class $\mathcal{D}$, we say that the function $f : \mathbb{N} \to \mathbb{N}$ is almost linear if $f(n) \leq \exp(f'(n)) \cdot n$ for some $f' \in \mathcal{D}$.

Turning back to the Boolean circuits $\phi_n : \{0,1\}^{\ell(n)+m(n)} \to \{0,1\}$ and $\pi_n : \{0,1\}^{\ell(n)} \to [n]^{m(n)}$, observe that their size is upper-bounded by the time that it takes to construct them, where the latter bound is exponential in the depth of these circuits. Thus, we may consider Boolean *formulae* of such size instead. (We also replace $\pi_n : \{0,1\}^{\ell(n)} \to [n]^{m(n)}$ by $\pi_{n,1}, ..., \pi_{n,m(n)} : \{0,1\}^{\ell(n)} \to [n]$.) With these preliminaries in place, we are finally ready to state the definition of the class of counting problems that we consider. Actually, we present a family of such classes, each corresponding to a different set of admissible functions.

Definition 2.2 (counting local patterns): *Let $\mathcal{D}$ be a set of admissible functions. For every $f \in \mathcal{D}$ and $\ell, m : \mathbb{N} \to \mathbb{N}$ such that $\ell : \mathbb{N} \to \mathbb{N}$ is a logarithmic function* (i.e., $\ell(n) = \lceil c \cdot \log n \rceil$ for some constant $c > 0$)*, let A be an algorithm that, on input n, runs for $\exp(f(n))$-time and outputs Boolean formulae $\phi_n : \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}$ and $\pi_{n,1}, ..., \pi_{n,m(n)} : \{0,1\}^{\ell(n)} \to [n]$. The* counting problem associated with *A, denoted $\#_A$, consists of counting, on input $x \in \{0,1\}^*$, the number of $w \in \{0,1\}^{\ell(|x|)}$ such that*

$$\Phi_x(w) \stackrel{\text{def}}{=} \phi_n(w, x_{\pi_{n,1}(w)}, ..., x_{\pi_{n,m(n)}(w)}) \tag{1}$$

equals 1. The class of counting problems associated with ℓ and $\mathcal{D}$ is denoted $\mathcal{C}_{\ell,\mathcal{D}}$.

Note that any problem in the class $\mathcal{C}_{\ell,\mathcal{D}}$ can be solved in polynomial-time; specifically, n-bit long instances can be solved in time $2^{\ell(n)} \cdot \exp(f(n)) = 2^{(1+o(1))\cdot\ell(n)}$ on a direct access machine. We mention that the doubly-efficient interactive proof systems presented by us in [13] apply to these counting problems, provided that the verifier is allowed to run in time $\exp(f(n)) \cdot n = n^{1+o(1)}$, for some $f \in \mathcal{D}$.

We mention that a simplified form, in which ϕ_n ignores its $\ell(n)$-bit long prefix (and so we may have $\phi_n : \{0,1\}^{m(n)} \to \{0,1\}$), suffices for capturing many natural problems. Specifically, for fixed $t \in \mathbb{N}$, when representing n-vertex graphs by their adjacency matrix, denoted $x = (x_{r,c})_{r,c\in[n]}$, the problem of counting the number of t-cliques in the graph is captured by $\Phi_x(i_1, ..., i_t) = \wedge_{j<k} x_{i_j,i_k}$; that is, we use $\phi_{n^2} : \{0,1\}^{t^2} \to \{0,1\}$ and $\pi_{n^2,(j,k)} : \{0,1\}^{t\log n} \to [n^2]$ (for $j,k \in [t]$) such that $\phi_{n^2}(z_{1,1}, ..., z_{t,t}) = \wedge_{j<k} z_{j,k}$ and $\pi_{n^2,(j,k)}(i_1, ..., i_t) = (i_j, i_k)$. Likewise, with some abuse of notation, the problem of counting t-tuples of integers in the input sequence $x = (x_1, ..., x_n) \in [-m, m]^n$

that sum-up to zero, is captured by $\Phi_x(i_1,...,i_t)=1$ if and only if $\sum_{j\in[t]} x_{i_j}=0$; that is, we use $\phi_n:[m]^t\to\{0,1\}$ such that $\phi_n(z_1,...,z_t)=\texttt{TruthValue}(\sum_{j\in[t]} z_j=0)$ and $\pi_{n,j}:\{0,1\}^{t\log n}\to[n]$ such that $\pi_{n,j}(i_1,...,i_t)=i_j$ for $j\in[t]$.

3 The worst-case to average-case reduction

We are now ready to (re)state our main result.

Theorem 3.1 (worst-case to average-case reduction, Theorem 1.1 formalized): *Let $\mathcal{D}$ be a set of admissible functions and ℓ be a logarithmic function. For every counting problem Π in $\mathcal{C}_{\ell,\mathcal{D}}$, there exists a counting problem Π' in $\mathcal{C}_{\ell,\mathcal{D}}$ and an almost-linear time randomized reduction of solving Π on the worst case to solving Π' on at least $0.75+\epsilon$ fraction of the domain, where ϵ is any positive constant.*

Recall that, for every $f\in\mathcal{D}$, the function $n\mapsto\exp(f(n))\cdot n=n^{1+o(1)}$ is considered almost linear. The reduction consists of two main steps.

1. An almost-linear time randomized reduction of solving Π *on the worst case* to evaluating certain Arithmetic expressions over a non-binary finite field, where the evaluation subroutine is correct on at least a $0.5+o(1)$ fraction of the instances.

2. An almost-linear time reduction of evaluating the foregoing Arithmetic expressions on at least a $0.5+o(1)$ fraction of the instances to solving Π' on at least $0.75+o(1)$ fraction of the instances.

We start by presenting the aforementioned arithmetic problem. Recall that an Arithmetic circuit (resp., formula) is a directed acyclic graph (resp., directed tree) with vertices (of bounded in-degree) that are labeled by multiplication-gates and linear-gates (i.e., gates that compute linear combination of their inputs). Actually, since we consider generic Arithmetic circuits, which are well defined for any field, the scalars allowed in the linear gates are only 0, 1 and -1.

Definition 3.2 (evaluating arithmetic expressions of a local type): *For $f\in\mathcal{D}$ and $\ell,m:\mathbb{N}\to\mathbb{N}$ as in Definition 2.2, let A' be an algorithm that, on input n, runs for $\exp(f(n))$-time and outputs an Arithmetic formula $\widehat{\phi}_n:\mathcal{F}^{\ell(n)}\times\mathcal{F}^{m(n)}\to\mathcal{F}$ and Boolean formulae $\pi_{n,1},...,\pi_{n,m(n)}:\{0,1\}^{\ell(n)}\to[n]$, where $\mathcal{F}$ is a generic finite field. Fixing a finite field $\mathcal{F}$, the* evaluation problem associated with A' and $\mathcal{F}$*, denoted $\texttt{EV}_{A',\mathcal{F}}$, consists of computing the function $\widehat{\Phi}_{A'}:\mathcal{F}^n\to\mathcal{F}$ such that*

$$\widehat{\Phi}_{A'}(X_1,...,X_n)\stackrel{\text{def}}{=}\sum_{w\in\{0,1\}^{\ell(n)}}\widehat{\phi}_n(w,X_{i^{(1)}_{n,w}},...,X_{i^{(m(n))}_{n,w}}),\tag{2}$$

where $i^{(j)}_{n,w}=\pi_{n,j}(w)$.

Indeed, as in Eq. (2), we shall often abuse notation and view t-long binary strings as t-long sequences over $\mathcal{F}$. The formally inclined reader should consider a mapping $\xi:\{0,1\}\to\mathcal{F}$ such that $\xi(0)=0\in\mathcal{F}$ and $\xi(1)=1\in\mathcal{F}$, and write $\widehat{\phi}_n(\xi(w_1),...,\xi(w_{\ell(n)}),X_{i^{(1)}_{n,w}},...,X_{i^{(m(n))}_{n,w}})$ instead of $\widehat{\phi}_n(w,X_{i^{(1)}_{n,w}},...,X_{i^{(m(n))}_{n,w}})$.

We shall first prove a weaker version of Theorem 3.1 in which the tolerated error rate is $1/O(\log n)$ rather than $0.25-o(1)$. This proof is presented in Section 3.1, and it will serves as a basis for the proof of Theorem 3.1 itself, which is presented in Section 3.2.

3.1 The vanilla version

Our first step is reducing the (Boolean) counting problem to the Arithmetic evaluation problem. This reduction is based on the folklore emulation of Boolean circuits by Arithmetic circuits, which yields a worst-case to worst-case reduction. The validity of the reduction relies on the choice of the field for the Arithmetic evaluation problem; specifically, we choose a finite field of characteristic that is larger than the number of terms in the counting problem. (The worst-case to average-case reduction, for the Arithmetic evaluation problem, will be performed in the next step, and it is enabled by keeping track of the degree of the polynomial that is computed by the Arithmetic circuit that is derived by the Boolean-to-Arithmetic reduction.)

Proposition 3.3 (reducing worst-case Boolean counting to worst-case arithmetic evaluation): *Solving the counting problem associated with an algorithm A* (and functions ℓ, m and f) *is reducible in almost-linear time to the evaluation problem that is associated with a related algorithm A', and any finite field of prime cardinality greater than $2^{\ell(n)}$. Furthermore, the reduction makes a single query, the degree of the polynomial $\widehat{\Phi}_{A'}$ is at most $\exp(f(n))$, and the functions ℓ, m and f equal those in the counting problem.*

Proof: The basic idea is to emulate the Boolean formula ϕ_n by the Arithmetic formula $\widehat{\phi}_n$, where we shall first transform the former formula into an almost-balanced one. The almost-balanced structure will yield a bound on the degree of the derived Arithmetic formula.

Let $\ell = \ell(n) = O(\log n)$. We may assume, without loss of generality, that the depth the Boolean formula ϕ_n is logarithmic in its size, which is upper-bounded by $s = \exp(f(n))$. Observe that the transformation of arbitrary formula to this (almost-balanced) form can be performed in polynomial (in s) time. Within the same complexity bound, we can construct an Arithmetic formula $\widehat{\phi}_n : \mathcal{F}^{\ell(n)+m(n)} \to \mathcal{F}$ that agrees with ϕ_n on $\{0,1\}^{\ell(n)+m(n)}$; that is, $\widehat{\phi}_n(w,z) = \phi_n(w,z)$ for every $(w,z) \in \{0,1\}^{\ell(n)+m(n)}$. This construction is obtained by replacing each `and`-gate by a multiplication gate, and replacing each negation-gate by a gate that computes the linear mapping $v \mapsto 1-v$. The crucial point is that $\widehat{\phi}_n$ preserves the depth of ϕ_n, and so the degree of the function computed by $\widehat{\phi}_n$ is upper-bounded by $D = \exp(O(\log s)) = \mathrm{poly}(s) \ll |\mathcal{F}|$, where $\mathcal{F}$ is chosen to be a finite field of prime cardinality that is larger than 2^ℓ.

Hence, we reduce the counting problem $\#_A$ (associated with an algorithm A) to the arithmetic evaluation problem $\mathtt{EV}_{A',\mathcal{F}}$ associated with an algorithm A' that computes the foregoing $\widehat{\phi}_n$ and $\pi_{n,j}$'s, by first invoking algorithm A, and then proceeding as outlined above. Defining $\widehat{\Phi}_{A'}$ as in Eq. (2), for every $x = x_1 \cdots x_n \in \{0,1\}^n \subset \mathcal{F}^n$, it holds that

$$\begin{aligned}
\widehat{\Phi}_{A'}(x_1,...,x_n) &= \sum_{w\in\{0,1\}^\ell} \widehat{\phi}_n(w, x_{i^{(1)}_{n,w}}, ..., x_{i^{(m(n))}_{n,w}}) \\
&= \sum_{w\in\{0,1\}^\ell} \phi_n(w, x_{\pi_{n,1}(w)}, ..., x_{\pi_{n,m(n)}(w)}) \\
&= \sum_{w\in\{0,1\}^\ell} \Phi_x(w)
\end{aligned}$$

where the equalities hold both over $\mathcal{F}$ and over the integers, because each term is in $\{0,1\}$ and $\mathcal{F}$ is a finite field of prime cardinality that is larger than 2^ℓ. Hence, $\#_A(x) = \mathtt{EV}_{A',\mathcal{F}}(x)$ for every $x \in \{0,1\}^n \subset \mathcal{F}^n$. Recalling that $D = \mathrm{poly}(s) = \exp(f(n))$, the furthermore clause follows. ■

Proposition 3.4 (folklore worst-case to average-case reduction for evaluating polynomials of bounded degree): *Let $P : \mathcal{F}^n \to \mathcal{F}$ be a polynomial of total degree $d < |\mathcal{F}|/3$. Then, evaluating P on any input can be randomly reduced to evaluating P correctly on at least a $8/9$ fraction of the domain, by invoking the latter evaluation procedure for $3d$ times.*

We stress that the reduction is randomized, and, on each input, it yields the correct answer with probability at least $2/3$.

Proof: On input $x \in \mathcal{F}^n$, we select $r \in \mathcal{F}^n$ uniformly at random, and invoke the evaluation procedure on the points $x + ir$, where $i = 1, ..., 3d$. (This description presumes that $\mathcal{F}$ is a prime field; but otherwise, we may use $3d$ distinct non-zero elements of $\mathcal{F}$ instead of the i's.) Note that the queried points are uniformly distributed in $\mathcal{F}^n$, and so, with probability at least $1/3$, the evaluation procedure answers correctly on at least two-thirds of the queried points. Using the Berlekamp–Welch algorithm, we reconstruct the unique degree d polynomial that agrees with these correct answers, and return its free-term (i.e., its value at 0, which corresponds to $P(x)$). This value is correct with probability at least $2/3$. ■

A pause. Combining Propositions 3.3 and 3.4, we obtain a reduction of solving the (Boolean) counting problem, on the worst-case, to solving the Arithmetic problem on the average (i.e., for a $8/9$ fraction of the instances). In order to prove Theorem 3.1, we have to reduce the latter problem to a (Boolean) counting problem, and this reduction has to be analyzed in the average-case regime. This raises a difficulty, since the Arithmetic problem (denoted $\mathtt{EV}_{A',\mathrm{GF}(p)}$) refers to any fixed prime $p > 2^{\ell(n)}$. Note that Proposition 3.3 does not specify how this prime is chosen. Indeed, such a prime can be selected at random by the reduction (of Proposition 3.3), but in such a case different invocations will yield different primes, and so we shall not have a single Arithmetic problem but rather a distribution over such problems. This difficulty could have been avoided if we had a deterministic $\exp(f(n))$-time algorithm for generating primes that are larger than $2^{\ell(n)}$, but such an algorithm is not known (since $\exp(f(n)) = n^{o(1)}$ whereas $2^{\ell(n)} = n^{\Omega(1)}$). We prepare to accommodate this difficulty by considering an extension of Definition 2.2 in which the algorithm that generates the Boolean formulae is given an auxiliary input (in addition to the length parameter n). Indeed, providing the algorithm associated with the Boolean problem with an auxiliary input that specifies the field used in the Arithmetic problem, allows us to reduce the Arithmetic problem to a Boolean problem. We stress that this reduction will be analyzed with respect to potential solvers that err on a noticeable fraction of the domain.

Proposition 3.5 (reducing average-case Arithmetic evaluation to average-case Boolean counting): *For a finite field $\mathcal{F}$ of prime size $p \leq \mathrm{poly}(n)$, let A' be as in Definition 3.2, and $\widehat{\Phi}_{A'}$ be the corresponding polynomial. Then, the problem of evaluating $\widehat{\Phi}_{A'}$ on at least a $8/9$ fraction of the domain $\mathcal{F}^n$ is randomly reducible in almost-linear time to solving a counting problem of the flavor of Definition 2.2 on at least $1 - (1/O(\log p))$ fraction of the domain $\{0,1\}^{n''}$, where $n'' = \widetilde{O}(n)$. The reduction makes $\log p$ queries to the counting problem, where each query has input length n''. Specifically, we consider an algorithm as in Definition 2.2, except that it is given the prime p in addition to its length parameter. Furthermore, if ℓ', f' and m' (resp., ℓ'', f'' and m'') are the functions used in the Arithmetic* (resp., Boolean) *problem, then $\ell'' = \ell'$, $f''(\widetilde{O}(n)) \leq \mathrm{poly}(\log\log p) \cdot f'(n)$, and $m''(\widetilde{O}(n)) = O(\log n) \cdot m'(n)$.*

Note that $m''(\widetilde{O}(n)) \leq O(\log n) \cdot \exp(f'(n)) = \exp(f''(n))$ for a suitable choice of f'' such that $f''(\widetilde{O}(n)) \leq \mathrm{poly}(\log\log p) \cdot f'(n)$. As for the prime provided to the algorithm that generates instances of the counting problem, this deviation from Definition 2.2 will be eliminated when applying Proposition 3.5. Specifically, in these applications the prime p will be chosen according to the length parameter and fed to the foregoing algorithm (which will be used as a subroutine). As hinted above, this will raise an issue that will be discussed and resolved in the proof of Corollary 3.6.

Proof: The basic idea is to let the Boolean formula emulate the computation of the Arithmetic formula $\widehat{\phi}_n$ that appears in Eq. (2). This will be done by using small Boolean formulae that emulate the operations of the field $\mathcal{F}$. Details follow.

We shall represent each element of $\mathcal{F}$ by an $O(\log n)$-bit long string. Actually, each element of $\mathcal{F} = \mathrm{GF}(p)$ will have $\mathrm{poly}(n)$-many possible representation by bit strings of length $\log p + O(\log n)$. When reducing the Arithmetic evaluation problem to a Boolean counting problem, we shall select at random one of them; that is, for each input symbol (in $\mathcal{F}$), we shall select at random one of these representations (as a bit string). This redundant representation is used in order to guarantee that all field elements have approximately the same number of representations (as $O(\log n)$-long bit strings).[4]

The Boolean formula will first map each such representation to the canonical representation, and then emulate the computation of the Arithmetic circuit using the canonical representations all along. Specifically, the canonical representation of $e \in \mathrm{GF}(p)$ will be the (zero-padded) binary expansion of e, and the other representations will correspond to the binary expansion of $e + i \cdot p$ for $i \in [\mathrm{poly}(n)]$. Hence, the output will be $t = \lceil \log p \rceil$ bits long, whereas in the input of the Boolean circuit each field element will be represented by a block of $t' = t + O(\log n)$ bits.

Next, we consider Boolean formula ϕ'_n that emulates the computation of the corresponding $\widehat{\phi}_n$ (over $\mathcal{F} = \mathrm{GF}(p)$). Specifically, each arithmetic gate will be replaced by an $\mathcal{NC}$ circuit that emulates the corresponding field operation.[5] Note that these gate-emulation circuits have depth $\mathrm{poly}(\log\log|\mathrm{GF}(p)|) = \mathrm{poly}(\log\log p)$, and that their construction depends on the prime p. In addition, we need to slightly modify the sequence of functions that determines the indices of the field elements fed to $\widehat{\phi}_n$. Suppose that the sequence fed to $\widehat{\phi}_n(w,\cdot)$ is determined by $\pi'_{n,1}, ..., \pi'_{n,m(n)} : \{0,1\}^{\ell'} \to [n]$. Then, each $\pi'_{n,j}$ determines the index of a field element in the input to $\widehat{\phi}_n$, and so it should be replaced by functions $\pi_{n,(t'-1)j+1}, ..., \pi_{n,t'j} : \{0,1\}^{\ell'} \to [t'n]$ that determine the corresponding bits in the input to ϕ'_n (i.e., $\pi_{n,(t'-1)j+k}(w) = (t'-1) \cdot \pi'_{n,j}(w) + k$ for $k \in [t']$).

Note that the formula ϕ'_n outputs $t = \log p$ bits, whereas we seek a Boolean formula with a single output bit. Such a Boolean formula is obtained by using an auxiliary input $i \in [t]$, which determines the bit to be output; that is, $\phi''_n(w,(\cdot,i))$ equals the i^{th} bit of $\phi'_n(w,\cdot)$. The corresponding function $\Phi''_{y,i}$ (per Eq. (1)) is such that $\Phi''_{y,i}(w) = \phi''_n(w,(y,i)_{\pi_n(w)})$, where π_n denote the sequence of $\pi_{n,j}$'s (augmented by functions that indicate the bit positions of i in the input (y,i)). Hence, we reduce the evaluation of of $\widehat{\Phi}_{A'}$ on input $x \in \mathcal{F}^n$ to counting the number of w's that satisfy $\Phi''_{y,i}(w) = 1$, for each $i \in [t]$, where $y \in \{0,1\}^{t'n}$ is a random representation of x. Specifically, the value of $\widehat{\Phi}_{A'}$ on $x = (x_1, ..., x_n) \in \mathcal{F}^n$ is obtained as follows.

[4] The source of trouble is that $\mathcal{F} = \mathrm{GF}(p)$ is a prime field and so representing its elements by $\lceil \log_2 p \rceil$-bit long strings will leave some of these strings unused. In such a case, a uniform distribution on $\mathrm{GF}(p)$ will be mapped to a uniform distribution on p of these strings, but not on all of them. The random representation used below will map the uniform distribution over $\mathrm{GF}(p)$ to a distribution that is almost uniform over all strings of length $\log_2 p + O(\log n)$.

[5] Recall that integer arithmetics is in $\mathcal{NC}$; see, e.g., [16, Lect. 30].

1. For each $k \in [n]$, randomly map $x_k \in \mathrm{GF}(p)$ to a t'-bit string, denoted y_k, that represents it. Recall that $y_k \in \{0,1\}^{t'} \equiv [2^{t'}]$ is a random integer that is congruent to x_k modulo p.

2. For each $i \in [t]$, compute the number of w's that satisfy $\Phi''_{y,i}(w) = 1$ by invoking the algorithm that supposedly solves the counting problem (on input $(y,i) = (y_1, ..., y_n, i)$), where $\Phi''_{y,i}$ is as above (i.e., $\Phi''_{y,i}(w) = \phi''_n(w, (y,i)_{\pi_n(w)})$, where π_n denote the sequence of $\pi_{n,j}$'s).

   Recall that the formulae $\pi_{n,j}$ determine integers in $[t'n + \log t]$ such that the value $\pi_{n,j}(w)$ determines the $(|w| + j)^{\text{th}}$ bit that will be fed to $\phi''_n(w, \cdot)$. (Note that the bits in locations $t'n + 1, ..., t'n + \log t$, which represent $i \in [t]$, will always be fed to ϕ''_n.)

3. Denoting by c_i the count obtained by the i^{th} call, output the value $\sum_{i=1}^{t} c_i \cdot 2^{i-1} \bmod p$.

The key observation is that

$$\begin{aligned} \textstyle\sum_{w \in \{0,1\}^{\ell'(n)}} \widehat{\phi}_n(w, x_{i^{(1)}_{n,w}}, ..., x_{i^{(m(n))}_{n,w}}) &\equiv \textstyle\sum_{w \in \{0,1\}^{\ell'(n)}} \sum_{i \in [t]} \phi''_n(w, (y,i)_{\pi_n(w)}) \cdot 2^{i-1} \pmod p \\ &\equiv \textstyle\sum_{i \in [t]} 2^{i-1} \cdot \sum_{w \in \{0,1\}^{\ell'(n)}} \phi''_n(w, (y,i)_{\pi_n(w)}) \pmod p. \end{aligned}$$

Hence, $\mathtt{EV}_{A',\mathrm{GF}(p)}(x) = \sum_{i \in [t]} 2^{i-1} \cdot \#_{A''(p)}(y,i)$, where A'' is the algorithm that (on input p) generates ϕ''_n (and π_n). Note that, when given a uniformly distributed $x \in \mathcal{F}^n$, the i^{th} query made by our reduction is almost uniformly distributed in $\{(y,i) : y \in \{0,1\}^{n \cdot t'}\}$, where the small deviation arises from the fact that some elements in $\mathcal{F} = \mathrm{GF}(p)$ have $\lfloor 2^{t'}/|\mathcal{F}| \rfloor$ representations, whereas others have $\lceil 2^{t'}/|\mathcal{F}| \rceil$ representations. Hence, if the invoked algorithm errs on at most an η fraction of its inputs (i.e., inputs in $\{0,1\}^{nt' + \log t}$), then our algorithm will err (with probability exceeding 0.1) on at most a $t \cdot (10 \cdot \eta + n \cdot (|\mathcal{F}|/2^{t'}))$ fraction of its inputs (i.e., inputs in $\mathcal{F}^n$), where the factor of t accounts for the different i's (and the factor of 10 accounts for input symbols that have more than 10% bad representations for a fixed i). Seeking to err on at most a 1/9 fraction of the inputs in $\mathcal{F}^n$, we may use any counting algorithm that errs on at most an $\eta = 0.1/t = 1/O(\log n)$ fraction of its inputs (which are strings in $\{0,1\}^{t'n}$).

The claim follows, except that our counting problem refers to $2^{\ell'(n)} = 2^{\ell''(n)}$ terms and to input-length of $n'' = t'n + \log t = \widetilde{O}(n)$, whereas the claim asserts a reduction to a counting problem that refers to $2^{\ell''(n'')}$ terms. This can be fixed by introducing $2^{\ell''(n'')} - 2^{\ell''(n)}$ dummy terms. Specifically, for $\lambda = \ell''(n'') - \ell''(n) \approx c \cdot \log(n''/n) \approx c \cdot \log t'$, we consider counting the number of $(w,v) \in \{0,1\}^{\ell''(n)+\lambda}$ that satisfy $\Phi_{y,i}$, where $\Phi_{y,i}(w, 1^\lambda) = \Phi''_{y,i}(w)$ and $\Phi_{y,i}(w,v) = 0$ for every $v \neq 1^\lambda$. ■

An intermediate conclusion. Combining Propositions 3.3–3.5, we obtain a weaker version of Theorem 3.1 in which the tolerated error rate is smaller and the class of admissible functions is slightly more restricted. Specifically:

Corollary 3.6 (weak version of Theorem 3.1): *Let $\mathcal{D}$ be a set of admissible functions that is further closed under multiplication by* $\mathrm{poly}(\log\log n)$ *factors, and ℓ be a logarithmic function. Then, for every counting problem Π in $\mathcal{C}_{\ell,\mathcal{D}}$, there exists a counting problem Π' in $\mathcal{C}_{\ell,\mathcal{D}}$ and an almost-linear time randomized reduction of solving Π on the worst case to solving Π' on at least $1 - (1/O(\log n))$ fraction of the domain.*

Note that the extra hypothesis regarding $\mathcal{D}$ is satisfied in the case that $\mathcal{D}$ contains a function f such that $f(n) \geq \exp((\log\log n)^{\Omega(1)})$.

Proof: We select a prime number $p \in (2^{\ell(n)}, 2^{\ell(n)+1})$ at random, and observe that (by Proposition 3.3) the original counting problem $\#_A$ is reduced (in the worst-case) to the Arithmetic evaluation problem $\mathtt{EV}_{A',\mathrm{GF}(p)}$ (i.e., evaluating the Arithmetic formula $\widehat{\Phi}_{A'}$ over $\mathrm{GF}(p)$). We apply the worst-case to average-case reduction of Proposition 3.4 to the evaluation of $\widehat{\Phi}_{A'}$ over $\mathrm{GF}(p)$, and next apply the reduction of Proposition 3.5, which yields a counting problem $\#_{B(p)}$ for some adequate algorithm B. Hence, we map the parameter $f' \in \mathcal{D}$ of $\widehat{\Phi}_{A'}$ to a parameter f'' of the new counting problem such that $f''(\widetilde{O}(n)) = \mathrm{poly}(\log\log n) \cdot f'(n)$.

We stress that the prime p is selected, upfront, by our reduction; that is, it is determined prior to applying Propositions 3.3–3.5. Thus, the algorithm underlying the final counting problem (i.e., B) views p as part of its input, whereas the algorithm in Definition 2.2 gets no such input. To bridge the gap, we include this input (i.e., p) in the input to the counting problem to which we reduce; that is, the input is now (p, y') rather than being $y' = (y, i)$ as in the proof of Proposition 3.5; that is, we consider the counting problem $\#_{B'}$ such that $\#_{B'}(p, y') = \#_{B(p)}(y')$. This requires modifying the algorithm so that the small circuits that emulate $\mathrm{GF}(p)$ computation take p as additional input, rather than being generated for a fixed p (by an algorithm that takes p as input). We also augment the sequence of functions $\pi_{n'}$ such that p is always fed to the formula $\phi_{n'}$. The minor increase in the length of the input of the counting problem may require additional padding of the index of the summation (but this is needed only if $\ell(n' + \ell(n')) > \ell(n')$, which is quite unlikely since $\ell(n) = O(\log n)$).

The analysis uses the fact that if the final counting problem (i.e., $\#_{B'}$) is solved correctly with probability $1 - \eta$, when the probability is take over pairs of the form (p, y'), then, with probability at least 0.9 over the choice of p, the residual counting problem (i.e., $\#_{B'}(p, \cdot) \equiv \#_{B(p)}$) is solved correctly with probability $1 - 10\eta$ (and the success probability of the entire reduction is $0.9 \cdot 2/3 \cdot 0.9$). ■

Digest. When tracing the cost of the reduction, one should focus on the depth of the various formulae, while assuming that they are in balanced form (i.e., that their depth is logarithmic in their size). The reduction in Proposition 3.3 preserves the depth, while the reduction Proposition 3.5 increases the depth by a $\mathrm{poly}(\log\log p) \leq \mathrm{poly}(\log\log n)$ factor, where p is the size of the field in use. Recall that Proposition 3.4 does not change the formula, and that the depth overhead is due to the implementation of the field's operations by Boolean formulae (in the proof of Proposition 3.5). Note that using Proposition 3.7 allows using $p = \exp(f(n))$ (rather than $p = \mathrm{poly}(n)$), which means that the depth increase is only a $\mathrm{poly}(\log f(n))$ factor.

3.2 Deriving the original conclusion

The small level of error rate that is allowed by Corollary 3.6 is rooted mainly in the fact that the arithmetic evaluation over a field of size $p > 2^{\ell(n)}$ (which refers to $\widehat{\phi}_n$ (and to the $i^{(j)}_{n,w}$'s)) is emulated by $t = \log_2 p$ invocations of a Boolean counting problem, which refers to the Boolean formula ϕ''_n (and to π_n). The key observation, made in the proof of Proposition 3.5, is that

$$\sum_{w \in \{0,1\}^{\ell(n)}} \widehat{\phi}_n(w, x_{i^{(1)}_{n,w}}, ..., x_{i^{(m(n))}_{n,w}}) \equiv \sum_{w \in \{0,1\}^{\ell(n)}} \sum_{i \in [t]} \phi''_n(w, (y, i)_{\pi_n(w)}) \cdot 2^{i-1} \pmod{p}. \quad (3)$$

In the proof of Proposition 3.5, we computed the r.h.s of Eq. (3) by counting, *separately, for each* $i \in [t]$, the number of w's that satisfy $\phi''_n(w,(y,i)_{\pi_n(w)})$, and taking a weighted (by 2^{i-1}) sum of these counts (modulo p). But an alternative solution is to replace these weights by auxilary summations; that is, the r.h.s of Eq. (3) can be computed by

$$\sum_{i\in[t]} \sum_{w\in\{0,1\}^{\ell(n)}} \sum_{u\in\{0,1\}^t} \phi'''_n(w,u,(y,i)_{\pi_n(w)}) \tag{4}$$

such that $\phi'''_n(w,u,(y,i)_{\pi_n(w)}) = \phi'''_n(w,(y,i)_{\pi_n(w)})$ if $u \in \{0,1\}^t$ ends with $t-(i-1)$ ones (i.e., $u \in \{u'1^{t-(i-1)} : u' \in \{0,1\}^{i-1}\}$) and $\phi'''_n(w,u,(y,i)_{\pi_n(w)}) = 0$ otherwise. The problem with this solution is that it doubles the length of the index of the summation (i.e., $|(i,w,u)| > \ell(n)+t \geq 2\ell(n) = 2|w|$). The source of the trouble is that the reduction of the (Boolean) counting to the Arithmetic evaluation (i.e., Proposition 3.3) uses a setting of $p > 2^{\ell(n)}$ (which implies $t = \log_2 p > \ell(n)$).

Instead, we can use a reduction to $O(\ell(n)/f(n))$ arithmetic problems that refer to fields of size $\exp(f(n)+\omega(1)) = n^{o(1)}$ (rather than of size $2^{\ell(n)} = \mathrm{poly}(n)$), where the the field size should be at least $\exp(f(n)+\omega(1))$ for the application of Proposition 3.4 (which requires the field to be larger than the degree bound). The corresponding $O(\ell(n)/f(n))$ values will be combined using Chinese Remaindering with errors (cf. [12]) so that we only incur a constant loss in the error rate (rather than a loss factor of $O(\ell(n)/f(n))$). This "CRT with errors" is reminiscent of the Berlekamp–Welch algorithm, which was employed in the proof of Proposition 3.4 (in order to obtain a constant error rate rather than an error rate that is inversely proportional to the degree of the polynomial). For sake of simplicity, we first present an alternative to Proposition 3.3 using CRT with no errors.

Proposition 3.7 (Proposition 3.3, revised): *Solving the counting problem associated with an algorithm A and functions ℓ, m and f is reducible in almost-linear time to $O(\ell(n)/f(n)) < \log n$ evaluation problems that are all associated with a related algorithm A', and finite fields of prime cardinality at least $s = \exp(f(n)+\omega(1))$. Furthermore, the reduction makes a single query to each problem, the degree of the polynomial $\widehat{\Phi}_{A'}$ is $o(s)$, and the functions ℓ, m and f equal those in the counting problem.*

Note that s is set as small as possible subject to being sufficiently larger than the degree of $\widehat{\Phi}_{A'}$. We wish s to be small because it determines the various overheads that we shall incur when emulating the Arithmetic formula by Boolean formula (in Proposition 3.9). (On the other hand, using Proposition 3.8 requires that s be sufficiently larger than the degree of $\widehat{\Phi}_{A'}$.)

Proof: Following the proof of Proposition 3.3, while using the field $\mathrm{GF}(p)$ for an arbitrary prime p, yields a (worst-case) reduction of "counting (mod p) problem $\#_A$" (i.e., counting the number of w's mod p that satisfy Φ_x) to the evaluation problem associated with A' and $\mathrm{GF}(p)$. Invoking this reduction for t primes p of size $\exp(f(n))$, and invoking the Chinese Remainder Theorem, the claim follows provided that $\exp(f(n))^t > 2^{\ell(n)}$, which solves to $t > \frac{\ell(n)}{f(n)}$. ∎

Terminology. For the rest of this section, it will be convenient to consider the average success probability of randomized algorithms, where the average is taken over all inputs with uniform probability distribution. Hence, rather than considering the fraction of the inputs on which the randomized algorithm succeeds with probability at least 2/3, we consider the average success probability of the algorithm, hereafter referred to as its success rate. (Indeed, an algorithm that succeeds with probability at least 2/3 on a ρ fraction of its inputs has success rate at least $\rho \cdot 2/3$.)

Proposition 3.8 (Proposition 3.4, revised): *Let $P : \mathcal{F}^n \to \mathcal{F}$ be a polynomial of total degree $d = o(\epsilon^2 \cdot |\mathcal{F}|)$. Then, evaluating P on any input can be randomly reduced to evaluating P with success rate at least $0.5 + \epsilon$, by invoking the latter evaluation procedure for $O(d/\epsilon^2)$ times.*

Recall that the reduction is randomized, and, on each input, it yields the correct answer with probability at least 2/3.

Proof: On input $x \in \mathcal{F}^n$, we select a random curve of degree two that passes through x, and invoke the evaluation procedure on the first $m = O(d/\epsilon^2)$ points of this curve. Note that the queried points are pairwise independent and uniformly distributed in $\mathcal{F}^n$. Hence, with probability at least 2/3, the evaluation procedure answers correctly on at least a $0.5 + 0.5\epsilon$ fraction of the queried points. Using the Berlekamp–Welch algorithm, we reconstruct the unique degree $2d$ polynomial that agrees with these correct answers, and return its value at x. This value is correct with probability at least 2/3. ■

Proposition 3.9 (Proposition 3.5, revised): *For a finite field $\mathcal{F}$ of prime size $p \leq \text{poly}(n)$, let A' be as in Definition 3.2, and $\widehat{\Phi}_{A'}$ be the corresponding polynomial. Then, the problem of evaluating $\widehat{\Phi}_{A'} : \mathcal{F}^n \to \mathcal{F}$ with success rate at least ρ, is randomly reducible in almost-linear time to solving a counting problem of the flavor of Definition 2.2 with success rate at least $\rho - (1/\text{poly}(n))$. Specifically, as in Proposition 3.5, we consider an algorithm as in Definition 2.2, except that it is given the prime p in addition to its length parameter. Let ℓ', f' and m'* (resp., ℓ'', f'' and m'') *be the functions used in the Arithmetic* (resp., Boolean) *problem. The reduction makes a single query to the counting problem, where the input length is $n'' = \exp(f'(n)) \cdot n$. Furthermore, $\ell'' = \ell'$, $f''(n'') \leq \text{poly}(\log\log p) \cdot f'(n)$, and $m''(n'') = O(\log n) \cdot m'(n)$.*

We shall use the setting $\rho = 0.5 + \epsilon > 0.5$.

Proof: Following the proof of Proposition 3.5, we derive the same formula ϕ'_n, and recall that the formula ϕ'_n outputs $t = \log p$ bits, whereas we seek a Boolean formula with a single output bit. Again, the latter formula is derived by using an auxiliary input $i \in [t]$, which determines the bit to be output; that is, $\phi''_n(w, i, z)$ equals the i^{th} bit of $\phi'_n(w, z)$. Unlike in the proof of Proposition 3.5, we shall not view i as part of the input to the counting problem (which is fed into ϕ''_n, just like z), but rather as part of the index of summartion (i.e., just as w). The corresponding Boolean formula Φ''_y (per Eq. (1)) is such that $\Phi''_y(w, i) = \phi''_n(w, i, y_{\pi_n(w)})$, where π_n denote the sequence of $\pi_{n,j}$'s. (That is, the input to this counting problem is y, and the desired output is the number of pairs (w, i) that satisfy Φ''_y.) Recall that, on input $x \in \mathcal{F}^n$, we do not wish to obtain $\sum_{w,i} \Phi''_y(w, i)$, but rather wish to obtain the related sum

$$\sum_{w \in \{0,1\}^{\ell'(n)}} \sum_{i \in [t]} 2^{i-1} \cdot \Phi''_y(w, i), \tag{5}$$

where $y \in \{0,1\}^{t'n}$ is a random representation of x. The difficulty is that Eq. (5) does not correspond to a counting problem, but this can be fixed by replacing the scalar multiplication by 2^{i-1} with a sum over 2^{i-1} terms. Specifically, consider $\Phi'''_y : \{0,1\}^{\ell'(n)} \times [t] \times \{0,1\}^t$ such that $\Phi'''_y(w, i, u) = \Phi''_y(w, i)$ if $u \in \{u'1^{t-(i-1)} : u' \in \{0,1\}^{i-1}\}$ and $\Phi'''_y(w, i, u) = 0$ otherwise. Thus, Eq. (5) equals

$$\sum_{w \in \{0,1\}^{\ell'(n)}} \sum_{i \in [t]} \sum_{u \in \{0,1\}^t} \Phi'''_y(w, i, u). \tag{6}$$

Hence, we reduce the evaluation of $\widehat{\Phi}_{A'}$ on $x \in \mathcal{F}^n$ to counting the number of (w,i,u)'s that satisfy Φ'''_y, where $y \in \{0,1\}^{t'n}$ is a random representation of x. Specifically, the value of $\widehat{\Phi}_{A'}$ on $x = (x_1, ..., x_n) \in \mathcal{F}^n$ is obtained as follows.

1. As in the proof of Proposition 3.5, for each $k \in [n]$, randomly map $x_k \in \mathrm{GF}(p)$ to a random t'-bit string, denoted y_k, that represents it. Recall that $y_k \in \{0,1\}^{t'} \equiv [2^{t'}]$ is a random integer that is congruent to x_k modulo p (and that $t' = t + O(\log n)$).

2. Compute the number of (w,i,u)'s that satisfy $\Phi'''_y(w,i,u) = 1$ by invoking the algorithm that supposedly solves the counting problem (on input $y = (y_1, ..., y_n)$), where Φ'''_y is as in Eq. (6).

3. Denoting by c the count obtained by the foregoing invocation, output the value $c \bmod p$.

As argued before, the key observation is that

$$\begin{aligned} \textstyle\sum_{w \in \{0,1\}^{\ell'(n)}} \widehat{\phi}_n(w, x_{i^{(1)}_{n,w}}, ..., x_{i^{(m(n))}_{n,w}}) &\equiv \textstyle\sum_{w \in \{0,1\}^{\ell'(n)}} \sum_{i \in [t]} \phi''_n(w, i, y_{\pi_n(w)}) \cdot 2^{i-1} \pmod p \\ &\equiv \textstyle\sum_{i \in [t]} \sum_{(w,u) \in \{0,1\}^{\ell'(n)+t}} \phi'''_n(w, i, u, y_{\pi_n(w)}) \pmod p, \end{aligned}$$

where $\phi'''_n(w,i,u,z) = \phi''_n(w,i,z)$ if $u \in \{0,1\}^{i-1} \times \{1\}^{t-i-1}$ and $\phi'''_n(w,i,u,z) = 0$ otherwise. Hence, $\mathtt{EV}_{A',\mathrm{GF}(p)}(x) = \#_{A'''(p)}(y,i)$, where A''' is the algorithm that (on input p) generates ϕ'''_n (and π_n).

Note that the query made by our algorithm is almost uniformly distributed in $\{0,1\}^{n \cdot t'}$, where the small deviation arises from the fact that some elements in $\mathcal{F}$ have $\lfloor 2^{t'}/|\mathcal{F}| \rfloor$ representations, whereas others have $\lceil 2^{t'}/|\mathcal{F}| \rceil$ representations. Hence, if the invoked algorithm has success rate ρ (on inputs in $\{0,1\}^{nt'}$), then our algorithm will will have success rate at least $\eta - n \cdot (|\mathcal{F}|/2^{t'})$ (on inputs in $\mathcal{F}^n$).

The claim follows, except that our counting problem refers to $2^{\ell'(n)+t+\log t}$ terms and to input-length of $t'n + \log t = \widetilde{O}(n)$, whereas the claim asserts a reduction to a counting problem that refers to $2^{\ell''(n')}$ terms and to input length $n'' = \exp(f'(n)) \cdot n$. This can be fixed by artificially padding the input to length n'' and noting that $\ell''(n'') \geq \ell'(n) + t + \log t$ (where $t = \log p = f'(n)$).[6] ■

Proof of Theorem 3.1. The claim of Theorem 3.1 follows by combining the revised reductions provided by Propositions 3.7–3.9. Specifically, using (a generalization of) Proposition 3.7, we reduce the original counting problem $\#_A$ to $\tau = O(\ell(n)/f(n))$ instances of the Arithmetic evaluation problem, where all instances refer to the same algorithm A' but to different primes $p > s = \exp(f(n)$. Note that we can afford to find and use the first τ such primes (since we can run for $\mathrm{poly}(s)$-time). Hence, in the final counting problem, denoted $\#_B$, the input will be prepended by an index $j \in [\tau]$ of one of these primes, and Proposition 3.9 will be applied to these primes. But before getting there, Proposition 3.8 will be applied to each value of $j \in [\tau]$ (i.e., to each of the foregoing τ primes), which means that we will employ error reduction to obtain the correct answer in all τ invocations.

(Note that applying Proposition 3.9 increases the depth of the formula by a factor of $\mathrm{poly}(\log f(n))$ (rather than $\mathrm{poly}(\log \log n)$), and increases the length of the input by a factor of $\exp(f(n))$.) More importantly, the τ values obtained by the τ applications of Proposition 3.9 (to different primes) will be combined using Chinese Remaindering with errors (cf. [12]). Indeed, the error-correcting version of the CRT will be used instead of the plain version used in Proposition 3.7, and the number of

[6] This uses the hypothesis that $\ell''(n) = c \log n$, which implies that $\ell''(\exp(f'(n)) \cdot n) = O(f'(n)) + \ell''(n)$.

primes is increased (from $(\ell(n)/f(n))+O(1)$ to $O(\ell(n)/f(n))=\tau$) so as to allow for such an error correcting feature.

Let us spell out the reduction that is obtained by combining all the above. Recall that we are given an input $x \in \{0,1\}^n$ to a counting problem $\#_A$ associated with an algorithm A. For an arbitrary small constant $\epsilon > 0$, we proceed in three steps, which correspond to the three foregoing reductions.

1. Denoting the first $\tau = O(\epsilon^{-1} \cdot \ell(n)/f(n))$ primes that are greater than $s = \exp(f(n))$ by $p_1, ..., p_\tau$, we consider the τ evaluation problems obtained by applying Proposition 3.7 to algorithm A (i.e., all these problems are associated with algorithm A'). The i^{th} such problem, denoted $\mathtt{EV}_{A',\mathrm{GF}(p_i)}$, consists of evaluating the polynomial $\widehat{\Phi}_{A'}$ at points in $\mathrm{GF}(p_i)^n$, and we shall apply it at the point $x \in \{0,1\}^n$, viewed as an element of $\mathrm{GF}(p_i)^n$.

2. For each $i \in [\tau]$, we use Proposition 3.8 to reduce the evaluation of $\widehat{\Phi}_{A'}$ at $x \in \mathrm{GF}(p_i)^n$ to its evaluation at $s' = O(s/\epsilon^2)$ points in $\mathrm{GF}(p_i)^n$, denoted $x^{(i,j)}$. Actually, this procedure is repeated for $O(\log \tau)$ times, and the plurality value is used, so that the probability of error is smaller than $0.1/\tau$ (rather than smaller than $1/3$).

   We stress that at this point we only determined the $x^{(i,j)}$'s, where $i \in [\tau]$ and $j \in [s'']$ (where $s'' = O(s' \log \tau)$).

3. For each $i \in [\tau]$, we apply Proposition 3.9 to $\widehat{\Phi}_{A'}$, while providing p_i as an auxiliary input. Denoting the resulting Boolean formulae by $\Phi^{(i)}$'s, we consider the Boolean formula Φ that is given i as auxiliary input and applies the relevant $\Phi^{(i)}$; that is, $\Phi(i,z) = \bigvee_{j\in[\tau]}(i{=}j \wedge \Phi^{(j)}(z))$. Denoting the corresponding algorithm that produces Φ by B, the foregoing specifies the counting problem $\#_B$.

   At this point, for each $i \in [\tau]$ and $j \in [s'']$, we select uniformly at random a representation $y^{(i,j)} \in \{0,1\}^{n'}$ of $x^{(i,j)} \in \mathrm{GF}(p_i)^n$, where $n' = \exp(f(n)) \cdot n$.

The actual computation (of the reduction) goes in the opposite direction. For every $i \in [\tau]$ and $j \in [s'']$, let $v^{(i,j)}$ denote the answer provided (by a hypothetical solver of the counting problem $\#_B$) for the input $y^{(i,j)}$; that is, $v^{(i,j)}$ is supposed to equal $\#_B(i, y^{(i,j)})$, where $\#_B(i, y^{(i,j)}) \equiv \mathtt{EV}_{A',\mathrm{GF}(p_i)}(x^{(i,j)}) \pmod{p_i}$. Now, for each $i \in [\tau]$, we apply the decoding procedure (of Proposition 3.8, with error reduction) to the values $v^{(i,1)}, ..., v^{(i,s'')}$, and obtain a value $v^{(i)} \in \mathrm{GF}(p_i)$, which is supposed to equal $\#_A(x) \bmod p_i$. Finally, we perform Chinese Remaindering with errors on the $v^{(i)}$'s, and obtain the desired value of $\#_A(x)$ (with high probability).

In the analysis, we observe that if some procedure P solves the final counting problem (i.e., $\#_B$) with success rate at least a $0.75+\epsilon$ (on instances $(i,y) \in [\tau] \times \{0,1\}^{n'}$), then, for at least $0.5+\epsilon$ of the $i \in [\tau]$, the procedure P solves this counting problem with success rate at least a $0.5+\epsilon$ (on instances of the form $(i,\cdot)$). Hence, for each of these majority i's, Proposition 3.9 yields a procedure that solves $\mathtt{EV}_{A',\mathrm{GF}(p_i)}$ with success rate at least $(0.5+\epsilon-o(1))$ (over instances in $\mathrm{GF}(p_i)^n$). This means that, for each of these i's, the hypothesis of Proposition 3.8 holds, and so (for each of these i's) the decoding procedure (of Proposition 3.8) obtains (w.h.p.) the value of $\#_A(x) \bmod p_i$. In this case, Chinese Remaindering with errors works, since the error rate is below its resiliency rate (i.e., $\frac{\log p_1}{\log p_1 + \log p_\tau} - \frac{\epsilon}{2}$). The theorem follows. ∎

4 The average-case to rare-case reduction

Fixing any admissible class $\mathcal{D}$ and a logarithmic function ℓ, and letting $\mathcal{C} = \mathcal{C}_{\ell,\mathcal{D}}$, Theorem 3.1 provides an almost-linear time randomized reduction of solving any problem Π in $\mathcal{C}$ on the worst-case to solving such some problem Π' in $\mathcal{C}$ on the average-case, where the notion of average-case requires solving the problem on at least a 0.76 fraction of the instances. In this section we show that, for every $f \in \mathcal{D}$, the latter (average-case) task can be reduced to solving a related problem on at least a $\exp(-f(n))$ fraction of the instances. Combined, these reductions show that solving the latter related problem on a noticeable fraction of its domain is essentially as hard as solving Π on the worst-case.

We show two related results of this flavor. The first result, stated in Theorem 1.3, shows a *non-uniform* reduction from the average-case task Π' to solving some problem Π'' in $\mathcal{C}$ on at least a $\exp(-f(n))$ fraction of the instances. The second result, stated in Theorem 4.4, shows a *uniform* reduction to solving some problem $\widehat{\Pi}$. While $\widehat{\Pi}$ is not in $\mathcal{C}$ it can be solved in $\mathrm{Dtime}(p(n) \cdot n^{1+o(1)})$, where $p(n) = 2^{\ell(n)} \cdot \exp(f(n))$ bounds the worst-case time to solve Π'. On a technical level, both results use *sample-aided reductions*: reductions that are given uniformly-distributed "solved instances" of the problem that they aim to solve. We provide a definition of this relaxed notion of a reduction between problems, which has been implicit in several past works, and which we find to be of independent interest.

Technical Overview. Our starting point is the realization that when referring to such low success rate (i.e., a $\exp(-f(n))$ fraction for some $f \in \mathcal{D}$), we are in the *regime of list decoding*. Hence, for starters, Proposition 3.8 should (and can) be replaced by the list decoding result of Sudan, Trevisan, and Vadhan [19, Thm. 29]. This result essentially asserts an explicit list of oracle machines such that if F agrees with a low-degree polynomial $P : \mathcal{F}^n \to \mathcal{F}$ on a noticeable fraction of the instances, then given oracle access to F one of these machines computes P correctly on all instances.

Trying to integrate this result in the procedure presented in Section 3.2 means that we proceed as follows: First, we reduce solving the Boolean counting problem Π' on the average-case (i.e., on 76% of the instances) to evaluating a polynomial $\widehat{\Phi}$ over $(\exp(f(n)))$ many different prime fields (on all instances). Next, we apply the foregoing reduction of [19, Thm. 29] in each of these fields, and finally we reduce the oracle calls made by the latter reductions to solving the Boolean counting problem Π'' (on a noticeable fraction of the instances). (Although the statement of [19, Thm. 29] only claims running time that is polynomial in all relevant parameters, it is clear that the running time is linear in the number of variables.)

As in Section 3.2, the input to Π'' is a pair of the form (p, y), where p is a prime and y represents an input to the problem of evaluating $\widehat{\Phi}$ in $\mathrm{GF}(p)$. Hence, solving Π'' on a noticeable fraction of its inputs implies that for a noticeable fraction of the primes p, we correctly solve Π'' on a noticeable fraction of the inputs of the form $(p, \cdot)$, which implies that one of the foregoing oracle machines computes $\widehat{\Phi} : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ correctly on all inputs. If we can identify these primes p and the corresponding oracle machines, then we can solve Π' (on all its instances). (Jumping ahead, we mention that we shall only solve Π' on 76% of its instances, because our identification of the good machines will be approximate in the sense that machines that are correct on almost all their inputs may also pass.)

Towards this end, let us assume that we have access to uniformly distributed "solved instances"

of Π' (i.e., we get a sample of pairs $(r, \Pi'(r))$ for uniformly distributed r's). Using such a sample of solved instances it is easy to estimate the probability that a procedure (e.g., an oracle machine equipt with an adequate oracle) correctly computes Π' mod p: We just compare the value provided for each of the sample points r to the value computed by the procedure. In particular, we can distinguish a procedure that is always correct from a procedure that errs on at least $1/\log n$ of the inputs. Hence, we can pick $\ell(n)/f(n)$ distinct primes (i.e., p's), and an oracle machine for each such prime p such that the chosen machine computes Π' mod p correctly on at least $1-(1/\log n)$ fraction of the inputs. Using Chinese Remaindering *without errors*, this allows us to compute Π' correctly on at least $1-\frac{\ell(n)}{f(n)}\cdot\frac{1}{\log n} = 1-o(1)$ fraction of the inputs. For details see Section 4.1.

This leaves us with the problem of obtaining uniformly distributed "solved instances" of Π'. There are two trivial solutions to this problem: The first is that such random solved instances may be available to us in the application, and the second is that such solved instances can be "hard-wired" in the reduction. This yields a non-uniform reduction, which establishes Theorem 1.3. A third solution, detailed in Section 4.2, is that such samples can be obtained by a downwards self-reduction, whereas each problem in $\mathcal{C}$ can be reduced to one that has a suitable downwards self-reduction.[7] Unfortunately, we were not able to apply this idea to the reduction of Π' to Π''; instead, we apply it to the ("internal") reduction of the Arithmetic problem (i.e., the reduction between the worst-case and average-case (or rather rare-case) versions of the Arithmetic problem). This yields the result stated in Theorem 4.4.

4.1 A sample-aided reduction

We start by spelling out the notion of a reduction that obtains uniformly distributed "solved instances" of the problem that it tries to solve.

Definition 4.1 (sample-aided reductions): *Suppose that M is an oracle machine that, on input $x \in \{0,1\}^n$, obtains as an auxiliary input a sequence of $s = s(n)$ pairs of the form $(r,v) \in \{0,1\}^{n+\ell(n)}$. We say that M is an* sample-aided reduction of solving Π' on ρ' of the instances to solving Π'' on ρ'' of the instances *if, for every procedure P that answers correctly on at least a ρ'' fraction of the instances of length n', it holds that*

$$\Pr_{r_1,...,r_s\in\{0,1\}^n}\left[\left|\mathrm{corr}_M^{P,\Pi'}(r_1,...,r_s)\right| \geq \rho'\cdot 2^n\right] \;>\; 2/3 \tag{7}$$

where

$$\mathrm{corr}_M^{P,\Pi'}(r_1,....,r_s) \stackrel{\mathrm{def}}{=} \left\{x\in\{0,1\}^n : \Pr[M^P(x;(r_1,\Pi'(r_1)),...,(r_s,\Pi'(r_s))) = \Pi'(x)] \geq 2/3\right\} \tag{8}$$

and the latter probability is taken over the coin tosses of the machine M and the procedure P.

The error probability bounds in Eq. (7)-(8) can be reduced at a moderate cost. This is straightforwards for Eq. (8), but the error reduction for Eq. (7) comes at a (small) cost and requires some care. Specifically, we have to approximate the size of the set $\mathrm{corr}_M^{P,\Pi'}(r_1,....,r_s)$, and this can be done using an auxiliary sample of the solved instances. (In particular, using $O(k\cdot s)+O(k/\epsilon^2)$ solved samples, we can output a sequence of s solved samples that, with probability at least $1-\exp(-k)$, has a $\mathrm{corr}_M^{P,\Pi'}$-value of at least $\rho'-\epsilon$.)

[7] Indeed, this follows a paradigm that can be traced to the work of Impagliazzo and Wigderson [14].

Although Definition 4.1 is stated in terms that fit average-case to rare-case (or average-case) reductions, the definition also applies to reductions from worst-case problems (by setting $\rho' = 1$).

We mention that sample-aided reductions are implicit in many known results (see, for example, [11, 19, 15]). Furthermore, any average-case to rare-case reduction of the "list-decoder" type (i.e., which outputs a short list of oracle machines that contains the correct one) yields a sample-aided reduction (as in Definition 4.1).[8]

Theorem 4.2 (sample-aided reduction of average-case to rare-case): *Let $\mathcal{D}$ be a set of admissible functions, $f \in \mathcal{D}$, and ℓ be a logarithmic function. For every counting problem Π' in $\mathcal{C}_{\ell,\mathcal{D}}$, there exists a counting problem Π'' in $\mathcal{C}_{\ell,\mathcal{D}}$ and an almost-linear time sample-aided reduction of solving Π' on at least 0.99 fraction of the domain to solving Π'' on at least $\exp(-f(n))$ fraction of the domain.*

In particular, this yields an almost-linear time *non-uniform* reduction of solving Π' on at least 0.99 fraction of the domain to solving Π'' on at least $\exp(-f(n))$ fraction of the domain. We mention that the following proof can be slightly adapted to yield a sample-aided reduction from solving Π' on at least a $1 - \exp(-f(n))$ fraction of the domain (to solving Π'' on at least $\exp(-f(n))$ fraction of the domain).

Proof Sketch: Given an input $x \in \{0,1\}^n$, we seek to output $\Pi'(x)$, and we are required to provide the correct output (with high probability) on 99% of the possible x's. (We shall reduce the counting problem Π' to the task of evaluating a polynomial $\widehat{\Phi}$ in many prime fields, moving from an average-case version of this problem to a rare-case version, and reduce the latter to a rare-case version of solving the counting problem Π''.)

We start by employing a reduction as in Proposition 3.7, except that here we use $\mathrm{poly}(D)$ primes of size $\mathrm{poly}(D)$, where $D = \exp(f(n))$ denotes the degree of the polynomial $\widehat{\Phi}$ that is derived in Proposition 3.7; specifically, we refer to the first $\Theta(D \cdot \ell(n))$ primes that are larger than $\mathrm{poly}(D)$. Note that at this point we only produce the polynomial $\widehat{\Phi}$ and determine the fields in which it will be evaluated. (Recall that $\widehat{\Phi}$ is a generic polynomial that will be evaluated over different finite fields. Jumping ahead, we note that, on input x, we shall only use the evaluation of $\widehat{\Phi}$ at x in $\ell(n)/f(n)$ of these fields, and combine these values using Chinese Remaindering (without errors).)

Now, for each prime p, we invoke [19, Thm. 29] to obtain $O(1/\epsilon)$ oracle machines that supposedly evaluate $\widehat{\Phi}$ on $\mathrm{GF}(p)^n$, when given oracle access to a procedure that computes $\widehat{\Phi}$ correctly on an ϵ fraction of the inputs, where $\epsilon = \exp(-f(n))$. Next, for each prime p, we invoke Proposition 3.9 in order to answer the queries of these oracle machines such that query $q \in \mathrm{GF}(p)^n$ is answered by querying Π'' on the pair (p, y), where y is a (random) representation of q. Let us denote the random representation so generated by $\mathtt{rr}$ (i.e., $y \leftarrow \mathtt{rr}(q)$). We also note that Π'' is actually fed the index of p in the said set of primes (or some other representation that selects such primes almost uniformly).[9]

(Recall that if the counting problem Π'' is solved correctly on an 2ϵ fraction of the inputs, then, for at least an ϵ fraction of the p's, the solution provided for at least ϵ fraction of the fairs $(p, \cdot)$ is correct. For each such p, at least one of the foregoing oracle machines will evaluate $\widehat{\Phi}$ correctly on $\mathrm{GF}(p)^n$ when fed with answers obtained from this "weak Π''-solver".)

[8] The auxiliary sample can be used to test the candidate oracle machines (as outlined in the foregoing discussion). Note that this allows to distinguish machines that are correct on all inputs from machines that err on a noticeable fraction of the inputs, but not to rule out machines that err on a negligible fraction of the inputs.

[9] Note that the formula Φ that underlies the counting problem Π'' can be implemented as a selector function that picks the corresponding formula $\Phi^{(p)}$ that emulates the computation of $\widehat{\Phi}$ in $\mathrm{GF}(p)^n$.

For each prime p and for each of the corresponding oracle machines M, we approximate the success probability of M in evaluating $\widehat{\Phi}$ over a random element in $\{0,1\}^n$, which is viewed as an element of $\mathrm{GF}(p)^n$. This is done by using the solved sample (for Π'). Specifically, for each pair $(r,v) \in \{0,1\}^{n+\ell(n)}$ in the sample (such that $v = \Pi'(r)$), we compare v to the output of M on input r, while answering the queries of M with the values obtained by the reduction to solving the counting problem Π'' such that the query $q \in \mathrm{GF}(p)^n$ is answered by taking the count provided for the input $(p, \mathtt{rr}(q))$ and reducing it modulo p. If any mismatch is found, then M is discarded as a candidate reduction, and if all oracle machines that correspond to p are eliminated then so is p itself.

Note that a (solved) sample of size $O(f(n) \cdot \log n)$ suffices to approximate all $\mathrm{poly}(D) = \exp(O(f(n))$ quantities such that, with high probability, all values that are below $1 - (1/\log n)$ are estimated as smaller than 1. Hence, with high probability, the list of surviving machines will only contain machines that are correct on at least a $1 - (1/\log n)$ fraction of the inputs in $\{0,1\}^n$. Needless to say, with high probability, the said list contains all machines that are correct on all inputs, where the small probability of error is due to the error probability of the (randomized) oracle machines. Hence, the said list contains more that $\ell(n)/f(n)$ machines that correspond to different primes, since $\epsilon \cdot \Omega(D \cdot \ell(n)) > \ell(n)$.

Lastly, we pick any $\ell(n)/f(n)$ surviving primes, and pick any surviving oracle machine for each of them. For each such prime p and machine M, we use M to solve "Π' mod p" on the original input $x \in \{0,1\}^n$, which is viewed as an element of $\mathrm{GF}(p)^n$.[10] As above, this is done while answering the queries of M with the values obtained by the reduction to solving the counting problem Π'' such that the query $q \in \mathrm{GF}(p)^n$ is answered by taking the count provided for the input $(p, \mathtt{rr}(q))$ and reducing it modolu p. Finally, we combine the values obtained for the count mod each of these primes, by using Chinese Remaindering (without errors).

The analysis amounts to showing that, with very high probability, each of the surviving machines is correct on at least a $1 - (1/\log n)$ fraction of the inputs in $\{0,1\}^n$. Hence, the value computed via the CRT is correct on at least a $1 - \frac{\ell(n)}{f(n)} \cdot \frac{1}{\log n} = 1 - o(1)$ fraction of $\{0,1\}^n$. The claim follows. ■

4.2 Obtaining solved samples via downwards self-reduction

A general paradigm that can be traced to the work of Impagliazzo and Wigderson [14] asserts that *if* Π' *is "downward self-reducible" and* Π' *has a sample-aided reduction to* Π''*, then* Π' *has a standard reduction to* Π''. On input $x \in \{0,1\}^n$, the standard reduction first generates a (solved) sample for Π', for each length $m \leq n$, where these samples are generated starting at $m = 1$ and going upwards to $m = n$. Specifically, when generating the sample for length m, we produce the answers by using the downwards self-reduction, which generates queries of length $m-1$, which in turn are answered by the sample-aided reduction of Π' to Π'', while using the (already generated) sample for length $m-1$. At the end, the answer to the original input x is found using the sample-aided reduction of Π' to Π'' , while using the sample for length n.

The foregoing outline works well in the context of worst-case to rare-case reductions, since in that case the solved sample generated for length $m-1$ is perfect (i.e., free of errors). But when

[10] Note that, since the solved instances given to the reduction are over boolean inputs, we can only estimate the success probability of a machine over $\{0,1\}^n$. In particular, for a surviving prime p and surviving oracle machine M for p, we cannot guarantee that M also gives correct answers on inputs that are outside $\{0,1\}^n$.

using an average-case to rare-case reduction (as in Section 4.1), we run into a problem: In that case, we can only guarantee that $1-\rho'$ fraction of the solutions obtained for the sampled $(m-1)$-bit instances are correct, and in such a case the fraction of errors in the solved sample of m-bit instances generated via downwards self-reduction may increase by a factor that equals the query complexity of the latter reduction, which is typically at least two. The error-doubling effect is disastrous, because the downwards self-reduction is applied many times.

In light of the above, we wish to apply the foregoing process to a worst-case to a rare-case reduction, and recall that the reduction of [19, Thm. 29] is actually of that type. Hence, starting from an arbitrary problem Π in $\mathcal{C}$, we first reduce it to a problem of evaluating low-degree polynomials that is downwards self-reducible. This problem will be a generalization of the evaluation problem presented in Definition 3.2. Before presenting this generalization, we introduce a more stringent notion of downwards self-reducibility, which is essential to our application. In particular, we require the reduction to work in almost-linear time (rather than in polynomial-time), and that the iterative process of downward self-reduction terminates after few steps (when reaching input length that is only slightly smaller than the original one). Specifically, we say that a problem Π is downward self-reducible if *there exists an almost-linear time oracle machine that, on input $x \in \{0,1\}^n$, makes queries of length $n-1$ only, and outputs $\Pi(x)$ provided that its queries are answered by Π*. In addition, we require that for a sufficiently dense set of input lengths, the problem can be solved in almost linear time without making any queries to shorter input lengths (i.e., for these input lengths, the oracle machine makes no queries at all). Specifically, *for some $f \in \mathcal{D}$, we require that for every $n \in \mathbb{N}$ the interval $[n+1, n+\exp(f(n))]$ contains such a length.* This additional requirement, hereafter referred to as Condition A, offers a crucial saving in the foregoing transformation from sample-aided reductions to standard reductions: The iterative downwards reduction procedure reaches the "base case" rather quickly; that is, on input $x \in \{0,1\}^n$, the iterative downwards reduction stops at length n' such that $n' \geq n - \exp(f(n))$ (where the downwards self-reduction makes no queries on inputs of length n'). Consequently, it suffices for the standard reduction to generate samples for lengths $n', n'+1, ..., n$.

Turning to the generalization of the evaluation problem presented in Definition 3.2, we seize the opportunity to pack together the different problems defined for the various fields. Recalling that the proof of Proposition 3.7 utilizes only $\ell(n)/f(n)$ different fields, which correspond to the first $\ell(n)/f(n)$ primes that are larger than $2^{f(n)}$, we make the index of the field part of the input. Actually, to guarantee that inputs that correspond to different fields have different lengths, we present the index of the field in unary.[11]

To obtain a downward self-reduction, we reduce the original n-bit long instance, which sums over the entire range of indices $\{0,1\}^{\ell(n)}$ (i.e., the index w in Eq. (2) of Definition 3.2), to (two instances of) summation over a smaller range $\{0,1\}^{\ell(n)-1}$. This is accomplished in the natural way by fixing the first bit of the summation-index (to 0 or 1), and making this bit part of an augmenged input. Similarly, for any $j \in \{0,\ldots,\ell(n)-1\}$, summation over a restricted range $\{0,1\}^{\ell(n)-j}$ is reduced to (two instances of) summation over the smaller range $\{0,1\}^{\ell(n)-j-1}$, which yields inputs that have $j+1$ bits of augmentation. Finally, when the "summation" is over a single element, which happens when the augmented input has length $n+\ell(n)$, the problem can be solved

[11] We shall reduce solving the original (worst-case) instance of length n to rare-case solving instances of various lengths, which correspond to different prime fields. Hence, for each input length for the original problem, we rely on being able to rarely solve the reduced problem on several input lengths (rather than on one input length as in the reduction presented so far). We note that this disadvantage is inherent to the downwards self-reduction paradigm of [14], so we may just take advantage of it for this additional purpose.

directly in nearly-linear time, which satisfies the additional condition (Condition A) of downward self-reducibility (see above). Note that this iterative process increases the length of the input in each iteration by one bit, whereas we want the input length to decrease in the iterations. This is achieved by padding the original input, and removing two bits of this padding in each iteration.

Thus, we divide the input to the problem into three parts, denoted U, V and X. The first part (i.e., U) includes the index of the field in unary, as described above, as well as the padding for guaranteeing that the input length shrinks when we fix a bit of the index. The second part (i.e., V) represents the prefix of the summation-index that has been fixed, and X is the "main" input. We shall use a multi-linear function $\mathtt{EQ} : \mathrm{GF}(p)^m \times \mathrm{GF}(p)^m \to \mathrm{GF}(p)$ that tests equality of m-bit long strings (viewed as m-long sequences over $\mathrm{GF}(p)$); that is, $\mathtt{EQ}(y_1 \cdots y_m, z_1 \cdots z_m) = \prod_{i\in[m]}(y_i z_i + (1-y_i)(1-z_i))$.

Definition 4.3 (Generalization of Definition 3.2): *For $f \in \mathcal{D}$ and $n \in \mathbb{N}$, let $p_1, ..., p_{\ell'(n)}$ denote the first $\ell'(n) = \ell(n)/f(n)$ primes that are larger than $2^{f(n)}$, and $L_n = \{0, 1, ..., \ell(n)\}$. For $A', \widehat{\phi}_n$ and $\pi_{n,1}, ..., \pi_{n,m(n)}$ as in Definition 3.2, the* generalized evaluation problem associated with A' *consists of computing the function $\overline{\Phi}_n : \bigcup_{i\in[\ell'(n)],j\in L_n} \mathrm{GF}(p_i)^{\ell(n)^2\cdot n+(\ell(n)+1)\cdot i-j} \times \mathrm{GF}(p_i)^n \to \bigcup_{i\in[\ell'(n)]} \mathrm{GF}(p_i)$ defined as follows: For every $i \in [\ell'(n)]$ and $j \in L_n$, and every $X = (X_1, ..., X_n) \in \mathrm{GF}(p_i)^n$ and $(U, V) \in \mathrm{GF}(p_i)^{\ell(n)^2\cdot n+(\ell(n)+1)\cdot i-2j} \times \mathrm{GF}(p_i)^j$, it holds that*

$$\overline{\Phi}_n(UV, X) \stackrel{\text{def}}{=} \sum_{w'\in\{0,1\}^{\ell(n)-j}} \widehat{\phi}_n(Vw', F_1(Vw', X), ..., F_{m(n)}(Vw', X)) \bmod p_i, \tag{9}$$

where $F_k(W, X) = \sum_{\alpha\in[n]} \mathtt{EQ}(\alpha, \widehat{\pi}_{n,k}(W)) \cdot X_\alpha$ and $\widehat{\pi}_{n,k} : \mathrm{GF}(p_i)^{\ell(n)} \to \mathrm{GF}(p_i)^{\log n}$ is a low degree polynomial that agrees with $\pi_{n,k} : \{0,1\}^{\ell(n)} \to [n]$ (cf. the proof of Proposition 3.3), and $[n] \equiv \{0,1\}^{\log n} \subset \mathrm{GF}(p_i)^{\log n}$.

Note that UV is a sequence of $\ell(n)^2 \cdot n + (\ell(n)+1) \cdot i - j$ elements of $\mathrm{GF}(p_i) \subset \{0,1\}^{f(n)+1}$, and that the said length determines both i and j. For simplicity, the reader may consider the length of the instance (UV, X) in terms of $\mathrm{GF}(p_i)$-elements (i.e., as equal $(\ell(n)^2+1) \cdot n + (\ell(n)+1) \cdot i - j$), but the following observations remain valid also when defining the length of that instance as $f(n)+1$ times longer (and observing that all p_i's satisfy $\lceil \log p_i \rceil = f(n)+1$).

1. The problem in Definition 3.2 (when restricted to any of fields $\mathrm{GF}(p_i)$) is (worst-case) reducible to the problem in Definition 4.3 (equiv., each counting problem in $\mathcal{C}$ is reducible to the generalized evaluation problem).

   The reduction may consist of mapping the instance $x \in \mathrm{GF}(p_i)^n$ to the instance $(1^{\ell(n)^2\cdot n+(\ell(n)+1)\cdot i}, x)$. (Equivalently, the proof of Proposition 3.7 can be similarly adapted.)

2. The mapping $(n, i, j) \to (\ell(n)^2+1) \cdot n + (\ell(n)+1) \cdot i - j$ is injective when restricted to $i \in [\ell'(n)]$ and $j \in L_n$, since $\ell'(n) < \ell(n)$ (and $L_n = \{0, 1, ..., \ell(n)\}$).

3. $\overline{\Phi}$ satisfies the additional condition (Condition A) of downward self-reducibility, since for every $x \in \mathrm{GF}(p_i)^n$ and $(u, v) \in \mathrm{GF}(p_i)^{\ell(n)^2\cdot n+(\ell(n)+1)\cdot i-2\ell(n)} \times \mathrm{GF}(p_i)^{\ell(n)}$, where $i \in [\ell'(n)]$, it holds that $\overline{\Phi}(uv, x) = \widehat{\phi}_n(v, F_1(v, x), ..., F_{m(n)}(v, x)) \bmod p_i$, can be computed in almost-linear time with no oracle calls. In particular, each $F_k(v, x) = \sum_{\alpha\in[n]} \mathtt{EQ}(\alpha, \widehat{\pi}_{n,k}(v)) \cdot x_\alpha$ can be computed in almost-linear time.

4. $\overline{\Phi}$ satisfies the main condition of downward self-reducibility, since for every $x \in \mathrm{GF}(p_i)^n$ and $(u,v) \in \mathrm{GF}(p_i)^{\ell(n)^2 \cdot n + (\ell(n)+1)\cdot i - 2j} \times \mathrm{GF}(p_i)^j$, where $i \in [\ell'(n)]$ and $j \in \{0,1,...,\ell(n)-1\}$, it holds that $\overline{\Phi}(uv,x) = \overline{\Phi}(u'0v,x) + \overline{\Phi}(u'1v,x)$, where u' is the $(\ell(n)^2 \cdot n + i \cdot \ell(n) - 2(j+1))$-long prefix of u (i.e., $u = u'\tau_1\tau_2$ for some $\tau_1, \tau_2 \in \mathrm{GF}(p_i)$).

   (Indeed, the fact that u' is two field-elements shorter than u, allows to extend v by one field-element, while yielding an input that is shorter than the original one.)

Theorem 4.4 (worst-case to rare-case reduction): *Let $\mathcal{D}$ be a set of admissible functions, $f \in \mathcal{D}$, and ℓ be a logarithmic function. For every counting problem Π in $\mathcal{C} = \mathcal{C}_{\ell,\mathcal{D}}$, there exist a generalized evaluation problem $\overline{\Phi}$* (as in Definition 4.3) *and an almost-linear time randomized reduction of solving Π* (in the worst-case) *to solving $\overline{\Phi}$ on at least $\exp(-f(n))$ fraction of the domain.*

Recall that the foregoing evaluation problem can be solved in time $2^{\ell(n)} \cdot n^{1+o(1)}$. Hence, we have reduced the worst-case complexity of $\mathcal{C}$ to the rare-case complexity of $\overline{\Phi}$, while upper-bounding the worst-case complexity of $\overline{\Phi}$ (alas not reducing $\overline{\Phi}$ to $\mathcal{C}$).

Proof Sketch: By the forgeoing discussion, solving Π reduces to solving $\overline{\Phi}$. Specifically, for every $x \in \{0,1\}^n$, the value of $\Pi(x)$ is obtained by applying the Chinese Remaindering (without errors) to the values of $\overline{\Phi}$ on the $\ell'(n)$ instances $(1^{\ell(n)^2 \cdot n + (\ell(n)+1)\cdot i}, x)$ for $i \in [\ell'(n)]$. Hence, fixing any ($n \in \mathbb{N}$ and) $i \in [\ell'(n)]$ and following the length convention stated above, we focus on reducing the task of computing $\overline{\Phi}$ on any instance of length $\widetilde{n} = (\ell(n)^2+1) \cdot n + (\ell(n)+1) \cdot i$ to obtaining this value of an $\epsilon = \exp(-f(n)/4)$ fraction of these instances, where the relevant domain is $\mathrm{GF}(p_i)^{\widetilde{n}}$.

(Note that we reduced obtaining the value of Π on an arbitrary (worst-case) $x \in \{0,1\}^n$ to obtaining values of $\overline{\Phi}$ on the $\ell'(n)$ instances, having varying lengths (i.e., the i^{th} instance has length $\ell(n)^2 \cdot n + (\ell(n)+1) \cdot i + n = \widetilde{n}$; it consists of padding x with $\widetilde{n} - n$ ones). Thus, we presented a worst-case to worst-case reduction from solving Π for *every* input length on *all inputs* to solving $\overline{\Phi}$ for *every* input length on *all inputs*. We stress that this reduction does not reduce solving Π on all inputs of a specific length n to solving $\overline{\Phi}$ on all inputs of some (possibly other) length n'. The next step, which is based on a downward self-reduction, will have a similar flavour (w.r.t input lengths): We shall present a worst-case to rare-case reduction from solving $\overline{\Phi}$ for *every* input length on *all inputs* to solving $\overline{\Phi}$ for *every* input length on *rare inputs*. Indeed, using many input lengths in the target of the reduction is inherent to the use of downward self-reductions.)

Focusing on the goal of presenting a worst-case to rare-case reduction for $\overline{\Phi}$, we observe that the result of [19, Thm. 29] provides a sample-aided reduction that achieves the desired goal. Specifically, as stated, their algorithm outputs a list of $O(1/\epsilon)$ oracle machines that contain machines that compute any polynomial that has agreement at least ϵ with the given oracle.[12] The list may contain also other machines. Still, using low-degree tests (e.g., [18]), we can guarantee that all machines on the list are very close to computing some low degree polynomial, and by employing self-correction procedures (e.g., [8]) we obtain machines that compute low degree polynomials. Next, using a solved sample of the polynomial that we seek to compute (i.e., $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}} \to \mathrm{GF}(p_i)$), we can identify a machine that computes the correct polynomial (since the other machines compute polynomials that disagree with the correct one on most inputs). Although this machine may not be unique, all surviving machines compute the same polynomial, and we may use any of them. Note that a sample of size $O(\log(1/\epsilon)) = O(f(n))$ suffices for identifying all bad machines with high probability.

[12] As noted in Section 4.1, although the statement of [19, Thm. 29] only claims running time that is polynomial in all relevant parameters, it is clear that the running time is linear in the number of variables.

The solved sample for $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}} \to \mathrm{GF}(p_i)$ is obtained by the foregoing downwards self-reduction, while using the fact that, for every $j \in L_n$, we can apply the foregoing argument to $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}-j} \to \mathrm{GF}(p_i)$. Specifically, on input $y \in \mathrm{GF}(p_i)^{\widetilde{n}}$, we proceed as follows.

1. Generate a random sample of $O(f(n))$ solved instances for $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}-\ell(n)} \to \mathrm{GF}(p_i)$, while relying on the fact that, for such an input length, the function $\overline{\Phi}$ can be computed in almost-linear time (Condition A of downward self-reducibility).

2. For $k = \widetilde{n} - \ell(n) + 1, ..., \widetilde{n}$ (equiv., using $k = \widetilde{n} - j$ for $j = \ell(n) - 1, ..., 0$), generate a random sample of $O(f(n))$ solved instances for $\overline{\Phi} : \mathrm{GF}(p_i)^k \to \mathrm{GF}(p_i)$, where the values of $\overline{\Phi}$ on $r = (uv, x) \in \mathrm{GF}(p_i)^{k-n} \times \mathrm{GF}(p_i)^n$ is obtained by invoking the downwards self-reducibilirty on input r and answering its queries by using the sample-aimed reduction for inputs of length $k - 1$ (while using the samples generated in the previous iteration). Recall that on input $r = (uv, x) \in \mathrm{GF}(p_i)^{\widetilde{n}-2(\widetilde{n}-k)+(\widetilde{n}-k)-n} \times \mathrm{GF}(p_i)^n$ the value of $\overline{\Phi}(r)$ is obtained by querying $\overline{\Phi}$ on $(u'v0, x)$ and on $(u'v1, x)$, where u' is the $(\widetilde{n} - 2(\widetilde{n} - k) - 2)$-long prefix of u, and that these inputs are of length $\widetilde{n} - (\widetilde{n} - k) - 1 = k - 1$.

3. Lastly, obtain the value of $\widehat{\Phi}(y)$ by invoking the sample-aimed reduction for inputs of length $\widetilde{n}$ (while using the samples generated in the previous step).

We stress that the invocations of the sample-aided reductions are invoked with a number of samples that guarantees that the probability of error in the invocation is smaller than $\exp(-f(n)) \ll 1/\ell(n)$. Hence, with high probability, our reduction provides the correct value of $\overline{\Phi}(x^{(i)}) \equiv \Pi(x) \pmod{p_i}$, where $x^{(i)} = (1^{\ell(n)^2 \cdot n + (\ell(n)+1) \cdot i}, x)$. ■

Digest of the reduction of Π to $\overline{\Phi}(r)$. The description of the reduction goes top-down (i.e., from Π to the generalized problem, then uses its worst-case to average-case reduction, and the downwards self-reduction that goes from the top level to the bottom one), but the algorithm that computes Π goes bottom-up. The original input $x \in \{0,1\}^n$ is transformed into $\ell'(n)$ inputs for the generalized problem, denoted $x^{(1)}, ..., x^{(\ell'(n))}$, such that $x^{(i)} = (1^{\widetilde{n}_i - n}, x)$ and $\widetilde{n}_i = (\ell(n)^2 + 1) \cdot n + (\ell(n) + 1) \cdot i$. The instance $x^{(i)}$ is viewed both as an $(f(n) + 1) \cdot \widetilde{n}_i$-bit long string and as an $\widetilde{n}_i$-long sequence over $\mathrm{GF}(p_i)$. The process of downwards self-reduction goes from length $\widetilde{n}_i$ to length $\widetilde{n}_i - \ell(n)$, whereas the actual generation of solved samples goes the other way around (i.e., from $\widetilde{n}_i - \ell(n)$ to $\widetilde{n}_i$). Specifically, a sample for length $\widetilde{n}_i - \ell(n)$ is generated by straightforward compuation of the value of $\overline{\Phi}$ on random instances length $\widetilde{n}_i - \ell(n)$, whereas a sample for length $k = \widetilde{n}_i - j$ is generated by using the downward self-reduction to length $k - 1$ and applying the sample-aided worst-case to rare-case reduction for length $k - 1$ while using the solved sample generated in the previous iteration. Lastly, the solution to $x^{(i)} = (1^{\widetilde{n}_i - n}, x)$ is found using the sample-aided worst-case to rare-case reduction for length $\widetilde{n}_i$ while using the solved sample generated above. Finaly, the solution to x is obtained by combining the solutions to the various $x^{(i)}$'s (using the CRT).

Acknowledgements

We are grateful to Madhu Sudan for many useful discussions regarding list decoding of multivariate polynomials and related issues.

References

[1] Miklos Ajtai. Σ_1^1-formulae on finite structures. *Ann. Pure Appl. Logic*, Vol. 24 (1), pages 1–48, 1983.

[2] Laszlo Babai. Random oracles separate PSPACE from the Polynomial-Time Hierarchy. *IPL*, Vol. 26, pages 51–53, 1987.

[3] Laszlo Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Complexity Theory*, Vol. 3, pages 307–318, 1993.

[4] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant N. Vasudevan. Average-Case Fine-Grained Hardness. *ECCC*, TR17-039, February 2017.

[5] Boaz Barak. A Probabilistic-Time Hierarchy Theorem for "Slightly Non-uniform" Algorithms. In the proceedings of the *6th RANDOM*, pages 194–208, 2002.

[6] Omer Barkol and Yuval Ishai. Secure Computation of Constant-Depth Circuits with Applications to Database Search Problems. In the proceedings of *CRYPTO*, pages 395–411, 2005.

[7] Andrej Bogdanov and Luca Trevisan. On Worst-Case to Average-Case Reductions for NP Problems. *SIAM Journal on Computing*, Vol. 36 (4), pages 1119–1159, 2006.

[8] Peter Gemmell, Richard J. Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-Testing/Correcting for Polynomials and for Approximate Functions . In the proceedings of *ACM Symposium on the Theory of Computing*, pages 32–42, 1991.

[9] Mikael Goldmann, Per Grape, and Johan Hastad. On Average Time Hierarchies. *Information Processing Letters*, Vol. 49 (1), pages 15–20, 1994.

[10] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[12] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese Remaindering with Errors. *IEEE Trans. Information Theory*, Vol. 46 (4), pages 1330–1338, 2000.

[13] Oded Goldreich and Guy N. Rothblum. Simple Doubly-Efficient Interactive Proof Systems for Locally-Characterizable Sets. *ECCC*, TR17-018, February 2017.

[14] Russell Impagliazzo and Avi Wigderson. Randomness vs Time: Derandomization under a Uniform Assumption. *Journal of Computer and System Science*, Vol. 63 (4), pages 672–688, 2001.

[15] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform Direct Product Theorems: Simplified, Optimized, and Derandomized. *SIAM Journal on Computing*, Vol. 39 (4), pages 1637–1665, 2010.

[16] Dexter Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, New York, 1991.

[17] Ronitt Rubinfeld and Madhu Sudan. Self-Testing Polynomial Functions Efficiently and Over Rational Domains. In the proceedings of *3rd SODA*, pages 23–32, 1992.

[18] Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25(2), pages 252–271, 1996. Unifies and extends part of the results contained in [8] and [17].

[19] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom Generators without the XOR Lemma. *Journal of Computer and System Science*, Vol. 62 (2), pages 236–266, 2001.

[20] Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis. In *10th Int. Sym. on Parameterized and Exact Computation*, pages 17–29, 2015.

Appendix: A related class (for context only)

Our starting point is the definition of locally-characterizable sets [13], which is slightly revised as follows.[13]

Definition A.5 (locally-characterizable sets): *A set S is* locally-characterizable *if there exist a constant c, a polynomial p and a polynomial-time algorithm that on input n outputs* $\mathrm{poly}(\log n)$*-sized formulae $\phi_n : \{0,1\}^{c\log n} \times \{0,1\}^{p(\log n)} \to \{0,1\}$ and $\pi_{n,1}, ..., \pi_{n,p(\log n)} : \{0,1\}^{c\log n} \to [n]$ such that, for every $x \in \{0,1\}^n$, it holds that $x \in S$ if and only if for all $w \in \{0,1\}^{c\log n}$*

$$\Phi_x(w) \stackrel{\text{def}}{=} \phi_n(w, x_{\pi_{n,1}(w)}, ..., x_{\pi_{n,p(\log n)}(w)}) \tag{10}$$

equals 0.[14]

That is, each value of $w \in \{0,1\}^{c\log n}$ yields a local condition that refers to polylogarithmically many locations in the input (i.e., the locations $\pi_{n,1}(w), ..., \pi_{n,p(\log n)}(w) \in [n]$). This local condition is captured by ϕ_n, and in its general form it depends both on the selected locations and on the value on the input in these locations. A simplified form, which suffices in many case, uses a local condition that only depends on the values of the input in these locations (i.e., $\phi_n : [n]^{p(\log n)} \times \{0,1\}^{p(\log n)} \to \{0,1\}$ only depends on the $p(\log n)$-bit suffix).

A locally-characterizable set corresponds to the set of inputs for the counting problem (of Definition 2.2) that have a zero count. The correspondence is not an equality, beacuse the sizes of the formulae in the two definitions are different. Whereas in Definition 2.2 the number of formulae and their sizes are exponential in a function f that is in the admissible class, in Definition A.5 the number and size is poly-logarithmic in n. In both definitions, the formulae are constructed in time that is polynomial in their number and size.

[13] In [13, Def. 2], the formula ϕ_n got as input $(\pi_{n,1}(w), ..., \pi_{n,p(\log n)}(w))$ as well as $(x_{\pi_{n,1}(w)}, ..., x_{\pi_{n,p(\log n)}(w)})$. This is equivalent to the form used here, since on the one hand ϕ_n can compute the $\pi_{n,i}$'s (given w), and on the other hand w can be reconstructed from c auxiliary $\pi_{n,i}$'s.

[14] Indeed, it is required that in case of inputs in S, the predicate ϕ_n evaluates to 0 (rather than to 1). This choice was made in [13] in order to simplify the exposition.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il