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

Oded Goldreich Guy N. Rothblum

December 5, 2019

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 t-cliques in a graph, for any fixed $t \geq 3$.

Specifically, 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. We show an almost-linear-time reduction from solving any such problem in the worst-case to solving some other problem (in the same class) on typical inputs. Furthermore, for some of these problems, we show that their average-case complexity almost equals their worst-case complexity.

En route we highlight a few issues and ideas such as sample-aided reductions, average-case analysis of randomized algorithms, and average-case emulation of arithmetic circuits by Boolean circuits. We also observe that adequately uniform versions of $\mathcal{AC}^0[2]$ admit worst-case to average-case reductions that run in almost linear-time.

Keywords: Worst-case to average-case reductions, fine-grained complexity,

Contents

1 Introduction

While most research in the theory of computation refers to worst-case complexity (cf., e.g., [13, Chap. 1–10.1] versus [13, Sec. 10.2]), the importance of average-case complexity is widely recognized. 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, until recently worst-case to average-case reductions were known only either for "very high" complexity classes, such as $\mathcal{E}$ and $\#\mathcal{P}$ (see [4] and [22, 9, 15][1], resp.), or for problems of "very low" complexity, such as `Parity` (cf. [1, 3]). In contrast, presenting a worst-case to average-case reduction for $\mathcal{NP}$ is a well-known open problem, which faces significant obstacles [10, 8].

A recent work by Ball, Rosen, Sabin, and Vasudevan [5] initiated the study of worst-case to average-case reductions in the context of fine-grained complexity.[2] The latter context focuses on the exact complexity of problems in $\mathcal{P}$ (see, e.g., survey by V. Williams [30]), 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 [5] (and followed here) – is to use almost linear-time reductions.[3]

The pioneering paper of Ball *et al.* [5] 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 some problems in $\mathcal{P}$. Furthermore, the worst-case complexity of the latter problems matches (approximately) the best algorithms *known* for the former problems (but note that the latter problems may actually have higher complexity than the former problems).

1.1 Our results

In this paper we strengthen the foregoing results in two ways. First, we present worst-case to average-case reductions (of almost linear-time) for a wide class of problems within $\mathcal{P}$ (see Theorem 1.1). Second, we show that the average-case complexity of some of these problems approximately equals their worst-case complexity, which is conjectured to be an arbitrary high polynomial (see Corollary 1.2). Details follow.

[1]The basic idea underlying the worst-case to average-case reduction of the "permanent" is due to Lipton [22], but his proof implicitly presumes that the field is somehow fixed as a function of the dimension. This issue was addressed independently by [9] and in the proceeding version of [15]. In the current work, we shall be faced with the very same issue.

[2]Actually, a worst-case to average-case reduction for a problem of this flavor was shown before by Goldreich and Wigderson [18]. They considered the problem of computing the function $f_n(A_1, ..., A_{\ell(n)}) = \sum_{S\subseteq[\ell(n)]} \mathtt{DET}(\sum_{i\in S} A_i)$, where the A_i's are n-by-n matrices over a finite field and $\ell(n) = O(\log n)$, conjectured that it cannot be computed in time $2^{\ell(n)/3}$, and showed that it is random self-reducible (by $O(n)$ queries). They also showed that it is downwards self-reducible when the field has the form $GF(2^{m(n)})$ such that $m(n) = 2^{\lceil \log_2 n\rceil}$ (or $m(n) = 2\cdot 3^{\lceil \log_3 n\rceil}$). We stress that the (subquadratic-time) reduction of [18] does not run in almost-linear time, and that the foregoing problem is not as well-studied as the problems considered in [5].

[3]Furthermore, typically, these reductions make a small number of queries (since their queries are of at least linear length). In any case, composing them with a T-time algorithm for the target problem yields an "almost T-time" algorithm for the reduced problem, provided that $T(n) \in [\Omega(n), \mathrm{poly}(n)]$.

Complexity classes having worst-case to average-case reductions. First, for each polynomial p, we define a worst-case complexity class $\mathcal{C}^{(p)}$ that is a subset of Dtime($p^{1+o(1)}$), and then show, 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. These classes are defined in Section 2, and they are related to the class of locally characterizable sets defined by us in [16] (see also Appendix A.1). 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` (e.g., the number of t-cliques in an n-vertex is expressed as the sum of $\binom{n}{t}$ local conditions, each referring to the subgraph induced by t specific vertices). We show that all theses classes have worst-case to average-case reductions.

Theorem 1.1 (worst-case to average-case reduction, loosely stated):[4] *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 a 0.76 fraction of the domain).[5]

Hence, worst-case to average-case reductions are shown to be a rather general phenomenon: They exist for each problem in a class that is defined in general terms (i.e., counting the number of local conditions that are satisfied by the input, where the local conditions are specified by formulae of bounded complexity). Moreover, the reductions preserve membership in this class (rather than moving to a problem of a different (arithmeticized) flavor as in [5]). In particular, Theorem 1.1 asserts that the average-case complexity of $\mathcal{C}^{(p)}$ is lower-bounded by the worst-case complexity of $\mathcal{C}^{(p)}$, provided that the latter is at least almost-linear. (Recall that the worst-case complexity of $\mathcal{C}^{(p)}$ is at most $p^{1+o(1)}$, and it is most likely that it is not almost-linear.)

A follow-up work of the authors [17] showed a worst-case to average-case reduction for t-`CLIQUE`, the problem of counting the number of t-cliques in a graph, for each $t \geq 3$. In contrast, the focus of the current work is on *classes* of problems, that are defined in terms of general complexity bounds. Specifically, the classes of counting problems studied here refer to a polynomially large (uniform) collection of small formulae that are applied to the same input.

Computational problems with average-case complexity that almost equals their worst-case complexity. In Theorem 1.1 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'). Our second improvement over [5] is in narrowing the potential gap between the complexities of Π and Π' (and, consequently, between the average-case and worst-case complexities of Π').

We do so by taking advantage of the fact that our reduction preserves membership in the class $\mathcal{C}^{(p)}$. Specifically, for every $\epsilon > 0$, assuming that the worst-case complexity of $\Pi \in \mathcal{C}^{(p)}$ is at least q, and invoking Theorem 1.1 for $O(1/\epsilon)$ times, we obtain a problem in $\mathcal{C}^{(p)} \setminus \text{Dtime}(q)$ whose *average-case* complexity is at most n^ϵ times smaller than its *worst-case* complexity. Hence, the average-case complexity of this problem is approximately equal to its worst-case complexity.

[4]See Theorem 3.1 for a precise statement.

[5]Here 0.76 stands for any constant greater than 3/4.

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

A result analogous to Corollary 1.2 was proved in our follow-up work [17], while assuming that t-`CLIQUE` is adequately hard. In contrast, the hypothesis of Corollary 1.2 requires only that *some* problem in the class $\mathcal{C}^{(p)}$ is adequately hard.

Proof: Starting with $\Pi^{(0)} \in \mathcal{C}^{(p)} \setminus \mathrm{Dtime}(q)$, 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 $\Pi^{(i-1)}$. 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 (i.e., the $\Pi^{(i)}$'s) constitute a non-decreasing sequence that is lower-bounded by q and upper-bounded by $p(n)^{1+o(1)}$, which is an upper bound on the complexity of any problem in $\mathcal{C}^{(p)}$. It follows that for some $i \in [O(1/\epsilon)]$, the ratio between the worst-case complexities of $\Pi^{(i)}$ and $\Pi^{(i-1)}$ is at most n^{ϵ}, 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 fraction of the instances (say, on a $n^{-o(1)}$ fraction of the n-bit long instances).[6] Using non-uniform reductions, we show that the rare-case complexity of $\mathcal{C}^{(p)}$ is lower-bounded by its (non-uniform) worst-case complexity (provided that the latter is at least almost-linear).

Theorem 1.3 (worst-case to rare-case reduction, loosely stated):[7] *For every polynomial p, and every problem Π in $\mathcal{C}^{(p)}$, there exists a 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.

1.2 Comparison to the known average-case hierarchy theorem

We mention the existence of an (unconditional) average-case hierarchy theorem of Goldmann, Grape, and Hastad [12], which is essentially proved 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

[6]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).

[7]See Theorem 4.2 for a precise statement. Note that Theorem 4.2 refers to sample-aided reductions, which imply non-uniform reductions. Theorem 1.3 refers to a liberal notion of almost-linearity that includes any function of the form $f(n) = n^{1+o(1)}$. Under a strict notion that postulates that only functions of the form $f(n) = n^{1+o(1)}$ are deemed almost-linear, the rare-case refers to an $1/\mathrm{poly}(\log n)$ fraction of the n-bit long instances.

unconditional. We point out several advantages of the current reductions (as well as those of [5]) 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. [6]).

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 [5]) 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 [12]): 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., [7]).

1.3 Techniques

The proofs of all our results are pivoted at *arithmetic versions* of the (Boolean) counting problems belonging to the class $\mathcal{C}^{(p)}$. These arithmetic versions refer to evaluating a small arithmetic formula on $p(n)$ different (short) projections of the n-symbol input, which is a sequence over a finite field (of varying size). The counting problem is easily reduced to the arithmetic problem, which is (implicitly) known to have a worst-case to average-case reduction (as well as an average-case to rare-case reduction). The tricky part is presenting a reduction of the arithmetic problem "back to" a Boolean counting problem in $\mathcal{C}^{(p)}$, where the reduction preserves the error rate (i.e., average-case or rare-case complexity).

1.3.1 On the proof of Theorem 1.1

A first attempt at a reduction from the arithmetic problem to a Boolean (counting) problem in $\mathcal{C}^{(p)}$, captured by Proposition 3.6, increases the error rate of the solver for the Boolean problem by a non-constant factor, yielding average-case hardness only with respect to error rate that tends to zero. Even this reduction is not straightforward; see discussion at the beginning of the proof of Proposition 3.6.

Deriving average-case hardness with respect to constant error rate requires a few refinements of the basic approach. First, rather than reducing the n-bit instance of the original counting problem to a single instance of the Arithmetic problem defined over a finite field of size $\Omega(p(n))$, we reduce it to $O(\log p(n))$ instances defined over different fields of much smaller size.[8] Since all these instances will be reduced to the same average-case complexity problem, some of the values obtained in these

[8]The sizes of these fields are lower bounded not only by $\Omega(\log p(n))$ but also by the size of the Boolean formula used in the original counting problem. The latter lower bound guarantees that the field is larger than the degree of the polynomial that is computed by the Arithmetic problem used in the reduction.

reductions may be wrong, and Chinese Remaindering *with errors* (cf. [15]) is used in combining them.

We still face the problem of reducing the Arithmetic problem (over fields of size $n^{o(1)}$ (rather than $n^{\Omega(1)}$)) to a counting problem in the class $\mathcal{C}^{(p)}$, while preserving the average-case complexity (equiv., the error rate of potential solvers). To do so, we use (a non-constant number of) Boolean formulae that compute each of the bits in the representation of the output of the arithmetic formula (which is a field element). However, we cannot afford to deal with each of the resulting formulae separately, since this will increase the error rate by a non-constant factor. Instead, we reduce evaluation of the arithmetic formula to solving a *single* counting problem in the class $\mathcal{C}^{(p)}$.

To do so, we construct a single Boolean formula that, when fed with an appropriate auxiliary input, outputs the corresponding bit in the representation of the arithmetic formula's output (which is a field element). The (weighted) summation of the Boolean outputs on the different auxiliary inputs (one per each bit in the representation of the field element) is performed by the counting problem itself (i.e., by the summation). This is done by using additional variables that emulate the weights (as detailed in Section 3.2), and yields a single-query reduction of the Arithmetic problem to the Boolean problem.

1.3.2 On the proofs of Theorems 1.3 and 4.4

Turning from average-case hardness to rare-case hardness (i.e., from the proof of Theorem 1.1 to the proofs of Theorems 1.3 and 4.4), we employ worst-case to rare-case reduction techniques at the Arithmetic level. Specifically, we use the list decoder of Sudan, Trevisan, and Vadhan [29].

For the proof of Theorem 4.4, we also employ a methodology heralded by Impagliazzo and Wigderson [20], which we distill here. The methodology is pivoted at the notion of *sample-aided reductions*, which is a relaxed notion of reduction between problems. Specifically, a sample-aided reduction from problem A to problem B receives random solved instances of problem A, which means that it implies an ordinary non-uniform reduction (of A to B). Furthermore, when coupled with a downwards self-reduction for the problem A, a sample-aided reduction of A to B implies a standard (uniform) reduction of A to B.

In the following definition, a task consists of a computational problem along with a required performance guarantee; for example, A may be "solving problem Π on the worst-case" and B may be "solving Π with success rate ρ". For sake of simplicity, we consider the case that the first task (i.e., A) is a worst-case function computation task (and that Π is a function (i.e., solving Π means computing Π)).

Definition 1.4 (sample-aided reductions): *Let $\ell, s : \mathbb{N} \to \mathbb{N}$, and suppose that M is an oracle machine that, on input $x \in \{0,1\}^n$, obtains a sequence of $s = s(n)$ pairs of the form $(r, v) \in \{0,1\}^{n+\ell(n)}$ as an auxiliary input. We say that M is an* sample-aided reduction of solving Π in the worst-case to the task $\mathtt{T}$ *if, for every procedure P that performs the task $\mathtt{T}$, it holds that*

$$\mathrm{Pr}_{r_1,\dots,r_s \in \{0,1\}^n}\left[\begin{array}{l}\forall x \in \{0,1\}^n \\ \qquad \Pr\left[M^P(x;(r_1,\Pi(r_1)),\dots,(r_s,\Pi(r_s))) = \Pi(x)\right] \geq \frac{2}{3}\end{array}\right] > \frac{2}{3} \qquad (1)$$

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

As stated upfront, a sample-aided reduction implies an ordinary non-uniform reduction. In fact, many known non-uniform reductions (e.g., [29]) are actually sample-aided reductions. Furthermore, coupled with a suitable downwards self-reduction for Π, a sample-aided reduction of solving Π in the worst-case to solving Π' on the average (resp., in the rare-case) implies a corresponding standard reduction (of worst-case to average-case (resp., to rare-case)).

Specifically, letting Π_n denote the restriction of Π to n-bit instances, suppose that solving Π_n on the worst-case reduces to solving Π'_n on the average, when provided a sample of solved instances of Π_n, and that we have an average-case solver for Π'. Then, using the downwards self-reduction of Π, a sample of solved instances of Π_n can be generated by employing the downwards reduction of Π_n to Π_{n-1} and using the worst-case to average-case reduction of Π_{n-1} (to Π'_{n-1}), which in turn requires a sample of solved instances of Π_{n-1}. Similarly, generating a sample of solved instances for Π_{n-1} is reduced to generating a sample of solve instances for Π_{n-2}, and ditto for Π_{n-i} via Π_{n-i-1}. We stress that the size of the sample remains unchanged at all levels: a sample of solved instances for Π_{n-i} allows for solving all instances of Π_{n-i} (by using the average-case solver of Π'_{n-i}).

The actual process of generating samples of solved instances goes in the other direction; that is, going from $j = 1$ to $j = n-1$, we use the sample of solved instances of Π_j in order to generate a sample of solved instances of Π_{j+1} (while using the sample-aided reduction of solving Π_{j+1} on the worst-case to solving Π'_{j+1} on the average-case). At the end, we solve the instance of Π_n given to us, by using the sample of solved instances of Π_n (while using the sample-aided reduction of solving Π_m on the worst-case to solving Π'_n on the average-case). Recall that, by the hypothesis, we have an average-case solver for Π', and that we can generate solved instances of Π_1 by ourselves (at the beginning of this process (i.e., for $j = 1$)).

Using the foregoing methodology we establish Theorem 4.4, which provides 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 Dtime(p'), where $p'(n) = p(n) \cdot n^{1+o(1)}$. The proof of Theorem 1.3, which asserts a non-uniform worst-case to rare-case reduction for $\mathcal{C}^{(p)}$, does not use this methodology; it rather uses the straightforward emulation of sample-aided reductions by non-uniform reductions.

We mention that in our follow-up [17] we have extended the notion of sample-aided reductions, allowing the $s(n)$ samples to be drawn from arbitrary distributions and be possibly dependent on one another. We believe that the notion of sample-aided reductions is of independent interest.

1.4 Worst-case to Average-case reduction for uniform $\mathcal{AC}^0[2]$

One may view our classes of counting problems as classes of (adequately uniform) circuits that consist of a top threshold gate that is fed by very small formulae (on joint variables). Given this perspective, one may ask which other classes of circuits admit worst-case to average-case reductions.

We show that adequately uniform versions of $\mathcal{AC}^0[2]$ (i.e., constant-depth polynomial-size circuits with parity gates) admit worst-case to average-case reductions that run in almost linear-time. Note that, while $\mathcal{AC}^0[2]$ cannot count, these classes contain "seemingly hard problems in $\mathcal{P}$" (e.g., the t-`CLIQUE` problem for n-vertex graphs can be expressed as a highly uniform DNF with n^t terms (each depending on $\binom{t}{2}$ variables)).

Specifically, letting $\mathcal{AC}^0_d[2]$ denote the class of decision problems regarding n-long inputs and the question of whether such an input satisfies some (adequately uniform) poly(n)-size circuit of depth d, we show an almost linear-time reduction of solving problems in $\mathcal{AC}^0_d[2]$ on the worst-case to solving problems in $\mathcal{AC}^0_{O(d)}[2]$ on 90% of the instances, where the constant in the O-notation is universal and small (but larger than 2).

The foregoing reduction uses some of the ideas that are used in the proof of Theorem 1.1. Again, the pivot is an arithmetic problem which admits a worst-case to average-case reduction, and the point is reducing the Boolean problem to it and reducing it back to a Boolean problem.

For the first reduction we use the approximation method of Razborov [25] and Smolensky [28]. Specifically, we obtain a poly(n)-size depth $O(d)$ Arithmetic circuit, over a sufficiently large extension field of GF(2), so that the standard worst-case to average-case reduction can be used; that is, the size of the extension field should exceed the degree of the polynomial computed by the Arithmetic circuit, which is low (i.e., polylogarithmic).[9] The approximation error is a non-issue, since we are only interested in the value of the original circuit at a given input, and the random choices made in the approximation method can be implemented by pseudorandom (i.e., small-bias) ones.

For the backward reduction we show that (adequately uniform) $\mathcal{AC}^0[2]$-circuits can emulate the computation of the foregoing Arithmetic circuit. In doing so, we rely on the following facts:

1. The multiplication gates (in the resulting Arithmetic circuit) have logarithmic fan-in.
2. The Arithmetic circuit is defined for an extension field of GF(2), and the size of this field is only polylogarithmic.
3. The class of Boolean circuits (i.e., $\mathcal{AC}^0[2]$) allows for parity gates of unbounded arity.

Combining Facts 1 and 2, each multiplication gate in the arithmetic circuit can be emulated by a constant-depth circuit of size $n^{o(1)}$. Facts 2 and 3 are used for emulating the addition gates. Finally, we let the circuit obtain an additional input, and output the corresponding bit in the Hadamard encoding of the output of the arithmetic circuit. This allows us to handle a constant error rate (rather than an error rate that is inversely related to the length of the representation of field elements). For more details, see Appendix A.2.

1.5 On average-case randomized algorithms

Our randomized of solving problem Π on the worst-case to solving problem Π' on the average-case (or rare-case) are naturally interpreted as implying a randomized procedure for solving Π that succeeded (with probability at least 1/3) on each instance of Π, when *given oracle access to any deterministic solver that is correct on a specified fraction of the instances of* Π'. Obviously, the same guaranteed holds when using any randomized procedure that for the same fraction of instances of Π' answers correctly with probability at least 1/3. (Indeed, error reduction may be applied both to the randomized solver of Π' and to the randomized reduction of Π to Π'; in fact, employing error reduction is typically essential before using the randomized procedure for Π' in order to derive a randomized procedure for Π.)[10]

In some of our intermediate results, it will be beneficial to use a seemingly relaxed notion of an average-case randomized solver. Specifically, we say that a randomized algorithm A solving P has success rate $\rho : \mathbb{N} \to [0,1]$ if *the probability that A solves a random n-bit instance of P is at least $\rho(n)$, where the probability is over the uniform choice of an n-bit input and the internal coin tosses of A.* Clearly, a randomized procedure that solves P on a ρ fraction of the instances can be converted to an (almost as efficient) algorithm for P that has error rate $(1 - o(1)) \cdot \rho$, by first reducing the error probability of A to $o(1)$. We observe that the converse holds too (essentially). Actually, we prove a stronger result.

[9]Indeed, the original goal of the approximation method in to obtain low degree approximations of $\mathcal{AC}^0[2]$ circuits.
[10]This is the case since typically the randomized procedure makes several queries to Π'.

Proposition 1.5 (from randomized to deterministic error rate): *Let A be a randomized procedure that solves P with error rate ρ such that $\rho(n) > 2^{-n/3}$. Then, there exists a randomized algorithm of about the same complexity as A that, on input 1^n, with probability $1 - 2^{-n/4}$, outputs a Boolean circuit that solves P on an $(1 - o(1)) \cdot \rho(n)$ fraction of the instances of length n.*

Proof: Letting t denote the running time of A, consider the corresponding residual deterministic algorithm A' that, on input (x, r) such that $|r| = t(|x|)$, invokes A on input x and coins r. Our circuit-generating algorithm selects a pairwise-independent ("hash") function $h : \{0,1\}^n \to \{0,1\}^{t(n)}$ (cf. [13, Apdx. D.2]), and outputs the circuit C_h such that $C_h(x) = A'(x, h(x))$. Using Chebyshev's inequality[11], it follows that, with probability at least $1 - \epsilon^{-2} \cdot 2^{-n/3}$ over the choice of h, the success rate of C_h is at least $(1 - \epsilon) \cdot \rho(n)$. Using $\epsilon = 2^{-n/24}$, the claim follows. ■

1.6 Terminology and organization

For a function $f : \mathbb{N} \to \mathbb{N}$, we often use $\exp(f(n))$ to denote $2^{O(f(n))}$. When we neglect to mention the base of a logarithmic function, the reader may assume it is 2. Integrality issues are often ignored. For example, $\log n$ may mean $\lceil \log_2 n \rceil$.

Throughout the paper, n denotes the length of the input (either in terms of bit or in terms of sequences over larger alphabet). For a set $S = \{i_1, ..., i_s\} \subseteq [n]$ such that $i_1 < i_2 < \cdots < i_s$ and $x = x_1 x_2 \cdots x_n \in \Sigma^n$, we denote by $x_S = x_{i_1} x_{i_2} \cdots x_{i_s}$ the projection of x at the coordinates S.

Almost linear functions. The notion of "almost linear" functions is given a variety of interpretations, in the literature, 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 maximum $f'(n) = (\log n)/(\log \log n)^{\omega(1)}$ (which means that $f(n) \leq n^{1+(\log \log n)^{-\omega(1)}}$) and at the very minimum $f'(n) = \widetilde{O}(\log \log n)$ (which means that $f(n) \leq (\log n)^{(\log \log \log n)^{O(1)}} \cdot n$). In these cases, we shall consider $\exp(f'(n)) = \mathrm{poly}(f(n)/n)$ to be *small*, and $\exp(-f'(n)) = \mathrm{poly}(n/f(n))$ to be *noticeable*.

Organization. In Section 2 we define the classes of counting problems that will be studied in this work. In Section 3 we present the proof of Theorem 1.1; our presentation proceeds in two steps: First, in Section 3.1, we prove a weaker result (which refers to an error rate that is noticeable but

[11] For each $x \in \{0,1\}^n$, define a 0-1 random variable $\zeta_x = \zeta_x(h)$ such that $\zeta_x(h) = 1$ if and only if $C_h(x)$ is correct. Then, $\mu_x \stackrel{\text{def}}{=} \mathrm{E}[\zeta_x]$ equals the probability that $A(x)$ is correct, and $S \stackrel{\text{def}}{=} \sum_{x \in \{0,1\}^n} \mu_x \geq \rho(n) \cdot 2^n$. Using Chebyshev's inequality, we get

$$\begin{aligned}
\Pr\left[\left|\sum_{x\in\{0,1\}^n} \zeta_x - S\right| \geq 2^n \cdot \epsilon \cdot \rho(n)\right] &\leq \frac{\mathrm{Var}\left[\sum_{x\in\{0,1\}^n} \zeta_x\right]}{(2^n \cdot \epsilon \cdot \rho(n))^2} \\
&= \frac{\sum_{x\in\{0,1\}^n} \mathrm{Var}[\zeta_x]}{2^{2n}\epsilon^2\rho(n)^2} \\
&< \frac{1}{2^n\epsilon^2\rho(n)^2}
\end{aligned}$$

where the equality is due to the pairwise-independence of the ζ_x's. Using $\rho(n) > 2^{-n/3}$, we obtain the desired bound (of $2^{-n/3}/\epsilon^2$).

tends to zero), and later (in Section 3.2) we derive the original claim. The proof of Theorem 1.3 is presented in Section 4.1, and in Section 4.2 we prove the incomparable result stated in Theorem 4.4.

2 Counting local patterns: defining the class of counting problems

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.

An archetypical example is the problem of counting t-cliques, for some fixed $t \in \mathbb{N}$, where the local conditions correspond to all t-subsets of the vertex set, and each local condition mandates that the corresponding vertices are adjacent in the graph. Hence, for an n-vertex input graph, represented by its adjacency matrix $X = (x_{i,j})$, we have local conditions of the form $\bigwedge_{j<k\in[t]} x_{i_j,i_k}$ for every t-subset $\{i_1, ..., i_t\} \in \binom{[n]}{t}$. Hence, counting t-cliques in a graph represented by X corresponds to counting the number of local conditions that are satisfied by X.

In general, for relatively small and efficiently described subsets $S_1^{(n)}, ..., S_{\text{poly}(n)}^{(n)} \subset [n]$ and formulae $\phi_1^{(n)}, ..., \phi_{\text{poly}(n)}^{(n)}$, we consider the problem of counting, on input $x \in \{0,1\}^n$, the number of subsets $S_i^{(n)}$ such that $\phi_i^{(n)}(x_{S_i}) = 1$. Hence, the $S_i^{(n)}$'s represents portions of (locations in) the input and the $\phi_i^{(n)}$'s represents the (local) conditions imposed on them. (Indeed, this class of counting problems is related to the class of locally characterizable sets defined by us in [16]; see more details in Appendix A.1.)

Key issues that were intentionally left unspecified in the foregoing description are what is the size of the subsets $S_i^{(n)}$'s and formulae $\phi_i^{(n)}$'s and what is meant by efficiently describing them. Starting from the latter question, we postulate that all subsets (i.e., $S_1^{(n)}, ..., S_{\text{poly}(n)}^{(n)} \subset [n]$) are described by an explicit sequence of formulae $\sigma_n : \{0,1\}^{\ell(n)} \to \binom{[n]}{m(n)}$, where $\ell(n) = O(\log n)$, such that $\sigma_n(i) = S_i^{(n)}$ for every $i \in [2^{\ell(n)}] \equiv \{0,1\}^{\ell(n)}$. Indeed, $2^{\ell(n)}$ equals the number of local conditions imposed on inputs of length n, and $m(n)$ is the number of locations considered in each local condition. Likewise, all ("local condition") formulae (i.e., $\phi_1^{(n)}, ..., \phi_{\text{poly}(n)}^{(n)}$) are described by an explicit sequence of formulae $\phi_n : \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}$ such that $\phi_n(i, z) = \phi_i^{(n)}(z)$ for every $i \in [2^{\ell(n)}] \equiv \{0,1\}^{\ell(n)}$ and $z \in \{0,1\}^{m(n)}$.

Still, we need to specify the function $m : \mathbb{N} \to \mathbb{N}$, a bound on the size of formulae σ_n and ϕ_n, and be more concrete regarding what we mean by "being explicit" (i.e., the complexity of constructing σ_n and ϕ_n). We actually prefer to leave these issues semi-open and only state minimal conditions regarding these aspects. For sure $m(n)$ and the formulae sizes should be between $n^{o(1)}$ and $\text{poly}(\log n)$, and, by default, *explicit* means that the object can be generated in time that is polynomial in its size.

In addition, since the bounds on the formulae sizes will be used to define complexity classes, we need to define families of bounds that are closed under some operations (so that the complexity classes may be closed under some natural algorithmic compositions). This will be done in Section 2.1, and once done we shall spell out the corresponding definition of the counting classes (in Section 2.2).

2.1 Admissible bounding functions

Our main complexity measure will be formula size (i.e., bounds on the sizes of the formulae σ_n and ϕ_n). Equivalently, we may just discuss formula depth, which is logarithmically related to their size. Recalling our minimal requirements reagrding the sizes of the formulae σ_n and ϕ_n, we focus on depth bounds that are between $o(\log n)$ and $\Omega(\log\log n)$.

Since our reductions may increase the depth of the formulae by a poly-logarithmic factor (i.e., go from $d(n)$ to $\widetilde{O}(d(n))$), we consider classes of (depth) bounding functions that are closed under multiplication by such a factor. Although this choice seems opportunistic, it is quite natural in the context of complexity measures: One typically allows closure under constant factors, and closure under polylogarithmic factors is but the next step. Lastly, we make the standard simplifying assumption that the bounding function is monotonically non-decreasing. Hence, we obtain the following admissible classes of bounding functions.

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

1. Minimal condition (i.e., d is neither too big nor too small): *The function d is sub-logarithmic and at least double-logarithmic; that is, $d(n) \in [\Omega(\log\log n), o(\log n)]$.*
2. Closure: *The function d' such that $d'(n) = \widetilde{O}(d(n))$ is in $\mathcal{D}$.*

   (Given that $d(n) = \Omega(\log\log n)$, it follows that $d''(n) = d(n) + \log\log n$ is also in $\mathcal{D}$).
3. Non-decreasing: $d(n+1) \geq d(n)$.

Examples of admissible classes of functions include

$$\begin{aligned}
\mathcal{D}_1 &= \{d \in \mathcal{L} : d(n) \leq \widetilde{O}(\log\log n)\} \\
\mathcal{D}_2 &= \{d \in \mathcal{L} : d(n) \leq \mathrm{poly}(\log\log n)\} \\
\mathcal{D}_3 &= \{d \in \mathcal{L} : d(n) \leq O(\log n)^c\} \ \text{ for any } c \in (0,1) \\
\mathcal{D}_4 &= \left\{d \in \mathcal{L} : d(n) \leq \frac{\log n}{(\log\log n)^{\omega(1)}}\right\}
\end{aligned}$$

where $\mathcal{L}$ denotes the class of functions that are at least double-logarithmic, and the unspecified constants (in the O-notation) allow for ignoring finitely many n's.

Definition 2.2 (a corresponding notion of almost-linear functions): *For a fixed admissible class $\mathcal{D}$, we say that the function $f : \mathbb{N} \to \mathbb{N}$ is* almost linear *if $f(n) \leq \exp(d(n)) \cdot n$ for some $d \in \mathcal{D}$.*

Indeed, this is shorthand for $\mathcal{D}$-almost-linear. Note that, for any admissible class $\mathcal{D}$, saying that f is $\mathcal{D}$-almost-linear means that $f(n) = n^{1+o(1)}$, although it may be that $f(n) > \widetilde{O}(n)$. (Even saying that f is $\mathcal{D}_1$-almost-linear only means that $f(n) \leq \exp(\widetilde{O}(\log\log n)) \cdot n$, which means that $f(n)/n$ is quasi-poly-logarithmic.)

2.2 The counting classes

Turning back to the Boolean formulae $\phi_n : \{0,1\}^{\ell(n)+m(n)} \to \{0,1\}$ and $\sigma_n : \{0,1\}^{\ell(n)} \to [n]^{m(n)}$, note that their size is upper-bounded by the time that it takes to construct them. Recalling that we may assume, without loss of generality, that the depth of a formula is logarithmic in its size, we make the running time of the constructing algorithm our main parameter. Hence, for any admissible class $\mathcal{D}$, which provides bounds for the depth of formuale, we shall consider algorithms that on input n run in time that is exponential in some function in $\mathcal{D}$.

For sake of convenience, we also replace $\sigma_n : \{0,1\}^{\ell(n)} \to [n]^{m(n)}$ by $\sigma_{n,1}, ..., \sigma_{n,m(n)} : \{0,1\}^{\ell(n)} \to [n]$, where $\sigma_{n,j}(i)$ specifies the j^{th} element in $\sigma_n(i)$ (which is the i^{th} subset specified by σ_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 (and to a logarithmic function ℓ).

Definition 2.3 (counting local patterns): *Let $\mathcal{D}$ be a set of admissible functions. For every $d \in \mathcal{D}$ 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* $\exp(d(n))$*-time and outputs Boolean formulae $\phi_n : \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}$ and $\sigma_{n,1}, ..., \sigma_{n,m(n)} : \{0,1\}^{\ell(n)} \to [n]$. 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*

$$\Phi_x(w) \stackrel{\text{def}}{=} \phi_n(w, x_{\sigma_{n,1}(w)}, ..., x_{\sigma_{n,m(n)}(w)}) \tag{2}$$

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

Hence, each problem in $\mathcal{C}_{\ell,\mathcal{D}}$ is specified by $2^{\ell(n)}$ local conditions reagrding the n-bit input. Each local condition refers to $m(n)$ locations in the input, and the predicate $\Phi_x(w)$ indicates whether or not the local condition associated with the index $w \in \{0,1\}^{\ell(n)} \equiv [2^{\ell(n)}]$ is satisfied by the input x. The corresponding counting problem is to compute the number of local conditions that are satisfies by x (i.e., the number of w's such that $\Phi_x(w) = 1$).

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 $2^{\ell(n)} \cdot \exp(d(n)) = 2^{(1+o(1))\cdot\ell(n)}$ on a direct access machine. We mention that the doubly-efficient interactive proof systems presented by us in [16] apply to these counting problems, provided that the verifier is allowed to run in time $\exp(d(n)) \cdot n = n^{1+o(1)}$, for some $d \in \mathcal{D}$.

A simplified form. In many natural cases, the local condition may depends only on the values of the bits in the input, and be oblivious of the index of the condition; that is, $\phi_n(w, z)$ may be oblivious of w. In this case, we may write $\phi_n : \{0,1\}^{m(n)} \to \{0,1\}$ (insread of $\phi_n : \{0,1\}^{\ell(n)+m(n)} \to \{0,1\}$), and have $\Phi_x(w) = \phi_n(x_{\sigma_{n,1}(w)}, ..., x_{\sigma_{n,m(n)}(w)})$. This holds for the problem of counting t-cliques, discussed in the beginning of this section. Likewise, this holds for the problem of counting t-tuples of integers in the input sequence $x = (x_1, ..., x_n) \in [-b, b]^n$ that sum-up to zero. Specifically, with some abuse of notation, this problem 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 $\phi_n : [-b, b]^t \to \{0,1\}$ such that $\phi_n(z_1, ..., z_t) = 1$ if and only if $\sum_{j\in[t]} z_j = 0$ (and $\sigma_{n,j} : \{0,1\}^{t \log n} \to [n]$ such that $\sigma_{n,j}(i_1, ..., i_t) = i_j$ 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. 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 $0.75+\epsilon$ fraction of the domain, where ϵ is any positive constant.*

Recall that, for every $d \in \mathcal{D}$, the function $n \mapsto \exp(d(n)) \cdot n = n^{1+o(1)}$ is considered *almost linear*. Actually, we may replace ϵ by $\exp(-d(n)) = o(1)$.

Outline of the proof. 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 field of size $\exp(d(n))$, 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 $d \in \mathcal{D}$ and $\ell, m : \mathbb{N} \to \mathbb{N}$ as in Definition 2.3, let A' be an algorithm that, on input n, runs for $\exp(d(n))$-time and outputs an Arithmetic formula $\widehat{\phi}_n : \mathcal{F}^{\ell(n)} \times \mathcal{F}^{m(n)} \to \mathcal{F}$ and Boolean formulae $\sigma_{n,1}, ..., \sigma_{n,m(n)} : \{0,1\}^{\ell(n)} \to [n]$, where $\mathcal{F} \supseteq \{0,1\}$ is a generic finite field. Fixing a finite field $\mathcal{F}$, the* evaluation problem associated with A' and $\mathcal{F}$, *denoted $\mathtt{EV}_{A',\mathcal{F}}$, consists of computing the function $\widehat{\Phi}_{A'} : \mathcal{F}^n \to \mathcal{F}$ such that*

$$\widehat{\Phi}_{A'}(X_1, ..., X_n) \stackrel{\text{def}}{=} \sum_{w \in \{0,1\}^{\ell(n)}} \widehat{\phi}_n(w, X_{i_{n,w}^{(1)}}, ..., X_{i_{n,w}^{(m(n))}}), \tag{3}$$

where $i_{n,w}^{(j)} = \sigma_{n,j}(w)$.

Indeed, as in Eq. (3), we shall often abuse notation and view binary strings as sequences over $\mathcal{F}$ (e.g., in Eq. (3) $w \in \{0,1\}^{\ell(n)}$ is viewed as an $\ell(n)$-long sequence 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 $\widehat{\phi}_n(\xi(w_1), ..., \xi(w_{\ell(n)}), X_{i_{n,w}^{(1)}}, ..., X_{i_{n,w}^{(m(n))}})$ instead of $\widehat{\phi}_n(w, X_{i_{n,w}^{(1)}}, ..., X_{i_{n,w}^{(m(n))}})$.

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

Slightly detailing the proof outline provided above, we note that it amounts to composing three basic reductions.

1. A (worst-case to worst-case) reduction of the Boolean counting problem to a corresponding Arithmetic evaluation problem.
2. A worst-case to average-case reduction of the Arithmetic evaluation problem to itself.
3. An average-case to average-case reduction of the Arithmetic evaluation problem to a new Boolean counting problem.

All reductions are performed in almost-linear time and preserve the relevant parameters (e.g., ℓ and $d \in \mathcal{D}$). These reduction are presented in Section 3.1.1, and the reduction deserving most attention is the third one (i.e., reducing the arithmetic problem to a Boolean problem), where the point is that this reduction is performed in the context of average-case analysis. Later, in Section 3.1.2, we shall combine these reductions and derive a weak version of Theorem 3.1.

3.1.1 Three basic reductions

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 in our context (of counting and summation problems) 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. We note that, for the sake of the worst-case to average-case reduction (for the Arithmetic evaluation problem), which will be performed in the next step, it is important to keep 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 $d \in \mathcal{D}$) 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 $\exp(d(n))$, and the functions ℓ, m and d 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 $s = \exp(d(n))$. 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 (and $s = n^{o(1)}$).

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 $\sigma_{n,j}$'s, by first invoking algorithm A, and then proceeding as outlined above. Defining $\widehat{\Phi}_{A'}$ as in Eq. (3), for every $x = x_1 \cdots x_n \in \{0,1\}^n \subset \mathcal{F}^n$, it holds that

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

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 $D = \mathrm{poly}(s) = \exp(d(n))$, the furthermore clause follows. ■

The second step: A worst-case to average-case reduction. The fact that the arithmetic evaluation problem (i.e., $\mathtt{EV}_{A',\mathcal{F}}$) admits a worst-case to average-case reduction is a special case of the well-known fact that such a reduction holds for any polynomial provided that the field is large enough (i.e., significantly larger than the degree of the polynomial). The latter fact is rooted in Lipton's work [22].

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 smaller than $D < |\mathcal{F}|/3$. 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 the evaluation procedure is expected to answer correctly at least an $8/9$ fraction of these queries. It follows that, with probability at least $2/3$, the evaluation procedure answers correctly on at least $2D$ of the queried points. Using the Berlekamp–Welch algorithm, we reconstruct the unique degree $D-1$ polynomial that agrees with at least $2D$ of the answers, and return its free-term (i.e., its value at 0, which corresponds to $P(x)$).[12] Hence, the recovered value is correct with probability at least $2/3$. ■

[12]This polynomial is unique since we cannot have two different polnomials of degree $D-1$ that each agree with $2D$ of the $3D$ points.

Towards the third step: A difficulty. 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 $p > 2^{\ell(n)}$, and the reduction (provided by Proposition 3.3) does not specify such a prime (i.e., Proposition 3.3 merely states that the reduction is valid for any sufficienyly large prime). This becomes a problem when we want to reduce the Arithmetic problem (which refers to an unspecified large field) to a Boolean problem.

Of course, an adequate 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 $\exp(d(n))$-time algorithm for generating primes that are larger than $2^{\ell(n)}$, but such an algorithm is not known (since $\exp(d(n)) = n^{o(1)}$ whereas $2^{\ell(n)} = n^{\Omega(1)}$).

We resolve this difficulty by considering an extension of Definition 2.3 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. Actually, it suffices to let the algorithm use this auxiliary information for the construction of the local conditions only (whereas the location formulae $\sigma_{n,j}$'s may remain oblivious of it).

Definition 3.5 (Definition 2.3, extended): *For $d \in \mathcal{D}$ and $\ell, m : \mathbb{N} \to \mathbb{N}$ as in Definition 2.3, let B be an algorithm that, on input (n, z) such $|z| \leq \exp(d(n))$, runs for $\exp(d(n))$-time and outputs Boolean formulae $\phi_{n,z} : \{0,1\}^{\ell(n)} \times \{0,1\}^{m(n)} \to \{0,1\}$ and $\sigma_{n,1}, ..., \sigma_{n,m(n)} : \{0,1\}^{\ell(n)} \to [n]$. The* counting problem associated with *$B(n, z)$ consists of counting, on input $x \in \{0,1\}^n$, the number of $w \in \{0,1\}^{\ell(n)}$ such that $\phi_{n,z}(w, x_{\sigma_{n,1}(w)}, ..., x_{\sigma_{n,m(n)}(w)}) = 1$.*

With this definition in place, we can reduce the Arithmetic problem $\mathtt{EV}_{A',\mathrm{GF}(p)}$, which refers to n-long sequences over $\mathrm{GF}(p)$, to a counting problem associated with $B(n, p)$, where B is as in Definition 3.5. We warn, however, that the following result reduces solving the arithmetic problem on 8/9 fraction on the instances to solving the Boolean problem on an $1 - \Omega(1/\log n)$ fraction of the instances; that is, the following result refers only to potential solvers (of the Boolean ptroblem) that err on a vanishing yet noticeable fraction of the domain.

Proposition 3.6 (reducing average-case Arithmetic evaluation to average-case Boolean counting): *Let A' be as in Definition 3.2, and $\widehat{\Phi}_{A'}$ be the corresponding polynomial. Then, for every prime $p \leq \mathrm{poly}(n)$, the problem of evaluating $\widehat{\Phi}_{A'}$ on at least a 8/9 fraction of the domain $\mathrm{GF}(p)^n$ is randomly reducible in almost-linear time to solving the counting problem associated with $B(n'', p)$ on at least $1 - (1/O(\log p))$ fraction of the domain $\{0,1\}^{n''}$, where B is as in Definition 3.5 and $n'' = \widetilde{O}(n)$. Furthermore, if ℓ', d' and m'* (resp., *ℓ'', d''* and *m'') are the functions used in the Arithmetic* (resp., Boolean) *problem, then $\ell'' = \ell'$, $d''(\widetilde{O}(n)) \leq \mathrm{poly}(\log\log p) \cdot d'(n)$, and $m''(\widetilde{O}(n)) = O(\log n) \cdot m'(n)$.*

Note that we are only guaranteed that $d''(\widetilde{O}(n)) \leq \mathrm{poly}(\log\log n) \cdot d'(n)$, which means that the Boolean counting problem is in $\mathcal{C}_{\ell,\mathcal{D}}$ only if $\mathcal{D}$ has a stronger closure property than postulated in Definition 2.1 (i.e., we need closure under multiplication by $\mathrm{poly}(\log\log n)$, whereas Definition 2.1

only postulates closure under multiplication by $\mathrm{poly}(\log d'(n))$). This drawback will be removed at a later stage (i.e., in Section 3.2), when dealing with the more acute drawback of referring only to vanishing error rates.

Proof: The basic idea is to let the Boolean formula emulate the computation of the Arithmetic formula $\widehat{\phi}_n$ that appears in Eq. (3). This will be done by using small Boolean formulae $\phi'_{n,p}$ that emulate the operations of the field $\mathrm{GF}(p)$, which are performed in $\widehat{\phi}_n$. This suggestion raises a couple of issues.

1. A straightforward representing of elements of $\mathrm{GF}(p)$ by $\lceil\log_2 p\rceil$-bit long strings is good enough for worst-case to worst-case reductions, but in the context of average-case to average-case reductions this may fail badly. The problem is that a constant fraction of the $\lceil\log_2 p\rceil$-bit long strings may have no preimage under the straightforward encoding, which means that the straightforward encoding may map n-long sequences of field elements to an $\exp(-n)$ fraction of the $n \cdot \lceil\log_2 p\rceil$-bit long strings.

   Our solution is to use a randomized encoding of the elements of $\mathrm{GF}(p)$; specifically, $e \in \mathrm{GF}(p)$ will be encoded by a randomly selected element of $\{e' \in [2^{O(\log(pn))}] : e' \equiv e \pmod p\}$.

2. The Boolean formula outputs a single bit, whereas the Arithmetic formula outputs an element of $\mathrm{GF}(p)$. Hence, in order to emulate a computation of the arithmetic formula, we need to combine $\log_2 p$ Boolean results. Furthermore, we need to compute the sum (in $\mathrm{GF}(p)$) ef many evaluations of an arithmetic formula, given a procedure that counts the number of times that a Boolean formula evaluates to 1.

   Our solution is to compute the sum of the $\mathrm{GF}(p)$-elements over the integers, and do so by computing the sum of each bit in the canonical representation of these elements (rather than in their randomized encodings per Item 1). The Boolean formula that we constructs starts by mapping the random representation to the canonical one, emulates the arithmetic formula, and then outputs the desired bit, which is specified as part of its input. The corresponding answer to the Boolean counting problem will give us the sums of the value of this bit over the $\mathrm{poly}(n)$-many evaluations of the Arithmetic formula.

   The small error rate that we can tolerate is due to the fact that we need to get the correct values to all $\log_2 p$ queries, which correspond to all bits in the binary expansion of the canonical representation of the elements of $\mathrm{GF}(p)$. (This can be improved upon using an additional idea, which we postpone to Section 3.2.)

Details follow.

We shall represent each element of $\mathcal{F} = \mathrm{GF}(p)$ by an $O(\log n)$-bit long string. Actually, each element of $\mathrm{GF}(p)$ will have $\mathrm{poly}(n)$-many possible representation by bit strings of length $\log_2 p + O(\log n) = O(\log n)$. When reducing the Arithmetic evaluation problem to a Boolean counting problem, we shall select at random one of these representations; that is, for each input symbol $e \in \mathrm{GF}(p)$, we shall select at random $e' \in [\mathrm{poly}(n)]$ such that $e' \equiv e \pmod p$. 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).

We construct a Boolean formula $\phi'_{n,p}$, which gets an input a sequence of $m'(n)$ redundant representations of elements in $\mathrm{GF}(p)$. It first map each such representation to the canonical representation, and then emulate the computation of the Arithmetic circuit (over $\mathrm{GF}(p)$) using the canonical representations all along. Specifically, the canonical representation of $e \in \mathrm{GF}(p)$ is the (zero-padded) binary expansion of e, and the other representations correspond to the binary expansions of $e + i \cdot p$ for each $i \in [\mathrm{poly}(n)]$. Hence, the output will be $t = \lceil \log_2 p \rceil$ 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, the Boolean formula $\phi'_{n,p}$ emulates the computation of the corresponding $\widehat{\phi}_n$ (over $\mathrm{GF}(p)$). Specifically, each arithmetic gate will be replaced by an $\mathcal{NC}$ circuit that emulates the corresponding field operation.[13] Note that these gate-emulation circuits operate on (pairs of) inputs of length $\log_2 |\mathrm{GF}(p)| = \log_2 p$, and so their depth is $\mathrm{poly}(\log\log p)$, 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 $\sigma'_{n,1}, ..., \sigma'_{n,m(n)} : \{0,1\}^{\ell'} \to [n]$. Then, each $\sigma'_{n,j}$ determines the index of a field element in the input to $\widehat{\phi}_n$, and so it should be replaced by functions $\sigma''_{n,(t'-1)j+1}, ..., \sigma''_{n,t'j} : \{0,1\}^{\ell'} \to [t'n]$ that determine the corresponding bits in the input to $\phi'_{n,p}$ (i.e., $\sigma''_{n,(t'-1)j+k}(w) = (t'-1) \cdot \sigma'_{n,j}(w) + k$ for $k \in [t']$).

Note that the formula $\phi'_{n,p}$ outputs $t = \log_2 p$ 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, $\phi''_{n,p}(w, (\cdot, i))$ equals the ith bit of $\phi'_{n,p}(w, \cdot)$. Hence, on input $(y, i) \in \{0,1\}^{n \cdot O(\log n)} \times [t]$, where $y = (y_1, ..., y_n)$ is a redudent representation of a sequence of n elements in $\mathrm{GF}(p)$, we consider the counting problem that corresponds to $\Phi''_{y,i}$ such that $\Phi''_{y,i}(w) = \phi''_{n,p}(w, (y,i)_{\sigma''_n(w)})$, where σ''_n denote the sequence of $\sigma''_{n,j}$'s (augmented by functions that indicate the bit positions of i in the input (y, i)).

In summary, we reduce the evaluation of of $\widehat{\Phi}_{A'}$ on input $x \in \mathrm{GF}(p)^n$ 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 $x = (x_1, ..., x_n) \in \mathrm{GF}(p)^n$ is obtained as follows.

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

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

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

The key observation is that

$$\sum_{w \in \{0,1\}^{\ell'(n)}} \widehat{\Phi}_{A'}(x_1, ..., x_n) \quad = \quad \sum_{w \in \{0,1\}^{\ell'(n)}} \widehat{\phi}_n(w, x_{i^{(1)}_{n,w}}, ..., x_{i^{(m(n))}_{n,w}})$$

[13] Recall that integer arithmetics is in $\mathcal{NC}$; see, e.g., [23, Lect. 30].

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

Hence, $\mathtt{EV}_{A',\mathrm{GF}(p)}(x) = \sum_{i\in[t]} 2^{i-1} \cdot \#_{B(n,p)}(y,i)$, where B is the algorithm that (on input n and p) generates $\phi''_{n,p}$ (and σ_n).

Note that, when given a uniformly distributed $x \in \mathrm{GF}(p)^n$, the ith query made by our reduction is almost uniformly distributed in $S_i \stackrel{\mathrm{def}}{=} \{(y,i) : y \in \{0,1\}^{n\cdot t'}\}$, where the small deviation arises from the fact that some elements in $\mathrm{GF}(p)$ have $\lfloor 2^{t'}/p \rfloor$ representations, whereas others have $\lceil 2^{t'}/p \rceil$ representations.

Now, suppose that we are given an algorithm for the Boolean counting problem associated with $B(n,p)$ such that for every $i \in [t]$ the algorithm errs on an η_i fraction of the inputs in S_i. Then, the probability that our reduction solves $\mathtt{EV}_{A',\mathrm{GF}(p)}$ on a random $x \in \mathrm{GF}(p)^n$ is at least $1-\sum_{i\in[t]} \eta_i - n/p$, where the n/p term is due to the aforementioned deviation. Letting $\eta = \mathrm{E}_{i\in[t]}[\eta_i]$ denote the fraction of the instances $(\{0,1\}^{n\cdot t'} \times [t])$ on which the counting algorithm errs, it follows that our reduction has error rate at most $t \cdot \eta + n/p$.

Hence, our reduction errs with probability exceeding $1/3$ on at most an $3 \cdot (t \cdot \eta + n/p) < 4t\eta$ fraction of the instances (in $\mathrm{GF}(p)^n$). Recalling that we seek to solve $\mathtt{EV}_{A',\mathrm{GF}(p)}$ on at least a $8/9$ fraction of the inputs in $\mathrm{GF}(p)^n$, we need a counting algorithm that errs on at most an $\eta = 1/36t = 1/O(\log n)$ fraction of its inputs.

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 formal discrepancy 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 $v \neq 1^\lambda$. ■

3.1.2 An intermediate conclusion

Combining Propositions 3.3, 3.4 and 3.6, 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 3.7 (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 d such that $d(n) \geq \exp((\log\log n)^{\Omega(1)})$.

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 (worst-case) counting problem $\#_A$ is reduced 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 3.6, which yields a counting problem $\#_{B(p)}$ for some adequate algorithm B (as in Definition 3.5). Hence, we map the parameter $d' \in \mathcal{D}$ of $\widehat{\Phi}_{A'}$ to a parameter d'' of the new counting problem such that $d''(\widetilde{O}(n)) = \mathrm{poly}(\log\log n) \cdot d'(n)$. Actually, since both reduction are randomized, we first apply error reduction so that for instances that are originally solved correctly (with probability at least 2/3) are solved correctly with probability at least 0.9.

We stress that the prime p is selected, upfront, by our composed reduction; that is, it is determined prior to applying Propositions 3.3, 3.4 and 3.6. Thus, the algorithm underlying the final counting problem (i.e., B) views p as part of its input (as in Definition 3.5), whereas the algorithm in Definition 2.3 gets no such input. To bridge the gap, we include this part of B's input (i.e., p) in the input to the counting problem to which we reduce; that is, the input to the counting problem is now $(p,(y,i))$ rather than being $y' \stackrel{\mathrm{def}}{=} (y,i)$ as in the proof of Proposition 3.6. Hence, we consider the counting problem $\#_{B'}$ such that $\#_{B'}(p,y') = \#_{B(p)}(y')$.

We stress that B' is a small variation on B (rather than being B itself): This is obvious given that B gets the input (n,p), whereas B' gets the input n only. In addition, $B'(n)$ should construct small circuits that emulate $\mathrm{GF}(p)$ operations, when $p \in (2^{\ell(n)}, 2^{\ell(n)+1})$ is part of their input, whereas $B(n,p)$ constructs small circuits that emulate $\mathrm{GF}(p)$ operations for a fixed p. Such circuits are still in uniform $\mathcal{NC}$. We also augment the formula $\sigma''_{n''}$ such that p is always fed to the formula $\phi''_{n''}$ (where $\sigma''_{n''}$ and $\phi''_{n''}$ are as in the proof of Proposition 3.6). 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 $\#_{B'}(p,\cdot) \equiv \#_{B(p)}$ is solved correctly with probability $1-10\eta$. Hence, the resulting composed reduction reduces solving $\#_A$ on the worst-case to solving $\#_{B'}$ on a $1-\eta$ fraction of the instances, provided that $10\eta \leq 1/36t$, where $t = \lceil \log_2 p \rceil = \ell(n)+1$ is as in the proof of Proposition 3.6. The success probability of the entire (composed) reduction (on each input) is at least $0.9^3 > 2/3$, where one 0.9 factor arises from the choice of p, and the other two factors are due to the two randomized reductions that we use (whose error probability was reduction to 0.1 (see above)). ■

Digest. When tracing the cost of the reduction captured by Corollary 3.7, 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 in Proposition 3.6 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 in Proposition 3.6 is due to the implementation of the field's operations by Boolean formulae. Jumping ahead, we note that using the following Proposition 3.8 allows using $p = \exp(d(n))$ (rather than $p = \mathrm{poly}(n)$), which means that the depth increase is only a $\mathrm{poly}(\log d(n))$ factor.

3.2 Deriving the original conclusion

The small level of error rate that is allowed by Corollary 3.7 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 $i_{n,w}^{(j)}$'s)) is emulated by $t = \log_2 p$ invocations of a Boolean counting problem, which refers to the Boolean formula $\phi''_{n,p}$ (and to σ''_n). The key observation, made in the proof of Proposition 3.6, is that

$$\sum_{w\in\{0,1\}^{\ell(n)}} \widehat{\phi}_n(w, x_{i_{n,w}^{(1)}}, ..., x_{i_{n,w}^{(m(n))}}) \equiv \sum_{w\in\{0,1\}^{\ell(n)}} \sum_{i\in[t]} \phi''_{n,p}(w, (y,i)_{\sigma''_n(w)}) \cdot 2^{i-1} \pmod p. \tag{4}$$

In the proof of Proposition 3.6, we computed the r.h.s of Eq. (4) by counting, *separately, for each* $i \in [t]$, the number of w's that satisfy $\phi''_n(w, (y,i)_{\sigma''_n(w)})$, 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. (4) can be computed by

$$\sum_{i\in[t]} \sum_{w\in\{0,1\}^{\ell(n)}} \sum_{u\in\{0,1\}^t} \phi'''_{n,p}(w, u, (y,i)_{\sigma''_n(w)}) \tag{5}$$

such that $\phi'''_{n,p}(w, u, (y,i)_{\sigma''_n(w)}) = \phi'''_{n,p}(w, (y,i)_{\sigma''_n(w)})$ 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 $\phi'''_{n,p}(w, u, (y,i)_{\sigma''_n(w)}) = 0$ 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 of the Boolean counting problem to $O(\ell(n)/d(n))$ arithmetic problems that refer to fields of size $\exp(d(n)) = n^{o(1)}$ (rather than of size $2^{\ell(n)} = \text{poly}(n)$), where the field size should be at least $\exp(d(n))$ for the application of Proposition 3.4 (which requires the field to be larger than the degree bound). The corresponding $O(\ell(n)/d(n))$ values will be combined using Chinese Remaindering with errors (cf. [15]) so that we only incur a constant loss in the error rate (rather than a loss factor of $O(\ell(n)/d(n))$). 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).

3.2.1 Revisiting the three basic reductions

We start by presenting revised versions of the three reductions presented in Section 3.1 (i.e., Propositions 3.3, 3.4 and 3.6). In the revised versions of Propositions 3.3 and 3.6 we use a field of size $\exp(d(n))$ (rather than of size $\text{poly}(n)$). This already yields improvement in some parameters. In the revised version of Proposition 3.4 (i.e., the worst-case to average-case reduction for the arithmetic problem) we use a more refined analysis that allows to reduce the required success rate from $8/9$ to $0.5+\epsilon$, for any $\epsilon > 0$, while requiring a field size that is $O(1/\epsilon^2)$ times larger than the degree of the polynomial. The most important improvement comes from the last reduction (i.e., a revision of Propositions 3.6), which executes the emulation of Eq. (4) by Eq. (5).

Revised reduction of the Boolean problem to the Arithmetic one. For sake of simplicity, we first present an alternative to Proposition 3.3 using CRT with no errors. Recall that Proposition 3.3 (as well as the following alternative) is a deterministic reduction from solving the Boolean problem (in the worst-case) to solving the Arithmetic problem (in the worst-case). In the following alternative, we consider the same evaluation problem for several fields, while assuming that the answers obtained for these different fields are all correct. In contrast, we shall use "CRT with errors" when actually proving Theorem 3.1, which will require dealing with a situation in which only the answers obtained for 51% of these fields are correct.

Proposition 3.8 (Proposition 3.3, revised): *Solving the counting problem associated with an algorithm A and functions ℓ, m and $d \in \mathcal{D}$ is reducible in almost-linear time to $O(\ell(n)/d(n)) = o(\log n)$ evaluation problems that are all associated with the same algorithm A', and finite fields of prime cardinality at least $s = \exp(d(n))$. 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 d 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 3.10). (On the other hand, using Proposition 3.9 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 the "counting (mod p) problem $\#_A$" to the evaluation problem associated with A' and $\mathrm{GF}(p)$, where by counting (mod p) problem $\#_A$ we mean computing $\sum_{w \in \{0,1\}^{\ell(n)}} \Phi_x(w) \bmod p$. Invoking this reduction for $\tau = O(\ell(n)/d(n))$ primes p of size $\exp(d(n))$, we obtain the sum modulo each of these primes. Finally, invoking the Chinese Remainder Theorem, the claim follows provided that $\exp(d(n))^\tau > 2^{\ell(n)}$, which holds for any $\tau > \frac{\ell(n)}{O(d(n))}$. ■

Revised worst-case to average-case reduction for the Arithmetic problem. Foreseeing that it will be more convenient to perform the third reduction when referring to the notion of success rate (see Section 1.5), we also present the second reduction in these terms.

Proposition 3.9 (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. (Indeed, this move from success rate (of $0.5 + \epsilon$) to (a $0.5 + 0.5\epsilon$) fraction of correct answers is analogous to the proof of Proposition 1.5; in the current context we get it "for free".)

Using the Berlekamp–Welch algorithm, we reconstruct the unique degree D polynomial that agrees with at least $(0.5 + 0.5\epsilon) \cdot m$ of the answers, and return its value at x. (Uniqueness is guaranteed by $D < \epsilon \cdot m$.) Hence, the returned value is correct with probability at least 2/3. ■

Revised average-case reduction from the Arithmetic problem to the Boolean one. We finally get to the core of the entire revision, which is the improved reduction that takes us back from the Arithmetic world to the Boolean world. This improved reduction is based on the emulation of Eq. (4) by Eq. (5).

Proposition 3.10 (Proposition 3.6, revised): *Let A' be as in Definition 3.2 and $\widehat{\Phi}_{A'}$ be the corresponding polynomial, where ℓ', d' and m' are the corresponding functions. Then, for every prime $p \leq \mathrm{poly}(n)$, the problem of evaluating $\widehat{\Phi}_{A'} : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ with success rate at least ρ is randomly reducible in almost-linear time to solving the counting problem associated with $B(n'', p)$ with success rate at least $\rho + (1/\mathrm{poly}(n))$, where B is as in Definition 3.5 and $n'' = \exp(d'(n)) \cdot n$. Furthermore, the reduction makes a single query, and the functions corresponding to the Boolean problem are $\ell'' = \ell'$, $d''(n'') \leq \mathrm{poly}(\log\log p) \cdot d'(n)$, and $m''(n'') = O(\log n) \cdot m'(n)$.*

We shall use the setting $\rho = 0.5 + \epsilon > 0.5$. We stress that Proposition 3.10 improves over Proposition 3.6 in its preservation of the success rate (where in Proposition 3.6 solving problem of evaluating $\widehat{\Phi}_{A'}$ with success rate 8/9 was reduced to solving the counting problem associated with $B(n'', p)$ with success rate $1 - 1/O(\log n)$).

Proof: Following the proof of Proposition 3.6, we derive the same formula $\phi'_{n,p}$, and recall that the formula $\phi'_{n,p}$ outputs $t = \log_2 p$ 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 $\phi'_n(w, z)$, where $z = y_{\sigma''_n(w)}$ is a projection of the bits of the input y to the counting problem. Unlike in the proof of Proposition 3.6, we shall not view i as part of the input to the counting problem (which is fed into $\phi''_{n,p}$, 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. (2)) is such that $\Phi''_y(w, i) = \phi''_{n,p}(w, i, y_{\sigma''_n(w)})$, where σ''_n denote the sequence of $\sigma''_{n,j}$'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 $x \in \mathrm{GF}(p)^n$, which is encoded by $y \in \{0,1\}^{t'n}$, we do not wish to obtain $\sum_{w,i} \Phi''_y(w, i)$, but rather wish to obtain the related sum

$$\sum_{w \in \{0,1\}^{\ell'(n)}} \sum_{i \in [t]} 2^{i-1} \cdot \Phi''_y(w, i), \tag{6}$$

where $y \in \{0,1\}^{t'n}$ is a random representation of x. The difficulty is that Eq. (6) 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. (6) equals

$$\sum_{w \in \{0,1\}^{\ell'(n)}} \sum_{i \in [t]} \sum_{u \in \{0,1\}^t} \Phi'''_y(w, i, u). \tag{7}$$

Hence, we reduce the evaluation of $\widehat{\Phi}_{A'}$ on $x \in \mathrm{GF}(p)^n$ 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 $x = (x_1, ..., x_n) \in \mathrm{GF}(p)^n$ is obtained as follows.

1. As in the proof of Proposition 3.6, for each $k \in [n]$, we 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. (7).

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

As outlined in the begining of this section (see Eq. (4)-(5)), the key observation is that

$$\begin{aligned}
\sum_{w\in\{0,1\}^{\ell'(n)}} \widehat{\Phi}_{A'}(x_1,...,x_n) &= \sum_{w\in\{0,1\}^{\ell'(n)}} \widehat{\phi}_n(w,x_{i^{(1)}_{n,w}},...,x_{i^{(m(n))}_{n,w}}) \\
&\equiv \sum_{w\in\{0,1\}^{\ell'(n)}} \sum_{i\in[t]} \phi''_{n,p}(w,(y,i)_{\sigma''_n(w)})\cdot 2^{i-1} \pmod{p} \\
&\equiv \sum_{i\in[t]} \sum_{(w,u)\in\{0,1\}^{\ell'(n)+t}} \phi'''_{n,p}(w,i,u,y_{\sigma''_n(w)}) \pmod{p}.
\end{aligned}$$

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

Note that the single 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 $\mathrm{GF}(p)$ have $\lfloor 2^{t'}/p\rfloor$ representations, whereas others have $\lceil 2^{t'}/p\rceil$ 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 $\rho'-n\cdot(p/2^{t'})$ (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 $n''=\exp(d'(n))\cdot n$. This can be fixed by artificially padding the input to length n'' and noting that $\ell''(n'')\geq \ell'(n)+t+\log t$ (where $t=\log p=d'(n)$).[14] ■

3.2.2 Proof of Theorem 3.1

The claim of Theorem 3.1 follows by combining the revised reductions provided by Propositions 3.8–3.10. Specifically, using (a generalization of) Proposition 3.8, we reduce the original counting problem $\#_A$ to $\tau=O(\ell(n)/d(n))$ instances of the Arithmetic evaluation problem, where all instances refer to the same algorithm A' but to different primes $p>s=\exp(d(n))$. Note that we can afford to find and use the first τ such primes (since we can run for 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 Propositions 3.9 and 3.10 will be applied to each of these primes.

As hinted above, we shall be using a generalization of Proposition 3.8, rather than Proposition 3.8 itself. This generalization will allow us to solve $\#_A$ on the worst-case even when solving, on the worst-case, only most of the Arithmetic evaluation problems associated with A', rather than all of them. Hence, in this step, we lose a factor of $2+o(1)$ in the error rate, rather than losing a factor of τ. The foregoing generalization is obtained by combining the τ results, which refer to the count modulo different primes, using Chinese Remaindering with errors (cf. [15]). Indeed, the error-correcting version of the CRT will be used instead of the plain version used in Proposition 3.8, and the number of primes is increased (from $(\ell(n)/d(n))+O(1)$ to $O(\ell(n)/d(n))=\tau$) so as to allow for such an error correcting feature.

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

(Recall that applying Proposition 3.10 increases the depth of the formula by a factor of $\text{poly}(\log d(n))$ (rather than $\text{poly}(\log\log n)$), and increases the length of the input by a factor of $\exp(d(n))$. This is all due to our using smaller primes (i.e., primes of size $\exp(d(n))$ rather than $\text{poly}(n)$).)

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 $\tau = O(\epsilon^{-1} \cdot \ell(n)/d(n))$ primes that are greater than $s = \exp(d(n))$ by $p_1, ..., p_\tau$, we consider the τ evaluation problems obtained by applying the generalized Proposition 3.8 to algorithm A (i.e., all these problems are associated with algorithm A'). The i^{th} such problem, denoted $\texttt{EV}_{A',\text{GF}(p_i)}$, consists of evaluating the polynomial $\widehat{\Phi}_{A'}$ at points in $\text{GF}(p_i)^n$, and we shall apply it at the point $x \in \{0,1\}^n$, viewed as an element of $\text{GF}(p_i)^n$.

   The difference between Proposition 3.8 and its generalization will arise when we reconstruct the answer to $\#_A$: We shall be employing CRT with errors and reconstruct $\#_A(x)$ even when obtaining the correct value of $\texttt{EV}_{A',\text{GF}(p_i)}(x)$ only for $(0.5 + \epsilon) \cdot \tau$ of the i's.

2. For each $i \in [\tau]$, we use Proposition 3.9 to reduce the evaluation of $\widehat{\Phi}_{A'}$ at $x \in \text{GF}(p_i)^n$ to its evaluation at $s' = O(s/\epsilon^2)$ points in $\text{GF}(p_i)^n$, 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 3.10 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 $x^{(i,j)} \in \text{GF}(p_i)^n$, where $n' = \exp(d(n)) \cdot n$.

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 $\#_B(i, y^{(i,j)}) \equiv \texttt{EV}_{A',\text{GF}(p_i)}(x^{(i,j)}) \pmod{p_i}$. Now, for each $i \in [\tau]$, we apply the decoding procedure (of Proposition 3.9, with error reduction) to the values $v^{(i,1)}, ..., v^{(i,s'')}$, and obtain a value $v^{(i)} \in \text{GF}(p_i)$, 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 3.10 yields a procedure that solves $\texttt{EV}_{A',\text{GF}(p_i)}$ with success rate at least $(0.5 + \epsilon - o(1))$ (over instances in $\text{GF}(p_i)^n$). This means that, for each of these i's, the hypothesis of Proposition 3.9 holds, and so (for each of these i's) the decoding procedure (of Proposition 3.9) 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 $d \in \mathcal{D}$, the latter (average-case) task can be reduced to solving a related problem on at least a $\exp(-d(n))$ 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 $\exp(-d(n))$ 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 $\text{Dtime}(p(n) \cdot n^{1+o(1)})$, where $p(n) = 2^{\ell(n)} \cdot \exp(d(n))$ bounds the worst-case time of solving 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.

Overview. Our starting point is the realization that when referring to such low success rate (i.e., a $\exp(-d(n))$ fraction for some $d \in \mathcal{D}$), we are in the *regime of list decoding*. Hence, for starters, Proposition 3.9 should (and can) be replaced by the list decoding result of Sudan, Trevisan, and Vadhan [29, Thm. 29]. This result essentially asserts the existence of an *explicit list of oracle machines such that, for every low-degree polynomial $P : \mathcal{F}^n \to \mathcal{F}$ and every $F : \mathcal{F}^n \to \mathcal{F}$ that agrees with P on a noticeable fraction of the instances, one of these machines computes P correctly on all instances when given oracle access to F.*

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 $\exp(d(n))$-many different prime fields (on all instances). Next, we apply the foregoing reduction of [29, 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 [29, 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 $\text{GF}(p)$. 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 $(p, \cdot)$. This implies that, when given oracle access to such a rare-case solver for $\Pi''(p, \cdot)$, one of the foregoing oracle machines computes $\widehat{\Phi} : \text{GF}(p)^n \to \text{GF}(p)$ 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 $1 - o(1) > 76\%$ fraction 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.)

To see how we can identify these primes and machines, suppose 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 $\Pi' \bmod p$: 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, given a list of oracle machines for each of the ($\exp(d(n))$-many) primes, we can pick $\ell(n)/d(n)$ distinct primes (i.e., p's), and an oracle machine for each such prime p such that the chosen machine computes $\Pi' \bmod p$ 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 $1 - \frac{\ell(n)}{d(n)} \cdot \frac{1}{\log n} = 1 - o(1)$ fraction of the inputs. For details see Section 4.1.

Three ways of obtaining a sample of solved instances. The foreging discussion 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. The second solution 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 a problem that has a suitable downwards self-reduction.[15] 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., to the reduction between the worst-case and 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. We call such a reduction *sample-aided*, and formulate it in the context of average-case to average-case reductions (while noting that worst-case settings can be derived as special cases).

Definition 4.1 (sample-aided reductions, Definition 1.4 revisited): *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*

$$\Pr_{r_1,...,r_s \in \{0,1\}^n}\left[\left|\mathtt{corr}_M^{P,\Pi'}(r_1,...,r_s)\right| \geq \rho' \cdot 2^n\right] \; > \; 2/3 \tag{8}$$

where $\mathtt{corr}_M^{P,\Pi'}(r_1,....,r_s)$ denotes the set of inputs on which the reduction returns the correct answer; that is,

$$\mathtt{corr}_M^{P,\Pi'}(r_1,....,r_s) \stackrel{\text{def}}{=} \left\{x \in \{0,1\}^n : \Pr[M^P(x; (r_1, \Pi'(r_1)), ..., (r_s, \Pi'(r_s))) = \Pi'(x)] \geq 2/3\right\} \tag{9}$$

[15]Indeed, this follows a paradigm that can be traced to the work of Impagliazzo and Wigderson [20].

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

The error probability bounds in Eq. (8)-(9) can be reduced at a moderate cost. This is straightforwards for Eq. (9), but the error reduction for Eq. (8) comes at a (small) cost and requires some care. Specifically, we have to approximate the size of the set $\texttt{corr}_M^{P,\Pi'}(r_1, ..., r_s)$. This can be done using an auxiliary sample of the solved instances; in particular, for every $\epsilon > 0$ and $k \in \mathbb{N}$, using $O(k \cdot s) + O(k/\epsilon^2)$ solved samples, we can output a sequence of s solved samples $\overline{r}$ such that, with probability at least $1 - \exp(-k)$, it holds that $\left|\texttt{corr}_M^{P,\Pi'}(\overline{r})\right| > (\rho' - \epsilon) \cdot 2^n$.

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, [14, 29, 21]). Furthermore, any average-case to rare-case reduction of the "list-decoding" 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).[16]

Our sample-aided reduction. Turning back to our specific context, we present a sample-aided reduction of solving any counting problem in $\mathcal{C}_{\ell,\mathcal{D}}$ on at least 99% of the inputs to solving some other problem in $\mathcal{C}_{\ell,\mathcal{D}}$ on 1% of the inputs. Actually, 99% and 1% can be replaced by $1-\exp(-d(n))$ and $\exp(-d(n))$, respectively, for any $d \in \mathcal{D}$.

Theorem 4.2 (sample-aided reduction of average-case to rare-case for counting problems): *Let $\mathcal{D}$ be a set of admissible functions, $d \in \mathcal{D}$, 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* $\Pi' : \{0,1\}^{n'} \to \mathbb{N}$ *on at least* $1 - \exp(-d(n'))$ *fraction of the domain to solving* $\Pi'' : \{0,1\}^{n''} \to \mathbb{N}$ *on at least* $\exp(-d(n''))$ *fraction of the domain.*

In particular, Theorem 4.2 yields an almost-linear time *non-uniform* reduction of solving Π' on at least $1 - \exp(-d(n'))$ fraction of the domain to solving Π'' on at least $\exp(-d(n''))$ fraction of the domain. Combining this non-uniform reduction with Theorem 3.1, we obtain Theorem 1.3.

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 at least a $1 - \exp(-d(n))$ fraction of the possible x's. The general plan is to mimic the proof of Theorem 3.1, while replacing the worst-case to average-case reduction (of the arithmetic problem) by a worst-case to rare-case reduction, and coping with the difficulties that arise. Hence, we first reduce the counting problem Π' to the task of evaluating a polynomial $\widehat{\Phi}$ in many prime fields, then move from a worst-case version of the aritmetic problem to a rare-case version of it, and reduce the latter to a rare-case version of solving the counting problem Π''.

We start by employing a reduction as in Proposition 3.8, except that here we use poly(D) primes of size poly(D), where $D = \exp(d(n))$ denotes the degree of the polynomial $\widehat{\Phi}$ that is derived in Proposition 3.8; specifically, we refer to the first $\Theta(D \cdot \ell(n))$ primes that are larger than poly(D).

[16]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. Hence, a worst-case to rare-case reduction of the list-decoding type only yields a sample-aided reduction from solving the original problem on a $1 - o(1)$ fraction of the instances.

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 $\ell(n)/d(n)$ of these fields, and combine these values using Chinese Remaindering (without errors).)

Next, for each prime p, we invoke [29, 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 $\epsilon = \exp(-d(n))$. Now, for each prime p, we invoke Proposition 3.10 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 generator by $\mathtt{rr}$; that is, $y \leftarrow \mathtt{rr}(q)$. 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).[17]

(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 "rare-case Π''-solver".)

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 $\mathrm{GF}(p)^n$. Recall that when restricted to $\{0,1\}^n$, the polynomial $\widehat{\Phi}$ equals the value of the counting problem Π'. Hence, we can approximate the success probability of M in evaluating $\widehat{\Phi}$ over $\{0,1\}^n$ by using the solved sample for Π'. (We stress that while $\widehat{\Phi}$ and M are defined over $\mathrm{GF}(p)^n$, we approximate their agreement rate over $\{0,1\}^n$, which suffices for our application.)[18]

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 $\exp(d(n))$ suffices to approximate all $\mathrm{poly}(D) = \exp(d(n))$ quantities such that, with high probability, all values that are below $1 - \exp(-d(n))$ 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 $1 - \exp(-d(n))$ 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 $\ell(n)/d(n)$ machines that correspond to different primes, since we can have $\epsilon \cdot \mathrm{poly}(D) > \ell(n)$.

Lastly, we pick any $\ell(n)/d(n)$ 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 $\mathrm{GF}(p)^n$. 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-value provided for the input $(p, \mathtt{rr}(q))$

[17]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 $\mathrm{GF}(p)^n$.

[18]In contrast, it is not clear how to approximate the agreement of $\widehat{\Phi}$ and M over $\mathrm{GF}(p)^n$, since we only have a solved sample of instances that are uniformly distributed in $\{0,1\}^n$.

and reducing it modulo p. Finally, we combine the values obtained for the count modulo each of these primes, by using Chinese Remaindering (without errors). Assuming that the surviving machines that we picked are typically correct on x, with high probability, the resulting value equals the value of $\Pi'(x)$.

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

Digest. The sample-aided reduction used in the proof of Theorem 4.2 is the result of composing three reduction, where the second reduction (i.e., the worst-case to average-case reduction of the evalutation of the polynomial $\widehat{\Phi}$) is actually the one that is sample-aided. However, this second reduction is sample-aided with respect to the set of instances $\mathrm{GF}(p)^n$, where p is a generic prime (which is sufficiently larger than the degree of $\widehat{\Phi}$), and the reduction uses its solved sample in order to estimate the success probability over $\mathrm{GF}(p)^n$ of several candidate solvers. If we were to compose this reduction with the (worst-case) reduction of solving the counting problem Π' to evalutating $\widehat{\Phi}$, we would not have obtained a sample-aided reduction of Π' to $\widehat{\Phi}$, since the solved sample would have been of n-bit inputs to Π' whereas the sample-aided reduction for $\widehat{\Phi}$ requires solved instances drawn uniformly from $\mathrm{GF}(p)^n$. Instead, we composed the first two reductions, while using the sample of solved instances for Π' in order to estimate the success rate of candidate solvers for $\widehat{\Phi}$ *in solving random instances in* $\{0,1\}^n$. This yields a sample-aided reduction of Π to $\widehat{\Phi}$, which reduces solving Π' with success rate $1-o(1)$ to solving $\widehat{\Phi}$ with success rate $o(1)$. (Combined with the success-rate preseving reduction of $\widehat{\Phi}$ to Π'', we establish Theorem 4.2.)

4.2 Obtaining solved samples via downwards self-reduction

A general paradigm that can be traced to the work of Impagliazzo and Wigderson [20] 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''. Loosely speaking, 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.

4.2.1 Difficulties and resolving them

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 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 [29, 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, and apply the methodology at that level (i.e., on the worst-case to rare-case of the polynomial evaluation problem). This problem will be a generalization of the evaluation problem presented in Definition 3.2, where the generalization is aimed at supporting an adequate notion of downwards self-reducibility.

Refining the notion of downwards self-reducibility. 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 $d \in \mathcal{D}$, we require that for every $n \in \mathbb{N}$ the interval $[n+1, n+\exp(d(n))]$ 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 $n' \geq n - \exp(d(n))$ (i.e., the downwards self-reduction makes no queries on inputs of length n'). Consequently, it suffices for the standard reduction to generate samples for lengths $n', n'+1, ..., n$.

Generalizing the evaluation problem. 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 3.8 utilizes only $\ell(n)/d(n)$ different fields, which correspond to the first $\ell(n)/d(n)$ primes that are larger than $2^{d(n)}$, we make the index of the field part of the input. Furthermore, we present the index of the field in unary so that inputs that correspond to different fields have different lengths, and rely on the fact that the rare-case solver is supposed to work for all input lengths.[19]

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. (3) 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 $j \in [\ell(n)-1]$, 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

[19]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 reductions presented so far). We note that this disadvantage is inherent to the downwards self-reduction paradigm of [20], so we may just take advantage of it for this additional purpose.

$j+1$ 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 directly in nearly-linear time, which satisfies the additional condition (Condition A) of downward self-reducibility.

Note that the foregoing 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 an additional bit of the summation-index. The second part (i.e., v) represents the prefix of the summation-index (which has been fixed), and x is the "main" input. We shall use a multi-linear extension $\texttt{SEL} : \mathrm{GF}(p)^{\log_2 n} \times \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ of the selection function $\texttt{sel} : \{0,1\}^{\log_2 n} \times \mathrm{GF}(p)^n \to \mathrm{GF}(p)$, which satisfies $\texttt{sel}(\alpha, x) = x_{\texttt{int}(\alpha)+1}$, where $\texttt{int}(\alpha)$ is the integer represented by the binary string α; that is,

$$\texttt{SEL}(\zeta, x) = \sum_{\beta\in\{0,1\}^{\log_2 n}} \prod_{k\in[\log_2 n]} (\beta_k\zeta_k + (1-\beta_k)(1-\zeta_k)) \cdot x_{\texttt{int}(\beta)+1}$$

(i.e., crucial point is that $\texttt{SEL}(\zeta, x)$ is the sum of n terms such that the i^{th} term is a multilinear function of the ζ_k's and x_i).

Definition 4.3 (generalization of Definition 3.2): *For $d \in \mathcal{D}$ and $n \in \mathbb{N}$, let $p_1, ..., p_{\ell'(n)}$ denote the first $\ell'(n) = \ell(n)/d(n)$ primes that are larger than $2^{d(n)}$, 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

$$\overline{\Phi}_n(uv, x) \stackrel{\text{def}}{=} \sum_{w'\in\{0,1\}^{\ell(n)-j}} \widehat{\phi}_n(vw', F_1(vw', x), ..., F_{m(n)}(vw', x)) \bmod p_i, \tag{10}$$

where $F_k(w, x) = \texttt{SEL}(\widehat{\pi}_{n,k}(w), x)$ 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).

Note that uv is a sequence of $\ell(n)^2 \cdot n + (\ell(n) + 1) \cdot i - j$ elements of $\mathrm{GF}(p_i)$, and that the said length determines both i and j.

The generalized evaluation problem is suitable for our application. We first observe that Definition 4.3 is suitable for our application, since it is downward self-reducible in an adequate sense and generalizes Definition 3.2. 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 $d(n) + 1$ times longer (and observing that all p_i's satisfy $\lceil \log p_i \rceil = d(n) + 1$).

1. The problem in Definition 3.2 (when restricted to any of the 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 consists of mapping the instance $x \in \mathrm{GF}(p_i)^n$ to the instance $(1^{\ell(n)^2 \cdot n+(\ell(n)+1)\cdot i}, x)$, where $1^{\ell(n)^2 \cdot n+(\ell(n)+1)\cdot i}$ is viewed as a sequence of length $\ell(n)^2 \cdot n + (\ell(n)+1)\cdot i$ over $\mathrm{GF}(p_i)$. (Equivalently, the proof of Proposition 3.8 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 $L_n = \{0,1,...,\ell(n)\}$).

   This means that instances associated with different (n,i,j)'s have different lengths, as posulated in the motiovating discussion (see also Footnote 19).

3. The function $\overline{\Phi}$ 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.)

4. The function $\overline{\Phi}$ 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 $\overline{\Phi}(uv,x) = \widehat{\phi}_n(v, F_1(v,x), ..., F_{m(n)}(v,x)) \bmod p_i$, because in this case $j = \ell(n)$.

   (Note that computing $\widehat{\phi}_n(v, F_1(v,x), ..., F_{m(n)}(v,x))$ essentially reduces to computing $F_k(w,x) = \mathtt{SEL}(\widehat{\pi}_{n,k}(w), x)$ for every $k \in [m(n)]$, we can be computed in almost-linear time.)

These observations will be used when proving the following.

Theorem 4.4 (worst-case to rare-case reduction): *Let $\mathcal{D}$ be a set of admissible functions, $d \in \mathcal{D}$, 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* $\exp(-d(n))$ *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 $z \in \{0,1\}^n$, the value of $\Pi(z)$ is obtained by applying the Chinese Remaindering (without errors) to the values of $\overline{\Phi}$ on the $\ell'(n)$ instances $(1^{\ell(n)^2 \cdot n+(\ell(n)+1)\cdot i}, z)$ for $i \in [\ell'(n)]$. Hence, fixing any ($n \in \mathbb{N}$ and) $i \in [\ell'(n)]$ and following the length convention stated above (i.e., considering the length of $\mathrm{GF}(p_i)$ sequences), we focus on reducing the task of computing $\overline{\Phi}$ on *any* instance of length $\widetilde{n}_i = (\ell(n)^2+1)\cdot n + (\ell(n)+1)\cdot i$ to obtaining this value of an $\epsilon = \exp(-d(n))$ fraction of these instances (i.e., members of $\mathrm{GF}(p_i)^{\widetilde{n}_i}$).

(Note that we reduced obtaining the value of Π on an arbitrary (worst-case) $z \in \{0,1\}^n$ to obtaining values of $\overline{\Phi}$ on $\ell'(n)$ related instances, having varying lengths, where the i^{th} instance has length $\ell(n)^2 \cdot n + (\ell(n)+1) \cdot i + n = \widetilde{n}_i$ (i.e., it consists of padding z with $\widetilde{n}_i - n$ 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 the evaluation of $\overline{\Phi}$ over $\mathrm{GF}(p_i)^{\widetilde{n}_i}$, we observe that the list-decoder algorithm of [29, Thm. 29] yields 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, and it works in time $\mathrm{poly}(1/\epsilon) \cdot \widetilde{O}(n) = \exp(d(n)) \cdot n$, which is almost-linear time.[20] The foreging list may contain also other machines. Still, using low-degree tests (e.g., [27]), 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., [11]) we obtain machines that compute low degree polynomials.

We next convert the list-decoder of [29, Thm. 29] into a sample-aided reduction, by identifying the machines that compute the correct polynomial. Specifically, using a solved sample for the evalution problem of the polynomial that we seek to compute (i.e., $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}_i} \to \mathrm{GF}(p_i)$), we can identify machines that computes the correct polynomial (since the other machines compute polynomials that disagree with the correct polynomial on most inputs).[21] 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 $O(\log(1/\epsilon)) = O(d(n))$ suffices for identifying all bad machines with high probability.

The solved sample for $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}_i} \to \mathrm{GF}(p_i)$ 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 $\overline{\Phi} : \mathrm{GF}(p_i)^{\widetilde{n}_i - j} \to \mathrm{GF}(p_i)$. Specifically, on input $y \in \mathrm{GF}(p_i)^{\widetilde{n}_i}$, we proceed as follows.

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

2. For $k = \widetilde{n}_i - \ell(n) + 1, ..., \widetilde{n}_i$ (equiv., using $k = \widetilde{n}_i - j$ for $j = \ell(n) - 1, ..., 0$), generate a random sample of $O(d(n))$ solved instances for $\overline{\Phi} : \mathrm{GF}(p_i)^k \to \mathrm{GF}(p_i)$, where the values of $\overline{\Phi}$ on each (random sample point) $r = (uv, x) \in \mathrm{GF}(p_i)^{k-n} \times \mathrm{GF}(p_i)^n$ 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 $r = (uv, x) \in \mathrm{GF}(p_i)^{\widetilde{n}_i - 2(\widetilde{n}_i - k) + (\widetilde{n}_i - k) - n} \times \mathrm{GF}(p_i)^n$ 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 $(\widetilde{n}_i - 2(\widetilde{n}_i - k) - 2)$-long prefix of u, and that these inputs are of length $\widetilde{n}_i - (\widetilde{n}_i - k) - 1 = k - 1$.

[20]As noted in Section 4.1, although the statement of [29, 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.

[21]We could not do this in Section 4.1, since the solved instances we obtained there were all in $\{0,1\}^{\widetilde{n}_i}$, whereas different low degree polynomials over $\mathrm{GF}(p_i)^{\widetilde{n}_i}$ may agree on $\{0,1\}^{\widetilde{n}_i}$.

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

We stress that the invocations of the sample-aided reductions are provided with a number of samples that guarantees that the probability of error in the invocation is smaller than $\exp(-d(n)) \ll 1/\ell(n)$. Hence, with very high probability, our reduction provides the correct value of $\overline{\Phi}(y)$. Recalling that $y = y^{(i)}$ was actually derived from the input z such that $\overline{\Phi}(y^{(i)}) \equiv \Pi(z) \pmod{p_i}$, we recover $\Pi(z)$ by applying Chinese Remaindering (without errors) to the values of $\overline{\Phi}$ on the $\overline{\Phi}(y^{(i)})$'s for $i \in [\ell'(n)]$. ■

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 $z \in \{0,1\}^n$ is transformed into $\ell'(n)$ inputs for the generalized problem, denoted $y^{(1)}, ..., y^{(\ell'(n))}$, such that $y^{(i)} = (1^{\widetilde{n}_i - n}, z)$ and $\widetilde{n}_i = (\ell(n)^2 + 1) \cdot n + (\ell(n) + 1) \cdot i$. The instance $y^{(i)}$ 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 value $\Pi(z)$ will be computed at the end of the process, based on the $\overline{\Phi}(y^{(i)})$'s.)

Likewise, 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 $\widetilde{n}_i - \ell(n)$, while relying on the fact that instances of such length are easy to solve, 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 $y^{(i)} = (1^{\widetilde{n}_i - n}, z)$ 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 z is obtained by combining the solutions to the various $y^{(i)}$'s (using the CRT).

Appendices

Appendix A.1 relates Definition 2.3 to [16, Def. 2], and Appendix A.2 presents a worst-case to average-case reduction for uniform $\mathcal{AC}^0[2]$.

A.1 A related class (for context only)

In this appendix, we review the definition of locally-characterizable sets [16], and discuss its relation to Definition 2.3.

Definition A.5 (locally-characterizable sets [16, Def. 2]):[22] *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* $\pi_{n,1}, ..., \pi_{n,p(\log n)}$:

[22] Actually, this is a slightly revised version, which is essentially equivalent to the original: In [16, Def. 2], the formula ϕ_n got as input the sequence $(\pi_{n,1}(w), ..., \pi_{n,p(\log n)}(w))$ (rather than w) as well as $(x_{\pi_{n,1}(w)}, ..., x_{\pi_{n,p(\log n)}(w)})$. This is essentially 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.

$\{0,1\}^{c\log n} \to [n]$ *such that, for every* $x \in \{0,1\}^n$, *it holds that* $x \in S$ *if and only if for all* $w \in \{0,1\}^{c\log n}$

$$\Phi_x(w) \stackrel{\rm def}{=} \phi_n(w, x_{\pi_{n,1}(w)}, ..., x_{\pi_{n,p(\log n)}(w)}) \tag{11}$$

equals 0.[23]

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 long suffix).

A locally-characterizable set corresponds to the set of inputs for the counting problem (of Definition 2.3) that have value 0 (i.e., the number of satisfied local conditions is 0). The correspondence is not an equality, beacuse the sizes of the formulae in the two definitions are different. Whereas in Definition 2.3 the number of formulae and their sizes are exponential in a function d 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.

A.2 Worst-case to Average-case reduction for uniform $\mathcal{AC}^0[2]$

For constants $c, d \in \mathbb{N}$, let $\mathtt{C}^{(d,c)}$ denote the class of decision problems on n-bit inputs that can be solved by uniform families of (unbounded fan-in) Boolean circuits of depth d and size n^c, with `and`, `or`, `parity`, and `not` gates. Specifically, a problem is parameterized by an efficient algorithm that on input n and $i, j \in [n^c]$ returns the type of the $i^{\rm th}$ gate and whether or not the $j^{\rm th}$ gate feeds into it (in the circuit that corresponds to n-bit inputs). The term "efficient" is left unspecified on purpose; possible choices include poly(n)-time and $O(\log n)$-space. Note that any decision problem having sufficiently uniform $\mathcal{AC}^0[2]$ circuits is in $\mathtt{C}^{(d,c)}$ for some constants $c, d \in \mathbb{N}$.

Theorem A.6 (worst-case to average case reduction for $\mathcal{AC}_0$): *There exists a universal constant* γ *such that for any* $c, d \in \mathbb{N}$, *solving any problem in* $\mathtt{C}^{(d,c)}$ *on the worst-case reduces in almost linear time to solving some problem in* $\mathtt{C}^{(\gamma\cdot d, c+o(1))}$ *on at least 90% of the instances.*

Proof Sketch: We proceed in three steps: First we reduce the Boolean problem (in $\mathtt{C}^{(d,c)}$) to an Arithmetic problem, next we show that the latter problem supports a worst-case to average-case reduction, and lastly we reduce the Arithmetic problem to a Boolean problem (in $\mathtt{C}^{(\gamma\cdot d, c+o(1))}$). This is very similar to what was done in the main part of this work, except that the first step is fundamentally different.

A straightfoward emulation of the Boolean circuit by the Arithmetic circuit would yield multiplication gates of polynomial fan-in, which in turn would mean that the polynomial computed by this circuit is of polynomial degree. In contrast, using the approximation method of Razborov [25] and Smolensky [28], we can get multiplication gates of logarithmic fan-in, and arithmetic circuits that compute polynomials of polylogarithmic degree. The approximation error is of no real concern,

[23]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 [16] in order to simplify the exponsion. We stress that since n is presented in binary, the algorithm runs in poly($\log n$)-time.

since we are actually interested in the value of the circuit at a single point. We need, however, to perform the foregoing randomized reduction using an almost linear amount of randomness, since we need to run in almost linear time, but this is possible using small-bias generators (cf. [24]). Details follow.

The first step is a randomized reduction of solving the Boolean problem in the worst-case to solving a corresponding Arithmetic problem on the worst-case. This reduction uses the ideas underlying the approximation method of Razborov [25] and Smolensky [28], while working with the field GF(2) (as [25], rather than with GF(p) for some prime $p > 2$ (as [28])).[24] When doing so, we replace the random choices made at each gate by pseudorandom choices that are generated by a small bias generator [24]; specifically, we use a highly uniform generator $G : \{0,1\}^{O(\log n^c)} \to \{0,1\}^{\widetilde{O}(n^c)}$ such that each bit of $G(s)$ is computed by a $n^{o(1)}$-size circuit of constant depth (and parity gates) [2, 19].[25]

Hence, for a fixed Boolean circuit C_n, on input $x \in \{0,1\}^n$, we uniformly select a seed $s \in \{0,1\}^{O(\log n)}$ for the aforementioned small-bias generator G, and consider the corresponding Arithmetics circuit $A_n^{(s)} : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$, in which `or`-gates of C_n are replaced by $O(c \log n)$-way multiplications of linear combinations of the original gates that are determined by $G(s)$. For any fixed x, we may have $\Pr_s[A_n^{(s)}(x) \neq C_n(x)] = 1/\mathrm{poly}(n)$. Note that the depth of $A_n^{(s)}$ is only $O(1)$ times larger than the depth of C_n, where the constant is determined by various (local) manipulations (which include replacing `and`-gates by `or`-gates, computing inner products of gates' values and generator outputs, and adding $O(\log n)$-wise multiplication gates). Furthermore, $A_n^{(s)}$ uses multiplication gates of $O(c \log n)$ arity, its size is at most $O(\log n)^d \cdot n^c$, and it computes a polynomial of degree $O(c \log n)^d$.

The next step is to embed GF(2) in an extension field of size greater than the foregoing degree so that the standard process of self-correction of polynomials can be performed. Hence, $A_n^{(s)}$ is now viewed as an arithmetic circuit over $\mathcal{F} = \mathrm{GF}(2^\ell)$ (i.e., $A_n^{(s)} : \mathcal{F}^n \to \mathcal{F}$), where $\ell = d \log\log n + O(d)$, since we need $2^\ell \geq O(c \cdot \log n)^d$. Now, a worst-case to average-case reduction is applied to $A_n^{(s)}$ (i.e., evaluating $A_n^{(s)}$ on the worst case reduces to evaluating $A_n^{(s)}$ correctly on at least a 51% fraction of the instances).

Lastly, we wish to get back to a class of Boolean problems. We can do so as follows. First, we replace each (unbounded) addition gate by ℓ parity gates (which add-up the ℓ corresponding bits in the sequence of field elements). Next, we replace each multiplication gate of arity at most $m = O(\log n)$ by a multiplication gate of arity $\sqrt{m}$ that is fed by $\sqrt{m}$ multiplication gate that cover the original m wires. Finally, we implement each of the latter gates by a small Boolean circuit of depth two (via a look-up table of size $|\mathcal{F}|^{\sqrt{m}} = \exp(\widetilde{O}(\sqrt{\log n})) = n^{o(1)}$). Hence, given the Arithmetic circuit $A_n^{(s)} : \mathrm{GF}(2^\ell)^n \to \mathrm{GF}(2^\ell)$, we obtain a Boolean circuit $B_n^{(s)} : \{0,1\}^{n\ell} \to \{0,1\}^\ell$

[24]Recall that this construction replaces each `or`-gate by a $O(\log n^c)$-way conjunction of random linear combination of the values that feed the original `or`-gate.

[25]Using the third construction in [2], we need to perform exponentiation in a field of size $2^{k/2}$, where $k = O(\log n^c)$ is the length of the seed. By [19, Thm. 4], this operation can be performed by a constant-depth circuit (with parity gates) of size $\exp(\widetilde{O}(\sqrt{k/2})) = n^{o(1)}$. The same pseudorandom sequence can be used for all gates in the circuit, and in each gate we can use $O(\log n^c)$ disjoint portions of the pseudorandom sequence for the $O(\log n^c)$ different linear combinations. (Recall that taking t independent linear combinations of the output of an ϵ-bias generator yields a distribution that equal 0^t with probability at most $2^{-t} + \epsilon$. Also recall that the aforementioned generator of [2] produces a n^c-bit long ϵ-biased sequence using a seed of length $O(\log(n^c/\epsilon))$, and so we can set $t = O(c \log n)$ and $\epsilon = 2^{-t}$.)

that emulates $A_n^{(s)}$. Furthermore, using the small circuits that produce the output bits of the small-bias generator G on seed $s \in \{0,1\}^k$, where $k = O(\log n)$, we obtain a Boolean circuit $B_n : \{0,1\}^{n\ell+k} \to \{0,1\}^\ell$ such that $B_n(y,s) = B_n^{(s)}(y)$. Recalling that the aforementioned circuits that compute the bits of G have constant depth and size $n^{o(1)}$, it follows that B_n has depth $O(d) + O(1)$ and size $n^{c+o(1)}$.

We are almost done, except that we need to reduce the evaluation of $B_n : \{0,1\}^{n\ell+k} \to \{0,1\}^\ell$ to the evaluation of a Boolean circuit that has a single output bit, and we need this reduction to work in the average-case setting. We can do so by using the Boolean circuit $B'_n : \{0,1\}^{n\ell+k+\ell} \to \{0,1\}$ that, on input $(z,r) \in \{0,1\}^{n\ell+k} \times \{0,1\}^\ell$, returns the inner product mod 2 of $B_n(z)$ and r. Note that the depth of B'_n exceeds the depth of B_n only by a constant term, and that we can correctly retrieve $B_n(z)$ (with high probability) if we can obtain the correct value of $B'_n(z,r)$ correctly on 0.76% of the r's. Recalling that evaluating C_n on x was reduced to evaluating B_n correctly on 0.51% of the instances, it follows that it suffices to compute B'_n correctly on a ρ fraction of the instances, provided that $\rho \geq 1 - 0.24 \cdot 0.49$. (Of course, 0.24 and 0.49 stand for any constants smaller than 0.25 and 0.5 respectively.)

To summarize, evaluating $C_n : \{0,1\}^n \to \{0,1\}$ in the worst case reduces to evaluating $B'_n : \{0,1\}^{\log\log n + O(\log n)} \to \{0,1\}$ on at least $1 - 0.24 \cdot 0.49 \approx 0.88$ fraction of the instances. Using an adequate indexing of the gates in B'_n, the uniformity of B'_n follows from the uniformity of C_n, since all modifications we have performed are local. ■

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] Noga Alon, Oded Goldreich, Johan Hastad, and Rene Peralta. Simple Construction of Almost k-wise Independent Random Variables. *Random Struct. Algorithms*, Vol. 3 (3), pages 289–304, 1992.

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

[4] 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.

[5] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant N. Vasudevan. Average-Case Fine-Grained Hardness. In the proceedings of *STOC*, pages 483–496, 2017.

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

[7] 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.

[8] 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.

[9] Jin-yi Cai, Aduri Pavan, and D. Sivakumar. On the Hardness of Permanent. In *16th STACS*, pages 90–99, 1999.

[10] Joan Feigenbaum and Lance Fortnow. Random-Self-Reducibility of Complete Sets. *SIAM Journal on Computing*, Vol. 22 (5), pages 994–1005, 1993.

[11] 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.

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

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

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

[15] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese Remaindering with Errors. *IEEE Trans. Information Theory*, Vol. 46 (4), pages 1330–1338, 2000. Preliminary version in *31st STOC*, 1999.

[16] Oded Goldreich and Guy N. Rothblum. Simple Doubly-Efficient Interactive Proof Systems for Locally-Characterizable Sets. In the proceedings of *ITCS*, 18:1–18:19, 2018.

[17] Oded Goldreich and Guy N. Rothblum. Counting t-Cliques: Worst-Case to Average-Case Reductions and Direct Interactive Proof Systems. In the proceedings of *FOCS*, pages 77–88, 2018.

[18] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. *ECCC*, TR02-039, 2002.

[19] Alexander Healy and Emanuele Viola. Constant-Depth Circuits for Arithmetic in Finite Fields of Characteristic Two. *ECCC* TR05-087, 2005.

[20] 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.

[21] 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.

[22] Richard J. Lipton. New directions in testing. *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt (ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematics Society, Vol. 2, pages 191–202, 1991.

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

[24] Joseph Naor and Moni Naor. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, Vol. 22 (4), pages 838–856, 1993. Preliminary version in *22nd STOC*, 1990.

[25] Alexander A. Razborov. Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. In *Matematicheskie Zametki*, Vol. 41, No. 4, pages 598–607, 1987 (in Russian). English translation in *Mathematical Notes of the Academy of Sci. of the USSR*, Vol. 41 (4), pages 333–338, 1987.

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

[27] 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 [11] and [26].

[28] Roman Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *19th ACM Symposium on the Theory of Computing* pages 77–82, 1987.

[29] 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.

[30] 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.