Learning Algorithms from Natural Proofs

Marco L. Carmosino[1], Russell Impagliazzo[2], Valentine Kabanets[3], and Antonina Kolokolova[4]

1 **Department of Computer Science, University of California, San Diego, USA**
mcarmosi@eng.ucsd.edu
2 **Department of Computer Science, University of California, San Diego, USA**
russell@eng.ucsd.edu
3 **School of Computing Science, Simon Fraser University, Burnaby, Canada**
kabanets@cs.sfu.ca
4 **Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada**
kol@mun.ca

Abstract

Based on Håstad's (1986) circuit lower bounds, Linial, Mansour, and Nisan (1993) gave a quasipolytime learning algorithm for AC^0 (constant-depth circuits with AND, OR, and NOT gates), in the PAC model over the uniform distribution. It was an open question to get a learning algorithm (of any kind) for the class of $\mathsf{AC}^0[p]$ circuits (constant-depth, with AND, OR, NOT, and MOD_p gates for a prime p). Our main result is a quasipolytime learning algorithm for $\mathsf{AC}^0[p]$ in the PAC model over the uniform distribution with membership queries. This algorithm is an application of a general connection we show to hold between natural proofs (in the sense of Razborov and Rudich (1997)) and learning algorithms. We argue that a natural proof of a circuit lower bound against any (sufficiently powerful) circuit class yields a learning algorithm for the same circuit class. As the lower bounds against $\mathsf{AC}^0[p]$ by Razborov (1987) and Smolensky (1987) are natural, we obtain our learning algorithm for $\mathsf{AC}^0[p]$.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases natural proofs, circuit complexity, lower bounds, learning, compression

Digital Object Identifier 10.4230/LIPIcs.CCC.2016.10

1 Introduction

Circuit analysis problems, problems whose input or output is a Boolean circuit, are a crucial link between designing algorithms and proving lower bounds. For example, Williams [41, 43, 42] shows how to convert non-trivial Circuit-SAT algorithms into circuit lower bounds. In the other direction, there have been many circuit analysis algorithms inspired by circuit lower bound techniques [25, 4, 32, 34, 19, 20, 3, 8, 7, 33, 6, 9, 37], but outside the setting of derandomization [28, 2, 21, 18, 39, 23], few formal implications giving generic improvements.

Here we make a step towards such generic connections. While we are not able to show that an *arbitrary* way to prove circuit lower bounds yields circuit analysis algorithms, we show that any circuit lower bound proved through the general *natural proofs paradigm* of Razborov and Rudich [31] does yield such algorithms. Our main general result is the following.

▶ **Theorem 1.1** (Learning Algorithms from Natural Lower Bounds: Informal version)**.** *Natural proofs of circuit lower bounds imply learning algorithms for the same circuit class.*

COMPUTATIONAL COMPLEXITY CONFERENCE
31st Conference on Computational Complexity (CCC 2016).
Editor: Ran Raz; Article No. 10; pp. 10:1–10:24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Using known natural lower bounds [30, 35, 31], we get quasipolynomial-time learning algorithms for the hypothesis class $\mathsf{AC}^0[p]$, for any prime p (polynomial-size constant-depth circuits with AND, OR, NOT, and MOD_p gates).

▶ **Theorem 1.2** (Learning for $\mathsf{AC}^0[p]$: Simplified version)**.** *For every prime $p \geq 2$, there is a randomized algorithm that, given membership queries to an arbitrary n-variate Boolean function $f \in \mathsf{AC}^0[p]$, runs in quasi-polynomial time $n^{\mathsf{poly}\log n}$ and finds a circuit that computes f on all but $1/\mathsf{poly}(n)$ fraction of inputs.*

No learning algorithms for $\mathsf{AC}^0[p]$ were previously known. For AC^0, a learning algorithm was given by Linial, Mansour, and Nisan [25][1], based on Håstad's proof of strong circuit lower bounds for AC^0 [15].

We also apply the general result to immediately obtain the following compression algorithm, first developed (with somewhat stronger parameters) by Srinivasan [36].

▶ **Theorem 1.3** (Compression for $\mathsf{AC}^0[p]$: Simplified version)**.** *For every prime $p \geq 2$, there is a randomized algorithm that, given the 2^n-bit truth table of an arbitrary n-variate Boolean function $f \in \mathsf{AC}^0[p]$, runs in time $\mathsf{poly}(2^n)$ (polynomial in the input size), and outputs a circuit computing f of the circuit size at most 2^{n-n^μ}, for some $0 < \mu < 1$.*

1.1 Compression and learning algorithms from natural lower bounds

Informally, a *natural* lower bound for a circuit class Λ contains an efficient algorithm that distinguishes between the truth tables of "easy" functions (of low Λ-circuit complexity) and those of random Boolean functions. This notion was introduced by Razborov and Rudich [31] to capture a common feature of most circuit lower bound proofs: such proofs usually come with efficient algorithms that say something nontrivial about the structure of easy functions in the corresponding circuit class. In [31], this observation was used to argue that any circuit class with a natural lower bound is too weak to support cryptography: no strong pseudorandom generator can be computed by a small circuit from the class.

We show that natural circuit lower bounds also imply algorithms for compression and learning of Boolean functions from the same circuit class (provided the circuit class is not too weak). More precisely, we show how to reduce the task of compressing (learning) Boolean functions in a circuit class Λ to the task of distinguishing between the truth tables of functions of low Λ-circuit complexity and those of random functions. The latter task is exactly what is solved by an efficient algorithm embedded in any natural proof of Λ-circuit lower bounds.

Compression. Recall the compression task for Boolean functions: given the truth table of a Boolean function f, print a circuit that computes f. If f is unrestricted, the best guarantee for the circuit size is $2^n/n$ [26, 27], and such a circuit can be found in time $\mathsf{poly}(2^n)$, polynomial in the truth table size. We might however be able to do much better for restricted classes of functions. Let Λ be the set of functions computed by some circuit class Λ. Recent work has shown that we can "mine" specific lower bounds against Λ to compress functions $g \in \Lambda$ better than the universal construction [7]. This work suggests that there should be some generic connection between circuit lower bounds and compression algorithms, but such a connection was not known.

[1] Their algorithm works in a more general learning model without membership queries, but with access to labeled examples $(x, f(x))$ for uniformly random x.

We show that any circuit lower bound that is natural in the sense of Razborov and Rudich [31] yields a generic compression algorithm for Boolean functions from the same circuit class, provided the circuit class is sufficiently powerful (e.g., containing $\mathsf{AC}^0[p]$ for some prime $p \geq 2$).

A compression algorithm may be viewed as a special case of a natural property: if the compression fails, the function must have high complexity, and compression must fail for most functions. Thus we get an equivalence between these two notions for the case of randomized compression algorithms and BPP-computable natural properties. That is, for an appropriate circuit complexity class $\mathcal{C}$, a BPP-computable natural property against $\mathcal{C}$ implies the existence of a related $\mathcal{C}$-compression algorithm in BPP, and a $\mathcal{C}$-compression algorithm in BPP implies a BPP-computable natural property against $\mathcal{C}$. As our compression algorithms are randomized, we don't get such an equivalence for the case of deterministic natural properties.

Learning. The first stage of our algorithm is a lossy compression of the function in the sense that we get a small circuit that computes the function on *most* inputs. Because this first stage only examines the truth table of the function in relatively few locations, we can view this stage as a *learning algorithm*. This algorithm produces a circuit that approximately computes the given function f with respect to the uniform distribution, and uses membership queries to f. So it fits the framework of PAC learning for the uniform distribution, with membership queries.

Minimum Circuit Size Problem: Search to decision reduction. Our main result also yields a certain "search-to-decision" reduction for the Minimum Circuit Size Problem (MCSP). Recall that in MCSP, one is given the truth table of a Boolean function f, and a parameter s, and needs to decide if the minimum circuit size of f is less than s. Since an efficient algorithm for MCSP would make it a natural property (with excellent parameters), our main result implies the following. If MCSP is in BPP, then, given oracle access to any n-variate Boolean function f of circuit complexity s, one can find (in randomized polynomial time) a circuit of size $\mathsf{poly}(s)$ that computes f on all but $1/\mathsf{poly}(n)$ fraction of inputs.

1.2 Our proof techniques

One of the main tools we use is the Nisan-Wigderson (NW) generator construction [28]. Informally, this construction takes as input the truth table of a Boolean function f, and outputs an algorithm for the new function G_f mapping "short" input strings to "long" output strings. The function G_f is intended to be a *pseudo-random generator (PRG)* in the sense that no "small" Boolean circuit can "distinguish" the uniform distribution from the distribution of G_f's outputs (on uniformly random inputs to G_f). A circuit that can distinguish these two distribution is said to break the generator, and is called a *distinguisher*. Nisan and Wigderson [28] prove that if the initial function f has "high" circuit complexity, then the function G_f is indeed a PRG. Moreover, their proof is constructive in the sense that there is an efficient *reconstruction* algorithm that, given a distinguisher for G_f and oracle access to f, outputs a "small" Boolean circuit that approximately computes f. (See Section 2 for the formal definitions and statements.)

Intuitively, we can use this reconstruction algorithm as a *learning algorithm* for a Boolean function f in some circuit class Λ, provided we manage to find an efficient distinguisher for the NW generator G_f. As we shall argue, such a distinguisher for G_f is supplied by any natural proof of Λ-circuit lower bounds (natural property for the circuit class Λ)!

Thus, the main idea of our lossy-compression algorithm is, given the truth table of a Boolean function f from a circuit class Λ,

- imagine using f as the basis for the NW generator G_f,
- argue that the natural property R for the class Λ is a distinguisher for G_f,
- apply the reconstruction algorithm to R to produce a small circuit that approximates f.

For the described approach to work, we need to ensure that (1) there is an efficient reconstruction algorithm that takes a distinguisher for G_f and constructs a small circuit for (approximately computing) f, and (2) the natural property for Λ is a distinguisher for G_f.

For (1), we use the known efficient randomized algorithm that takes a distinguisher for G_f and constructs a small circuit approximately computing f, provided the algorithm is given oracle access to f. The existence of such a uniform algorithm was first observed by Impagliazzo and Wigderson [22] (based on [28, 2]) in the context of derandomizing BPP under uniform complexity assumptions. Simulating oracle access to f in the framework of [22] was quite nontrivial (and required the downward self-reducibility of f). In contrast, we are explicitly given the truth table of f (or allowed membership queries to f), and so oracle access to f is not an issue!

For (2), we must show that each output of the NW generator, when viewed as the truth table of a Boolean function, is computable by a small circuit from the circuit class for which we have a natural lower bound (and so the natural property algorithm can be used as a distinguisher to break the generator). Looking inside the construction of the NW generator, we note that, for a fixed seed (input) of G_f, each bit of the output of G_f is the value of f on some substring of the seed (chosen via a certain combinatorial structure, the NW design). We argue that the circuit complexity of the truth table output by the NW generator G_f is closely related to the circuit complexity of the original function f.

In particular, we show that if f is in $\mathsf{AC}^0[p]$, and the NW generator has exponential stretch (from $\mathsf{poly}(n)$ bits to 2^{n^γ} bits, for some $\gamma > 0$), then each string output by the NW generator is also a function in $\mathsf{AC}^0[p]$. If, on the other hand, we take the NW generator with certain polynomial stretch, we get that its output strings will be Boolean functions computable by $\mathsf{AC}^0[p]$ circuits of subexponential size. The trade-off between the chosen stretch of the NW generator and the circuit complexity of the string it outputs will be very important for the efficiency of our learning algorithms: the runtime of the learning algorithm will depend polynomially on the stretch of the NW generator. This makes our setting somewhat different than most applications of the NW generator. We will want to make the stretch as small as possible, but must set it above a threshold determined by the quantitative strength of the circuit lower bound that we start from. Thus, the *larger* the circuit size for which we have lower bounds, the *faster* the learning algorithms we get.

Note that if we break the NW generator based on a function f, we only get a circuit that agrees with f on slightly more than half of all inputs. To get a better approximation of f, we employ a standard "hardness amplification" encoding of f, getting a new, amplified function h, and then use h as the basis for the NW generator. The analysis of such hardness amplification is also constructive: it yields an efficient *reconstruction* algorithm that takes a circuit C_0 computing h on more than $1/2$ of the inputs, and constructs a new circuit C that computes the original f on most inputs.

For this amplification to work in our context, we need to ensure that the amplified function h is in the same circuit class as f, and is of related circuit complexity. We show that standard tools such as the Direct Product and XOR constructions have the required properties for $\mathsf{AC}^0[2]$. For $\mathsf{AC}^0[p]$ where p is prime other than 2, we can't use the XOR construction (as

PARITY cannot be computed in $\mathsf{AC}^0[p]$ for any prime $p > 2$ by Smolensky's lower bound [35]). We argue that the MOD_p function can be used for the required amplification within $\mathsf{AC}^0[p]$[2].

Thus, our actual lossy-compression algorithm for a circuit class Λ is as follows:

Given the truth table of a function $f \in \Lambda$,

1. Run the reconstruction algorithm for the NW generator G_h with the natural property for Λ as a distinguisher, where h is the amplified version of f. This produces a circuit C_0 computing h on more than $1/2$ of inputs.
2. Run the reconstruction algorithm for hardness amplification to get from C_0 a new circuit C that computes f on most inputs.

To turn this algorithm into an exact compression algorithm, we just patch up the errors by table lookup. Since there are relatively few errors, the size of the patched-up circuit will still be less that the trivial size $2^n/n$.

More interestingly, our lossy compression algorithm described above also yields a *learning* algorithm! The idea is that the reconstruction algorithm for the NW generator G_f runs in time polynomial in the size of the output of the generator, and so only needs at most that many oracle queries to the function f. Rather than being given the full truth table of f, such an algorithm can be simulated with just membership queries to f. Thus we get a learning algorithm with membership queries in the PAC model over the uniform distribution.

Since the runtime of this learning algorithm (and hence also the size of the circuit for f it produces) will be polynomial in the output length of the NW generator that we use to learn f, we would like to minimize the stretch of the NW generator[3]. However, as noted above, *shorter stretch* of the generator means *higher circuit complexity* of the truth table it outputs. This in turn means that we need a natural property that works for Boolean functions of higher circuit complexity (i.e., natural properties useful against large circuits). In the extreme case, to learn a polysize Boolean function f in polynomial time, we need to use the NW generator with polynomial stretch, and hence need a natural property useful against circuits of exponential size. In general, there will be a trade-off between the efficiency of our learning algorithm for the circuit class Λ and the usefulness of a natural circuit lower bound for Λ: the larger the size s such that a natural property is useful against Λ-circuits of size s, the more efficient the learning algorithm for Λ.

Razborov and Rudich [31] showed the $\mathsf{AC}^0[p]$ circuit lower bounds due to Razborov [30] and Smolensky [35] can be made into natural properties that are useful against circuits of weakly exponential size 2^{n^γ}, for some $\gamma > 0$ (dependent on the depth of the circuit). Plugging this natural property into our framework, we get our quasi-polynomial-time learning algorithm for $\mathsf{AC}^0[p]$, for any prime p.

We remark that our approach is quite similar to the way Razborov and Rudich [31] used natural properties to get new algorithms. They used natural properties to break the cryptographic pseudorandom function generator of [11], which by definition outputs functions of low circuit complexity. Breaking such a generator based on an assumed one-way function

[2] We stress that for our purposes it is important that the *forward direction* of the conditional PRG construction, from a given function f to a generator based on that f, be computable in some low nonuniform circuit class (such as $\mathsf{AC}^0[p]$). In contrast, in the setting of conditional derandomization, it is usually important that the *reverse direction*, from a distinguisher to a small circuit (approximately) computing the original function f, be computable in some low (nonuniform) circuit class (thereby contradicting the assumed hardness of f for that circuit class). One notable exception is hardness amplification within NP [29, 16, 38].

[3] This is in sharp contrast to the setting of derandomization where one wants to *maximize* the stretch of the generator, as it leads to a more efficient derandomization algorithm.

F leads to an efficient algorithm for inverting this function F well on average (contradicting the one-wayness of F). We, on the other hand, use the NW generator based on a given function f. The properties of the NW generator construction can be used to show that it outputs (the truth tables of) functions of low circuit complexity, relative to the circuit complexity of f. Thus a natural property for the appropriate circuit complexity class (with an appropriate size parameter) can be used to break this NW generator, yielding an efficient algorithm for producing a small circuit approximating f.

Discussion. One counter-intuitive development in the theory of pseudorandomness has been the prevalence of "win-win" arguments. Typically, in a win-win argument in pseudorandomness, one takes a construction of pseudorandom generator from a hardness assumption (such as the NW generator mentioned above) and applies it to a function that is *not known* to actually be hard. If the construction is still a PRG, that is a win; if it is not, one learns that the function in question is not hard, and perhaps finds a circuit computing it. Here, we take this paradigm one step further; ours is a "play-to-lose" argument. We apply the pseudorandom generator construction to a function f we *know* not to be hard, in such a way as to guarantee that the resulting generator G_f is *not* pseudorandom. The win in this argument is that the proof of the hardness to pseudorandomness connection gives a way of converting the non-randomness of the generator G_f into a way of computing f, thus translating the knowledge that f is easy to compute into an actual circuit computing f.

1.3 Related work

This work was prompted by results that circuit analysis algorithms imply circuit lower bounds. A natural question is: given that these algorithms are *sufficient* for circuit lower bounds, to what degree are they *necessary*? Apart from derandomization, no other equivalences between circuit analysis algorithms and circuit lower bounds are known. Some of the known circuit-analytic algorithmic tasks that would imply circuit lower bounds include: derandomization [18, 23, 1, 5], deterministic (lossy) compression or MCSP [7, 18], deterministic learning [10, 24], and deterministic (QBF) SAT algorithms [41, 33].

Bracketing the hardness vs. randomness setting, special cases of using circuit lower bounds to construct circuit analysis algorithms abound. Often, lower bounds are the *only* way that we know to construct these algorithms. Each of the following results uses the proof of a lower bound to construct an algorithm. The character and number of these results gives empirical evidence that there should be generic algorithms for circuit analysis based on generic lower bounds.

- Parity $\notin \mathsf{AC}^0 \rightsquigarrow \mathsf{AC}^0$-Learning [25], AC^0-SAT [19], and AC^0-Compression [7]
- $\mathrm{MOD}_q \notin \mathsf{AC}^0[p]$, p, q distinct primes, $\rightsquigarrow \mathsf{AC}^0[p]$-Compression [36]
- Andreev's function $\notin \mathrm{deMorgan}[n^{3-\epsilon}] \rightsquigarrow$ subcubic formula Compression [7]

All the lower bounds listed above belong to the natural proofs framework. Given these results, the obvious conjecture was that natural proofs imply some kind of generic circuit analysis algorithm. For instance, [7] suggested that every natural circuit lower bound should imply a compression algorithm. We take a step towards proving such an implication by showing that any natural circuit lower bound for a sufficiently powerful circuit class ($\mathsf{AC}^0[p]$ or bigger) does indeed lead to a randomized compression algorithm for the same circuit class.

The remainder of the paper. e give the necessary background in Section 2. Sections 3 and 4 summarize the useful properties of past constructions of black-box generators and black-box amplifications, which we revisit and modify to implement in $\mathsf{AC}^0[p]$. In Section 5, we use those tools to prove our main result that natural properties yield learning algorithms for circuit classes $\mathsf{AC}^0[p]$ and above, using a novel "play-to-lose" interpretation of pseudorandomness. On the other hand, in Section 6, we argue that our main result cannot be applied directly to AC^0 because the construction of Section 3 is impossible in AC^0. Section 7 contains concluding remarks and open questions.

2 Definitions and tools

2.1 Circuits and circuit construction tasks

For a circuit class Λ and a set of size functions $\mathcal{S}$, we denote by $\Lambda[\mathcal{S}]$ the set of $\mathcal{S}$-size n-input circuits of type Λ. When no $\mathcal{S}$ is explicitly given, it is assumed to be $\mathsf{poly}(n)$.

▶ **Definition 2.1** (Circuits (Approximately) Computing f)**.** Let $f : \{0,1\}^n \to \{0,1\}$ be some Boolean function, and let $\epsilon : \mathbb{N} \to [0,1]$ be an approximation bound. Then $\mathsf{CKT}_n(f)$ denotes the set of circuits that compute the function f on all n-bit inputs, and $\widetilde{\mathsf{CKT}}_n(f,\epsilon)$ the set of all circuits that compute f on all but an ϵ fraction of inputs.

▶ **Definition 2.2** (Circuit Builder Declarations (adapted from [22]))**.** Let A and B be indexed sets of circuits. A $T(n)$-*construction of* B *from* A is a probabilistic machine $\mathcal{M}(n,\alpha,A_n)$ which outputs a member of B_n with probability at least $1-\alpha$ in time $T(n)$, where the size of B_n is $\mathsf{poly}(|A_n|)$. We declare that such a machine exists by writing: $\textsc{Cons}(A \to B;\ T(n))$. Read this notation as "from A we can construct B in time $T(n)$." To assert the existence of a $T(n)$-*construction of* B *from* A*, with oracle* $\mathcal{O}$, where the machine $\mathcal{M}$ is equipped with an oracle for the language $\mathcal{O}$ but otherwise is as above, write: $\textsc{Cons}^{\mathcal{O}}(A \to B;\ T(n))$.

2.2 Learning and compression tasks

Let $f \in \Lambda$ be some Boolean function. The learner is allowed membership queries to f. That is, the learner may query an input $x \in \{0,1\}^n$ to the oracle, getting back the value $f(x)$.

▶ **Definition 2.3** (PAC learning over the uniform distribution with membership queries)**.** Let Λ be any class of Boolean functions. An algorithm A PAC-learns Λ if for any n-variate $f \in \Lambda$ and for any $\epsilon, \delta > 0$, given membership query access to f algorithm A prints with probability at least $1-\delta$ over its internal randomness a circuit $C \in \widetilde{\mathsf{CKT}}_n(f,\epsilon)$. The runtime of A is measured as a function $T = T(n, 1/\epsilon, 1/\delta, \mathrm{size}(f))$.

▶ **Definition 2.4** (Λ-Compression)**.** Given the truth table of n-variate Boolean function $f \in \Lambda$, print some Boolean circuit $C \in \mathsf{CKT}_n(f)$ computing f such that $|C| < 2^n/n$, the trivial bound.

▶ **Definition 2.5** (ϵ-Lossy Λ-Compression)**.** Given the truth table of n-variate Boolean function $f \in \Lambda$, print some Boolean circuit $C \in \widetilde{\mathsf{CKT}}_n(f,\epsilon)$ such that $|C| < 2^n/n$, the trivial bound.

The relevant parameters for compression are runtime and printed circuit size. We say that a compression algorithm is efficient if it runs in time $\mathsf{poly}(2^n)$, which is polynomial in the size of the truth-table supplied to the algorithm. Though we count any output circuit of size less than $2^n/n$ as a successful compression, we will of course want to optimize this. In previous

work, the size of the resulting circuits approximately matches the size of circuits for which we have lower bounds.

We remark that we do not obtain "proper" learning or compression: the output of the learning (compression) algorithm is an unrestricted circuit, not necessarily from the class to be learned (compressed).

2.3 Natural properties

Let F_n be the collection of all Boolean functions on n variables. Λ and Γ denote complexity classes. A combinatorial property is a sequence of subsets of F_n for each n.

▶ **Definition 2.6** (Natural Property [31]). A combinatorial property R_n is Γ-natural against Λ with density δ_n if it satisfies the following three conditions:

Constructivity: The predicate $f_n \stackrel{?}{\in} R_n$ is computable in Γ

Largeness: $|R_n| \geq \delta_n \cdot |F_n|$

Usefulness: For any sequence of functions f_n, if $f_n \in \Lambda$ then $f_n \notin R_n$, almost everywhere.

For each n, δ_n is a lower bound on the probability that $g \in F_n$ has R_n. The original definition in [31] sets $\delta_n \geq 2^{-O(n)}$. However, we show (see Lemma 2.7 below) that one may usually assume that $\delta_n \geq 1/2$. Note that in the wild, nearly all natural properties have δ_n close to one and $\Gamma \subseteq \mathsf{NC}^2$.

▶ **Lemma 2.7** (Largeness for natural properties). *Suppose P is a P-natural property of n-variate Boolean functions that is useful against class Λ of size $s(n)$, and has largeness $\delta_n \geq 2^{-cn}$, for some constant $c \geq 0$. Then there is another P-natural property P' that is useful against the class Λ of size $s'(n) := s(n/(c+1))$, and has largeness $\delta'_n \geq 1/2$.*

Proof. Define P' as follows:

> The truth table of a given $f : \{0,1\}^n \to \{0,1\}$ is in P' iff for at least one string $a \in \{0,1\}^k$, for $k = cn/(c+1)$, the restriction
>
> $$f_a(y_1, \ldots, y_{n-k}) := f(a_1, \ldots, a_k, y_1, \ldots, y_{n-k})$$
>
> is in P (as a function on $n-k = n/(c+1)$ variables).

Observe that testing P' on a given n-variate Boolean function f can be done in time $O(2^k) \cdot \mathsf{poly}(2^{n-k}) \leq \mathsf{poly}(2^n)$; so we have constructivity for P'. Next, if $f : \{0,1\}^n \to \{0,1\}$ has a Λ circuit of size less $s'(n)$, then each restricted subfunction $f_a : \{0,1\}^{n-k} \to \{0,1\}$ has a Λ circuit of size less than $s(n-k) \leq s'(n)$. Finally, a random function $f : \{0,1\}^n \to \{0,1\}$ yields 2^k independent random subfunctions, on $n-k$ variables each, and the probability that at least one of these $(n-k)$-variate functions satisfies P is at least $1 - (1 - 2^{-c(n-k)})^{2^k} = 1 - (1 - 2^{-k})^{2^k}$, which is at least $1/2$, as required. ◀

2.4 NW generator

▶ **Definition 2.8** (NW Design). For parameters $n, m, L \in \mathbb{N}$, a sequence of sets $S_1, \ldots, S_L \subseteq [m]$ is called an *NW design* if

- $|S_i| = n$, for all $1 \leq i \leq L$, and
- $|S_i \cap S_j| \leq \log L$, for all $1 \leq i \neq j \leq L$.

It is well-known that NW designs exist and can be efficiently constructed for any n, $m = O(n^2)$, and $L < 2^n$ [28]. In Section 3.1 below, we review the construction of NW designs from [28], and show that it can be implemented in $\mathsf{AC}^0[p]$ (Theorem 3.3). The efficiency of this construction of designs is necessary for our transfer theorem.

▶ **Definition 2.9** (NW Generator)**.** Let $f\colon \{0,1\}^n \to \{0,1\}$. For $m = n^2$ and a stretch function $L(m)\colon \mathbb{N} \to \mathbb{N}$, where $L(m) < 2^n$, let $S_1, \ldots, S_L \subseteq [m]$ be an NW design. Define the NW generator $G_f\colon \{0,1\}^m \to \{0,1\}^{L(m)}$ as:

$$G_f(z) = f(z|_{S_1})f(z|_{S_2})\ldots f(z|_{S_{L(m)}}), \tag{1}$$

where $z|_S$ denotes the $|S|$-length bit-string obtained by restricting z to the bit positions indexed by the set S.

Recall the notion of a distinguisher, a circuit that breaks a given generator.

▶ **Definition 2.10** (Distinguishers)**.** Let $L\colon \mathbb{N} \to \mathbb{N}$ be a stretch function, let $0 < \epsilon < 1$ be an error bound, and let $\mathcal{G} = \{g_m\colon \{0,1\}^m \to \{0,1\}^{L(m)}\}$ be a sequence of functions. Define $\mathsf{DIS}(\mathcal{G}, \epsilon)$ to be the set of all Boolean circuits D on $L(m)$-bit inputs satisfying:

$$\Pr_{z\in\{0,1\}^m}[D(g_m(z))] - \Pr_{y\in\{0,1\}^{L(m)}}[D(y)] > \epsilon.$$

▶ **Theorem 2.11** (NW Reconstruction [28, 22])**.** *We have*

$$\textsc{Cons}^f(\mathsf{DIS}(G_f, 1/5) \to \widetilde{\mathsf{CKT}}(f, 1/2 - 1/L(m));\ \mathsf{poly}(L(m))).$$

NW Reconstruction Algorithm. Since the reconstruction algorithm from the proof of Theorem 2.11 above is an essential ingredient in our learning algorithms, we sketch this algorithm below (omitting the correctness proof, which can be found in [28, 22]).

Let $G_f\colon \{0,1\}^m \to \{0,1\}^L$ be the NW generator based on a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, using the NW design $S_1, \ldots, S_L \subseteq [m]$. Suppose D is a distinguisher for G_f such that $D \in \mathsf{DIS}(G_f, 1/5)$. The following randomized algorithm will produce, with probability at least $1/\mathsf{poly}(L)$, a circuit C computing f on at least $1/2 + \Omega(1/L)$ fraction of inputs. It consists of a preprocessing stage, and a circuit construction stage.

Preprocessing

1. Pick a random $i \in [L]$.
2. For each $i \le j \le L$, fix the jth input of the distinguisher D to a random bit w_j.
3. For each $j \in [m] \setminus S_i$, fix the jth input of the generator G_f to a random bit z_j.
4. For each $1 \le j < i$, enumerate all $x \in \{0,1\}^n$ consistent with the partial assignment $z|_{S_j}$ from the previous step, query $f(x)$, and build the table T of pairs $(x, f(x))$.

Circuit construction

Using T, w_j's, and z_j's from preprocessing, build a circuit C following the template:
"On input $x \in \{0,1\}^n$,

1. Assign the inputs $z|_{S_i}$ of G_f to x, getting a fully specified input $z \in \{0,1\}^m$.
2. For each $1 \le j < i$, fix the jth input of D to $w_j = f(z|_{S_j})$, via table lookup in T.
3. If $D(w_1, \ldots, w_L) = 1$, then output w_i; otherwise, output $1 - w_i$."

To boost the probability of producing a good circuit C, we repeat the algorithm above $\mathsf{poly}(L)$ times, and estimate, using random sampling and membership queries to f, the agreement between f and each produced circuit C. We output the best circuit on our list.

3 Black-box generators

The main tool we need for our learning algorithms is a transformation, which we call *black-box generator*, taking a given function $f\colon \{0,1\}^n \to \{0,1\}$ to a family $G = \{g_z\}_{z\in I}$ of new Boolean functions $g_z\colon \{0,1\}^{n'} \to \{0,1\}$ satisfying the following properties:

Nonuniform Efficiency: Each function g_z has "small" circuit complexity relative to the circuit complexity of f.

Reconstruction: Any circuit distinguishing a random function g_z (for a uniformly random $z \in I$) from a random n'-variate Boolean function can be used (by an efficient randomized algorithm with oracle access to f) to construct a good approximating circuit for f.

Once we have such a black-box generator, we get our learning algorithm as follows. To learn a function $f\colon \{0,1\}^n \to \{0,1\}$, use the natural property as a distinguisher that rejects (the truth tables of) all functions g_z, $z \in I$, but accepts a constant fraction of truly random functions; apply the efficient reconstruction procedure to learn a circuit approximating f. Intuitively, we use the nonuniform efficiency property to argue that if f is an easy function in some circuit class Λ, then so is each function g_z, $z \in I$.

Next we give a more formal definition of a black-box generator. For a function f, we denote by Λ^f the class of oracle circuits in Λ that have f-oracle gates. Also recall that $\Lambda[s]$ denotes the class of Λ-circuits of size at most s.

▶ **Definition 3.1** (Black-Box (ϵ, L)-Generator Within Λ)**.** For a given error parameter $\epsilon\colon \mathbb{N} \to [0,1]$ and a stretch function $L\colon \mathbb{N} \to \mathbb{N}$, a *black-box* (ϵ, L)*-generator within* Λ is a mapping $\textsc{Gen}$ that associates with a given function $f\colon \{0,1\}^n \to \{0,1\}$ a family $\textsc{Gen}(f) = \{g_z\}_{z\in\{0,1\}^m}$ of Boolean functions $g_z\colon \{0,1\}^\ell \to \{0,1\}$, where $\ell = \log L(n)$, satisfying the following conditions for every $f : \{0,1\}^n \to \{0,1\}$:

Small Family Size: $m \le \mathsf{poly}(n, 1/\epsilon)$,

Nonuniform Λ-Efficiency: for all $z \in \{0,1\}^m$, $g_z \in \Lambda^f[\mathsf{poly}(m)]$, and

Reconstruction: $\textsc{Cons}^f(\mathsf{DIS}(\textsc{Gen}(f), 1/5) \to \widetilde{\mathsf{CKT}}(f,\epsilon); \mathsf{poly}(n, 1/\epsilon, L(n)))$, where we think of $\textsc{Gen}(f)$ as the distribution over the truth tables of functions $g_z \in \textsc{Gen}(f)$, for uniformly random $z \in \{0,1\}^m$.

We will prove the following.

▶ **Theorem 3.2.** *Let p be any prime. For every $\epsilon\colon \mathbb{N} \to [0,1]$ and $L\colon \mathbb{N} \to \mathbb{N}$ such that $L(n) \le 2^n$, there exists a black-box (ϵ, L)-generator within $\mathsf{AC}^0[p]$.*

We will use the NW generator as our black-box generator. For it to be within $\mathsf{AC}^0[p]$, we need NW designs to be computable within $\mathsf{AC}^0[p]$. We prove the following in the next subsection (see the proof of Theorem 3.7 in Section 3.1).

▶ **Theorem 3.3.** *Let p be any prime. There exists a constant $d_{MX} \ge 1$ such that, for any n and $L < 2^n$, there exists an NW design $S_1, \ldots, S_L \subseteq [m]$ with $m = O(n^2)$, each $|S_i| = n$, and $|S_i \cap S_j| \le \ell = \log L$ for all $1 \le i \ne j \le L$, such that the function $MX_{NW} : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^n$, defined by $MX_{NW}(i,z) = z|_{S_i}$, is computable by an $\mathsf{AC}^0[p]$ circuit of size $O(\ell \cdot n^3 \log n)$ and depth d_{MX}.*

Another ingredient we need for the proof of Theorem 3.2 is the following notion of black-box amplification. Let Λ be any circuit class.

▶ **Definition 3.4** (Black-Box (ϵ,δ)-Amplification within Λ)**.** For given $\epsilon, \delta > 0$, (ϵ,δ)*-amplification within* Λ is a mapping that associates with a given function $f\colon \{0,1\}^n \to \{0,1\}$ its *amplified* version, $\textsc{Amp}(f)\colon \{0,1\}^{n'} \to \{0,1\}$, satisfying the following conditions for every $f\colon \{0,1\}^n \to \{0,1\}$:

Short Input: $n' \leq \mathsf{poly}(n, 1/\epsilon, \log 1/\delta)$,
Nonuniform Λ-Efficiency: $\textsc{Amp}(f) \in \Lambda^f[\mathsf{poly}(n')]$,
Uniform P-Efficiency: $\textsc{Amp}(f) \in \mathsf{P}^f$, and
Reconstruction: $\textsc{Cons}^f(\widetilde{\mathsf{CKT}}(\textsc{Amp}(f), 1/2 - \delta) \to \widetilde{\mathsf{CKT}}(f, \epsilon);\ \mathsf{poly}(n, 1/\epsilon, 1/\delta))$.

We prove the following in the next section (see Theorems 4.3 and 4.8).

▶ **Lemma 3.5.** *Let p be any fixed prime. For all $0 < \epsilon, \delta < 1$, there is black-box (ϵ, δ)-amplification within $\mathsf{AC}^0[p]$.*

Now we are ready to prove Theorem 3.2.

Proof of Theorem 3.2. For a given n-variate Boolean function f, consider its amplified version $f^\star = (\epsilon(n), 1/L(n))\text{-}\textsc{Amp}(f)$, for the black-box amplification within $\mathsf{AC}^0[p]$ that exists by Lemma 3.5. We have that $f^\star$ is a function on $n' = \mathsf{poly}(n, 1/\epsilon, \log L) = \mathsf{poly}(n, 1/\epsilon)$ variables (using the assumption that $L(n) \leq 2^n$).

Let $G_{f^\star} \colon \{0,1\}^m \to \{0,1\}^{L(n)}$ be the NW generator based on the function $f^\star$, with the seed size $m = (n')^2$. Define $\textsc{Gen}(f) = \{g_z\}_{z \in \{0,1\}^m}$, where $g_z = G_{f^\star}(z)$. We claim that this $\textsc{Gen}(f)$ is an (ϵ, L)-black-box generator within $\mathsf{AC}^0[p]$. We verify each necessary property:

Small Family Size: $m = (n')^2 \leq \mathsf{poly}(n, 1/\epsilon)$.

Nonuniform $\mathsf{AC}^0[p]$-Efficiency: We know that $f^\star = \textsc{Amp}(f) \in (\mathsf{AC}^0[p])^f[\mathsf{poly}(m)]$. For each fixed $z \in \{0,1\}^m$, we have $g_z(i) = (G_{f^\star}(z))_i$, for $i \in \{0,1\}^\ell$, where $\ell = \log L(n)$. By the definition of the NW generator, $g_z(i) = f^\star(z|_{S_i})$. By Theorem 3.3, the restriction $z|_{S_i}$, as a function of z and i, is computable in $\mathsf{AC}^0[p]$ of size $\mathsf{poly}(n')$ and some fixed depth d_{MX}. It follows that each g_z is computable in $(\mathsf{AC}^0[p])^f[\mathsf{poly}(m)]$.

Reconstruction: The input to reconstruction is $D \in \mathsf{DIS}(G_{\textsc{Amp}(f)}, 1/5)$. Let $\mathcal{M}_{NW}$ be the reconstruction machine from the NW construction, and let $\mathcal{M}_{\textsc{Amp}}$ be the reconstruction machine from $(\epsilon, 1/L)$-amplification. We first run $\mathcal{M}_{NW}^{\textsc{Amp}(f)}(D)$ to get, in time $\mathsf{poly}(L)$, a circuit $C \in \widetilde{\mathsf{CKT}}(\textsc{Amp}(f), 1/2 - 1/L(n))$; note that we can provide this reconstruction algorithm oracle access to $\textsc{Amp}(f)$, since $\textsc{Amp}(f) \in \mathsf{P}^f$ by the uniform P-efficiency property of black-box amplification. Next we run $\mathcal{M}_{\textsc{Amp}}^f$ on C to get $C' \in \widetilde{\mathsf{CKT}}(f, \epsilon)$, in time $\mathsf{poly}(n, 1/\epsilon, L(n))$. ◀

3.1 NW designs in $\mathsf{AC}^0[p]$

Here we show that the particular NW designs we need in our compression and learning algorithms can be constructed by small $\mathsf{AC}^0[p]$ circuits, for any fixed prime p. Consider an NW design $S_1, \ldots, S_L \subseteq [m]$, for $m = O(n^2)$, where

- each set S_i is of size n,
- the number of sets is $L = 2^\ell$ for $\ell \leq n$, and
- for any two distinct sets S_i and S_j, $i \neq j$, we have $|S_i \cap S_j| \leq \ell$.

We show a particular construction of such a design that has the following property: the index set $[m]$ is partitioned into n disjoint subsets $U_1, \ldots, U_n$ of equal size $(m/n) \in O(n)$. For each $1 \leq i \leq L$, the set S_i contains exactly one element from each subset U_j, over all $1 \leq j \leq n$. For $1 \leq j \leq n$ and $1 \leq k \leq O(n)$, we denote by $(U_j)_k$ the kth element in the subset U_j.

To describe such a design, we use the following Boolean function g: for $1 \le i \le L$, $1 \le j \le n$, and $1 \le k \le O(n)$, we define $g(i,j,k) = 1$ iff $(U_j)_k \in S_i$. We will prove the following.

▶ **Theorem 3.6.** *There exists a constant $d_{NW} \ge 1$ such that, for any prime p, there exists a family of functions $g : \{0,1\}^{\ell+2\log n} \to \{0,1\}$ that are the characteristic functions for some NW design with the parameters as above, so that $g \in \mathsf{AC}^0[p]$ of size $O(n^2 \log n)$ and depth d_{NW}.*

Proof. Recall the standard construction of NW designs from [28]. Let F be a field of size $O(n)$. Consider an enumeration of L polynomials of degree at most ℓ over F, with all coefficients in $\{0,1\}$; there are at least $2^\ell = L$ such polynomials. We associate each such polynomial with a binary string $i = i_1 \dots i_\ell \in \{0,1\}^\ell$, so that i corresponds to the polynomial

$$A_i(x) = \sum_{j=1}^{\ell} i_j \cdot x^{j-1}$$

over the field F. Let $r_1, \dots, r_{|F|}$ be some canonical enumeration of the elements of F. For each binary string $i \in \{0,1\}^\ell$, we define a set $S_i = \{(r_j, A_i(r_j)) \mid 1 \le j \le n\}$. Note that $|S_i| = n$, and S_i defines a set of n pairs in the universe $F \times F$ of $O(n^2)$ pairs (hence the universe size for this construction is $O(n^2)$). Finally, any two distinct degree $(\ell-1)$ polynomials $A_i(x)$ and $A_j(x)$ may agree on at most ℓ points $r \in F$, and so we have $|S_i \cap S_j| \le \ell$ for the sets S_i and S_j, corresponding to the polynomials $A_i(x)$ and $A_j(x)$.

Arrange the elements of the universe $[m]$ on an $n \times (m/n)$ grid. The n rows of the grid are indexed by the first n field elements $r_1, \dots, r_n$, and the columns by all fields elements $r_1, \dots, r_|F|$. For each j, $1 \le j \le n$, define U_j to be the elements of $[m]$ that belong to the row j of the grid. We get that every set $S_i = \{(r_j, A_i(r_j)) \mid 1 \le j \le n\}$ picks exactly one element from each of the n sets $U_1, \dots, U_n$.

We will argue that this particular design construction is computable in $\mathsf{AC}^0[p]$ of size polynomial in ℓ, for each prime p. Let p be any fixed prime (which we think of as a constant). Let F be an extension field over $\mathbf{GF}(p)$ of the least size so that $|F| \ge n$; such a field is described by some polynomial over $\mathbf{GF}(p)$ of degree $O(\log_p n)$, and is of size at most $pn = O(n)$. As before, let $r_1, \dots, r_{|F|}$ be a canonical enumeration of the field elements in F.

Define the following $n \times \ell$ matrix M: for $1 \le j \le n$ and $1 \le k \le \ell$, we have $M_{j,k} = (r_j)^k$, where the power $(r_j)^k$ is computed within the field F. Then the values $A_i(r_1), \dots, A_i(r_n)$ may be read off from the column vector obtained by multiplying the matrix M by the column vector $i \in \{0,1\}^\ell$, in the field F. For a particular $1 \le j \le n$, we have $A_i(r_j) = \sum_{k=1}^{\ell} M_{j,k} \cdot i_k$. Since each $i_k \in \{0,1\}$, the latter reduces to the task of adding a subset of ℓ field elements. Each field element of F is a polynomial over $\mathbf{GF}(p)$ of degree $k \le O(\log n)$, and so adding a collection of elements from F reduces to the coordinate-wise summation modulo p of k-element vectors in $(\mathbf{GF}(p))^k$. The latter task is easy to do in $\mathsf{AC}^0[p]$[4].

For any $1 \le i \le L$, $1 \le j \le n$, and $1 \le k \le |F|$, $g(i,j,k) = 1$ iff $A_i(r_j) = r_k$. To compute $g(i,j,k)$, we need to evaluate the polynomial $A_i(x)$ at r_j, and then check if the result is

[4] We code elements of $\mathbf{GF}(p)$ by p-wire bundles, where wire i is on iff the bundle codes the ith element of $\mathbf{GF}(p)$. An addition, multiplication, or inverse in the field $\mathbf{GF}(p)$ can be implemented in AC^0. To add up a tuple of field elements, we first convert each field element from the representation above to the unary representation (using constant-depth selection circuits). Then we lead these unary encodings into a layer of p gates, $\oplus_p^j$, for $0 \le j \le p-1$, where $\oplus_p^j$ is the gate $\oplus_p$ with $p-j$ extra inputs 1. Thus the gate $\oplus_p^j$ on inputs $x_1, \dots, x_n \in \mathbf{GF}(p)$ outputs 1 iff $x_1 + \dots + x_n = j \mod p$. Note that exactly one of the gates $\oplus_p^j$ will output 1, giving us the desired field element in our encoding.

equal to r_k. To this end, we "hard-code" the matrix M into the circuit (which incurs the cost at most $O(n\ell \log n)$ bits of advice). We compute $A_i(r_j)$ by computing the matrix-vector product $M \cdot i$, and restricting to the jth coordinate of the resulting column vector. This computation involves $O(\log n)$ summations of ℓ field elements of $\mathbf{GF}(p)$ modulo p, over n rows of the matrix M. The resulting field element is described an $O(\log n)$-element vector of elements from the underlying field $\mathbf{GF}(p)$. Using $O(\log n)$ operations over $\mathbf{GF}(p)$, we can check if this vector equals the vector corresponding to r_k.

It is easy to see that this computation can be done in some fixed constant depth d_{NW} by an $\mathsf{AC}^0[p]$ circuit of size $O(\ell \cdot n \log n)$, which can be bounded by $O(n^2 \log n)$, as required. ◀

As a corollary, we get Theorem 3.3, which we re-state below.

▶ **Theorem 3.7.** *Let p be any prime. There exists a constant $d_{MX} \geq 1$ such that, for any n and $L < 2^n$, there exists an NW design $S_1, \ldots, S_L \subseteq [m]$ with $m = O(n^2)$, each $|S_i| = n$, and $|S_i \cap S_j| \leq \ell = \log L$ for all $1 \leq i \neq j \leq L$, such that the function $MX_{NW} : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^n$, defined by $MX_{NW}(i,z) = z|_{S_i}$, is computable by an $\mathsf{AC}^0[p]$ circuit of size $O(\ell \cdot n^3 \log n)$ and depth d_{MX}.*

Proof. Let $g(i,j,k)$ be the characteristic function for the NW design from Theorem 3.6, where $|i| = \ell$, $|j| = \log n$, and $|k| = \log n + \log c$, for some constant $c \geq 1$. We have $g \in \mathsf{AC}^0[p]$ of size $O(\ell \cdot n \log n)$ and depth d_{NW}. Let $U_1, \ldots, U_n \subseteq [m]$ be the sets of size cn each that partition $[m]$ so that every S_i contains exactly one element from every U_j, $1 \leq j \leq n$.

Let $i_1, \ldots, i_\ell$ and $z_1, \ldots, z_m$ denote the input gates of MX_{NW}, and let $y_1, \ldots, y_n$ denote its output gates. Associate each gate y_j with the set U_j of indices in $[m]$, for $1 \leq j \leq n$. For each $1 \leq i \leq L$ and each $1 \leq j \leq n$, define

$$y_j = \vee_{k=1}^{cn} \; g(i,j,k) \wedge (z|_{U_j})_k.$$

Clearly, the defined circuit computes MX_{NW}. It has size $O(\ell \cdot n^3 \log n)$ and depth $d_{MX} \leq d_{NW} + 2$, as required. ◀

Let G_f be the NW generator based on a function f, using the NW design $S_1, \ldots, S_L$ from Theorem 3.3. For each fixed seed z, define the function $g_z : \{0,1\}^\ell \to \{0,1\}$, for $\ell = \log L$, as $g_z(i) = (G_f(z))_i = f(z|_{S_i})$, where $1 \leq i \leq L$. By Theorem 3.3, we get $g_z \in (\mathsf{AC}^0[p])^f$. See Figure 1 for the description of a small circuit for g_z that combines the $\mathsf{AC}^0[p]$ circuit for MX_{NW} with a circuit for f.

4 Black-box amplification

Here we show that black-box amplification (Definition 3.4) is possible within $\mathsf{AC}^0[p]$, for any prime $p \geq 2$, as required for the proof that black-box generators within $\mathsf{AC}^0[p]$ exist (Theorem 3.2). For $\mathsf{AC}^0[2]$, we shall use standard hardness amplification tools from pseudorandomness: Direct Product and XOR construction. For $\mathsf{AC}^0[p]$, $p \neq 2$, we will need to use something else in place of XOR, as small $\mathsf{AC}^0[p]$ circuits can't compute Parity [35]. We will replace XOR with a MOD_p function, also using an efficient conversion from $\{0,1,\ldots,p-1\}$-valued functions to Boolean functions, which preserves the required amplification parameters.

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and a parameter $k \in \mathbb{N}$, the k-wise direct product of f is $f^k : \{0,1\}^{nk} \to \{0,1\}^k$, where $f^k(x_1, \ldots, x_k) = (f(x_1), \ldots, f(x_k))$ for $x_i \in \{0,1\}^n$, $1 \leq i \leq k$. It is well-known that the Direct Product (DP) construction amplifies hardness of a given function f in the sense that a circuit somewhat nontrivially

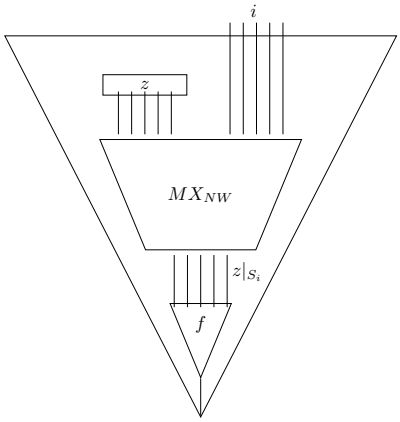

Figure 1 A circuit for $g_z(i) = f(z|_{S_i})$.

approximating the function f^k may be used to get a new circuit that approximates the original function f quite well [13], and, moreover, this new circuit for f can be constructed efficiently uniformly [22]. We shall use the following algorithm due to [17] that has optimal parameters (up to constant factors).

▶ **Theorem 4.1** (DP Reconstruction [17]). *There is a constant c and a probabilistic algorithm $\mathcal{A}$ with the following property. Let $k \in \mathbb{N}$, and let $0 < \epsilon, \delta < 1$ be such that $\delta > e^{-\epsilon k/c}$. For a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, let C' be any circuit in $\widetilde{\mathsf{CKT}}(f^k, 1-\delta)$. Given such a circuit C', algorithm $\mathcal{A}$ outputs with probability $\Omega(\delta)$ a circuit $C \in \widetilde{\mathsf{CKT}}(f, \epsilon)$.*

DP Reconstruction Algorithm. The algorithm $\mathcal{A}$ in Theorem 4.1 is a uniform randomized NC^0 algorithm (with one C'-oracle gate), and the produced circuit C is an AC^0 circuit of size $\mathsf{poly}(n, k, \log 1/\epsilon, 1/\delta)$ (with $O((\log 1/\epsilon)/\delta)$ of C'-oracle gates). We sketch this algorithm below. It consists of a preprocessing stage and a circuit construction stage. For simplicity, we allow the constructed circuit to be randomized; it can easily be made deterministic by choosing all required randomness in the preprocessing stage.

Preprocessing

Randomly pick a set B_0 of k strings in $\{0,1\}^n$. Pick a random subset $A \subset B_0$ of size $k/2$. Evaluate C' on a k-tuple $\vec{b_0}$ that is a random permutation of the strings in B_0, and note the answers $\vec{a}$ given by $C'(\vec{b_0})$ for the strings in A.

Circuit construction

Using A and $\vec{a}$ from preprocessing, build a randomized circuit C following the template:
"On input $x \in \{0,1\}^n$, if $x \in A$, then output the corresponding answer in $\vec{a}$. Otherwise, for $m = O((\log 1/\epsilon)/\delta)$ times,
1. sample a random k-set B such that $A \cup \{x\} \subset B$;
2. evaluate C' on a k-tuple $\vec{b}$ that is a random permutation of the strings in B;
3. if the answers of $C'(\vec{b})$ for A are consistent with $\vec{a}$, then output $C'(\vec{b})_x$ (the answer C' gave for x), and stop.
If no output is produced after m iterations, output a random bit."

Next, we need to convert a non-Boolean function $f^k\colon \{0,1\}^{kn} \to \{0,1\}^k$ into a Boolean function h such that a circuit approximately computing h would uniformly efficiently yield

a circuit approximately computing f^k, where the quality of approximation is essentially preserved. To this end, we "collapse" the k-bit output of f^k to a single number modulo a prime p, using the Goldreich-Levin construction [12] over $F = \mathbf{GF}(p)$: For $g\colon \{0,1\}^m \to \{0,1\}^k$, define $g^{GL}\colon \{0,1\}^m \times F^k \to F$ to be

$$g^{GL}(x_1,\ldots,x_m,r_1,\ldots,r_k) = \sum_{i=1}^{k} r_i \cdot g(x_1,\ldots,x_m)_i,$$

where all arithmetic is over the field F.

We will describe an efficient reconstruction algorithm that takes a circuit computing the function g^{GL} on more than $1/p+\gamma$ fraction of inputs, for some $\gamma > 0$, and produces a circuit that computes g on more than $\Omega(\gamma^3)$ fraction of inputs. The main ingredient of this algorithm is the following result first proved by Goldreich and Levin [12] for the case of $p = 2$, and later generalized by Goldreich, Rubinfeld, and Sudan [14] to all primes p.

▶ **Theorem 4.2** (GL Reconstruction [12, 14]). *There is a probabilistic algorithm $\mathcal{A}$ with the following property. Let $h \in F^k$ be arbitrary, and let $B\colon F^k \to F$ be such that $\mathbf{Pr}_{r\in F^k}[B(r) = \langle h, r\rangle] \geq 1/p+\gamma$ for some $\gamma > 0$, where $\langle x, y\rangle = \sum_{i=1}^{k} x_i \cdot y_i \mod p$. Then, given oracle access to B and the parameter γ, the algorithm $\mathcal{A}$ runs in time $\mathsf{poly}(k, 1/\gamma)$ and outputs a list of size $O(1/\gamma^2)$ such that, with probability at least $1/2$, the tuple h is on the list.*

GL Reconstruction Algorithm. We sketch below the algorithm $\mathcal{A}$ of Theorem 4.2.

Proceed in k rounds, maintaining after round i a list $\mathcal{H}_i$ of length-i tuples in F^i; the list after round k is the final output. In round i:

1. Extend each tuple in $\mathcal{H}_{i-1}$ by one element in all $|F|$ possible ways.
2. For each extended tuple $\vec{c} \in F^i$, include $\vec{c}$ in $\mathcal{H}_i$ iff it passes the following test:

   Randomly pick $m = \mathsf{poly}(k/\gamma)$ tuples $\vec{s}_1,\ldots,\vec{s}_m \in F^{k-i}$. For each $\vec{s}_i$ and each $\sigma \in F$, estimate $\mathbf{Pr}_{\vec{r}\in F^i}[B(\vec{r},\vec{s}) = \langle \vec{c},\vec{r}\rangle + \sigma]$. If at least one of these estimates is significantly larger than $1/p$, then accept; otherwise, reject.

4.1 Case of $\mathsf{AC}^0[2]$

Theorems 4.1 and 4.2 imply the following.

▶ **Theorem 4.3** (Black-Box Amplification within $\mathsf{AC}^0[2]$). *For any $0 < \epsilon, \gamma < 1$, there is black-box (ϵ,γ)-amplification within $\mathsf{AC}^0[2]$.*

Proof. Given $f\colon \{0,1\}^n \to \{0,1\}$ in $\mathsf{AC}^0[2]$ of size s, and given $0 < \epsilon, \delta < 1$, define $\textsc{Amp}(f)$ as follows:

1. Set $k = \lceil (3c)\cdot 1/\epsilon \cdot \ln 1/\gamma\rceil + 1$, where c is the constant in Theorem 4.1.
2. Define g to be the direct product $f^k\colon \{0,1\}^{nk} \to \{0,1\}^k$.
3. Define $\textsc{Amp}(f)$ to be $g^{GL}\colon \{0,1\}^{nk+k} \to \{0,1\}$ over $F = \mathbf{GF}(2)$.

▶ **Claim 4.4.** *For any $\gamma > 0$, we have*

$$\textsc{Cons}^f(\widetilde{\mathsf{CKT}}(g^{GL}, 1/2-\gamma) \to \widetilde{\mathsf{CKT}}(g, 1-\Omega(\gamma^3)); \mathsf{poly}(n,k,1/\gamma)).$$

Proof. Suppose we are given a circuit $C' \in \widetilde{\mathsf{CKT}}(g^{GL}, 1/2-\gamma)$. Let $\mathcal{A}_{GL}$ be the Goldreich-Levin algorithm of Theorem 4.2. Consider the following algorithm $\mathcal{A}_1$ that attempts to compute g:

For a given input $x \in \{0,1\}^{nk}$, define a circuit $B_x(r) := C'(x,r)$, for $r \in \{0,1\}^k$. Run $\mathcal{A}_{GL}$ on B_x, with parameter $\gamma/2$, getting a list L of k-bit strings. Output a uniformly random k-bit string from the list L.

Correctness Analysis of $\mathcal{A}_1$: By averaging, for each of at least $\gamma/2$ fraction of strings $x \in \{0,1\}^{nk}$, the circuit $B_x(r) := C'(x,r)$ agrees with $g^{GL}(x,r) = \langle g(x), r\rangle$ on at least $1/2 + \gamma/2$ fraction of strings $r \in \{0,1\}^k$. For each such x, the circuit B_x satisfies the condition of Theorem 4.2, and so the GL algorithm will find, with probability at least $1/2$, a list L of $O(1/\gamma^2)$ strings in $\{0,1\}^k$ that contains the string $g(x)$. Conditioned on the list containing the string $g(x)$, if we output a random string on that list, we get $g(x)$ with probability at least $1/|L| \geq \Omega(\gamma^2)$. Overall, the fraction of inputs x where $\mathcal{A}_1$ correctly computes $g(x)$ is at least $\frac{\gamma}{2} \cdot \frac{1}{2} \cdot \Omega(\gamma^2) \geq \Omega(\gamma^3)$. The runtime of $\mathcal{A}_1$ is $\mathsf{poly}(|C'|, k, n, 1/\gamma)$. ◀

By Theorem 4.1, we have

$$\textsc{Cons}^f(\widetilde{\mathsf{CKT}}(f^k, 1-\mu) \to \widetilde{\mathsf{CKT}}(f,\epsilon); \mathsf{poly}(n,k,\log 1/\epsilon, 1/\mu)),$$

as long as $\mu > e^{-\epsilon k/c}$, for some fixed constant $c > 0$. Combining this with Claim 4.4 yields

$$\textsc{Cons}^f(\widetilde{\mathsf{CKT}}(\textsc{Amp}(f), 1/2-\gamma) \to \widetilde{\mathsf{CKT}}(f,\epsilon); \mathsf{poly}(n,1/\epsilon,1/\gamma)),$$

as long as $\gamma^3 > e^{-\epsilon k/c}$, which is equivalent to $\gamma > e^{-\epsilon k/c'}$, for $c' = 3c$. Our choice of k satisfies this condition.

Finally, we verify that $\textsc{Amp}(f)$ also satisfies the other conditions of black-box amplification:

- $(f^k)^{GL}$ is defined on inputs of size $kn + k \leq O(n \cdot 1/\epsilon \cdot \log 1/\gamma)$.
- If $f \in \mathsf{AC}^0[2]$ of size s, then f^k is in $\mathsf{AC}^0[2]$ of size $O(s \cdot k) = O(s \cdot 1/\epsilon \cdot \log 1/\gamma)$, and $(f^k)^{GL}$ is of size at most the additive term $O(k)$ larger.
- $(f^k)^{GL}$ is in P^f.

Thus, $\textsc{Amp}(f)$ defined above is black-box (ϵ,γ)-amplification of f, as required. ◀

4.2 Case of $\mathsf{AC}^0[p]$ for primes $p > 2$

For $\mathsf{AC}^0[p]$ circuits, with $p > 2$, we can't use the XOR construction above, as Parity is not computable by small $\mathsf{AC}^0[p]$ circuits [35]. A natural idea to amplify a given function f is to apply the Goldreich-Levin construction g^{GL} over the field $F = \mathbf{GF}(p)$ to the direct-product function $g = f^k$, for an appropriate value of k. Theorem 4.2 guarantees that if we have a circuit that computes g^{GL} on more than $1/p + \gamma$ fraction of inputs, then we can efficiently construct a circuit that computes g on $\Omega(\gamma^3)$ fraction of inputs; the proof is identical to that of Claim 4.4 inside the proof of Theorem 4.3 for the case of $\mathsf{AC}^0[2]$ above.

The only problem is that the function g^{GL} defined here is *not* Boolean-valued, but we need a Boolean function to plug into the NW generator in order to complete our construction of a black-box generator within $\mathsf{AC}^0[p]$. We need to convert g^{GL} into a Boolean function h in such a way that if h can be computed by some circuit on at least $1/2 + \mu$ fraction of inputs, then g^{GL} can be computed by a related circuit on at least $1/p + \mu'$ fraction of inputs, where μ and μ' are close to each other.

We use von Neumann's idea for converting a coin of unknown bias into a perfectly unbiased coin [40]. Given a coin that is "heads" with some (unknown) probability $0 < p < 1$, flip the coin twice in a row, independently, and output 0 if the trials were ("heads", "tails"), or 1 if the trials were ("tails", "heads"). In case both trials came up the same (i.e., both "heads", or both "tails"), flip the coins again.

Observe that, conditioned on producing an answer $b \in \{0,1\}$, the value b is uniform over $\{0,1\}$ (as both conditional probabilities are equal to $p(1-p)/(1-p^2-(1-p)^2)$. The probability of not producing an answer in one attempt is $p^2+(1-p)^2$, the collision probability of the distribution $(p, 1-p)$. If p is far away from 0 and 1, the probability that we need to repeat the flipping for more than t trials diminishes exponentially fast in t.

In our case, we can think of the value of g^{GL} on a uniformly random input as a distribution over F. Assuming that this distribution is close to uniform over F, we will define a new Boolean function h based on g^{GL} so that the output of h on a uniformly random input is close to uniform over $\{0,1\}$. Our analysis of h will be constructive in the following sense. If we are given a circuit that distinguishes the distribution of the outputs of h from uniform, then we can efficiently construct a circuit that distinguishes the distribution of the outputs of g^{GL} from uniform over F. Finally, using the standard tools from pseudorandomness (converting distinguishers into predictors), we will efficiently construct from this distinguisher circuit a new circuit that computes g^{GL} on noticeably more than $1/p$ fraction of inputs.

The construction of this function h follows the von Neumann trick above. Formally we have the following.

▶ **Definition 4.5** (von Neumann trick function)**.** For an integer parameter $t > 0$, define the function $E^{vN}\colon (F^2)^t \to \{0,1\}$ as follows: For pairs $(a_1,b_1),\ldots,(a_t,b_t) \in F \times F$, set

$$E^{vN}((a_1,b_1),\ldots,(a_t,b_t)) = \begin{cases} 1 & \text{if, for each } 1 \le i \le t,\ a_i = b_i \\ 1 & \text{if } (a_i,b_i) \text{ is the first unequal pair and } a_i > b_i \\ 0 & \text{if } (a_i,b_i) \text{ is the first unequal pair and } a_i < b_i \end{cases}$$

It is not hard to see that E^{vN} is computable in AC^0. Moreover, for independent uniformly distributed inputs, the output of E^{vN} is a random coin flip, with bias at most $(1/p)^t$.

▶ **Claim 4.6.** *Let $\mathcal{F}$ be the uniform distribution over the field $F = \mathbf{GF}(p)$, and let $\mathcal{G} = (\mathcal{F}^2)^t$ be the uniform distribution over sequences of t pairs of elements from F. Then $\left|\mathbf{Pr}_{r\in\mathcal{G}}[E^{vN}(r)=1] - \mathbf{Pr}_{r\in\mathcal{G}}[E^{vN}(r)=0]\right| \le p^{-t}$.*

Proof. Conditioned on having some unequal pair in the sample from $\mathcal{G}$, the bias of the random variable $E^{vN}(\mathcal{G})$ is 0. Conditioned on having no such unequal pair, the bias is at most 1. Note that the collision probability of the uniform distribution over $\mathbf{GF}(p)$ is $\sum_{i=1}^{p} p^{-2} = p^{-1}$. So the probability of having collisions in all t independent samples from $\mathcal{F}^2$ is p^{-t}. Thus, the overall bias is at most p^{-t}. ◀

Next, given $g^{GL}\colon D \to F$, for the domain $D = \{0,1\}^m \times F^k$, define $h^{vN}\colon (D^2)^t \to \{0,1\}$ as follows:

$$h^{vN}((a_1,b_1),\ldots,(a_t,b_t)) = E^{vN}((g^{GL}(a_1),g^{GL}(b_1)),\ldots,(g^{GL}(a_t),g^{GL}(b_t))).$$

▶ **Theorem 4.7.** *For any $0 < \mu < 1$ and $1 > \gamma > \Omega(\mu/(\log 1/\mu))$, we have*

$$\textsc{Cons}^f(\widetilde{\mathsf{CKT}}(h^{vN}, 1/2-\mu) \to \widetilde{\mathsf{CKT}}(g^{GL}, 1-1/p-\gamma);\ \mathsf{poly}(k,m,\mathsf{poly}(1/\mu))).$$

Proof. Recall some basic definition from pseudorandomness theory. We say that distributions X and Y are computationally (η,s)-indistinguishable, denoted by $X \overset{\eta,s}{\approx} Y$ if, for any circuit T of size s, the probability that T accepts a sample from X is the same as the probability T accepts a sample from Y, to within $\pm\eta$.

We want to show that if h^{vN} is predictable with probability better than $1/2$, then g^{GL} is predictable with probability better than $1/p$. We will argue the contrapositive: suppose g^{GL} is unpredictable, and show that h^{vN} is unpredictable. This will take a sequence of steps.

Let $\mathcal{D}$ denote the uniform distribution over D, $\mathcal{F}$ the uniform distribution over F, and $\mathcal{U}$ the uniform distribution over $\{0,1\}$. Assume g^{GL} is unpredictable by circuits of size s with probability better than $1/p+\gamma$, for some $\gamma > 0$. This implies the following sequence of statements:

1. $(\mathcal{D}, g^{GL}(\mathcal{D})) \overset{2\gamma,\Omega(s)}{\approx} (\mathcal{D},\mathcal{F})$ (unpredictable $\Rightarrow$ indistinguishable)
2. $(\mathcal{D}^{2t}, g^{GL}(\mathcal{D})^{2t}) \overset{4t\gamma,\Omega(s/t)}{\approx} (\mathcal{D}^{2t}, F^{2t})$ (hybrid argument)
3. $(\mathcal{D}^{2t}, E^{vN}(g^{GL}(\mathcal{D})^{2t})) \overset{4t\gamma,\Omega((s/t)-\mathsf{poly}(t))}{\approx} (\mathcal{D}^{2t}, E^{vN}(F^{2t}))$ (applying h^{vN})
4. $(\mathcal{D}^{2t}, h^{vN}(\mathcal{D}^{2t})) \overset{4t\gamma+p^{-t},\Omega((s/t)-\mathsf{poly}(t))}{\approx} (\mathcal{D}^{2t},\mathcal{U})$ (by Claim 4.6)

Finally, the last statement implies (via the "indistinguishable to unpredictable" direction) that h^{vN} cannot be computed on more than $1/2+\mu$ fraction of inputs by any circuit of size $\Omega((s/t)-\mathsf{poly}(t))$, where $\mu = \Omega(t\gamma + p^{-t})$. For $t = O(\log 1/\mu)$, we get $\gamma \geq \Omega(\mu/(\log 1/\mu))$.

In the standard way, the sequence of implications above yields an efficient randomized algorithm, with the runtime $\mathsf{poly}(k, m, \log 1/\mu)$, for going in the reverse direction: from a predictor circuit for h^{vN} to a predictor circuit for g^{GL}. To be able to carry out the hybrid argument with uniform algorithms, we need efficient sampleability of the distribution $(\mathcal{D}, g^{GL}(\mathcal{D}))$. Such sampling is possible when we have membership queries to f (as $g^{GL} \in \mathsf{P}^f$); in fact, here it would suffice to have access to uniformly random labeled examples $(x, f(x))$. Another issue is that we need to sample uniformly from $\mathbb{Z}_p$, while we only have access to uniformly random bits. However, it is easy to devise an efficient sampling algorithm for $\mathbb{Z}_p$, with the distribution statistically almost indistinguishable from uniform over $\mathbb{Z}_p$.[5] ◀

We now have all the ingredients to prove the following.

▶ **Theorem 4.8** (Black-Box Amplification within $\mathsf{AC}^0[p]$)**.** *For any $0 < \epsilon, \gamma < 1$, there is black-box (ϵ,γ)-amplification within $\mathsf{AC}^0[p]$.*

Proof. The proof is similar to that of Theorem 4.3. To amplify a given function f, we first apply the Direct Product construction to get $g = f^k$ (for an appropriate parameter k), then use the Goldreich-Levin construction to get g^{GL}, and finally apply the von Neumann construction h^{vN}. The only difference is the use of the von Neumann construction of Theorem 4.7. But the only consequence of this extra step for the parameters of the amplification procedure is the slightly worse dependence on $1/\gamma$: from $1/\gamma$ to $(1/\gamma)\cdot\log 1/\gamma \leq 1/\gamma^2$. ◀

5 Natural properties imply randomized learning

In this section, we prove the general implication from natural properties to learning algorithms. First we prove the generic reduction from learning (and compression) to natural properties. Then, as our main application, we use the known natural properties for $\mathsf{AC}^0[p]$, to get learning and compression algorithms for $\mathsf{AC}^0[p]$.

[5] We divide an interval $[0, 2^{k-1}]$ into p almost equal pieces (all but the last piece are equal to $\lfloor 2^k/p \rfloor$), and check in AC^0 which piece we fall into. The statistical difference between the uniform distribution over $\mathbb{Z}_p$ and this distribution is at most $p/2^k$. So we can make it negligible by choosing k to be a large enough polynomial in the relevant parameters.

5.1 A generic reduction from learning to natural properties

▶ **Theorem 5.1** (Learning from a natural property). *Let Λ be any circuit class containing $\mathsf{AC}^0[p]$ for some prime p. Let R be a P-natural property, with largeness at least $1/5$, that is useful against $\Lambda[u]$, for some size function $u\colon \mathbb{N} \to \mathbb{N}$. Then there is a randomized algorithm that, given oracle access to any function $f\colon \{0,1\}^n \to \{0,1\}$ from $\Lambda[s_f]$, produces a circuit $C \in \widetilde{\mathsf{CKT}}(f,\epsilon)$ in time $\mathsf{poly}(n, 1/\epsilon, 2^{u^{-1}(\mathsf{poly}(n,1/\epsilon,s_f))})$.*

Proof. Let $\textsc{Gen}(f) = \{g_z\}$ be an (ϵ, L)-black-box generator based on f, for $L(n)$ such that $\log L(n) > u^{-1}(\mathsf{poly}(n, 1/\epsilon(n), s_f))$. Using the nonuniform Λ-efficiency of black-box generators, we have that $g_z \in \Lambda^f[\mathsf{poly}(n, 1/\epsilon)]$, for every z. Hence, we get, by replacing the f-oracle with the Λ-circuit for f, that $g_z \in \Lambda[s_g]$, for some $s_g \leq \mathsf{poly}(n, 1/\epsilon, s_f)$. We want $s_g < u(\log L(n))$. This is equivalent to $u^{-1}(s_g) < \log L(n)$.

Let D be the circuit obtained from the natural property R restricted to truth tables of size $L(n)$. By usefulness, we have $\Pr_z[\neg D(g_z) = 1] = 1$, and by largeness, $\Pr_y[\neg D(y) = 1] \leq 1 - 1/5$. So $\neg D$ is a $1/5$-distinguisher for $\textsc{Gen}(f)$. By the reconstruction property of black-box generators, we have a randomized algorithm that constructs a circuit $C \in \widetilde{\mathsf{CKT}}(f,\epsilon)$ in time $\mathsf{poly}(n, 1/\epsilon(n), L(n)) = \mathsf{poly}(n, 1/\epsilon, 2^{u^{-1}(\mathsf{poly}(n,1/\epsilon,s_f))})$, as required. ◀

For different usefulness bounds u, we get different runtimes for our learning algorithm:

- polynomial $\mathsf{poly}(ns_f/\epsilon)$, for $u(n) = 2^{\Omega(n)}$,
- quasi-polynomial $\mathsf{quasi\text{-}poly}(ns_f/\epsilon)$, for $u(n) = 2^{n^\alpha}$ where $\alpha < 1$, and
- subexponential $\mathsf{poly}(n, 1/\epsilon, 2^{(ns_f/\epsilon)^{o(1)}})$, for $u(n) = n^{\omega(1)}$.

▶ **Corollary 5.2.** *Under the same assumptions as in Theorem 5.1, we also get randomized compression for $\Lambda[\mathsf{poly}]$ to the circuit size at most $O(\epsilon(n) \cdot 2^n \cdot n)$, for any $0 < \epsilon(n) < 1$ such that $\log(\epsilon(n) \cdot 2^n \cdot n) \geq u^{-1}(\mathsf{poly}(n, 1/\epsilon))$.*

Proof. We use Theorem 5.1 to learn a small circuit that computes f on all but at most $\epsilon \cdot 2^n$ inputs, and then patch up this circuit by hardwiring all the error inputs, using extra circuitry of size at most $O(\epsilon \cdot 2^n \cdot n)$. This size will dominate the overall size of the patched-up circuit for the ϵ satisfying the stated condition. ◀

5.2 Application: Learning and compression algorithms for $\mathsf{AC}^0[p]$

We have natural properties useful against the class of AC^0 circuits with mod p gates, for any fixed prime p, as given in [31]. The lower bound of Razborov [30] (showing that $\textsc{Majority}$ is not in $\mathsf{AC}^0[2]$) embeds a natural property against $\mathsf{AC}^0[2]$, and the lower bound of Smolensky [35] (showing that $\textsc{Parity}$ is not in $\mathsf{AC}^0[p]$, for any prime $p \neq 2$) embeds a natural property against $\mathsf{AC}^0[p]$ for any prime $p > 2$. In both cases, the natural property is NC^2-computable, and is useful for circuit size up to $2^{\Omega(n^{1/(2d)})}$, where d is the circuit depth, and n is the input size.

▶ **Theorem 5.3** ([31]). *For every prime p, there is an NC^2-natural property of n-variate Boolean functions, with largeness at least $1/2$, that is useful against $\mathsf{AC}^0[p]$ circuits of depth d of size up to $2^{\Omega(n^{1/(2d)})}$.*

Below we sketch the corresponding natural properties; see the full paper for more details.

Natural Property useful against $\mathsf{AC}^0[2]$. For $0 \le a, b \le n$, define a linear transformation $A_{a,b}$ that maps a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ to a matrix $M = A_{a,b}(f)$ of dimension $\binom{n}{a} \times \binom{n}{b}$, whose rows are indexed by size a subsets of $[n]$, and rows by size b subsets of $[n]$. For every $K \subseteq [n]$, define the set $Z(K) = \{(x_1, \ldots, x_n) \in \{0,1\}^n \mid \forall i \in K,\ x_i = 0\}$. For a size a subset $I \subseteq [n]$ and size b subset $J \subseteq [n]$, define $M_{I,J} = \oplus_{x \in Z(I \cup J)} f(x)$.

The natural property of Theorem 5.3 for $\mathsf{AC}^0[2]$ is the following algorithm:

> Given an n-variate Boolean function f, construct matrices $M_b = A_{a,b}(f)$ for $a = n/2 - \sqrt{n}$ and for every $0 \le b \le a$. Accept f if, for at least one b, $\mathrm{rank}(M_b) \ge \frac{2^n}{140n^2}$.

Natural Property useful against $\mathsf{AC}^0[p]$, for primes $p > 2$. Let f be a given n-variate Boolean function. Without loss of generality, assume n is odd. Denote by L the vector space of all multilinear polynomials of degree less than $n/2$ over $\mathbf{GF}(p)$. Let $\bar{f}$ be the unique multilinear polynomial over $\mathbf{GF}(p)$ that represents f on the Boolean cube $\{-1,1\}^n$ (after the linear transformation mapping the Boolean 0 to $1 \mod p$, and the Boolean 1 to $-1 \mod p$), i.e., f and $\bar{f}$ agree over all points of $\{-1,1\}^n$.

The natural property of Theorem 5.3 for $\mathsf{AC}^0[p]$ is the following algorithm:

> Given an n-variate Boolean function f, construct its unique multilinear polynomial extension $\bar{f}$ over $\mathbf{GF}(p)$. Accept f if $\dim(\bar{f}L + L) \ge \frac{3}{4} \cdot 2^n$ (over $\mathbf{GF}(p)$).

Theorem 5.3, in conjunction with Theorem 5.1, immediately yields our main application.

▶ **Corollary 5.4** (Learning $\mathsf{AC}^0[p]$ in quasipolytime)**.** *For every prime p, there is a randomized algorithm that, using membership queries, learns a given n-variate Boolean function $f \in \mathsf{AC}^0[p]$ of size s_f to within error ϵ over the uniform distribution, in time* $\mathsf{quasi\text{-}poly}(ns_f/\epsilon)$.

Using Corollary 5.2, we also immediately get the following compression result, first proved (with somewhat stronger parameters) by Srinivasan [36].

▶ **Corollary 5.5.** *There is a randomized compression algorithm for depth-d $\mathsf{AC}^0[p]$ functions that compresses an n-variate function to the circuit size at most $2^{n-n^{\mu}}$, for $\mu \ge \Omega(1/d)$.*

5.3 Sketch of Complete Algorithm

Here, we sketch the algorithm implied by Theorem 5.1 for the case of $\mathsf{AC}^0[2]$. Let $f\colon \{0,1\}^n \to \{0,1\}$ be a function in $\mathsf{AC}^0[2]$ to be learned, given via membership oracle. Let R be a natural property, and let $L = n^{\mathsf{poly}(\log n)}$.

1. Design a subroutine for computing $\textsc{Amp}(f) = (f^k)^{GL}$ (Theorem 4.3) using f as an oracle.
2. Let D be a circuit simulating the natural property R_L. D is a distinguisher between $G_{\textsc{Amp}(f)}(s)$ for a random s and uniform, as shown in the proof of Theorem 5.1.
3. Convert D into C, a weak predictor for $\textsc{Amp}(f)$ on $(1/2 + \Omega(1/L))$-fraction of inputs, using the NW reconstruction algorithm (Section 2.9) and oracle for $\textsc{Amp}(f)$.
4. Use C as the oracle for the Goldreich-Levin reconstruction algorithm (Theorem 4.2), obtaining a predictor C' for the direct product of f.
5. Use C' as input to the Direct Product reconstruction algorithm of Theorem 4.1, and print the resulting circuit.

For the case of $\mathsf{AC}^0[p]$ with $p \neq 2$, the algorithm is essentially the same, but requires an additional step in the definition of $\textsc{Amp}(f)$: the von Neumann construction (Theorem 4.7) applied to $(f^k)^{GL}$. Thus, we need the von Neumann reconstruction step inserted between steps 3 and 4 of the complete algorithm above.

6 NW designs cannot be computed in AC^0

In Section 3.1 we showed that NW designs (with parameters of interest to us) are computable by small $\mathsf{AC}^0[p]$ circuits, for any prime p. It is natural to ask if one can compute such NW designs by small AC^0 circuits, without modular gates. Here we show that this is *not* possible.

Consider an NW design $S_1, \ldots, S_L \subseteq [n^2]$, where

- each set S_i is of size n,
- the number of sets is $L = 2^\ell$ for $\ell = n^\epsilon$ (for some $\epsilon > 0$), and
- for any two distinct sets S_i and S_j, $i \neq j$, we have $|S_i \cap S_j| \leq \ell$.

To describe such a design, we use the following Boolean function g: for $1 \leq i \leq L$, and for $1 \leq k \leq n^2$, define $g(i,k) = 1$ iff $k \in S_i$.

We will prove the following.

▶ **Theorem 6.1.** *Let $g\colon \{0,1\}^{\ell+2\log n} \to \{0,1\}$ be the characteristic function for any NW design with the above parameters. Then g requires depth d AC^0 circuits of size $\exp(\ell^{1/d})$.*

To prove this result, we shall define a family of Boolean functions f_T, parameterized by sets $T \subseteq [n^2]$: for $1 \leq i \leq L$, we define $f_T(i) = 1$ iff $T \cap S_i \neq \emptyset$. Observe that if $g(i,k)$ is computable by AC^0 circuits of depth d and size s, then, for every set T, the function $f_T(i) = \vee_{k \in T}\ g(i,k)$ is computable by AC^0 circuits of depth at most $d+1$ and size $O(s \cdot |T|)$. Therefore, to prove Theorem 6.1, it will suffice to prove the following.

▶ **Lemma 6.2.** *There exists a set $T \subseteq [n^2]$ such that $f_T\colon \{0,1\}^\ell \to \{0,1\}$ requires depth $d+1$ AC^0 circuits of size at least $\exp(\ell^{1/d})$.*

The idea of the proof of Lemma 6.2 is to show that for a random set T (of expected size $O(n)$), the function f_T has high average sensitivity (i.e., is likely to flip its value for many Hamming neighbors of a randomly chosen input). By averaging, we get the existence of a particular function f_T of high average sensitivity. On the other hand, it is well-known that AC^0 functions have low average sensitivity. This will imply that f_T must require large AC^0 circuits. We refer the reader to the full version of the paper for more details.

7 Conclusions

For our applications, we need Λ strong enough to carry out a (version of) the construction, yet weak enough to have a natural property useful against it. Here we show that $\Lambda = \mathsf{AC}^0[p]$ for any prime p satisfies both conditions. A logical next step would be ACC^0: if one can get a natural property useful against ACC^0, for example by naturalizing Williams's [43] proof, then a learning algorithm for ACC^0 would follow. (As MOD_p can be simulated with MOD_m, $m = p \cdot a$ gates by duplicating each input to the Mod_m gate a times (without any penalty in the number of gates), our construction for MOD_p can be applied directly by taking p to be any prime factor of m.)

Connections between learning algorithms and lower bounds could also be explored in other settings. In particular, it would be interesting to give such a connection for arithmetic circuits. In [23], the NW generator is used to derandomize polynomial identity testing based on a polynomial with a large arithmetic circuit lower bound. Since the main reduction is constructive, one might hope to use it to design learning (or interpolation) algorithms for multivariate polynomials of small circuit complexity. However, it is unclear what the analogy of "natural property" would be in this setting.

We conclude with the following open questions. Can we get an *exact* compression algorithm for $\mathsf{AC}^0[p]$ (or even AC^0) functions that would produce circuits of *subexponential*

size? Can our learning algorithm be derandomized? Is there a way to get nontrivial SAT algorithms from natural properties? Finally, are there more applications of "play-to-lose" pseudorandom constructions?

Acknowledgments. This work was partially supported by the Simons Foundation and NSF grants #CNS-1523467 and CCF-121351 (M. Carmosino, R. Impagliazzo) and by NSERC Discovery grants (V. Kabanets, A. Kolokolova). This work was done in part while all authors were visiting Simons Institute for the Theory of Computing. We thank the anonymous referees for their helpful suggestions.

References

1 Baris Aydinlioglu and Dieter van Melkebeek. Nondeterministic circuit lower bounds from mildly de-randomizing Arthur-Merlin games. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 269–279, 2012.

2 László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993. doi:10.1007/BF01275486.

3 Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#AC^0$SAT. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 117–125, 2012.

4 Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *Journal of the ACM*, 57:28:1–28:10, 2010.

5 Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Tighter connections between derandomization and circuit lower bounds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX-/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 645–658, 2015.

6 Ruiwen Chen and Valentine Kabanets. Correlation bounds and #SAT algorithms for small linear-size circuits. In Dachuan Xu, Donglei Du, and Dingzhu Du, editors, *Computing and Combinatorics – 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, volume 9198 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2015. doi:10.1007/978-3-319-21398-9_17.

7 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015. doi:10.1007/s00037-015-0100-0.

8 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small de Morgan formulas. In *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 165–176, 2014.

9 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, pages 33–45, 2015. doi:10.1007/978-3-319-24318-4_4.

10 Lance Fortnow and Adam R. Klivans. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.*, 75(1):27–36, 2009. doi:10.1016/j.jcss.2008.07.006.

11 Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

12 Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32. ACM, 1989. doi:10.1145/73007.73010.

13 Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-lemma. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 273–301. Springer, 2011.

14 Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries: The highly noisy case. *SIAM J. Discrete Math.*, 13(4):535–570, 2000. doi:10.1137/S0895480198344540.

15 Johan Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation*, pages 143–170, Greenwich, Connecticut, 1989. Advances in Computing Research, vol. 5, JAI Press.

16 Alexander Healy, Salil Vadhan, and Emanuele Viola. Using nondeterminism to amplify hardness. *SIAM Journal on Computing*, 35(4):903–931, 2006.

17 Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: Simplified, optimized, and derandomized. *SIAM J. Comput.*, 39(4):1637–1665, 2010. doi:10.1137/080734030.

18 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002. doi:10.1016/S0022-0000(02)00024-7.

19 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 961–972. SIAM, 2012. doi:10.1137/1.9781611973099.

20 Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from shrinkage. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 111–119, 2012. doi:10.1109/FOCS.2012.78.

21 Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229. ACM, 1997. doi:10.1145/258533.258590.

22 Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63(4):672–688, 2001. doi:10.1006/jcss.2001.1780.

23 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. doi:10.1007/s00037-004-0182-6.

24 Adam Klivans, Pravesh Kothari, and Igor Carboni Oliveira. Constructing hard functions using learning algorithms. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, Palo Alto, California, USA, 5-7 June, 2013*, pages 86–97. IEEE, 2013. doi:10.1109/CCC.2013.18.

25 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. doi:10.1145/174130.174138.

26 Oleg B. Lupanov. On the synthesis of switching circuits. *Soviet Mathematics*, 119(1):23–26, 1958. English translation in *Soviet Mathematics Doklady*.

27 Oleg B. Lupanov. A method of circuit synthesis. *Izvestiya VUZ, Radiofizika*, 1(1):120–140, 1959. (in Russian).

28 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.

29 Ryan O'Donnell. Hardness amplification within NP. *J. Comput. Syst. Sci.*, 69(1):68–94, 2004.

30 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41(4):333–338, 1987.

31 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997. doi:10.1006/jcss.1997.1494.

32 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.25.

33 Rahul Santhanam and Richard Ryan Williams. Beating exhaustive search for quantified boolean formulas and connections to circuit complexity. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 231–241, 2015. doi:10.1137/1.9781611973730.18.

34 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. In *Proceedings of the Twenty-Seventh Annual IEEE Conference on Computational Complexity*, pages 107–116, 2012.

35 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82. ACM, 1987.

36 Srikanth Srinivasan. A compression algorithm for $AC^0[\oplus]$ circuits using certifying polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:142, 2015. URL: http://eccc.hpi-web.de/report/2015/142.

37 Avishay Tal. #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:114, 2015. URL: http://eccc.hpi-web.de/report/2015/114.

38 Luca Trevisan. On uniform amplification of hardness in NP. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 31–38. ACM, 2005.

39 Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003. doi:10.1016/S0022-0000(03)00046-1.

40 John von Neumann. Various techniques used in connection with random digits. *J. Research Nat. Bur. Stand., Appl. Math. Series*, 12:36–38, 1951.

41 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. doi:10.1137/10080703X.

42 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 194–202, 2014. doi:10.1145/2591796.2591858.

43 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903.