Weak derandomization of weak algorithms: explicit versions of Yao's lemma*

Ronen Shaltiel†
University of Haifa

March 7, 2010

Abstract

A simple averaging argument shows that given a randomized algorithm A and a function f such that for every input x, $\Pr[A(x) = f(x)] \geq 1 - \rho$ (where the probability is over the coin tosses of A), there exists a *nonuniform* deterministic algorithm B "of roughly the same complexity" such that $\Pr[B(x) = f(x)] \geq 1 - \rho$ (where the probability is over a uniformly chosen input x). This implication is often referred to as "the easy direction of Yao's lemma" and can be thought of as "weak derandomization" in the sense that B is deterministic but only succeeds on most inputs. The implication follows as there exists a fixed value r' for the random coins of A such that "hardwiring r' into A" produces a deterministic algorithm B. However, this argument does not give a way to *explicitly construct* B.

In this paper we consider the task of proving *uniform versions* of the implication above. That is, how to *explicitly construct* a deterministic algorithm B when given a randomized algorithm A. We prove such derandomization results for several classes of randomized algorithms. These include: randomized communication protocols, randomized decision trees (here we improve a previous result by Zimand), randomized streaming algorithms and randomized algorithms computed by polynomial-size constant-depth circuits.

Our proof uses an approach suggested by Goldreich and Wigderson and "extracts randomness from the input". We introduce a new type of (seedless) extractors that extract randomness from distributions that are "recognizable" by the given randomized algorithm. We show that such extractors produce randomness that is in some sense not correlated with the input.

*A preliminary version of this paper appeared in CCC 2009.
†This research was supported by BSF grant 2004329 and ISF grant 686/07.

1 Introduction

1.1 Background

Randomized algorithms and derandomization Randomized algorithms are algorithms that get an additional input which is a sequence of independent coin tosses and are allowed to err with small probability. For some choices of computational resources it is known that randomized algorithms can be significantly more efficient than deterministic ones. For example, when measuring communication complexity (that is the amount of communication exchanged by two parties each holding "half of the input") there are tasks that require a linear amount of communication by deterministic algorithms whereas randomized algorithms can use a logarithmic amount of communication. When the complexity measure is running time, a long-standing open problem asks whether $BPP = P$ (namely, can any randomized polynomial-time algorithm be simulated by a polynomial-time deterministic algorithm). A long line of research is devoted to studying this problem (see e.g. [32, 24, 20] for survey articles). Some highlights of this research are "hardness versus randomness tradeoffs" showing that $BPP = P$ assuming certain circuit lower bounds [22] (see also [6, 54, 38, 4, 48, 47, 51]), and that the statement $BPP = P$ entails certain circuit lower bounds that seem hard to prove using current techniques [25].

Deterministic algorithms that do well on a random input A simple averaging argument (that is due to Yao [56]) shows that given a randomized algorithm A that computes a function f correctly with high probability over its random coins, there exists a deterministic algorithm B that computes f correctly with high probability over a random input. More precisely, given functions $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, $f : \{0,1\}^n \to \{0,1\}$ and a number $0 < \rho < 1/2$ we say that A *computes* f with *success* $1 - \rho$ if

$$\forall x \in \{0,1\}^n : \Pr_{R \leftarrow \{0,1\}^m}[A(x,R) = f(x)] \geq 1 - \rho \tag{1}$$

An averaging argument gives that under this assumption there exists a string $r' \in \{0,1\}^m$ such that setting $B(x) = A(x,r')$ gives that the *deterministic algorithm* B satisfies:

$$\Pr_{X \leftarrow \{0,1\}^n}[B(X) = f(X)] \geq 1 - \rho \tag{2}$$

Yao's lemma The aforementioned implication is often referred to as the "easy direction" of Yao's lemma [56]. In this paper we are only interested in the "easy direction" and we refer to it as "Yao's lemma". (The "hard direction" which follows using von-Neumann's min-max theorem shows that if (2) holds for any probability distribution over the inputs then (1) holds). We remark that the proof of the "easy direction" gives a stronger implication than stated above in which (2) holds for any probability distribution over the inputs. Yao's lemma can be viewed as a "weak derandomization" of randomized algorithms in the sense that it converts a randomized algorithm A into a deterministic algorithm B. For many complexity measures (e.g. communication complexity, circuit complexity) the deterministic algorithm B has essentially the same complexity as A. However, the argument also has two obvious weaknesses:

- The deterministic algorithm B doesn't necessarily succeed on *all* inputs and is only guaranteed to succeed with high probability on a random input.
- The averaging argument only shows the *existence* of the deterministic algorithm B but it does not give an *explicit* way to construct it.

To demonstrate the second weakness, note that if A is computable by a polynomial-time Turing machine we cannot deduce that there exists a polynomial-time Turing machine B that satisfies (2). This is because the

averaging argument only shows the *existence* of a string r' and doesn't give an explicit way to find it. We can however deduce that there is a polynomial-size *circuit* B that satisfies (2) because we can hardwire the constant string r' into A to create the circuit $B(x) = A(x, r')$.

Summing up this discussion we want to distinguish between the *complexity* of an algorithm (which is the amount of resources used by the algorithm) and the *uniformity* of an algorithm (namely, whether it can be *explicitly constructed* in uniform polynomial-time). Using this terminology, when given a randomized algorithm A that satisfies (1), Yao's lemma gives a deterministic algorithm B that satisfies (2) such that B has essentially the same complexity as A but the lemma does not guarantee that B is explicitly constructible.

We will be interested in giving transformations that given a randomized algorithm A that succeeds with high probability on every input, *explicitly construct* a deterministic algorithm B (with roughly the same complexity as A) that succeeds with high probability on a random input. We will refer to this goal as achieving "an explicit version of Yao's Lemma" or "explicit weak derandomization".

Adelman's theorem [1] states that $BPP \subseteq P/poly$, namely that any uniform polynomial-time randomized algorithm can be simulated by a family of polynomial-size circuits and the simulation succeeds on *all inputs*. The source of non-uniformity in Adelman's Theorem is that Yao's Lemma does not provide an explicitly constructible deterministic algorithm. More precisely, the proof of Adelman's theorem works in two steps:

Amplification: Given a polynomial-time randomized algorithm $A(x, r)$ which succeeds in computing a function f on all inputs $x \in \{0,1\}^n$ with probability $2/3$ the algorithm is amplified (by running it several times with independent random coins and taking the majority vote) so that it succeeds with probability $1 - \rho$ for $\rho = 2^{-2n}$.

Yao's Lemma Applying Yao's Lemma, there exists a deterministic polynomial-size circuit $B(x)$ such that $\Pr_{X \leftarrow \{0,1\}^n}[B(X) \neq f(X)] < \rho = 2^{-2n} < 2^{-n}$. Note that any positive probability in this probability space is at least 2^{-n} and therefore the probability above is zero. This implies that B succeeds on *all inputs*.

In particular, giving an explicit version of Yao's Lemma that applies to all polynomial-time randomized algorithms and every choice of ρ implies $BPP = P$.

1.2 Explicit versions of Yao's lemma for weak algorithms

We want to prove unconditional results and therefore we limit our attention to weak classes of randomized algorithms. In this paper we prove explicit versions of Yao's lemma for communication protocols and decision trees. (These results require that the number of random coins tossed by the algorithm is smaller than the input length). In addition we prove explicit versions of Yao's Lemma for streaming algorithms and algorithms implemented by constant-depth circuits (here we can allow the randomized algorithm to toss a polynomial number of coins). We also give *conditional* results that give explicit versions of Yao's lemma for *general* polynomial-time algorithms assuming that certain hard functions exists. These conditional results are incomparable to known "hardness versus randomness tradeoffs" in the sense that they use different assumptions and provide derandomization that only succeeds on most inputs. We survey our results below.

1.2.1 Communication protocols

Communication complexity was first defined by Yao [55] (see the book [31] for a comprehensive treatment). In this setup we think of an input $x \in \{0,1\}^n$ as a pair of inputs $x = (x_1, x_2)$ where $x_1, x_2 \in \{0,1\}^{n/2}$. There are two parties P_1, P_2 where P_i receives x_i and the two parties want to compute a function $f(x_1, x_2)$ by exchanging few bits of communication. Both deterministic and randomized communication protocols are

considered and the complexity of a communication protocol is the number of bits exchanged on the worst input x.

Yao's lemma gives "weak derandomization" for communication protocols and we cannot expect a "strong derandomization" of the form of Adelman's theorem in the setting of communication complexity. This is because there are examples of exponential gaps between deterministic and randomized communication complexity.[1] We prove an explicit version of Yao's lemma for communication protocols. Namely, when given a randomized (public coin) communication protocol A we show how to explicitly construct a deterministic protocol B with related complexity that succeeds on "most inputs". In order to state our results we define the notion of *explicitly constructible* (or in other words *uniform*) communication protocols.

Loosely speaking, a protocol is *explicitly constructible* if whenever a party P_i needs to send a bit in the communication protocol, this bit can be computed in *polynomial-time* as a function of the party's view. For deterministic protocols the view consists of the input x_i and the previously communicated bits. For randomized protocols the view also includes the string of public coin tosses. Note that as this is an asymptotic notion, we need to consider families $A = \{A_n\}$ of communication protocols (where A_n takes inputs of length n) for this to make sense. A precise formal definition appears in Section 4.1.1.

The issue of whether protocols are explicitly constructible is rarely addressed in communication complexity. Indeed, most of the research in communication complexity is concerned with *lower bounds* and can handle protocols that are not explicitly constructible. However, we believe that when providing *upper bounds* by designing communication protocols we should prefer communication protocols that are *explicitly constructible*. The majority of protocols that appear in the literature are explicitly constructible. However, there are some exceptions.[2]

We show that given a randomized communication protocol with communication complexity q and m coins we can *explicitly construct* a deterministic communication protocol of communication complexity $O(q + m)$ that simulates A on most inputs.

Theorem 1.1 (Explicit version of Yao's lemma for communication protocols)**.** *There exists a constant $\beta > 0$ such that the following holds: Let $A = \{A_n\}$ be an explicitly constructible family of randomized communication protocols with complexity $q(n)$ and $m(n) \leq \beta n - q(n)$ coins such that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then, there is an explicitly constructible family $B = \{B_n\}$ of deterministic communication protocols with complexity $O(q(n) + m(n))$ such that for every n, $\Pr_{X \leftarrow \{0,1\}^n}[B_n(X) = f_n(X)] \geq 1 - \rho(n)$.*

Under the weaker assumption that A succeeds with probability $1 - \rho(n)$ when *both* the input and the random coins are chosen uniformly at random, namely that for every n,

$$\Pr_{X \leftarrow \{0,1\}^n, R \leftarrow \{0,1\}^{m(n)}}[A_n(X, R) = f_n(X)] \geq 1 - \rho(n)$$

we obtain that B succeeds with probability $1 - 3\rho(n) - 2^{-10m(n)}$ on a uniformly chosen input. This also applies for our results on decision trees and streaming algorithms that are described next.

[1] The reason that the aforementioned proof of Adelman's Theorem does not go through is that the *amplification step* does not preserve the complexity of communication protocols. More precisely, amplifying from $\rho = 1/3$ to $\rho = 2^{-2n}$ requires multiplying the complexity by $\log(1/\rho) \geq n$ which gives a trivial communication protocol (as any function can be computed by a deterministic communication protocol of complexity n). We remark that less ambitious amplification to larger values of ρ (e.g. $\rho = 1/n$) is sometimes "affordable" for communication protocols.

[2] For example, a classical result by Newman [34] states that any randomized communication protocol can be simulated by one that has $m = O(\log n)$ coins (this is used to show the equivalence between public coin and private coin models of randomized communication complexity). It is interesting to note that the proof uses a probabilistic argument and does not give an explicitly constructible protocol. It is open whether one can explicitly construct such a communication protocol. There are also other examples in the literature where the computation performed by the parties takes exponential time. One such example is the proof that deterministic communication complexity and nondeterministic communication complexity are polynomially related.

1.2.2 Decision trees

A decision tree of complexity q is a procedure that is allowed to adaptively make q queries of the form "what is the i'th bit of the input x" and outputs a value at the end. The reader is referred to [8] for a survey article on decision trees. We want to discuss explicit versions of Yao's lemma and consider *explicitly constructible* (or *uniform*) decision trees. We say that a (family of) decision trees is *explicitly constructible* if there is a polynomial-time Turing machine which implements the decision tree. Namely when given the answers to the queries made so far the machine produces the index i of the bit to be queried next, and runs in time polynomial in $|i| = \log n$ and $q(n)$ (where $q(n)$ is the depth of the tree on inputs of length n). For a randomized decision tree with $m(n)$ coins the machine also receives a string $r \in \{0,1\}^{m(n)}$. A precise formal definition appears in Section 4.2.1. In particular, sublinear time randomized algorithms that run in time $q(n) < n$ and have random access to the input, are captured by randomized decision trees with complexity $q(n)$ and $m(n) \leq q(n)$ random coins.

Zimand [58] shows how to achieve weak derandomization of sublinear time algorithms. In our terminology Zimand's result can be seen as an explicit version of Yao's Lemma for decision trees. Namely, given a randomized decision tree A with complexity $q \ll n$ and $m = q$ coins, one can explicitly construct a deterministic decision tree B that simulates A correctly on most inputs. A precise formulation using our notation follows:[3]

Theorem 1.2 (Explicit version of Yao's lemma for decision trees [58])**.** *There exists a constant $\alpha > 0$ such that the following hold: Let $A = \{A_n\}$ be an explicitly constructible family of randomized decision trees with complexity $q(n) \leq n^\alpha$ and $m(n) = q(n)$ coins such that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then, there is an explicitly constructible family $B = \{B_n\}$ of deterministic decision trees with complexity $q(n)^{O(1)}$ such that for every n, $\Pr_{X \leftarrow \{0,1\}^n}[B_n(X) = f_n(X)] \geq 1 - \rho(n)$.*

For decision trees that *compute* functions it is known that there is at most a polynomial gap between deterministic complexity and randomized complexity [35]. (Note that this is very different than communication protocols in which there may be exponential gaps). However, Theorem 1.2 (and all our results) also apply to a more general setup of algorithms that "approximately count" or solve "search problems". We elaborate on these issues in Section 3.4. We stress that in such setups there are exponential gaps between the complexity of deterministic and randomized decision trees. One such example is that by sampling, randomized decision trees can approximately count the number of ones in the input with logarithmic complexity whereas deterministic decision trees require linear complexity for this task.

Our techniques provide an alternative proof of Zimand's result. Our proof gives a quantitative improvement over Zimand's result in the sense that we get a deterministic decision tree with complexity that is *linear* in $q(n)$ whereas Zimand gives a deterministic decision tree of complexity that is a large polynomial in $q(n)$. (In his paper Zimand estimates that the polynomial in his proof is $q(n)^{24}$). The parameters that come up in the formal statement below are identical to those in Theorem 1.1.

Theorem 1.3 (Improved explicit version of Yao's lemma for decision trees)**.** *There exists a constant $\beta > 0$ such that the following holds: Let $A = \{A_n\}$ be an explicitly constructible family of randomized decision trees with complexity $q(n)$ and $m(n) \leq \beta n - q(n)$ coins such that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then, there is an explicitly constructible family $B = \{B_n\}$ of deterministic decision trees with complexity $O(q(n)+m(n))$ such that for every n, $\Pr_{X \leftarrow \{0,1\}^n}[B_n(X) = f_n(X)] \geq 1 - \rho(n)$.*

[3] Zimand states a stronger quantitative result in which B succeeds with probability $1 - 2^{-\Omega(q(n)\log q(n))}$ for $\rho = 1/3$. The two formulations are equivalent as if one wants to improve the success probability from say $1 - \rho = 2/3$ to $1 - 2^{-q(n)\log q(n)}$ it is possible to first amplify the success probability of A to that value. This increases the complexity of A by at most a polynomial. One can then apply the weaker statement of Theorem 1.2 on the amplified algorithm to get the stronger statement in Zimand's paper.

1.2.3 Streaming algorithms

Streaming algorithms (see e.g. [3, 15]) are algorithms which read the input in one pass using sublinear space. Here the complexity measure is the space used by the algorithm. Randomized streaming algorithms $A(x, r)$ come in two flavors depending on how they access their random coins. Algorithms with *one-way access to randomness* receive one-way access to both the input x and the "sequence of random coins" r that is stored on a separate tape. This model captures streaming algorithms that "toss coins on the fly". We can also consider algorithms with *two-way access to randomness*. This is a less natural model as such algorithms are not charged for storing their random coins and thus have more power in case the number of random coins used is larger than the space. (Precise definitions of these models appear in Section 4.3).

We say that a streaming algorithm is *explicitly constructible* if it can be implemented by a polynomial-time Turing machine. More precisely, that the operation of updating the internal memory following reading a symbol from the input or "random coin sequence" can be done in polynomial-time. A precise formal definition appears in Section 4.3.

Our techniques give an explicit version of Yao's lemma for randomized streaming algorithms even when allowing two-way access to randomness. The parameters we obtain are identical to the parameters as Theorems 1.1 and 1.3.

Theorem 1.4 (Explicit version of Yao's lemma for streaming algorithms with two-way access to coins)**.** *There exists a constant $\beta > 0$ such that the following holds: Let $A = \{A_n\}$ be an explicitly constructible family of randomized streaming algorithms with two-way access to randomness. Assume that the family A has complexity $q(n)$ and $m(n) \leq \beta n - q(n)$ coins and that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then, there is an explicitly constructible family $B = \{B_n\}$ of deterministic streaming algorithms with complexity $O(q(n) + m(n))$ such that for every n, $\Pr_{X \leftarrow \{0,1\}^n}[B_n(X) = f_n(X)] \geq 1 - \rho(n)$.*

Theorem 1.4 achieves weak derandomization for the stronger and less natural model of randomized algorithms with two-way access to randomness. We can do better for the more standard case of one-way access and handle randomized algorithm that toss a polynomial number of coins (rather than a sublinear number of coins).

Theorem 1.5 (Explicit version of Yao's lemma for streaming algorithms with one-way access to coins)**.** *There exists a constant $\beta > 0$ such that the following holds: Let $A = \{A_n\}$ be an explicitly constructible family of randomized streaming algorithms with one-way access to randomness. Assume that the family A has complexity $\log n \leq q(n) \leq \beta n / \log n$ and a polynomial number of coins and that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then, there is an explicitly constructible family $B = \{B_n\}$ of deterministic streaming algorithms with complexity $O(q(n) \cdot \log n)$ such that for every n, $\Pr_{X \leftarrow \{0,1\}^n}[B_n(X) = f_n(X)] \geq 1 - \rho(n) - 2^{-q(n)}$.*

Theorem 1.5 follows from Theorem 1.4 because using pseudorandom generators for bounded space algorithms [36, 21, 42] one can simulate any randomized streaming algorithm with one-way access to a polynomial number of random coins by a randomized streaming algorithm with two way access to $O(q \log n)$ random coins. The multiplicative factor of $\log n$ can be removed if one can construct pseudorandom generators with optimal seed length for bounded space algorithms. Achieving this goal is a longstanding open problem.

We remark that in the setup of streaming algorithms pseudorandom generators do not yield a derandomization that succeeds on all inputs. This is because the standard approach of enumerating all seeds of the pseudorandom generator yields a deterministic algorithm that is not a streaming algorithm as it requires two-way access to the input.

1.2.4 Constant-depth algorithms

We consider algorithms that are computable by uniform families of polynomial-size constant-depth circuits. This class is called uniform-AC^0. Uniformity means that there is a uniform machine that on input n constructs the appropriate circuit in the family. Different variants of this notion are obtained when considering machines from different complexity classes. In this paper we use the standard notion that the machine runs in space logarithmic in n and the size of the circuit. However, our results apply also to other notions. We refer to such a family of circuits as a *deterministic uniform AC^0 algorithm*. There is also a corresponding notion of randomized algorithms $A(x, r)$. Loosely speaking, it is required that $|r|$ is polynomial in $|x|$ and that the function $A(x, r)$ is in uniform-AC^0. More precisely, a *uniform randomized AC^0 algorithm* is a family of functions $A = \{A_n\}$ such that $A_n : \{0,1\}^n \times \{0,1\}^{m=\text{poly}(n)} \to \{0,1\}$ and the family A is in uniform-AC^0. (This class is sometimes referred to as $BPAC^0$).

Full derandomization a-la Adelman's theorem applies in this setup (using the fact that approximate majority is in AC^0 [2]) and gives that nonuniform randomized AC^0 and nonuniform deterministic AC^0 coincide. For uniform algorithms there are beautiful constructions of pseudorandom generators against AC^0 [38]. These constructions are based on the hardness on average of the parity function against AC^0 [19]. Applying these generators gives that every uniform randomized AC^0 algorithm can be simulated by a (deterministic) uniform family of constant-depth circuits that has *quasi-polynomial* size [38, 29]. In other words, these results give uniform *strong* derandomization (a simulation that succeeds on *all* inputs) but the resulting algorithm has quasi-polynomial-size (that is size $n^{O(\log n)}$) rather than polynomial-size.

We prove an explicit version of Yao's lemma for AC^0 algorithms. This gives the following incomparable result: For every uniform randomized AC^0 algorithm there is a uniform deterministic AC^0 algorithm that succeeds on most inputs.

Theorem 1.6 (Explicit version of Yao's lemma for AC^0)**.** *Let $A = \{A_n\}$ be a randomized uniform AC^0 algorithm such that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then there is a deterministic uniform AC^0 algorithm $B = \{B_n\}$ such that for every n, $\Pr_{X \leftarrow \{0,1\}^n}[B_n(X) = f_n(X)] \geq 1 - \rho(n) - n^{-v}$ for every constant $v > 1$.*

Note that in contrast to some of our previous results, Theorem 1.6 does not assume that the number of random coins of A is smaller than the length of the input. This is because by the aforementioned pseudorandom generators we can assume w.l.o.g. that a uniform randomized algorithm uses polylog(n) random coins. The standard approach to derandomizing a randomized algorithm with polylog$n$ random coins is to run the algorithm for all possible choices for random coins and take the majority vote. A drawback of this approach is that this takes quasipolynomial-time (indeed this is the reason [38] only gets circuits of quasipolynomial-size). In contrast, our approach runs the randomized algorithm only once using one choice for random coins that is computed as a deterministic function of the input. We elaborate more on our approach in Section 1.3.

We remark that if one could improve Theorem 1.6 and remove the factor n^{-v} then this implies a deterministic algorithm B that succeeds on all inputs. This is because by amplifying the success probability we can assume without loss of generality that $\rho(n) \leq 2^{-2n}$. Thus, if one could remove the additive factor of n^{-v} one would get a deterministic algorithm that succeeds on all but a 2^{-2n} fraction of inputs of length n, which in turn implies that the deterministic algorithm succeeds on all inputs. We stress however that we cannot expect to prove such a result using the approach of this paper.

A subsequent work [28] gives a simpler proof of Theorem 1.6. The approach of [28] has advantages over our approach in the sense that it also gives results on constant depth circuits with parity gates. We elaborate on the approach of [28] in Section 5.

1.2.5 Conditional weak derandomization for general polynomial-time algorithms

We now consider the class of *general* polynomial-time randomized algorithms. Here we give an explicit version of Yao's lemma under a hardness assumption. This result follows by imitating the proof of Theorem 1.6 and replacing the parity function (which is hard for AC^0) with a function that is assumed to be hard on average for polynomial-size circuits. Loosely speaking, a useful property of the parity function is that it is very hard on average for "weak" circuits, yet can be computed precisely by circuits with slightly larger resources. Generalizing this to our setup gives the assumption that for every constant c there are functions that are hard on average for circuits of size n^c and yet are computable in time n^d for some constant $d > c$.

Theorem 1.7 (Conditional explicit version of Yao's lemma for polynomial-time algorithms)**.** *Assume that there exists a constant $\gamma > 0$ such that for every constant c there exists a constant d and a family of functions $h = \{h_n\}$ where $h_n : \{0,1\}^n \to \{0,1\}$, such that h is computable in time n^d and for every sufficiently large n and every circuit C of size n^c, $\Pr_{X \leftarrow \{0,1\}^n}[C(X) = h_n(X)] \leq 1/2 + 2^{-n^\gamma}$. Then there exists a constant $\eta > 0$ such that for every language $L \in BPP$ there is a polynomial-time deterministic machine B that for every n, algorithm B decides the language L correctly on a $1 - 2^{-n^\eta}$ fraction of inputs of length n.*

It is natural to compare Theorem 1.7 to hardness versus randomness tradeoffs [38, 22]. The latter give a stronger conclusion of a deterministic algorithm that succeeds on all inputs. On the other hand, the assumption of Theorem 1.7 requires lower bounds against circuits of polynomial-size. This is usually referred to as "the low-end of hardness assumptions" in the literature on hardness versus randomness tradeoffs and these tradeoffs produce deterministic algorithms that run in subexponential time (and not polynomial-time). Another difference is that Theorem 1.7 requires that the hard function is computable in polynomial-time whereas hardness versus randomness tradeoffs allow the hard function to be computable in exponential time. In the latter setup there are hardness amplification results transforming functions that are hard on the worst case into functions that are hard on average. Therefore, it is sufficient to assume that the hard function is hard on the worst case. In summary, the assumption used in Theorem 1.7 is incomparable to those used in hardness versus randomness tradeoffs.

Some previous work on hardness versus randomness tradeoffs [23, 50] considered "uniform tradeoffs" in which the deterministic algorithm is guaranteed to succeed with high probability on every samplable distribution. This notion of "weak derandomization" is stronger than the one in this paper which only considers the uniform distribution. We point out that similar to the aforementioned "non-uniform" hardness versus randomness tradeoffs, the tradeoffs in [23, 50] only produce subexponential time deterministic algorithms when starting from a "low-end hardness assumption". We remark that the "uniform tradeoffs" only need to assume hardness for uniform randomized algorithms rather than circuits. In summary, the assumption used in Theorem 1.7 is incomparable to those used in uniform tradeoffs.

Theorem 1.7 can be compared to a result of [18] that also gets weak derandomization under the assumptions that there exist functions that are hard on average. However, in [18] one requires the function to be hard for circuits that have a SATISFIABILITY oracle whereas Theorem 1.7 does not. Even though we require lower bounds against a weaker class of circuits, our result is incomparable to that of [18]. This is because the quantities measuring the hardness on average of the function and the success probability of the deterministic algorithm are different. Theorem 1.7 requires a quantitatively stronger guarantee of hardness on average and gives a weaker guarantee on the success probability of the deterministic algorithm.

A subsequent work [28] shows an improved version of Theorem 1.7 that gives weak derandomization of every language in BPP starting from a weaker assumption than the one given in Theorem 1.7. Specifically, it is sufficient that the function h_n in Theorem 1.7 is only "mildly hard on average" in the sense that every circuit C of size n^c attempting to compute h_n errs on a $1/n$ fraction of the inputs. This result is incomparable to Theorem 1.7 as under the weaker assumption, [28] obtains weak derandomization that may err on more inputs then in Theorem 1.7. However, the approach of [28] is stronger in the sense that under the assumption

of Theorem 1.7, [28] can obtain the same conclusion of Theorem 1.7. We elaborate on the approach of [28] in Section 5.

1.3 Technique

We now highlight some of the new ideas in the paper. We start by reviewing some ideas from previous work.

1.3.1 Extracting randomness from the input

The goal of achieving what we call "weak derandomization" was considered by Goldreich and Wigderson [18]. They consider randomized algorithms $A(x, r)$ in which the number of random coins is smaller than the input length. Their high level idea is that in such cases one can try and "extract" randomness r from the input x. An obvious difficulty is that this makes the input and random coins correlated. Consider for example a randomized algorithm $A(x, r)$ where $|x| = |r|$ and define the deterministic algorithm $B(x) = A(x, x)$. It may be the case that for every x, all $r \neq x$ have $A(x, r) = f(x)$ while $A(x, x) \neq f(x)$ and then B errs on every input. As this example shows, to extract randomness from the input we need to somehow control the correlation between input and random coins.

Goldreich and Wigderson do not try to handle this correlation. Instead they restrict their attention to randomized algorithms in which there exist many coin outcomes r that are each good for all inputs x. (Note that in this case the aforementioned problem does not come up). Surprisingly, they show that there are interesting algorithms with this property in the literature. Moreover, they show that under a hardness assumption any polynomial-time randomized algorithm can be converted to one with the aforementioned property. We remark that the construction presented in [18] also uses "seeded extractors" to amplify the success probability of A before the derandomization. More specifically, the deterministic algorithm presented in [18] is $B(x) = \mathsf{majority}_{y \in \{0,1\}^d} A(x, E(x, y))$ where $E : \{0,1\}^n \times \{0,1\}^{d=O(\log n)} \rightarrow \{0,1\}^m$ is an explicit (that is polynomial-time computable) "seeded extractor" (see e.g. [37, 45, 52] for survey articles on seeded extractors). However, the proof can be viewed as applying the aforementioned idea of $B(x) = A'(x, x)$ on the randomized algorithm $A'(x, r) = \mathsf{majority}_{y \in \{0,1\}^d} A(x, E(r, y))$ (which by [59] computes the same function computed by A with improved success probability).

Zimand's result [58] that we stated in Theorem 1.2 also uses the approach of extracting randomness from the input. This is done using a new variant of seeded extractors (called "exposure resilient extractors" [57]). Zimand shows that using the approach of Goldreich and Wigderson with exposure resilient extractors gives an explicit version of Yao's lemma for decision trees. This suggests that one can try to control the correlation in special cases.

1.3.2 Our approach

In this paper we develop a general technique to prove explicit versions of Yao's Lemma. Our approach is also based on extracting randomness from the input. The high level idea is that the use of specific extractors (that we introduce in this paper) allows us to control the correlation between the input and the extracted random coins. More specifically, we show that any class of randomized algorithms that use $m \leq n$ random coins defines a class of probability distributions such that if $E : \{0,1\}^n \rightarrow \{0,1\}^m$ is an extractor with very small error for the class of distributions (meaning that for any distribution X in the class, $E(X)$ has statistical distance less than 2^{-m} from the uniform distribution on m bit strings) then the deterministic algorithm $B(x) = A(x, E(x))$ weakly derandomizes any randomized algorithm A in the class.

The use of seedless extractors Note that in contrast to aforementioned previous work of [18, 58] the extractor function E in the discussion above does not use a seed. To explain this point let us review the

two notions of randomness extractors: Seeded extractors expect to receive a short seed of few independent random bits when extracting randomness. There are constructions of such extractors that are able to extract randomness in the most general case: namely, from *any* distribution that "contains sufficient randomness" (that is, has high enough min-entropy). Seedless extractors have the advantage that they do not need to receive an independent seed. However,seedless extractors cannot exist for the general case above. Instead, seedless extractors are designed for specific families of distributions. For our analysis we need extractors with error smaller than 2^{-m}. By the lower bounds of [39, 40], seeded extractors for general distributions cannot achieve such error unless they use huge seeds (namely seeds that are at least as long as the output of the extractor). Moreover, even if we restrict our attention to seeded extractors for specific families of distributions, we want to use extractors with seed length $d \ll m$ and error $\ll 2^{-m}$. For these parameters any seeded extractor can be transformed into a seedless extractor by choosing an arbitrary fixed seed. Thus, the low error requirement dictates the use of seedless extractors.

Our analysis is different than that of [18, 58]. The analysis of [18, 58] relies on the ability of seeded extractors to extract randomness from any distribution with sufficient min-entropy. We need to be more careful as we cannot expect seedless extractors with this property. Instead, we use extractors that are tailored to the specific family of randomized algorithms considered.

Extractors for recognizable distributions We introduce a new notion of extractors which we call "extractors for recognizable distributions". This notion is defined in Section 3.1. We show that any class of randomized algorithms can be weakly derandomized if one can explicitly construct an extractor for distributions recognizable by the class. Our weak derandomization results then follow by using specific explicit extractors. In some cases we can use "off the shelf" explicit constructions of randomness extractors. For communication protocols we observe that "2-source extractors" extract randomness from distributions recognizable by communication protocols, and this allows us to use existing constructions of 2-source extractors. Similarly, for decision trees we observe that "extractors for bit-fixing sources" extract randomness from distributions recognizable by decision trees. For other results we construct appropriate extractors. We use 2-source extractors to construct extractors for distributions recognizable by streaming algorithms. We also show that functions that are hard on average can be used to construct extractors for recognizable distributions. This allows us to use the hardness of the parity function [19] to construct an extractor for distributions recognizable by constant-depth algorithms. More generally, assuming a function that is hard on average for polynomial-size circuits, we can construct extractors for distributions recognizable by general polynomial-time algorithms.

1.4 Organization of the paper

We give some preliminaries in Section 2. In Section 3 we state and prove our general theorem that shows how to use seedless extractors to get derandomization. In Section 4 we show how to use the general theorem and prove our main results. Finally, we mention some open problems and subsequent work in Section 5.

2 Preliminaries

General notation: We use $x \circ y$ to denote the concatenation of two strings x, y. We use $[n]$ to denote the set $\{1, \ldots, n\}$. Given a string $x \in \{0,1\}^n$ and a set $S \subseteq [n]$ we use x_S to denote the restriction of x to S (which is a string of length $|S|$).

Randomized algorithms: We model randomized procedures as functions $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$. We say that A *computes* a function $f : \{0,1\}^n \to \{0,1\}$ with *success* $1 - \rho$ if for every $x \in \{0,1\}^n$,

$\Pr_{R \leftarrow U_m}[A(x,R) = f(x)] \geq 1-\rho$. For $r \in \{0,1\}^m$, $A_r : \{0,1\}^n \to \{0,1\}$ is defined by $A_r(x) = A(x,r)$.

Probability distributions: Given a probability distribution P we use $X \leftarrow P$ to denote the experiment in which X is chosen at random according to P. Given a set S we use $X \leftarrow S$ to denote the experiment in which X is chosen uniformly at random from S. We use U_m to denote the uniform distribution on $\{0,1\}^m$. We define $\mathsf{Supp}(X) = \{x : \Pr[X = x] > 0\}$.

We say that two distributions P, Q on a set Ω are ϵ-close if $\frac{1}{2} \cdot \sum_{x \in \Omega} |\Pr[P = x] - \Pr[Q = x]| \leq \epsilon$. The min-entropy of a distribution X on Ω is defined by $H_\infty(X) = min_{x \in \Omega} \frac{1}{\Pr[X=x]}$.
We also need the following standard lemmata. See e.g., [46] for a proof.

Lemma 2.1. *Let X be a distribution over $\{0,1\}^n$ and let $S \subseteq [n]$ then $H_\infty(X_S) \geq H_\infty(X) - (n - |S|)$.*

Lemma 2.2. *Let X, Y be random variables such that Y is over $\{0,1\}^\ell$. Then, for any $\epsilon > 0$, with probability $1 - \epsilon$ over choosing $y \leftarrow Y$ we have that $H_\infty(X|Y = y) \geq H_\infty(X) - \ell - \log(1/\epsilon)$.*

3 A general approach to weak derandomization

In this section we introduce a general approach for proving weak derandomization results. The approach builds on a new concept of "extractors for recognizable distributions" defined in Section 3.1. In Section 3.2 we state a general theorem that is used to derive all our results.

3.1 Extractors for recognizable distributions

The field of deterministic (seedless) extractors attempts to explicitly construct randomness extractors for "interesting classes of distributions". Many classes are considered in the literature (see e.g. [45] for details). The definition below considers sources that are uniform over sets that can be efficiently recognized.

Definition 3.1 (Extractors for recognizable distributions)**.** *Let $C : \{0,1\}^n \to \{0,1\}$ be a function. Let U_C denote the uniform distribution over $\{x \in \{0,1\}^n : C(x) = 1\}$. We refer to U_C as the distribution* recognized *by C. Let $\mathcal{C}$ be a collection of functions $C : \{0,1\}^n \to \{0,1\}$. We say that a distribution Y over $\{0,1\}^n$ is* recognized *by $\mathcal{C}$ if there exists $C \in \mathcal{C}$ such that Y is recognized by C. We refer to the class of such distributions as* recognizable *by $\mathcal{C}$. A function $E : \{0,1\}^n \to \{0,1\}^m$ is a (k,ϵ)-extractor for distributions recognizable by $\mathcal{C}$ if for every distribution Y that is recognizable by $\mathcal{C}$ with $H_\infty(Y) \geq k$, $E(Y)$ is ϵ-close to U_m.*

Remark 3.2 (Relationship to previous work on extractors)**.** *To the best of our knowledge this notion was not explicitly defined before. Nevertheless, it is implicit in some previous work on extractors. For example, extractors for bit-fixing sources (defined in Section 4.2.2) can be viewed as extractors for distributions recognizable by projections or by low complexity decision trees (see Section 4.2.3 for details). 2-source extractors (defined in Section 4.1.2) can be viewed as extractors for distributions recognizable by rectangles or by low complexity communication protocols (see Section 4.1.3 for details). Extractors for affine sources can be viewed as extractors for distributions recognized by affine functions.*

Trevisan and Vadhan [49] introduced an alternative way in which a class of functions $\mathcal{C}$ induces a class of distributions. Their notion of "extractors for samplable distributions" is orthogonal to our notion as it asks that the distribution can be sampled efficiently rather than recognized efficiently. To illustrate the difference we mention two recent works on extractors for low degree polynomials: [14] is concerned with extractors for sources samplable by low degree polynomials, whereas [13] is concerned with extractors for sources recognizable by low degree polynomials. We also remark that the two notions coincide in case of linear polynomials (that is affine sources).

3.2 Extractors yield weak derandomization

We show that one can weakly derandomize a given algorithm A using an extractor for distributions recognizable by a family of functions $\mathcal{C}_A$. The definition of $\mathcal{C}_A$ follows:

Definition 3.3. *Given a function $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ and $r, r' \in \{0,1\}^m$ we define a function $A_{r,r'} : \{0,1\}^n \to \{0,1\}^m$ by $A_{r,r'}(x) = 1$ iff $A(x,r) = A(x,r')$. Let $\mathcal{C}_A$ denote the class of all such functions. Namely, $\mathcal{C}_A = \left\{A_{r,r'} : r, r' \in \{0,1\}^m\right\}$.*

We remark that the complexity of $A_{r,r'}$ is typically bounded by twice the complexity of A as computing $A_{r,r'}$ reduces to computing A twice. Our main theorem (stated below) shows that if $E : \{0,1\}^n \to \{0,1\}^m$ ia an extractor for distributions recognizable by $\mathcal{C}_A$ then the deterministic algorithm $B(x) = A(x, E(x))$ weakly derandomizes A.

Theorem 3.4. *Let $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be a function, let $k = n - 100m$ and $\epsilon = 2^{-100m}$. Let $E : \{0,1\}^n \to \{0,1\}^m$ be a (k,ϵ)-extractor for distributions recognizable by $\mathcal{C}_A$. Let $\rho \leq 1/3$ and let $f : \{0,1\}^n \to \{0,1\}$ be some function and assume that*

$$\Pr_{X \leftarrow U_n, R \leftarrow U_m}[A(X,R) = f(X)] \geq 1 - \rho$$

Then

$$\Pr_{X \leftarrow U_n}[A(X, E(X)) = f(X)] \geq 1 - 3\rho - 2^{-10m}$$

Remark 3.5 (Comments on Theorem 3.4 and its application)**.**

- *We made no attempt to optimize the constants $10, 100$ in the theorem and the relationship between constants can be made tighter.*
- *Theorem 3.4 does not require that A* computes *f. Recall that A computes f if*

  $$\forall x \in \{0,1\}^n, \ \Pr_{R \leftarrow U_m}[A(x,R) = f(x)] \geq 1 - \rho.$$

  Instead the theorem makes the weaker requirement that

  $$\Pr_{X \leftarrow U_n, R \leftarrow U_m}[A(X,R) = f(x)] \geq 1 - \rho.$$

  When starting from the stronger assumption we can often amplify the success probability of A which improves the success probability of the induced deterministic algorithm $B(x) = A(x, E(x))$. In Theorems 1.1,1.3 and 1.4 the stronger assumption allows us to slightly amplify the success probability of A before applying the theorem and this gives a clean statement in which the success probability of the deterministic algorithm is $1 - \rho$ rather than $1 - 3\rho - 2^{-10m}$. In Theorems 1.6 and 1.7 we are indifferent to polynomial slowdown and therefore can amplify the success probability so that the error is exponentially small before we start.
- *Theorem 3.4 does not require that A is "explicitly constructible". It applies to any randomized algorithm A. The assumption made in Theorems 1.1,1.3,1.4,1.6 and 1.7 that A is explicitly constructible is only used to argue that $B(x) = A(x, E(x))$ is explicitly constructible and is not used in the analysis showing that B weakly derandomizes A.*

Some additional extensions of Theorem 3.4 are discussed in Section 3.4. In the next section we prove Theorem 3.4.

3.3 Proof of Theorem 3.4

We are given functions $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, $f : \{0,1\}^n \to \{0,1\}$ and $E : \{0,1\}^n \to \{0,1\}^m$ that satisfy the requirements of the theorem. In the remainder of the proof we consider a probability space with two independent random variables X and R where X is uniformly distributed over $\{0,1\}^n$ and R is uniformly distributed over $\{0,1\}^m$. We define:

$$\alpha = \Pr[A(X,R) = f(X)]$$

$$\beta = \Pr[A(X,E(X)) = f(X)]$$

We have that $\alpha \geq 1 - \rho$ and our goal is to show that $\beta \geq 1 - 3\rho - 2^{-10m}$. Our strategy is to try and express the two quantities α and β in a way that will allow us to relate them. Before doing so, it is convenient to replace the random variable $f(X)$ with a different random variable on which we have better control. We note that by an averaging argument there exists $r' \in \{0,1\}^m$ such that:

$$\Pr[A(X,r') = f(X)] \geq 1 - \rho$$

In other words, the random variable $A(X,r')$ is a good approximation to $f(X)$. We now consider modified versions of the quantities α and β in which $f(X)$ is replaced by $A(X,r')$:

$$\alpha' = \Pr[A(X,R) = A(X,r')]$$

$$\beta' = \Pr[A(X,E(X)) = A(X,r')]$$

We have that $\Pr[A(X,r') \neq f(X)] \leq \rho$ and therefore $|\alpha - \alpha'| \leq \rho$ and $|\beta - \beta'| \leq \rho$. Thus, the task of proving Theorem 3.4 and showing that $\beta \geq 1 - 3\rho - 2^{-10m}$ reduces to the following lemma.

Lemma 3.6. $\beta' \geq \alpha' - 2^{-10m}$.

In the remainder of this section we prove Lemma 3.6. For $r \in \{0,1\}^m$ we will be interested in the event

$$\left\{A(X,r) = A(X,r')\right\} = \left\{A_{r,r'}(X) = 1\right\}.$$

We say that r is *tiny* if $\Pr[A(X,r) = A(X,r')] < 2^{k-n}$ and denote the set of tiny strings by T. The parameters were chosen so that tiny strings take a "negligible fraction" of the weight:

$$\sum_{r \in T} \Pr[A(X,r) = A(X,r')] \leq \sum_{r \in T} 2^{k-n} \leq 2^m \cdot 2^{k-n} \leq 2^{-50m} \tag{3}$$

where the last step follows because $k = n - 100m$. Lemma 3.6 follows from the following two lemmata:

Lemma 3.7. $\alpha' \leq 2^{-m} \cdot \sum_{r \notin T} \Pr[A(X,r) = A(X,r')] + 2^{-50m}$.

Lemma 3.8. $\beta' \geq 2^{-m} \cdot \sum_{r \notin T} \Pr[A(X,r) = A(X,r')] - 2^{-50m}$.

To conclude the proof we give the proofs of the two lemmata above.

Proof. (of Lemma 3.7) Recall that $\alpha' = \Pr[A(X,R) = A(X,r')]$.

$$\begin{aligned}
\alpha' &= \sum_{r\in\{0,1\}^m} \Pr[A(X,r) = A(X,r') \wedge R = r] \\
&= \sum_{r\in\{0,1\}^m} \Pr[A(X,r) = A(X,r')] \cdot \Pr[R = r] \\
&= 2^{-m} \cdot \sum_{r\in\{0,1\}^m} \Pr[A(X,r) = A(X,r')] \\
&= 2^{-m} \cdot \left(\sum_{r\in T} \Pr[A(X,r) = A(X,r')] + \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] \right) \\
&\leq 2^{-m} \cdot \left(2^{-50m} + \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] \right) \\
&\leq 2^{-50m} + 2^{-m} \cdot \sum_{r\notin T} \Pr[A(X,r) = A(X,r')]
\end{aligned}$$

where the first inequality follows from (3). □

We are now ready to prove Lemma 3.8.

Proof. (of Lemma 3.8) Recall that $\beta' = \Pr[A(X,E(X)) = f'(X)]$. We will analyze β' in the same manner as in Lemma 3.7. The difference is that the events $\{A(X,r) = A(X,r')\}$ and $\{E(X) = r\}$ are correlated (whereas in Lemma 3.7 the events $\{A(X,r) = A(X,r')\}$ and $\{R = r\}$ are independent). In order to handle the correlation we use the fact that E is an extractor for recognizable distributions. Specifically, for every $r \notin T$, The distribution

$$Y = (X|A(X,r) = A(X,r')) = (X|A_{r,r'}(X) = 1) = U_{A_{r,r'}}$$

is recognizable by $\mathcal{C}_A$ and has min-entropy at least k. It follows that $E(Y)$ is ϵ-close to uniform and in particular, $\Pr[E(Y) = r] \geq 2^{-m} - \epsilon$. Recalling that $\epsilon = 2^{-100m}$ this can be expressed as

$$\Pr[E(X) = r|A(X,r) = A(X,r')] \geq 2^{-m} - 2^{-100m}$$

and will be used in the calculation below.

$$\begin{aligned}
\beta' &= \sum_{r\in\{0,1\}^m} \Pr[A(X,r) = A(X,r') \wedge E(X) = r] \\
&= \sum_{r\in\{0,1\}^m} \Pr[A(X,r) = A(X,r')] \cdot \Pr[E(X) = r | A(X,r) = A(X,r')] \\
&\geq \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] \cdot \Pr[E(X) = r | A(X,r) = A(X,r')] \\
&\geq \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] \cdot (2^{-m} - 2^{-100m}) \\
&= 2^{-m} \cdot \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] - 2^{-100m} \cdot \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] \\
&\geq 2^{-m} \cdot \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] - 2^{-100m} \cdot 2^m \\
&\geq 2^{-m} \cdot \sum_{r\notin T} \Pr[A(X,r) = A(X,r')] - 2^{-50m}
\end{aligned}$$

□

3.4 Extensions of the argument

Theorem 3.4 is quite general and in Section 4 we apply it on several classes of algorithms. We now explain that the theorem can be further generalized in several ways and that these extensions can be used to prove stronger versions of Theorems 1.1,1.3,1.4,1.6 and 1.7.

3.4.1 Non-boolean functions

The definitions of deterministic and randomized algorithms discuss algorithms that output one bit (that is algorithms that solve decision problems). The definitions can be extended to algorithms that output values in any set. This stronger version of Theorem 3.4 follows just the same as we never used the fact that the output is Boolean.

3.4.2 Approximation algorithms

Given a deterministic algorithm B and a function f we say that B is successful on x if $B(x) = f(x)$. This notion is inherited by randomized algorithms A where we required that $\Pr_{R\leftarrow U_m}[A(x,R) = f(X)] \geq 1 - \rho \geq 2/3$. Following the previous item we can discuss algorithms and functions that output real values. In that case we may be interested in approximation algorithms and say that deterministic algorithm B is successful on x if $|B(x) - f(x)| \leq \mu$ where μ is some real number. We can extend this notion to randomized algorithms and say that a randomized algorithm A is successful with probability $1 - \rho$ if $\Pr_{R\leftarrow U_m}[|A(x,R) - f(x)| \leq \mu] \geq 1 - \rho \geq 2/3$.

Theorem 3.4 also applies in this setup. We only need to replace events of the form $\{A(\cdot,\cdot) = f(\cdot)\}$ with $\{|A(\cdot,\cdot) - f(\cdot)| \leq \mu\}$. This replacement should be made in the definition of $\mathcal{C}_A$ and in the proof. The

deterministic algorithm $B(x) = A(x, E(x))$ that we obtain will be successful (with respect to the quantity 2μ) on a $1 - 2\rho - 2^{-10m}$ fraction of the inputs.

4 Explicit versions of Yao's lemma

In this section we prove explicit versions of Yao's lemma for several concrete models of computations. The proofs all follow by using Theorem 3.4 with a suitable extractor.

4.1 Communication protocols

In this section we prove Theorem 1.1 which gives an explicit version of Yao's lemma for communication protocols. In Section 4.1.1 we give a precise formal definition of the notion of *explicitly constructible communication protocols* that is defined in loose terms in the introduction. In Section 4.1.2 we define 2-source extractors and survey some constructions. In Section 4.1.3 we observe that 2-source extractors can be viewed as extractors for distributions recognizable by low complexity communication protocols. Finally, in Section 4.1.4 we apply Theorem 3.4 and prove Theorem 1.1.

4.1.1 Definitions of explicitly constructible communication protocols

Communication complexity was first considered by Yao [55] (see the book [31] by Nisan and Kushilevitz for a comprehensive treatment). In this setup we think of the input $x \in \{0,1\}^n$ as a pair of inputs $x = (x_1, x_2)$ where $x_1, x_2 \in \{0,1\}^{n/2}$. We imagine that there are two parties P_1, P_2 where P_i receives x_i. The two parties want to compute a function $f(x) = f(x_1, x_2)$ by exchanging few bits of communication. For the sake of simplifying the definitions we consider the setup in which P_1 sends a bit on uneven rounds and P_2 sends a bit on even rounds. (This at most doubles the amount of bits exchanged compared to the standard definition of communication protocols). The output of the protocol is the last bit in the transcript and the complexity is the number of bits in the transcript. A formal definition follows:

Definition 4.1 (communication protocols)**.** *A* deterministic communication protocol *over* $\{0,1\}^n$ *with* complexity q *is a function* $B : \{0,1\}^n \to \{0,1\}$ *defined as follows: We think of* $x \in \{0,1\}^n$ *as* $x = (x_1, x_2)$ *where* $|x_1| = |x_2| = n/2$*. We require that there exists a* protocol function π *which is a function that given* $x' \in \{0,1\}^{n/2}$ *and a string* v *of length at most* q *outputs a bit (which specifies "the next bit to be sent in the protocol"). Given* $x \in \{0,1\}^n$ *we define* the bit communicated at step i *(denoted by* $M_i(x)$*) as follows:* $M_1(x) = \pi(x_1, \epsilon)$ *(where* ϵ *is the empty string) and for* $j \geq 1$*,* $M_{2j+1}(x) = \pi(x_1, M_1(x) \circ \ldots \circ M_{2j}(x))$ *and* $M_{2j+2}(x) = \pi(x_2, M_1(x) \circ \ldots \circ M_{2j+1}(x))$*. The* transcript *of the protocol is* $Q(x) = M_1(x) \circ \ldots \circ M_q(x)$ *and finally the output is given by the last bit in the transcript, that is* $B(x) = M_q(x)$*. (The definition can be extended to the case where* B *outputs many bits in which case we require that the output of* B *is determined by the transcript).*

A randomized communication protocol *over* $\{0,1\}^n$ *with* complexity q *and* m coins *is a function* $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *such that for every* $r \in \{0,1\}^m$ *the function* $A_r(x) = A(x, r)$ *is a deterministic communication protocol.*

We now define *explicitly constructible* communication protocols. Loosely speaking, the definition requires that the strategy of each of the two parties is computable by a polynomial-time machine.

Definition 4.2 (Explicitly constructible communication protocols)**.** *Let* $q(n)$ *be an integer function computable in time polynomial in* n*. Let* $B = \{B_n\}$ *be a family of deterministic communication protocols where* B_n *is over* $\{0,1\}^n$ *and has complexity* $q(n)$*. The family is* explicitly constructible *if there exists a*

polynomial-time Turing machine that given input (n, y) (where y is an input to the protocol function π_n associated with B_n) outputs $\pi_n(y)$.

Let $q(n), m(n)$ be integer functions computable in time polynomial in n. A family $A = \{A_n\}$ of randomized communication protocols (where A_n is over $\{0,1\}^n$ with complexity $q(n)$ and $m(n)$ coins) is explicitly constructible *if there exists a polynomial-time Turing machine which given input (n, r, y) (where $r \in \{0,1\}^{m(n)}$ and y is an input to the protocol function $\pi_{n,r}$ associated with $A_n(\cdot, r)$) outputs $\pi_{n,r}(y)$.*

In particular Definition 4.2 requires that whenever a party needs to send a bit in the communication protocol, this bit can be computed in polynomial-time as a function of the party's view (that is his input, previously exchanged communication and public random string if one exists). In contrast, note that definition 4.1 allows protocols in which computing the next bit to be sent takes exponential time.

4.1.2 2-source extractors

2-source extractors were introduced by [44, 9]. A definition follows.

Definition 4.3. *A function $E : \{0,1\}^{n/2} \times \{0,1\}^{n/2} \to \{0,1\}^m$ is a (k, ϵ)-*2-source-extractor *if for every two independent distributions X_1, X_2 over $\{0,1\}^{n/2}$ such that $H_\infty(X_1) + H_\infty(X_2) \geq k$ the distribution $E(X_1, X_2)$ is ϵ-close to U_m.*

We sometimes think of E as a function that takes one input of length n. We remark that the definition we use here is less common in the literature and that it is often required that $min(H_\infty(X_1), H_\infty(X_2))$ is large. We choose the formulation above as it makes the calculations we need to make more similar to those that come up next for decision trees and streaming algorithms. We stress however that we could have used the more standard formulation with a slightly more careful analysis.

There are explicit constructions of 2-source extractors in the literature [9, 53, 7]. We are interested in a "less challenging" setup in which $k = 9n/10$ and use the following theorem due to [53, 9] that variants of the Hadamard matrix yield extractors (see e.g. [12] for a proof).

Theorem 4.4. *[53, 9, 12] There exists a constant $\eta > 0$ such that for sufficiently large n there is a function $E_{Had} : \{0,1\}^{n/2} \times \{0,1\}^{n/2} \to \{0,1\}^{\eta n}$ that is computable in polynomial-time and linear space such that E is a $(9n/10, 2^{-300\eta n})$-2-source extractor.*

We are going to use the extractor E_{Had} to construct a 2-source extractor E that can extract from distributions with min-entropy $k = n - \Delta$ and can be implemented by explicitly constructible deterministic communication protocols with complexity $O(\Delta)$.

Theorem 4.5. *Let $\Delta(n) \geq m(n)$ be integer functions computable in time polynomial in n. Assume that $\Delta(n) \leq \eta n$ where η is the constant from Theorem 4.4. There is an explicitly constructible family of deterministic communication protocols $E = \{E_n\}$ of complexity $\Delta(n)/\eta$ such that for every sufficiently large n, $E_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ is an $(n - \Delta(n), 2^{-300m(n)})$-2-source extractor.*

Proof. Given functions $\Delta(n) \geq m(n)$ we consider the extractor E_{Had} when applied on inputs of length $\Delta(n)/\eta \leq n$. The output length of this extractor is $\Delta(n)$ and we chop it to length $m(n)$. Note that this extractor has error $\epsilon \leq 2^{-300m(n)}$.

Given $x = (x_1, x_2)$ of length n, we construct $E_n(x)$ by applying E_{Had} on the first $\Delta(n)/2\eta$ bits of x_1 and x_2. More precisely, Let $T = \{1, \ldots, \Delta(n)/2\eta\}$ and define $E_n(x_1, x_2) = E_{had}(x_1|_T, x_2|_T)$. We first note that this indeed gives that $E = \{E_n\}$ is an explicitly constructible family of deterministic communication protocols of complexity $\Delta(n)/\eta$. More precisely, each party sends the first $\Delta(n)/2\eta$ bits of his input and then each of the parties can apply the explicit extractor E_{Had}.

Let X_1, X_2 be independent distributions (where each one is over $\{0,1\}^{n/2}$) and $H_\infty(X_1)+H_\infty(X_2) \geq n - \Delta(n)$. Let $X_1' = X_1|_T$ and $X_2' = X_2|_T$. By Lemma 2.1 we have that for $i \in \{1,2\}$, $H_\infty(X_i') \geq H_\infty(X_i) - (n/2 - |T|)$. Note that X_1', X_2' are independent and

$$H_\infty(X_1') + H_\infty(X_2') \geq H_\infty(X_1) + H_\infty(X_2) - 2(n/2 - |T|) \geq 2|T| - \Delta(n)$$

We can assume w.l.o.g. that $\eta \leq 1/10$ and therefore:

$$H_\infty(X_1') + H_\infty(X_2') \geq \Delta(n)/\eta - \Delta(n) \geq \frac{9}{10} \cdot \frac{\Delta(n)}{\eta}$$

It follows that $E_n(X_1, X_2) = E_{Had}(X_1', X_2')$ is $2^{-300m(n)}$-close to uniform as required. □

4.1.3 Extractors for distributions recognizable by communication protocols

We now observe that 2-source extractors can be viewed as extractors for distributions recognizable by low complexity communication protocols.

Theorem 4.6. *If $E : \{0,1\}^{n/2} \times \{0,1\}^{n/2} \to \{0,1\}^m$ is a $(k - q - \log(1/\epsilon), \epsilon)$-2-source extractor then E is a $(k, 2\epsilon)$-extractor for distributions recognizable by communication protocols of complexity q.*

Proof. Let X be a distribution over $\{0,1\}^n$ that is recognizable by communication protocols of complexity q and let $B : \{0,1\}^{n/2} \times \{0,1\}^{n/2} \to \{0,1\}$ be a communication protocol that recognizes it. We are assuming that $H_\infty(X) \geq k$ which means that $|\mathsf{Supp}(X)| \geq 2^k$.

Let $Q : \{0,1\}^n \to \{0,1\}^q$ be the function that maps the input $x = (x_1, x_2)$ to the transcript $Q(x)$ of the protocol $B(x)$. Let $S_v = \{x : Q(x) = v\}$. As B is a communication protocol we have that for every $v \in \{0,1\}^q$ there exist sets $T_v^1, T_v^2 \subseteq \{0,1\}^{n/2}$ such that $S_v = T_v^1 \times T_v^2$. Furthermore as the answer of B on x is determined by the transcript $Q(x)$, there exists a subset $V \subseteq \{0,1\}^q$ such that

$$\mathsf{Supp}(X) = \{x : B(x) = 1\} = \bigcup_{v \in V} S_v = \bigcup_{v \in V} (T_v^1 \times T_v^2).$$

We say that v is tiny if $|S_v| \leq 2^{k-q-\log(1/\epsilon)}$ which implies $\Pr[Q(X) = v] \leq 2^{-(q+\log(1/\epsilon))}$. We denote the set of tiny v by T. Note that

$$\sum_{v \in V \cap T} \Pr[Q(X) = v] \leq 2^q \cdot 2^{-(q+\log(1/\epsilon))} = \epsilon$$

For every $v \in V \setminus T$ we have that the distribution $X_v = (X|Q(X) = v)$ can be written as a distribution (X_1, X_2) where X_1, X_2 are independent and $H_\infty(X_1) + H_\infty(X_2) = \log|S_v| \geq k - q - \log(1/\epsilon)$. Thus, for every such v, $E(X_v)$ is ϵ-close to uniform. We can express X as a convex combination

$$X = \sum_{v \in V} \Pr[Q(X) = v] \cdot X_v$$

and thus,

$$E(X) = \sum_{v \in V} \Pr[Q(X) = v] \cdot E(X_v) = \sum_{v \in V \setminus T} \Pr[Q(X) = v] \cdot E(X_v) + \sum_{v \in V \cap T} \Pr[Q(X) = v] \cdot E(X_v)$$

the distributions on the left term are all ϵ-close to uniform and the overall weight of the right term is bounded by ϵ. Overall we have that $E(X)$ is 2ϵ-close to uniform. □

4.1.4 Applying the main theorem

We prove Theorem 1.1 by applying Theorem 3.4 using the extractor E from Theorem 4.5.

Proof. (of Theorem 1.1) Let $A = \{A_n\}$ be an explicitly constructible family of randomized communication protocols with complexity $q(n)$ and $m(n)$ coins. Let $\eta > 0$ be the constant from Theorems 4.4 and 4.5. Fix some integer n and let $\Delta(n) = 2q(n) + 300m(n)$. In the assumption of Theorem 1.1 we assume that there exist a constant $\beta > 0$ such that $m(n) \leq \beta n - q(n)$. We can choose β so that $\Delta(n) \leq \eta n$. Note that by the definition of $\Delta(n)$ we have that $m(n) \leq \Delta(n)$. We meet the conditions of Theorem 4.5 and let E_n be the $(n - \Delta(n), 2^{-300m(n)})$-2-source extractor from Theorem 4.5. Let $k = n - 100m(n)$ and note that by Theorem 4.6 E_n is a $(k, 2^{-100m(n)})$-extractor for distributions recognizable by communication protocols of complexity $2q(n)$. Every function $C \in \mathcal{C}_{A_n}$ can be computed by a communication protocol of complexity $2q(n)$. This is because C is determined by the outputs of $A_n(x, r)$ and $A_n(x, r')$ for some $r, r' \in \{0,1\}^m$ and both $A_n(\cdot, r), A_n(\cdot, r')$ have communication protocols of complexity $q(n)$. Overall, we meet the conditions of Theorem 3.4 and can conclude that the function $B_n(x) = A_n(x, E_n(x))$ satisfies:

$$\Pr_{X \leftarrow U_n}[B_n(X) = f_n(X)] \geq 1 - 3\rho(n) - 2^{-10m(n)}$$

It is left to verify that $B = \{B_n\}$ is an explicitly constructible family of deterministic communication protocols with complexity $O(m(n)+q(n))$. This follows as by Theorem 4.5 E is an explicitly constructible family of deterministic communication protocols with complexity $O(\Delta(n)) = O(m(n) + q(n))$. Thus, we can implement B_n by first running E_n and then applying A_n.

Finally, note that the success of B is $1 - 3\rho(n) - 2^{-10m(n)}$ and not $1 - \rho(n)$ as promised. However, this can be fixed if we slightly amplify the success probability of A before the argument. First note that the Theorem holds trivially if $\rho = 0$. Thus we can assume that $\rho > 0$ and as A tosses $m(n)$ coins this implies that $\rho(n) \geq 2^{-m(n)}$. Thus, $1-3\rho-2^{-10m(n)} \geq 1-4\rho$. By a constant number of repetitions we can amplify the success probability of A from $1 - \rho$ to $1 - \rho(n)/4$ and this amplification only increases $m(n)$ and $q(n)$ by a constant factor and does not affect the asymptotic behavior of the parameters in the theorem. Applying Theorem 3.4 we now get that the success of B is indeed at least $1 - \rho(n)$ as required. □

4.2 Decision trees

In this section we prove Theorem 1.3 which gives an explicit version of Yao's lemma for decision trees. In Section 4.2.1 we give a precise formal definition of the notion of *explicitly constructible decision trees* that is defined loose terms in the introduction. In Section 4.2.2 we define extractors for bit-fixing sources and survey some constructions. In Section 4.2.3 we observe that extractors for bit-fixing sources can be viewed as extractors for distributions recognizable by low complexity decision trees. Finally, in Section 4.2.4 we apply Theorem 3.4 and prove Theorem 1.3.

4.2.1 Formal definition of explicitly constructible decision trees

Decision trees are "sublinear time" algorithms which have random access to the input and need to compute a function of the input by making few queries into the input. The reader is referred to [8] for a survey article on decision trees. A formal definition follows:

Definition 4.7 (Decision trees)**.** *A* deterministic decision tree *over* $\{0,1\}^n$ *with* complexity q *is a function* $B : \{0,1\}^n \to \{0,1\}$ *defined as follows: We consider the full binary tree* T_q *of height* q *and require that there exists a* labeling function L *that labels the nodes of* T_q *in the following way: internal nodes* v *of* T_q *are labeled by a number* $L(v) \in [n]$ *and leaves* v *of* T_q *are labeled by a bit* $L(v)$*. Given an input* x*,* $B(x)$

is defined by "following the path that corresponds to x in T_q". More precisely, starting at the root, on any node v we choose the child of v according to the value of $x_{L(v)}$. This process defines a leaf v of the tree T_q we denote this leaf by $Q(x)$. Finally, we define $B(x) = L(Q(x))$ that is "the label of the leaf that is associated with x in the tree". (Note that this also makes sense in case that B outputs many bits.)

A randomized decision tree *over $\{0,1\}^n$ with* complexity *q and m* coins *is a function $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ such that for every $r \in \{0,1\}^m$ the function $A_r(x) = A(x,r)$ is a deterministic decision tree.*

We now define *explicitly constructible* decision trees. Loosely speaking, the definition requires that the decision which variable to query can be computed efficiently. In the definition below "efficiently" means time polynomial in $\log n + q(n)$.

Definition 4.8 (Explicitly constructible decision trees)**.** *Let $q(n)$ be an integer function that can be computed in time polynomial in $\log n$. Let $B = \{B_n\}$ be a family of deterministic decision trees where B_n is over $\{0,1\}^n$ and has complexity $q(n)$. The family is* explicitly constructible *if there exists a polynomial-time Turing machine which given input (n, v) (where v is a node in the binary tree $T_{q(n)}$ encoded as $q(n)+1$ bit long string) outputs $L(v)$ where L is the labeling function associated with B_n.*

Let $q(n), m(n)$ be integer functions that can be computed in time polynomial in $\log n$. A family $A = \{A_n\}$ of randomized decision trees (where A_n is over $\{0,1\}^n$ with complexity $q(n)$ and $m(n)$ coins) is explicitly constructible *if there exists a polynomial-time Turing machine which given input (n, r, v) (where $r \in \{0,1\}^{m(n)}$ and v is a node of $T_{q(n)}$) outputs $L(v)$ where L is the labeling function associated with $A_n(\cdot, r)$.*

We remark that explicitness of decision trees can be defined in several natural ways and the results do not depend on the precise definition given in Definition 4.8.

Remark 4.9 (Decision trees and sublinear time algorithms)**.** *In definition 4.8 we are requiring that whenever the tree needs to make a new query, that query can be computed in time polynomial in $\log n + q(n)$. Thus, for $q(n) \geq \log n$ the overall computation can be simulated by a machine that runs in time polynomial in $q(n)$ that has random access to the input. Using this definition gives that our model of explicitly constructible deterministic decision trees captures uniform deterministic sublinear time algorithms that run in time polynomial in $q(n)$, and that for $m(n) = q(n)^{O(1)}$ our model of randomized decision trees captures uniform randomized sublinear algorithms that run in time polynomial in $q(n)$. An alternative definition would allow the machine to run in time polynomial in n (rather than polynomial in $\log n + q(n)$). (We can also allow $q(n), m(n)$ to be computable in time polynomial in n rather than $\log n$). This less stringent requirement still gives that any function computed by an explicitly constructible decision trees is also computable by polynomial-time machines. Our results also hold in the less stringent model.*

4.2.2 Extractors for bit-fixing sources

Bit-fixing sources were introduced by Chor et al. [10]. (We remark that the variant we consider in this paper is sometimes called *oblivious* bit-fixing sources).

Definition 4.10 (bit-fixing source)**.** *A distribution X over $\{0,1\}^n$ is a k-*bit-fixing source *if there exists a subset $S = \{i_1, \ldots, i_k\} \subseteq [n]$ such that $X_{i_1}, X_{i_2}, \ldots, X_{i_k}$ is uniformly distributed over $\{0,1\}^k$ and for every $i \notin S$, X_i is constant.*

We now define extractors for bit-fixing sources.

Definition 4.11. *A function $E : \{0,1\}^n \to \{0,1\}^m$ is a (k, ϵ)-*bit-fixing source extractor *if for every k-bit-fixing source X over $\{0,1\}^n$ the distribution $E(X)$ is ϵ-close to U_m.*

There are many constructions of bit-fixing source extractors in the literature [10, 5, 11, 33, 27, 16, 41]. We are interested in a "less challenging" setup in which $k = 9n/10$ and use the following theorem which is a special case of a theorem by Kamp and Zuckerman.

Theorem 4.12. *[27] There exists a constant $\eta > 0$ such that for sufficiently large n there is a poly(n)-time computable function $E_{KZ} : \{0,1\}^n \to \{0,1\}^{\eta n}$ that is a $(9n/10, 2^{-300\eta n})$-bit-fixing source extractor.*

We are going to use the extractor E_{KZ} to construct an extractor E that can extract from k-bit-fixing sources with $k = n - \Delta$ and can be implemented by explicitly constructible deterministic decision trees with complexity $O(\Delta)$. The argument below is very similar to the proof of Theorem 4.5.

Theorem 4.13. *Let $\Delta(n) \geq m(n)$ be integer functions that can be computed in time polynomial in $\log n$. Assume that $\Delta(n) \leq \eta n$ where η is the constant from Theorem 4.12. There is an explicitly constructible family of deterministic decision trees $E = \{E_n\}$ of complexity $\Delta(n)/\eta$ such that for every sufficiently large n, $E_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ is a $(n - \Delta(n), 2^{-300m(n)})$-bit-fixing source extractor.*

Proof. Given functions $\Delta(n) \geq m(n)$ we consider the extractor E_{KZ} when applied on inputs of length $\Delta(n)/\eta \leq n$. The output length of this extractor is $\Delta(n)$ and we chop it to length $m(n)$. Note that this extractor has error $\epsilon \leq 2^{-300m(n)}$.

Given x of length n, we construct $E_n(x)$ by applying E_{KZ} on the first $\Delta(n)/\eta$ bits of x. More precisely, Let $T = \{1, \ldots, \Delta(n)/\eta\}$ and define $E_n(x) = E_{KZ}(x_T)$. We first note that this indeed gives that $E = \{E_n\}$ is an explicitly constructible family of deterministic decision trees of complexity $\Delta(n)/\eta$. This is because the function E_{KZ} can be computed in polynomial-time.

Let X be a $(n - \Delta(n))$-bit-fixing source over $\{0,1\}^n$ and let $S \subset [n]$ be the set of size $n - \Delta(n)$ that is associated with it. We can assume w.l.o.g. that $\eta \leq 1/10$ and therefore:

$$|S \cap T| \geq |T| - \Delta(n) \geq \Delta(n)/\eta - \Delta(n) \geq \frac{9}{10} \cdot \frac{\Delta(n)}{\eta}$$

It follows that X_T is a k-bit-fixing source with $k \geq \frac{9}{10} \cdot \frac{\Delta(n)}{\eta}$ and therefore $E_n(X) = E_{KZ}(X_T)$ is $2^{-300m(n)}$-close to uniform as required. □

4.2.3 Extractors for distributions recognizable by decision trees

We now observe that extractors for bit-fixing sources can be viewed as extractors for distributions recognizable by low complexity decision trees.

Theorem 4.14. *If $E : \{0,1\}^n \to \{0,1\}^m$ is a $(n - q, \epsilon)$-bit-fixing source extractor then for every k, E is a (k, ϵ)-extractor for distributions recognizable by decision trees of complexity q.*

Proof. Let X be a distribution over $\{0,1\}^n$ that is recognizable by decision trees of complexity q and let $B : \{0,1\}^n \to \{0,1\}$ be a decision tree that recognizes it. Let $Q : \{0,1\}^n \to \{0,1\}^q$ be the function that maps the input x to the leaf $Q(x)$ of the decision tree $B(x)$. Let $S_v = \{x : Q(x) = v\}$. As B is a decision tree we have that for every leaf $v \in \{0,1\}^q$ there exist $i_1, \ldots, i_q \in [n]$ (the variables queried on the path to v) such that $S_v = \left\{x \in \{0,1\}^n : \forall j \in [q],\ x_{i_j} = v_j\right\}$. Furthermore as the answer of B on x is determined by the leaf $Q(x)$, there exists a subset $V \subseteq \{0,1\}^q$ such that

$$\mathsf{Supp}(X) = \{x : B(x) = 1\} = \bigcup_{v \in V} S_v.$$

For every $v \in V$ we have that the distribution $X_v = (X|Q(X) = v)$ can be written as a $(n-q)$-bit-fixing source. Thus, for every such v, $E(X_v)$ is ϵ-close to uniform. We can express X as a convex combination

$$X = \sum_{v \in V} \Pr[Q(X) = v] \cdot X_v$$

and thus,

$$E(X) = \sum_{v \in V} \Pr[Q(X) = v] \cdot E(X_v)$$

Overall we have that $E(X)$ is ϵ-close to uniform. □

4.2.4 Applying the main theorem

Theorem 1.3 follows by applying Theorem 3.4 using the extractor E from Theorem 4.13 and noting that by Theorem 4.14 this extractor meets the requirements of Theorem 3.4. The argument and parameter choices are identical to that used in Section 4.1.4 to prove Theorem 3.4. (In fact, Theorem 4.14 has better parameters than Theorem 4.6 but we do not gain when using the superior parameters). We omit the precise details.

4.3 Streaming algorithms

In this section we prove Theorem 1.4 which gives an explicit version of Yao's lemma for streaming algorithms. In Section 4.3.1 we give a precise formal definition of the notion of *explicitly constructible streaming algorithms* that is defined in loose terms in the introduction. In Section 4.3.2 we construct the extractors that we need in this application using 2-source extractors. Finally, in Section 4.3.3 we apply Theorem 3.4 and prove Theorem 1.4.

4.3.1 Formal definition of explicitly constructible streaming algorithms

A streaming algorithm (see e.g. [3, 15]) is an algorithm that reads its input "in one pass" using sublinear space. The complexity q of such an algorithm is the length (in bits) of the state that it keeps when scanning the input. At each step the algorithm $B(x)$ reads a bit from the input and updates its state by some transition function. The algorithm is explicitly constructible if updating the state (that is running the transition function) can be done in polynomial-time and linear space. A precise definition of deterministic streaming algorithms follows.

Definition 4.15 (Deterministic streaming algorithms)**.** *A* deterministic streaming algorithm *over* $\{0,1\}^n$ *with* complexity q *is a function* $B : \{0,1\}^n \to \{0,1\}$ *defined as follows: We require that there exists a* transition function $\delta : \{0,1\}^q \times \{0,1\} \to \{0,1\}^q$*. Given* $x \in \{0,1\}^n$ *we define the* state at step i *(denoted by* $\tau_i(x)$*) as follows:* $\tau_0(x)$ *is the string of* q *zeros and for* $j \geq 1$*,* $\tau_j(x) = \delta(\tau_{j-1}(x), x_j)$*. The output of* $B(x)$ *is the first bit in the final state.*

Let $q(n)$ *be an integer function that can be computed in time polynomial in* $\log n$ *and space* $O(\log n)$*. Let* $B = \{B_n\}$ *be a family of deterministic streaming algorithms where* B_n *is over* $\{0,1\}^n$ *and has complexity* $q(n)$*. The family is* explicitly constructible *if there exists a polynomial-time Turing machine that runs in linear space and when given input* (n, v, b) *where* $v \in \{0,1\}^{q(n)}$ *and* $b \in \{0,1\}$*, computes* $\delta_n(v, b)$ *where* δ_n *is the transition function associated with* B_n*.*

Remark 4.16 (Streaming algorithms that output many bits)**.** *There are two possible definitions of streaming algorithms that output many bits. We can require that the algorithm produces its entire output at the end (as we do in Definition 4.15). Alternatively, we can allow the algorithm to "print" output bits on the fly.*

To demonstrate the difference between models, note that the second notion allows a streaming algorithm to compute the identity function with sublinear complexity whereas the first notion does not (as the algorithm must remember the output when it finishes reading the input). When we refer to streaming algorithms that output many bits we use the first notion. In particular, Theorem 1.4 and 1.5 also hold for streaming algorithms that output many bits using the first notion. The proofs fail if we use the second notion. This is because the proofs rely on the fact that the output of the algorithm is determined by its final state.

We consider two notions of randomized streaming algorithms depending on whether the algorithm has one-way access or two-way access to its random coins. In the one-way access case we consider algorithms that toss coins on the fly. At each step, such an algorithm $A(x, r)$ may either read the next bit from the input x or the next bit from the "sequence of random coins" r. This is decided by a value of a "decision function" d that is applied on the internal state. The algorithm is explicitly constructible if both the transition and decision functions can be computed in polynomial-time and linear space.

Definition 4.17 (Randomized streaming algorithms with one-way access to randomness)**.** *A* randomized streaming algorithm with one-way access to randomness *over* $\{0,1\}^n$ *with* complexity q *and* m *random coins is a function* $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *defined as follows: We require that there exist a* transition function $\delta : \{0,1\}^q \times \{0,1\} \to \{0,1\}^q$ *and a* decision function $d : \{0,1\}^q \to \{0,1\}$*. Given* $x \in \{0,1\}^n$ *and* $r \in \{0,1\}^m$ *we define the* state at step i *(denoted by* $\tau_i(x,r)$*) as follows:* $\tau_0(x,r)$ *is the string of* q *zeros. For* $j \geq 1$ *we define* $\tau_j(x,r) = \delta(\tau_{j-1}(x,r), b)$ *where* b *is the first bit not yet read from* x *if* $d(\tau_{j-1}(x,r)) = 0$*, and* b *is the first bit not yet read from* r *otherwise. The output of the algorithm* $A(x,r)$ *is the first bit of the final state.*

Let $q(n), m(n)$ *be integer functions computable in time polynomial in* $\log n$ *and space* $O(\log n)$*. A family* $A = \{A_n\}$ *of randomized streaming algorithms with one-way access to randomness (where* A_n *is over* $\{0,1\}^n$ *with complexity* $q(n)$ *and* $m(n)$ *coins) is* explicitly constructible *if there exist two polynomial-time Turing machines that run in linear space and when given input* (n, v, b) *where* $v \in \{0,1\}^{q(n)}$ *and* $b \in \{0,1\}$ *the first machine computes* $\delta_n(v,b)$ *and the second machine computes* $d_n(v)$ *where* δ_n, d_n *are the transition and decision functions associated with* A_n*.*

We now consider randomized algorithms $A(x, r)$ that are allowed two-way access to their randomness. The definition

The two-way access case resembles the way we previously defined randomized communication protocols and streaming algorithms. In this case we only require that the algorithm $A(\cdot, r)$ is a deterministic streaming algorithm for every r.

Definition 4.18 (Randomized streaming algorithms with two-way access to randomness)**.** *A* randomized streaming algorithm with two-way access to randomness *over* $\{0,1\}^n$ *with* complexity q *and* m coins *is a function* $A : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *such that for every* $r \in \{0,1\}^m$ *the function* $A_r(x) = A(x,r)$ *is a deterministic streaming algorithm.*

Let $q(n), m(n)$ *be integer functions computable in time polynomial in* $\log n$ *and space* $O(\log n)$*. A family* $A = \{A_n\}$ *of randomized streaming algorithms with two-way access to randomness (where* A_n *is over* $\{0,1\}^n$ *with complexity* $q(n)$ *and* $m(n)$ *coins) is* explicitly constructible *if there exists a polynomial-time Turing machines that runs in linear space and when given input* (n, v, b, r) *where* $v \in \{0,1\}^{q(n)}$*,* $r \in \{0,1\}^m$ *and* $b \in \{0,1\}$ *computes* $\delta_{n,r}(v,b)$ *where* $\delta_{n,r}$ *is the transition function associated with* $A_n(\cdot, r)$*.*

In the definition above the input to the machine is of length $\log n + q(n) + m(n)$. Thus, the running time and space may be larger than allowed to algorithms with one-way access if $m(n) \gg q(n) \geq \log n$. This gives algorithms with two-way access to randomness an "unfair" advantage over algorithms with one-way access. Nevertheless, we use this definition as it makes our results in Theorem 1.5 stronger.

In the remainder of this section we prove Theorem 1.4 that shows how to weakly derandomize randomized streaming algorithms that toss a sublinear amount of random coins even when allowing 2-way access to randomness. Before proceeding with the proof, we first show that Theorem 1.5 follows from Theorem 1.4. Theorem 1.5 discusses randomized algorithm with one-way access to randomness that toss a polynomial number of coins. As we now explain, it is possible to reduce the number of random bits used by such algorithms while only slightly increasing their complecity. A celebrated result by Nisan [36] (see also [21, 42]) explicitly constructs pseudorandom generators against bounded space algorithms that have one-way access to randomness. Using these pseudorandom generators any randomized algorithm with complexity q and one-way access to poly(n) random coins can be converted to one that uses only $O(q \log n)$ random coins (where q is the complexity of the initial algorithm). The target algorithm runs the pseudorandom generator using its input randomness as seed whenever the original algorithm tosses a coin and needs 2-way access to randomness. This can be bypassed by storing the random coins in the internal memory and this requires additional storage of $O(q \log n)$ bits. We state a corollary of Nisan's pseudorandom generator for explicitly constructible streaming algorithms below.

Theorem 4.19 (Corollary of [36])**.** *Let $A = \{A_n\}$ be an explicitly constructible family of randomized streaming algorithms with one-way access to randomness. Assume that the family A has complexity $q(n) \geq \log n$ and a polynomial number of coins and that for every n, A_n computes a function $f_n : \{0,1\}^n \to \{0,1\}$ with success $1 - \rho(n) \geq 2/3$. Then, there is an explicitly constructible family $A' = \{A'_n\}$ of randomized streaming algorithms with one-way access to randomness, complexity $O(q(n) \log n)$ and $O(q(n) \log n)$ coins such that for every n, A'_n computes f_n with success $1 - \rho(n) - 2^{-q(n)}$.*

Together, Theorem 1.4 and Theorem 4.19 imply Theorem 1.5. In the remainder of this section we prove Theorem 1.4.

4.3.2 Extractors for distributions recognizable by streaming algorithms

We are interested in constructing extractors for distributions recognizable by streaming algorithms. We show how to construct such extractors using ideas similar to those used in [30, 26] in the context of extractors for sources samplable by small width branching programs. In fact, our task will be easier as we are once again interested in the "high entropy case".

Theorem 4.20. *There exists a constant $\lambda > 0$ such that the following holds: Let $q(n), m(n)$ be integer functions that can be computed in time polynomial in $\log n$ and space $O(\log n)$ and assume that $q(n) + m(n) \leq \lambda n$. There is an explicitly constructible family of deterministic streaming algorithms $E = \{E_n\}$ of complexity $O(q(n) + m(n))$ such that for every sufficiently large n, $E_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ is a $(n - 100m(n), 2^{-100m(n)})$-extractor for distributions recognizable by streaming algorithms of complexity $q(n)$. Furthermore, E_n only depends on a prefix of its input of length $O(q(n) + m(n))$.*

Proof. Let $\Delta(n) = q(n) + 300m(n)$ and let $\eta > 0$ be the constant from Theorem 4.4. Let $\ell(n) = max(10\Delta(n), m(n)/\eta)$ and note that $\ell(n) = O(q(n) + m(n))$. We can choose the constant $\lambda > 0$ in the theorem statement to be sufficiently small so that $\ell(n) \leq n$. We apply Theorem 4.4 and let $E'_n : \{0,1\}^{\ell(n)/2} \times \{0,1\}^{\ell(n)/2} \to \{0,1\}^{\eta\ell(n)}$ be a $(9\ell(n)/10, 2^{-300\eta\ell(n)})$-2-source extractor from the theorem. We have that $m(n) \leq \eta\ell(n)$ and can chop the output of E'_n to length $m(n)$ and then the error of E'_n is less than $2^{-300m(n)}$. Let $T_1 = \{1, \ldots, \ell(n)/2\}$, $T_2 = \{\ell(n)/2 + 1, \ldots, \ell(n)\}$ and define $E_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ by

$$E_n(x) = E'_n(x_{T_1}, x_{T_2})$$

We now observe that $E = \{E_n\}$ can trivially be implemented by an explicitly constructible family of deterministic streaming algorithms with complexity $O(q(n) + m(n))$. Consider a streaming algorithm that

given $x \in \{0,1\}^n$ reads $x_{T_1 \cup T_2}$ and stores it in its memory (this takes space $\ell(n) = O(q(n) + m(n))$). At this point it applies E'_n (that can be computed in time polynomial in $\ell(n)$ and space $O(\ell(n))$ by Theorem 4.4).

We now show that E_n is a $(n - 100m(n), 2^{-100m(n)})$-extractor for distributions recognizable by streaming algorithms of complexity $q(n)$. Fix some integer n and let X be a distribution on $\{0,1\}^n$ with $H_\infty(X) \geq n - 100m(n)$ and such that X is recognized by a deterministic streaming algorithm B of complexity $q(n)$. For $x \in \{0,1\}^n$ let $s(x)$ denote the state of B after reading $\ell(n)/2$ bits of x. For $y \in \{0,1\}^{q(n)}$ let $X^y = (X|s(X) = y)$. We have that X is uniform over $\{x \in \{0,1\}^n : B(x) = 1\}$ and therefore X^y is uniform over $\{x \in \{0,1\}^n : B(x) = 1,\ s(X) = y\}$. It follows that for every $y \in \{0,1\}^{q(n)}$ the distributions $X^y_{T_1}, X^y_{T_2}$ are independent. This is because after reading $\ell(n)/2$ bits, B only remembers the state y. We define $Y = s(X)$ and using Lemma 2.2 we have that with probability $1 - 2^{-200m(n)}$ over $y \leftarrow Y$,

$$H_\infty(X^y) \geq H_\infty(X) - q(n) - 200m(n) \geq n - q(n) - 300m(n).$$

We call $y \in \{0,1\}^q$ good if the condition above holds. For each good y by Lemma 2.1

$$H_\infty(X^y_{T_1 \cup T_2}) \geq \ell(n) - q(n) - 300m(n) \geq 9\ell(n)/10$$

It follows that for every good y, $E_n(X^y) = E'_n(X^y_{T_1}, X^y_{T_2}))$ is $2^{-300m(n)}$-close to uniform. Overall we have that $E_n(X)$ is $2^{-100m(n)}$-close to uniform as required. □

4.3.3 Applying the main theorem

Theorem 1.4 follows by applying Theorem 3.4 using the extractor E from Theorem 4.20. The proof is very similar to the previous arguments for communication protocols and decision trees and therefore we only highlight the differences. Similarly to the previous proofs, given a randomized streaming algorithm $A = \{A_n\}$ with complexity $q(n)$ and $m(n)$ coins we have that every function in $\mathcal{C}_{A_n}$ can be computed by a deterministic streaming algorithm of complexity $2q(n)$. Therefore, Theorem 4.20 (setting the complexity to $2q(n)$) is precisely what we need in Theorem 3.4. Finally we note that $B(x) = A(x, E(x))$ can be implemented by an explicitly constructible streaming algorithm of complexity $O(q(n) + m(n))$. This is done by first reading $\ell = O(q(n) + m(n))$ bits of the input x and storing them in memory. By Theorem 4.20 we can now compute $E(x)$ (without reading the rest of the input x). At this point we simulate $A(x, E(x))$ which requires one-way access to x. As we already read ℓ bits of x we use the bits stored in memory first and once A wants to read bit $(\ell + 1)$ of x we access x directly and continue reading more bits from the input x.

4.4 Constant-depth algorithms

In this section we prove Theorem 1.6 which gives an explicit version of Yao's lemma for AC^0 algorithms. In Section 4.4.1 we construct the extractors that we need in this application. These extractors are based on the hardness of the parity function. Finally, in Section 4.4.2 we apply Theorem 3.4 and prove Theorem 1.6.

4.4.1 Extractors for distributions recognized by constant-depth circuits

For this application we construct the following extractor.

Theorem 4.21. *For any constants $c, d, e > 1$ there is a constant $d' > 1$ and a uniform family $E = \{E_n\}$ of circuits of polynomial-size and depth d' such that $E_n : \{0,1\}^n \to \{0,1\}^m$ for $m(n) = (\log n)^e$ and E_n is a $(n - 100m(n), 2^{-100m(n)})$-extractor for sources recognizable by circuits of size n^c and depth d.*

The key ingredient in the proof is the classical results of Hastad on hardness of the parity function.

Theorem 4.22 (Hardness of parity [19]). *Let $p_n : \{0,1\}^n \to \{0,1\}$ denote the parity function on n bits. For every integer d, sufficiently large n, and every circuit C of size $t = 2^{n^{1/(d+1)}}$ and depth d, $\Pr_{Y \leftarrow U_n}[C(Y) = p_n(Y)] \leq 1/2 + t^{-1}$.*

It is easy to see that Theorem 4.22 implies that the parity function is an extractor for high values of k.

Theorem 4.23 (Parity is an extractor). *For every integer d and large enough n, let $t = 2^{n^{1/(d+1)}}$ and assume that $\Delta \leq n^{1/3d}$ (so that $2^{-2\Delta} > t^{-1}$). Then p_n is a $(n - \Delta, 2^{-\Delta})$-extractor for sources recognizable by circuits of size $s = t - O(1)$ and depth d.*

Proof. Assume otherwise. Then there exists a distribution X over $\{0,1\}^n$ that is recognizable by circuits of size s and depth d such that $H_\infty(X) \geq n - \Delta$ and $p_n(X)$ is not $(2^{-\Delta})$-close to uniform. Let $R(x)$ be a circuit that recognizes X and consider the following probabilistic circuit $C(x)$ which on input $x \in \{0,1\}^n$ runs $R(x)$. If $R(x)$ outputs one, then C outputs the more likely value of p_n on X (this constant a_n can be hardwired to C). Otherwise, C outputs a random bit. We now compute the success probability of C on a uniformly chosen input $Y \leftarrow \{0,1\}^n$.

$$\Pr[C(Y) = p_n(Y)] \geq 1/2 + \Pr[R(Y) = 1] \cdot 2^{-\Delta} \geq 1/2 + 2^{-2\Delta} > 1/2 + t^{-1}$$

Note that C has size $s + O(1) \leq t$ and depth $d + 1$. By a standard argument we can transform C into a deterministic circuit with the same complexity and thus, we get a contradiction to Theorem 4.22.

In fact, when transforming C into a deterministic circuit we obtain one of the following four circuits (depending on the constant a_n and the choice of the random bit): the constant zero, the constant one, the circuit R or the negation of R. All these circuits have depth that is bounded by the depth of R and so the final depth is d and not $d + 1$. □

For our application we need extractors that output many bits. We do this by applying the parity function on disjoint blocks of the input. The theorem below shows that using the construction above produces an extractor. We introduce the following notation: Let n and ℓ be integers. Given a string $x \in \{0,1\}^n$ we consider a partition of x into n/ℓ substrings where each is of length ℓ. We denote these strings by $x[1], \ldots, x[\ell]$.

Theorem 4.24. *Let $m \leq n/\ell$ and let $E' : \{0,1\}^\ell \to \{0,1\}$ be a $(\ell - (\Delta + \log(3m/\epsilon)), \epsilon/2m)$-extractor for sources recognizable by size s and depth d. Let $E : \{0,1\}^n \to \{0,1\}^m$ be defined by $E(x) = E'(x[1]), \ldots, E'(x[m])$. Then E is a $(n - \Delta, \epsilon)$-extractor for distributions recognizable by circuits of size $s - n$ and depth d.*

Proof. Assume otherwise. Then there exists a distribution X over $\{0,1\}^n$ that is recognizable by circuits of size $s - n$ and depth d such that $H_\infty(X) \geq n - \Delta$ and $E(X)$ is not ϵ-close to uniform. Let $R(x)$ be a circuit that recognizes X. By a standard hybrid argument the fact that $E(X)$ is not ϵ-close to uniform gives that it is possible to predict the next bit of $E(X)$ at some index i. More precisely, that there exists $1 \leq i \leq m$ and a "predictor function" $P : \{0,1\}^{i-1} \to \{0,1\}$ such that:

$$\Pr[P(E'(X[1]), \ldots, E'(X[i-1])) = E'(X[i])] \geq 1/2 + \epsilon/m$$

Given $x \in \{0,1\}^n$ and $1 \leq i \leq n/\ell$ we use x^{-i} to denote $x[1], \ldots, x[i-1], x[i+1], \ldots, x[n/\ell]$ (that is all the coordinates of x except $x[i]$). For $a \in (\{0,1\}^\ell)^{n/\ell - 1}$ we consider the event $A_a = \{X^{-i} = a\}$. By an averaging argument we have that with probability at least $\epsilon/2m$ over the choice of $a \leftarrow X^{-i}$ it holds that

$$\Pr[P(E'(X[1]),\ldots,E'(X[i-1])) = E'(X[i]) \mid A_a] \geq 1/2 + \epsilon/2m$$

We call an a that satisfies the condition above "predicting". We call an a such that $H_\infty(X[i]|A_a) \geq \ell - \Delta - \log(3m/\epsilon)$ "heavy". By Lemma 2.2 with probability at least $1-\epsilon/3m$ over the choice of $a \leftarrow X^{-i}$ the element a is heavy. It follows that there exists an element a in the support of X^{-i} that is both predicting and heavy. Fix such an a and let X' denote the distribution $(X[i]|A_a)$. This is a distribution over $\{0,1\}^\ell$. Note that X' is recognizable by a circuit R' which given $x' \in \{0,1\}^\ell$ is defined by:

$$R'(x') = R(a_1,\ldots,a_{i-1},x',a_i,\ldots,a_{\ell-1})$$

Thus, X' is recognizable by circuits of size $(s-n)+n \leq s$ and depth d. Note that $P(E'(a_1),\ldots,E'(a_{i-1}))$ is a constant. We denote it by $b \in \{0,1\}$. It follows that

$$\begin{aligned}
\Pr[b = E'(X')] &= \Pr[b = E'(X[i]) \mid A_a] \\
&= \Pr[P(E'(a_1),\ldots,E'(a_{i-1})) = E'(X[i]) \mid A_a] \\
&\geq 1/2 + \epsilon/2m
\end{aligned}$$

Which contradicts the fact that E' is a $(\ell - (\Delta + \log(3m/\epsilon)), \epsilon/2m)$-extractor for sources recognizable by circuits of size s and depth d. □

Using Theorem 4.24 we can get extractors that output m bits using the parity function on ℓ bits. There are several ways to set the parameters. The natural one is to set $\ell = n/m$ and this gives the following corollary:

Corollary 4.25. *There is a constant $\alpha > 0$ such that for every sufficiently large n and $m \leq n^{1/\alpha d}$ the function $E : \{0,1\}^n \to \{0,1\}^m$ defined by $E(x) = (p_{n/m}(x[1]),\ldots,p_{n/m}(x[m]))$ is a $(n - n^{1/\alpha d}, 2^{-100m})$-extractor for sources recognizable by circuits of size $2^{n^{1/\alpha d}}$ and depth d.*

The extractor above can be computed in polynomial time and could be plugged into Theorem 3.4 and give exponentially small failure probability. However, we want extractors that are computable by polynomial-size constant-depth circuits. We choose $\ell = (\log n)^a$ where a is a constant that depends on the depth so that parity on ℓ bits can be computed by constant-depth circuits with size polynomial in n. This gives an extractor with weaker parameters that are still sufficient for our application. Specifically, this gives the extractor of Theorem 4.21.

Proof. (of Theorem 4.21) We are given n and $m(n) = (\log n)^e$. We choose ℓ so that $2^{\ell^{1/d+1}}$ is larger than n^{3c}. Note that this can be done for $\ell = (\log n)^a$ for some sufficiently large constant a that depends on c and d. We also require that a is large enough so that $\ell^{1/3d} = (\log n)^{a/3d} \geq m(n)^2 = (\log n)^{2e}$. For this choice, Theorem 4.23 gives that p_ℓ is a $(n - 200m(n), 2^{-200m(n)})$-extractor for sources recognizable by circuits of size n^{2c} and depth d. By applying Theorem 4.24 we get an $(n - 100m(n), 2^{-100m(n)})$-extractor for sources recognizable by circuits of size n^c and depth d as required.

For every $d' > 2$, the function p_ℓ can be computed by a uniform family of circuits of size $2^{O(\ell^{\frac{1}{d'-1}})}$ and depth d'. Thus, for sufficiently large d', p_ℓ can be computed by a uniform family of circuits of size polynomial in n and depth d'. It follows that the family E is computable by a uniform family of polynomial-size constant-depth circuits. □

4.4.2 Applying the main theorem

We now explain how to prove Theorem 1.6. An advantage of this setup is that we can reduce the number of random bits of uniform randomized AC^0 algorithms with appropriate pseudorandom generators.

Theorem 4.26 (Pseudorandom generators against AC^0 [38, 29])**.** *For every randomized uniform AC^0 algorithm A that computes some function f with success $1 - \rho(n)$ there is randomized uniform AC^0 algorithm A' such that for every constant $v > 1$ A' computes f with success $1 - \rho(n) - n^{-v}$ and uses only $(\log n)^e$ random coins where e is a constant that depends on the depth of A.*

We are given a randomized uniform AC^0 algorithm that computes some function f with success $1 - \rho(n) \geq 2/3$. We can apply Theorem 4.26 to obtain a randomized uniform AC^0 algorithm which uses $m(n) = (\log n)^e$ coins for some constant e that has success probability $1 - \rho(n) - n^{-v}$ for any constant $v > 1$. Now that we have an algorithm with $m(n) = (\log n)^e$ coins we can apply Theorem 3.4 using the extractor of Theorem 4.21 and obtain Theorem 1.6. The argument is similar to that used in the previous sections. We remark that by reviewing the argument of [38] the term n^{-v} in Theorem 4.26 can be replaced by $2^{-(\log n)^v}$. As a consequence (and by slightly modifying the parameters in our proof) it is also possible to achieve the same replacement in Theorem 1.6.

4.5 General polynomial-time algorithms

We now explain how to prove Theorem 1.7 that gives a conditional explicit version of Yao's lemma for general polynomial-time algorithms. The proof of Theorem 1.7 imitates the proof of Theorem 1.6. The latter can be seen as providing a reduction that given a randomized algorithm $A(x, r)$ that is not weakly derandomized by the suggested extractor provides a circuit C that computes the parity function well on average where the size and depth of C are not much larger than that of A. We stress that the reduction and its proof do not use any properties of the parity function. Thus, we can replace the parity function with any function h and repeat the construction of the extractor from Theorem 4.21 replacing parity with h. The same argument would show that if there is a polynomial-time randomized algorithm $A(x, r)$ that is not weakly derandomized using the extractor then h can be computed well on the average by a polynomial-size circuit (where the precise polynomial depends on the running time of A). This implies that given a language in BPP that is accepted by some polynomial-time randomized algorithm A we can compute a polynomial n^c that depends on the running time of A such that given a function h that is sufficiently hard on average for circuits of size n^c , the deterministic algorithm $B(x) = A(x, E(x))$ (where E is the extractor of Theorem 4.21 using h instead of parity) weakly derandomizes A.

This high level summary hides a lot of details. Specifically, we want B to be computed in polynomial-time and therefore require that E is computed in polynomial-time. Note that the computation of E needs to apply h on certain blocks of the input x. In the case of parity, we used the fact that for size $\approx 2^{\ell^{1/d}}$, parity on ℓ bits is very hard on average for circuits of depth d but is easy for circuits of depth $O(d)$. We chose $\ell = \text{polylog}(n)$ so that $2^{\ell^{1/d}}$ is polynomial in n and the extractor construction applied parity on inputs of length ℓ. In the current setup we need that h behaves like parity in the sense that h is very hard on average for algorithms with limited resources but easy for algorithms with larger resources. Specifically we want that the function h is very hard on average for circuits of size n^c yet easy for circuits of larger polynomial-size. This is indeed the assumption made in Theorem 1.7 and this allows B to be computed in polynomial-time.

While the overall argument proving Theorem 1.7 is similar to that made in the proof of Theorem 1.6 the parameter choices are somewhat different. We omit the precise details as subsequent to this paper [28] gave a simpler proof of an improved version of Theorem 1.7. See Section 5 for a discussion.

5 Open problems and subsequent work

In this paper we present a general technique to prove explicit versions of Yao's lemma and apply it in several computational settings. A natural research direction is to handle other classes of randomized algorithms. Specifically, the approach of "extracting randomness from the input" was used in [17] to give a weak derandomization of the class SL. (These results are superseded by Reingold's breakthrough result that $SL = L$ [43]). It is open whether $RL = L$ and we find it interesting to try and get weak derandomization of RL.

The notion of weak derandomization that we consider only requires that the deterministic algorithm succeeds with high probability when the input is chosen according to the uniform distribution. A stronger notion considered in [23, 50] requires the deterministic algorithm to succeed with high probability on every distribution that is efficiently samplable. We find it interesting to extend the results of this paper to hold for "interesting distributions". We remark that the approach of this paper can be extended to allow every high min-entropy distribution that is recognizable by the corresponding class of algorithms. Loosely speaking, this is because our extractors already extract randomness from such distributions. More generally, in Theorem 3.4 it is possible to replace the uniform distribution over inputs with any distribution P of sufficiently high min-entropy by modifying the definition of recognizable distributions (and appropriately modifying the parameters). A sketch of the argument is that Definition 3.1 says that a distribution is recognizable by a function C if it is of the form $(X|C(X) = 1)$ where $X \leftarrow U_n$. We can repeat the argument for a distribution P with sufficiently high min-entropy by modifying the definition and replacing the experiment $X \leftarrow U_n$ with $X \leftarrow P$ in the definition of recognizable distributions.

Subsequent work: Subsequent to this work Salil Vadhan and Dieter van Melkebeek suggested an alternative proof for Theorems 1.6 and 1.7. This approach is described and further developed by Jeff Kinne, Dieter van Melkebeek and the author in [28]. The approach of [28] uses "seed-extending pseudoramdom generators". These are pseudorandom generators that remain secure even when the distinguisher receives the seed of the pseudorandom generator. It is shown in [28] that every class of randomized algorithms can be weakly derandomized if one can construct a seed-extending pseudorandom generator for the "non-uniform version" of the class. In some setups, one can construct such generators from functions that are hard one average for the "non-uniform version" of the class by using the pseudorandom generator construction of [38] (which is seed-extending). This approach gives simpler and proofs of Theorems 1.6 and 1.7. Furthermore, it proves stronger versions of these theorems as explained in the introduction

The approach of [28] can also be used for communication protocols. In particular, [28] extends Theorem 1.1 to the setup of k-party communication protocols for $k \geq 2$. However, the approach of [28] gives inferior parameters than Theorem 1.1 in case $k = 2$. Specifically, for 2-party communication protocols and the setup of Theorem 1.1 [28] gives a deterministic communication protocol with complexity $\Omega(q(n) \cdot m(n))$ whereas Theorem 1.1 gives a protocol with complexity $O(q(n) + m(n))$. A natural open problem is to obtain the behavior of Theorem 1.1 for k-party protocols. One way to achieve this is to explicitly construct an extractor for distributions recognizable by k-party communication protocols.

In [28] there is a formal comparison between the extractor based approach of this paper and the pseudorandom generator based approach. It is observed that the technique used to prove Theorem 3.4 can be used to argue that extractors for recognizable distributions give rise to seed-extending pseudorandom generators. In particular, this means that the results in this paper can be recast as following using the pseudorandom generator approach. The reader is referred to [28] for precise details.

Acknowledgements

I am grateful to Oded Goldreich, Dieter van Melkebeek, Chris Umans, Salil Vadhan and Avi Wigderson for interesting discussions on "weak derandomization". I am grateful to Eyal Kushilevitz and Ilan Newman for interesting discussions on communication complexity and decision trees. I thank anonymous referees for comments that improved the presentation of this paper.

References

[1] L. Adelman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, 1978.

[2] M. Ajtai. Approximate counting with uniform constant depth circuits. *Advances in computational complexity theory*, pages 1–20, 1990.

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

[4] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[5] M. Ben-Or and N. Linial. Collective coin flipping. *ADVCR: Advances in Computing Research*, 5:91–115, 1989.

[6] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, Nov. 1984.

[7] J. Bourgain. More on the sum-product phenomenon in prime fields and its applications. *International Journal of Number Theory*, 1:1–32, 2005.

[8] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.

[9] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, Apr. 1988. Special issue on cryptography.

[10] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky. The bit extraction problem or t-resilient functions. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 396–407, 1985.

[11] A. Cohen and A. Wigderson. Dispersers, deterministic amplification and weak random sources. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 14–25, 1989.

[12] Y. Dodis, A. Elbaz, R. Oliveira, and R. Raz. Improved randomness extraction from two independent sources. In *RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 334–344, 2004.

[13] Z. Dvir. Extractors for varieties. In *24th Annual IEEE Conference on Computational Complexity*, pages 102–113, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Z. Dvir, A. Gabizon, and A. Wigderson. Extractors and rank extractors for polynomial sources. *Computational Complexity*, 18(1):1–58, 2009.

[15] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams (extended abstract). In *SODA*, pages 165–174, 2000.

[16] A. Gabizon, R. Raz, and R. Shaltiel. Deterministic extractors for bit-fixing sources by obtaining an independent seed. *SICOMP: SIAM Journal on Computing*, 36(4):1072–1094, 2006.

[17] O. Goldreich and A. Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

[18] O. Goldreich and A. Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA*, pages 209–223, 2002.

[19] J. Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20. ACM, 1986.

[20] R. Impagliazzo. Can every randomized algorithm be derandomized? In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 373–374, 2006.

[21] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, 1994.

[22] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[23] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science*. IEEE, 1998.

[24] V. Kabanets. Derandomization: A brief overview. In *Electronic Colloquium on Computational Complexity, technical reports, TR 02-008*, 2002.

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

[26] J. Kamp, A. Rao, S. Vadhan, and D. Zuckerman. Deterministic extractors for small-space sources. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 691–700, 2006.

[27] J. Kamp and D. Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM J. Comput*, 36(5):1231–1247, 2007.

[28] J. Kinne, D. van Melkebeek, and R. Shaltiel. Pseudorandom generators and typically-correct derandomization. In *APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 574–587. Springer, 2009.

[29] A. Klivans. On the derandomization of constant depth circuits. In M. X. Goemans, K. Jansen, J. D. P. Rolim, and L. Trevisan, editors, *RANDOM-APPROX*, volume 2129 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2001.

[30] R. König and U. M. Maurer. Generalized strong extractors and deterministic privacy amplification. In *IMA Int. Conf.*, pages 322–339, 2005.

[31] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[32] P. B. Miltersen. Derandomizing complexity classes. In *Handbook of Randomized Computing, Kluwer*, pages 843–941, 2001.

[33] E. Mossel and C. Umans. On the complexity of approximating the vc dimension. In *Sixteenth Annual IEEE Conference on Computational Complexity*, pages 220–225, 2001.

[34] I. Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991.

[35] N. Nisan. Crew prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991.

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

[37] N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

[38] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, Oct. 1994.

[39] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[40] J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, 2000.

[41] A. Rao. Extractors for low weight affine sources. *Unpublished Manuscript*, 2008.

[42] R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 159–168, 1999.

[43] O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[44] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33:75–87, 1986.

[45] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[46] R. Shaltiel. How to get more mileage from randomness extractors. In *CCC '06: Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, pages 46–60, 2006.

[47] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[48] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the xor lemma. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[49] L. Trevisan and S. Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[50] L. Trevisan and S. P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[51] C. Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the Thirty-fourth Annual ACM Symposium on the Theory of Computing*, 2002.

[52] S. Vadhan. Randomness extractors and their many guises. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 9–12, 2002.

[53] U. Vazirani. Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources. *Combinatorica*, 7:375–392, 1987.

[54] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 Nov. 1982. IEEE.

[55] A. C.-C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing, 30 April-2 May, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979.

[56] A. C.-C. Yao. Lower bounds by probabilistic arguments (extended abstract). In *24th Annual Symposium on Foundations of Computer Science, 7-9 November 1983, Tucson, Arizona, USA*, pages 420–428, 1983.

[57] M. Zimand. Exposure-resilient extractors. In *IEEE Conference on Computational Complexity*, pages 61–72, 2006.

[58] M. Zimand. On derandomizing probabilistic sublinear-time algorithms. In *IEEE Conference on Computational Complexity*, pages 1–9, 2007.

[59] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11:345–367, 1997.