Electronic Colloquium on Computational Complexity, Report No. 184 (2023)

Towards Stronger Depth Lower Bounds

Gabriel Bathie
LaBRI, Université de Bordeaux
DIENS, PSL Research University

R. Ryan Williams
CSAIL, Massachusetts Institute of Technology

Abstract

A fundamental problem in circuit complexity is to find explicit functions that require large depth to compute. When considering the natural DeMorgan basis of $\{\mathrm{OR}, \mathrm{AND}\}$, where negations incur no cost, the best known depth lower bounds for an explicit function in NP have the form $(3-o(1))\log_2 n$, established by Håstad (building on others) in the early 1990s. We make progress on the problem of improving this factor of 3, in two different ways:

- We consider an "algorithmic method" approach to proving stronger depth lower bounds for non-uniform circuits in the DeMorgan basis. We show that slightly faster algorithms (than what is known) for counting the number of satisfying assignments on *subcubic*-size DeMorgan formulas would imply *supercubic*-size DeMorgan formula lower bounds, implying that the depth must be at least $(3+\varepsilon)\log_2 n$ for some $\varepsilon > 0$. For example, if #SAT on formulas of size $n^{2+2\varepsilon}$ can be solved in $2^{n-n^{1-\varepsilon}\log^k n}$ time for some $\varepsilon > 0$ and a sufficiently large constant k, then there is a function computable in $2^{O(n)}$ time with a SAT oracle which does not have $n^{3+\varepsilon}$-size formulas. In fact, the #SAT algorithm only has to work on formulas that are a conjunction of $n^{1-\varepsilon}$ subformulas, each of which is $n^{1+3\varepsilon}$ size, in order to obtain the supercubic lower bound. As a proof of concept, we show that our new algorithms-to-lower-bounds connection can be applied to prove new lower bounds for "hybrid" DeMorgan formula models which compute interesting functions at their leaves.
- Turning to the $\{\mathrm{NAND}\}$ basis, we establish a greater-than-$(3\log_2 n)$ depth lower bound against *uniform* circuits solving the SAT problem, using an extension of the "indirect diagonalization" method for NAND formulas. Note that circuits over the NAND basis are a special case of circuits over the DeMorgan basis; however, hard functions such as Andreev's function (known to require depth $(3-o(1))\log_2 n$ in the DeMorgan basis) can still be computed with NAND circuits of depth $(3+o(1))\log_2 n$. Our results imply that SAT requires polylogtime-uniform NAND circuits of depth at least $3.603\log_2 n$.

2012 ACM Subject Classification

Keywords and phrases DeMorgan formulas, depth complexity, circuit complexity, lower bounds, #SAT, NAND gates, SAT

Funding *Gabriel Bathie*: Portions of this work were completed while the author was visiting MIT.
R. Ryan Williams: Supported by NSF CCF-2127597 and a Frank Quick Faculty Research Innovation Fellowship. Portions of this work was completed while the author was visiting the Simons Institute for the Theory of Computing, at the University of California at Berkeley.

ISSN 1433-8092

1 Introduction

How deep must Boolean circuits be, in order to compute explicit functions? A simple counting argument shows that, with high probability, a random function $f : \{0,1\}^n \to \{0,1\}$ requires circuit depth at least $\Omega(n)$, for circuits over any bounded fan-in basis. However, finding specific (interesting) functions that require high depth is a major open problem. For example, it is open whether there are functions computable with a circuit family of size $s(n)$ that require circuits of $\omega(\log s(n))$ depth: this is essentially equivalent to the famous P vs NC^1 question which asks whether every polynomial-time problem can be simulated in parallel in only $O(\log n)$ depth.[1] Focusing on the natural case of the DeMorgan basis, where the gates of our circuits are ANDs and ORs of fan-in two with NOTs for free, it is only known that there is a function computable in P that requires circuits of depth $(3 - o(1)) \log_2 n$. This lower bound was established thirty years ago by Håstad in FOCS 1993 (following earlier work), with some low-order improvements since then [3, 26, 39, 22, 46]. (For more general bases, e.g., where all functions $g : \{0,1\}^2 \to \{0,1\}$ are allowed as gates, only a $(2 - o(1)) \log_2 n$ depth lower bound is known [38].)

There has been a stream of recent work, motivated by a compelling conjecture of Karcher-Raz-Wigderson [31], working towards stronger depth lower bounds via communication complexity [18, 21, 17, 35]. For example, a recent work of Dinur and Meir [18] recovers a $(3 - o(1)) \log_2 n$ depth lower bound for Andreev's function using communication complexity techniques. Mihajlin and Sofronova [35] use communication complexity methods to show (for instance) that a modified version of Andreev's function [3] requires $3.15 \log_2(n)$ depth when the top $0.29 \log_2(n)$ layers are AND gates.[2]

Note: *All logarithms in this paper will be in base two. From here on, we shall drop the subscript 2 in our logarithms.*

1.1 Our Results

In this paper, we provide two rather different lines of attack on the notorious problem of finding functions with a provable $(3 + \alpha) \log n$ depth lower bound, for some $\alpha > 0$. We first give a plausible "algorithmic method" approach for the non-uniform case, showing how faster algorithms for solving $\#\mathsf{SAT}$ on subcubic-size DeMorgan formulas would imply *supercubic*-size DeMorgan formula lower bounds (thus implying a $(3 + \alpha) \log n$ depth lower bound as well)[3], and we present unconditional depth lower bounds against *uniform* circuits attempting to solve the SAT problem, based on an extension of "indirect diagonalization" arguments to formulas.

An Approach via #SAT Algorithms.

Over the last decade, many connections have been shown between non-trivial algorithms for analyzing small circuits, and the task of constructing non-trivial functions that do not have small non-uniform circuits (e.g., [41, 24, 56, 57, 52, 15, 37, 28, 12] for a sample).

[1] As is standard, we say the size of a circuit is its number of gates, and the size of a formula is its number of leaves.

[2] Similar results (for a different modified Andreev's function) follow from average-case lower bounds via restriction-based techniques (Komargodski-Raz-Tal [32]) instead of communication complexity.

[3] Note that while an n^c size lower bound against formulas implies a $c \log n$ depth lower bound, the converse is not immediately true. One may obtain an n^c size lower bound from an $\alpha c \log n$ depth lower bound, where $\alpha > 1$ depends on the gate basis, by applying the results of Brent [8] and Spira [44].

The setting most relevant to us is that of subcubic-size DeMorgan formulas. Explicit lower bounds against them have been known for decades [45, 3, 26, 39, 22] by "shrinkage arguments", in which one sets variables to the formula in a careful way, and argues that the formula must "shrink" drastically in size from a relatively small number of variable settings. Analogues of these shrinkage arguments have been applied to derive faster algorithms for the Formula Satisfiability (Formula-SAT) problem and its counting version #Formula-SAT [41, 15, 14, 16, 47]. For example, it is known that, for $\varepsilon > 0$:

- [16] there is a #SAT algorithm running in $2^{n-n^{\varepsilon/1.63}}$ time on $n^{2.63-\varepsilon}$-size formulas, and
- [47] there is a #SAT algorithm running in $2^{n-n^{\varepsilon}}$ time on formulas of $n^{3-16\varepsilon}$ size.

In the case of "slightly superquadratic" formulas of size $n^{2.001}$ (for example), the above results imply #SAT algorithms running in $2^{n-n^{0.385}}$ time and $2^{n-n^{0.06}}$ time, respectively. Our first main result is that a seemingly minor-looking improvement on the running time of #SAT algorithms for "slightly superquadratic" formulas would already yield a non-trivial function with a supercubic DeMorgan formula lower bound.

▶ **Theorem 1.1** (#SAT For "Small" Formulas Implies "Large" Formula Lower Bounds). *There is a universal constant $k > 0$ such that, for every $\varepsilon \in (0, 1)$, if #SAT for $n^{2+2\varepsilon+o(1)}$ size formulas can be solved in $2^{n-n^{1-\varepsilon}\log^k n}$ time (even in zero-error randomized time), then there is a function in $\mathsf{E}^{\mathsf{NP}} = \mathsf{TIME}[2^{O(n)}]^{\mathsf{SAT}}$ that does not have $n^{3+\varepsilon}$ size formulas.*

In fact, the desired #SAT algorithm only has to succeed against DeMorgan formulas which are the conjunction of $n^{1-\varepsilon+o(1)}$ subformulas, where each subformula has $O(n^{1+3\varepsilon})$ size. It looks quite plausible that such a #SAT algorithm may be derivable from known methods: it is already known that such formulas are significantly more limited than general $n^{2+2\varepsilon+o(1)}$-size formulas [32, 35], and current algorithms [16] can solve #SAT on $n^{1+3\varepsilon}$-size formulas in time $O(2^{n-n^{1-1.8\varepsilon}})$, for small $\varepsilon > 0$.

The proof of Theorem 1.1 proceeds by first showing that the #SAT algorithmic hypothesis in fact implies a "non-trivial derandomization" for large, $n^{3+\varepsilon}$-size formulas: for any desired constant $\beta > 0$, there is an algorithm running in $2^n/n^{\omega(1)}$ time which can distinguish unsatisfiable $n^{3+\varepsilon}$-size formulas from those with at least $\beta \cdot 2^n$ satisfying assignments (a.k.a., a Gap-SAT algorithm, see Section 3 for a definition). Our reduction from Gap-SAT on "large" formulas to #SAT on "small" formulas works by applying *approximating polynomials* for DeMorgan formulas to a significant portion of the given "large" formula, in such a way that the counting of SAT assignments implicitly performs a portion of the large formula evaluation for us, allowing us to distinguish between the unsatisfiable and "highly satisfiable" cases. From there, we carefully adapt known strong connections between Gap-SAT algorithms for circuits and circuit lower bounds [13], to show an "ultra-fine-grained" connection: a non-trivial Gap-SAT algorithm for formulas of size $O(s(n))$ implies size-$s(n)$ formula lower bounds. That is, the difference between the size handled by the Gap-SAT algorithm and the size lower bound is only *constant factor.*

Generalization to Other Circuit Classes.

Along the way to Theorem 1.1, the connection proved between Gap-SAT algorithms and circuit lower bounds is rather general: any non-trivial Gap-SAT algorithm for a size-$O(s(n))$ circuit class $\mathcal{C}$ such that

(a) $\mathcal{C}$ can compute PARITY in size $s(n)$,
(b) $\mathcal{C}$ is closed under the OR of 3 copies of $\mathcal{C}$, and
(c) $\mathcal{C}$ is closed under negations of their outputs, and projections (0-1 assignments to some of the variables, and negations to some of the variables),

implies size-$s(n)$ lower bounds against $\mathcal{C}$. Two other natural examples of such $\mathcal{C}$ are $n^{2+\varepsilon}$-size formulas over the full binary basis, and $O(n)$-size circuits over any complete basis.

We use this connection to conclude some new unconditional circuit lower bounds. Kabanets, Koroth, Lu, Myrisiotis, and Oliveira [28] studied the class $\textsc{Formula}[s] \circ \mathcal{G}$ of Boolean functions computed by size-s DeMorgan formulas where the leaves may be any Boolean functions from a class $\mathcal{G}$. They showed that when the functions in $\mathcal{G}$ have low communication complexity, one can give interesting #SAT algorithms and pseudorandom generators for $\textsc{Formula}[n^{2-\varepsilon}] \circ \mathcal{G}$, and consequently also lower bounds (where $\varepsilon > 0$ denotes an arbitrarily small constant). Using our connection, we can prove lower bounds on $\textsc{Formula}[n^{2-\varepsilon}] \circ \mathcal{G}$ for classes $\mathcal{G}$ that do not have low communication complexity. The general statement is:

▶ **Theorem 1.2.** *For every $\varepsilon > 0$ and every class of functions $\mathcal{G}$ such that:*

- *$\mathcal{G}$ is closed under variable projections,*
- *$\textsc{Formula}[n^{2-\varepsilon}] \circ \mathcal{G}$ contains PARITY, and*
- *the acceptance probability of a conjunction of $O(n)$ functions from $\mathcal{G}$, over n variables, can be additively approximated to within $1/2^{n^{1-\varepsilon/3}}$ in $2^{n-\Omega(n^{1-\varepsilon/3})}$ deterministic time,*

E^{NP} does not have $\textsc{Formula}[n^{2-\varepsilon}] \circ \mathcal{G}$ circuits.

Applying Theorem 1.2 and prior work, we can conclude lower bounds against DeMorgan formulas composed with low-degree $\mathbb{F}_2$-polynomials.

▶ **Corollary 1.3.** *For all $\varepsilon > 0$, E^{NP} does not have $\textsc{Formula}[n^{2-\varepsilon}] \circ \mathcal{G}$ circuits, where $\mathcal{G}$ is the class of $n^{\varepsilon/3}$-degree polynomials over $\mathbb{F}_2$.*

Note that such circuits require $2^{\Omega(n^{\varepsilon})}$ bits to describe, to encode the polynomials.

An Approach via Indirect Diagonalization Methods.

A line of work initiated by Fortnow [19, 20, 48] proved model-independent time-space lower bounds (or equivalently, lower bounds against small-space algorithms) for SAT and related problems within the polynomial-time hierarchy. This approach was extended by Williams [53, 54, 55] into a framework which culminated in a proof that SAT does not have an $n^{o(1)}$-space algorithm running in time $n^{2\cos(\pi/7)} \simeq n^{1.801}$. (Buss and Williams [9] later proved that the $2\cos(\pi/7)$ exponent is optimal for this framework.)

An interesting feature of this line of work is that the proofs are highly algorithmic: one assumes SAT has a sufficiently efficient algorithm, and applies that algorithm in various ways until a contradiction to a known lower bound (by diagonalization) is reached. This type of reasoning is sometimes called "indirect diagonalization"; many other examples of such lower bounds can be found in the literature (e.g., [30, 2, 49, 48, 36]).

Here, we develop an indirect diagonalization method for lower bounds against uniform $O(\log n)$-depth circuits (and equivalently, $O(\log n)$-depth formulas). We establish supercubic depth lower bounds on solving SAT with uniform circuits over the NAND basis, where every gate computes the function

$$\mathsf{NAND} : x, y \mapsto \neg(x \wedge y).$$

There are no internal negations; only input variables can be negated. It is well-known that the NAND basis is universal (every Boolean function can be computed with a NAND circuit), and it is surprisingly expressive. For example, it is known that every boolean function on n variables has NAND formulas of depth $n + O(\log^* n)$ [34], nearly matching the $n + O(1)$ depth in DeMorgan's basis. Furthermore, the hard example function for DeMorgan formulas, Andreev's function, has NAND formulas of depth $(3 + o(1)) \log n$ (and therefore has NAND formulas of size $n^{3+o(1)}$), matching the lower bound for the DeMorgan basis up to low-order terms [22]. (See Appendix A for a construction of low-depth formulas for Andreev's function). Therefore, one cannot use Andreev's function to prove a significantly stronger depth lower bound for NAND formulas, and the lower bound proved in this work have deeper implications than a weak expressiveness of NAND formulas.

Here, our main result is that SAT requires supercubic depth for *uniform* circuits composed of NAND gates:

▶ **Theorem 1.4.** *For every $c < 4\cos(\pi/7) \simeq 3.603$, SAT does not have uniform NAND formulas of depth $(c - o(1)) \log n$.*

Our notion of uniformity can be defined as follows.

▶ **Definition 1.5** ("Input-aware" polylog-time uniform formulas)**.** *A family of formulas $(\varphi_n)_{n\geq 1}$ of depth $(c + o(1)) \log n$ is* polylog-time uniform *if there exists a* RAM *machine A_φ such that for every n, every $x \in \{0,1\}^n$ and every $b \in \{0,1\}^{\leq (c+o(1)) \log n}$, $A_\varphi(x, b, n)$ outputs a description of the gate at position b in φ_n in time $\log^{O(1)} n$. The gate at position b in*

Here, for a bitstring b of length at most $(c + o(1)) \log n$, the "gate at position b" in φ_n is the gate reached by iterating over all bits of b, starting from the root, moving to the left child of the current gate when reading a 0, and to its right child otherwise. We will see the need for this stronger definition in Section 4.2, where we need to give $O(\log n)$ bits of information from the input to the descriptor to dynamically select a subformula.

Observe that this is a more general notion than (say) PLOGTIME-uniformity, which is common in the literature [51], hence the lower bounds that we give against this model also hold for PLOGTIME-uniform formulas.

Under the hood, indirect diagonalization results often provide an unconditional method for simulating weak algorithms super-efficiently, using more powerful models. For example, all proofs that SAT cannot be solved in $n^{1+\delta}$ time and $\text{poly}\log(n)$ space, for various $\delta > 0$, exploit the fact that algorithms using small space can be quickly simulated by algorithms that use a small number of alternations [11] (a generalization of nondeterminism). Theorem 1.4 is obtained from constructing an extremely efficient method for evaluating a uniform NAND formula, with an algorithm using a small number of alternations. This evaluation method is combined with other ideas from previous indirect diagonalization lower bounds (in particular, results on "alternation-trading proofs" [55, 9]) to obtain the result.

Organization of the Rest of the Paper. Section 2 covers some preliminaries. Section 3 proves that faster #SAT algorithms for superquadratic-size formulas would imply supercubic-size lower bounds, along with an "ultra-fine-grained" connection from Gap-SAT algorithms to circuit lower bounds. Section 4 proves SAT lower bounds against NAND circuits of depth $3.6 \log n$. Section 5 concludes with a discussion of further work.

2 Preliminaries

We first recall some basic notions in circuit complexity that are relevant for the paper. Later in the preliminaries, we discuss some notions particular to the paper.

Formulas.

A *Boolean formula ϕ over n inputs* is a rooted full binary tree[4] whose internal nodes, called gates, are labeled by a function $f : \{0,1\}^2 \to \{0,1\}$, and leaves are labeled with either a variable x_i, its negation $\neg x_i$ for some $i, 1 \leq i \leq n$, or a constant $c \in \{0,1\}$. The depth of a formula is the depth of the underlying tree. A formula is *read-once* if each of its input variables appears on at most one leaf. Given an input $x \in \{0,1\}^n$, the value of ϕ at x can be defined as the value of its root on input x, where the value of a gate is defined inductively as follows:

$$\phi(g) = \begin{cases} c & \text{if } g \text{ is labeled with the constant } c \in \{0,1\} \\ x_i & \text{if } g \text{ is labeled with the variable } x_i \\ \neg x_i & \text{if } g \text{ is labeled with } \neg x_i \\ f(\phi(g_1), \phi(g_2)) & \text{if } g \text{ is labeled } f\text{, and } g_1, g_2 \text{ are the children of } g \end{cases}$$

Throughout this paper, we are concerned with formulas over the DeMorgan basis of functions $f : \{0,1\}^2 \to \{0,1\}$, including all 2-input Boolean functions except the XOR and EQUALS function. It is not hard to show that this is equivalent to considering formulas over AND/OR/NOT, with NOT gates considered to be "free" (zero cost in size and depth).

In the half of this work on SAT lower bounds (4), we focus on NAND formulas, that is, whose gates are all labeled with the function $\mathsf{NAND} : x, y \mapsto \neg(x \wedge y)$. DeMorgan's law implies that $\mathsf{NAND}(x, y) = \neg x \vee \neg y$, therefore NAND formulas are equivalent to "alternating OR/AND" formulas (up to negations of some inputs) where all gates of even depth are ORs, and odd-depth gates are ANDs (and the root is said to have depth 0). This "alternating" point of view will be useful when describing procedures to compute the value of a formula.

A *formula family* is a sequence $\{\phi_n\}_{n\in\mathbb{N}}$ of formulas such that for every n, ϕ_n is a formula on n inputs $x_1, \ldots, x_n$. A language $L \subseteq \{0,1\}^*$ is computed by[5] the formula family $\{\phi_n\}_{n\in\mathbb{N}}$ if for every n and every $x \in \{0,1\}^n$, $\phi_n(x) = 1 \Leftrightarrow x \in L$.

Since a separate formula is allowed for each input length, there are formula families of *constant* size and depth computing undecidable languages.

The *depth complexity* of a function $f : \{0,1\}^\star \to \{0,1\}$ is the function $d : \mathbb{N} \to \mathbb{N}$ such that $d(n)$ is the minimum depth of any formula computing f_n, the restriction of f to n-bit inputs. Observe that the notion does not change when we replace "formula" with "circuit": any circuit of depth d can be simulated by a formula of depth d and size $O(2^d)$. For this reason, formula size lower bounds imply depth complexity lower bounds, but the converse does not necessarily hold.

Uniformity.

In the half of this paper on SAT lower bounds (Section 4), we restrict our attention to uniform formulas: formula families such that the description of the n-input formula can be computed efficiently. We recall here the notion of uniformity, stated in the introduction, that we use.

▶ **Definition 1.5** ("Input-aware" polylog-time uniform formulas). *A family of formulas $(\varphi_n)_{n\geq 1}$ of depth $(c + o(1))\log n$ is* polylog-time uniform *if there exists a* RAM *machine A_φ such*

[4] *Full* means that every node has either zero or two children.
[5] In what follows, we may use "L has formulas ..." for "L is computed by a formula family..."

that for every n, every $x \in \{0,1\}^n$ and every $b \in \{0,1\}^{\leq (c+o(1))\log n}$, $A_\varphi(x,b,n)$ outputs a description of the gate at position b in φ_n in time $\log^{O(1)} n$. The gate at position b in

New to This Paper: An Important Complexity Class.

Finally, for any $c \geq 0$, we define $\mathsf{NDepth}\,[c\log n]$ to be the class of languages that have uniform polylog-time NAND formulas of depth$(c+o(1))\log n$. Observe that uniform log-depth NAND formulas can be efficiently simulated by random-access machines, due to the negligible runtime overhead of the description: for every $c > 1$, we have

$$\mathsf{NDepth}\,[c\log n] \subseteq \mathsf{TISP}\left[n^c, n^{o(1)}\right],$$

where $\mathsf{TISP}\left[n^c, n^{o(1)}\right]$ denotes the class of languages solvable by an algorithm (in a standard random-access model) running in $n^{c+o(1)}$ time and $n^{o(1)}$ space.

Notation for Alternating Complexity Classes

Along with the above notation for NAND formula complexity classes, in Section 4 it will be extremely convenient to use quantifier notation to express alternating complexity classes, that define a fined-grained version of the polynomial hierarchy, just as was used in prior work on alternation-trading proofs [54, 55, 9, 36]).

Given a complexity class $\mathcal{C}$ (defined by some machine model) and constants $k, (a_i)_{i\leq k}$, we consider the *alternating* complexity class $(Q_1\ n^{a_1})\ldots(Q_k\ n^{a_k})\ \mathcal{C}$, where, for every i, Q_i is one of $\forall$ or $\exists$. Computation in an alternating complexity class is defined as follows:

▶ **Definition 2.1** (Alternating computation). *A language L is in $(Q_1\ n^{a_1})\ldots(Q_k\ n^{a_k})\ \mathcal{C}$ is defined by k RAM machines $A_1,\ldots,A_k$ and a $\mathcal{C}$ machine M such that, given an input x of length n: for every $i \leq k$, A_i takes as input x_i along with y_i, a string of $n^{a_i+o(1)}$ nondeterministic bits, runs in time $n^{a_i+o(1)}$ and outputs a string x_i of length at most $n^{a_i+o(1)}$, with the convention that $x_0 := x$. The output of the last stage, x_k, is passed to the machine M.*

The computation accepts x starting from stage i if either:

- *$i = k+1$ and M accepts x_k,*
- *Q_i is $\exists$, and there exists a y_i such that the computation accepts x_i starting from stage $i+1$,*
- *Q_i is $\forall$, and for every y_i, the computation accepts x_i starting from stage $i+1$.*

An input x is in L if the computation x accepts starting from stage 1.

3 Counting SAT Assignments to “Small” Formulas Implies Lower Bounds for “Large” Formulas

In this section, we prove the following theorem:

Reminder of Theorem 1.1. *There is a universal constant $k > 0$ such that, for every $\varepsilon \in (0,1)$, if #SAT for $n^{2+2\varepsilon}$ size formulas can be solved in $2^{n-n^{1-\varepsilon}\log^k n}$ time (even in randomized zero-error time), then there is a function in $\mathsf{E}^{\mathsf{NP}} = \mathsf{TIME}[2^{O(n)}]^{\mathsf{SAT}}$ that does not have $n^{3+\varepsilon}$ size formulas.*

The proof is broken into two parts. First, in Section 3.1, we show that the #SAT hypothesis of Theorem 1.1 implies that the following promise problem can be solved in $2^n/n^{\omega(1)}$ time, for all constants $\beta > 0$.

Problem: GAP-FORMULA-SAT with gap $\beta \in (0,1)$
Input: A DeMorgan formula F on n inputs, with $n^{3+\varepsilon}$ size.
Promise: Either F is unsatisfiable, or at least $\beta \cdot 2^n$ of its 2^n assignments are satisfying.
Decide: which is the case.

Second, in Section 3.2, we show that the resulting algorithm for GAP-FORMULA-SAT for all constant $\beta > 0$ implies the circuit lower bound, building on prior work [56, 6, 13].

3.1 Obtaining a Gap-Formula-SAT Algorithm

In this section, we prove the following lemma:

▶ **Lemma 3.1.** *There is a universal constant $k > 0$ such that, for every $\varepsilon \in (0,1)$, if #SAT for $n^{2+2\varepsilon+o(1)}$-size formulas can be solved in $2^{n-n^{1-\varepsilon}\log^k n}$ time, then for any $\beta \in (0,1)$, the GAP-FORMULA-SAT problem with gap β on DeMorgan formulas of n inputs and size $n^{3+\varepsilon}$ can be solved in $2^n/n^{\omega(1)}$ time.*

Our Gap-Formula-SAT requires a couple of ingredients. First, we will use the following fundamental result for approximating DeMorgan formulas with low-degree polynomials, which built on a long line of prior work.

▶ **Theorem 3.2** (Reichardt [40], Kabanets-Koroth-Lu-Myrisiotis-Oliveira [29]). *For every positive integer s and $\alpha \in (0, 1/2)$, and for every DeMorgan formula F of size s, there is a polynomial P_F of degree $t \leq O(\sqrt{s}\log(1/\alpha))$, with rational coefficients of bit-length $O(t \cdot \log^2 s \cdot \log(1/\alpha))$, such that for every $a \in \{0,1\}^n$,*

$$|P_F(a) - F(a)| \leq \alpha.$$

Furthermore, given the description of F, a description of a P_F with the above property and degree $O(\sqrt{s}\log(1/\alpha)\log s)$ can be constructed in $s^{O(\sqrt{s}\log(1/\alpha)\log(s))}$ time.

The proof of Reichardt only shows the existence of such a polynomial P_F, and does not consider the cost of constructing it. Kabanets-Koroth-Lu-Myrisiotis-Oliveira [29] credit an anonymous reviewer with an algorithm for constructing the polynomial P_F; see their paper for details.

The second ingredient we need (which incidentally, is also used in the algorithm of Theorem 3.2) is the following decomposition result for formulas:

▶ **Theorem 3.3** (Impagliazzo-Meka-Zuckerman [25], Tal [46]). *There is a polynomial-time algorithm that, given a formula F of size S and a parameter $K \in [S]$, outputs a read-once formula F' of size $t \leq O(K)$ and subformulas $G_1, \dots, G_t$ of F of size $O(S/K)$, such that F is the composition of F' with $G_1, \dots, G_t$ (that is, inserting G_i into the i-th input of F', yields F).*

We now turn to proving Lemma 3.1. Suppose we are given a formula F of size $n^{3+\varepsilon}$, and we are promised that F is either unsatisfiable, or has at least $\beta 2^n$ satisfying assignments. We want to determine which is the case, using a #SAT algorithm for formulas of only $n^{2+2\varepsilon+o(1)}$ size.

First, we apply Theorem 3.3 with $K := n^{2-2\varepsilon}$, decomposing F into an F' of size $O(n^{2-2\varepsilon})$ and $t = O(n^{2-2\varepsilon})$ subformulas $G_1, \dots, G_t$ of size $O(n^{1+3\varepsilon})$.

Second, we compute an approximating polynomial $P_{F'}$ of degree $O(n^{1-\varepsilon}\log n \cdot \log(1/\alpha))$ for the polynomial F', with error $\alpha \in (0,1)$. (For now, let us leave α as a parameter.) By Theorem 3.2, such a $P_{F'}$ can be computed in $2^{\tilde{O}(n^{1-\varepsilon}\log(1/\alpha))}$ time, and has $T := 2^{\tilde{O}(n^{1-\varepsilon}\log(1/\alpha))}$ terms.

Over the boolean hypercube, the polynomial $P_{F'}$ can be naturally written as a sum of T conjunctions on $t = O(n^{2-2\varepsilon})$ variables each, where each conjunction has width $O(n^{1-\varepsilon}\log n \log(1/\alpha))$. Composing $P_{F'}$ with the subformulas $G_1, \ldots, G_t$, our $n^{3+\varepsilon}$-size formula F is close (in the sense of Theorem 3.2) to an expression of the generic form:

$$S(x) := \sum_{2^{\tilde{O}(n^{1-\varepsilon}\log(1/\alpha))}} c \cdot \bigwedge_{O(n^{1-\varepsilon}\log n \log(1/\alpha))} [\text{formula of } O(n^{1+3\varepsilon}) \text{ size}].$$

That is, $S(x)$ is a sum of $T = 2^{\tilde{O}(n^{1-\varepsilon}\log(1/\alpha))}$ terms, where each term is a conjunction of $t = O(n^{1-\varepsilon}\log n \log(1/\alpha))$ formulas, each formula has $O(n^{1+3\varepsilon})$ size, and each coefficient c has bit-length $O(t \cdot \log^2 s \cdot \log(1/\alpha)) \leq O(n)$. By Theorem 3.2, the sum $S(x)$ approximately computes $F(x)$, in that for all $a \in \{0,1\}^n$, $|F(a) - S(a)| < \alpha$.

Now, suppose we have an algorithm that can solve #SAT for formulas of size $n^{2+2\varepsilon+o(1)}$ in time $O(2^n/T^2) \leq 2^{n-n^{1-\varepsilon}\log(1/\alpha)\log^k n}$, for a sufficiently large constant $k > 0$. Then, as every summand of $S(x)$, i.e. each expression of the form

$$\bigwedge_{O(n^{1-\varepsilon})} [\text{formula of } O(n^{1+3\varepsilon}) \text{ size}],$$

has size $\tilde{O}(n^{2+2\varepsilon}) \leq n^{2+2\varepsilon+o(1)}$, we can compute the sum $\sum_a S(a)$ over all inputs, which has the form

$$\sum_{a\in\{0,1\}^n} S(a) = \sum_{2^{\tilde{O}(n^{1-\varepsilon}\log(1/\alpha))}} \left(\sum_a \bigwedge_{\tilde{O}(n^{1-\varepsilon}\log(1/\alpha))} [\text{formula of } O(n^{1+3\varepsilon}) \text{ size}] \right),$$

in time $T \cdot O(2^n/T^2) \leq O(2^n/T) \ll 2^n/n^{\omega(1)}$.

Finally, we observe that computing $\sum_a S(a)$ is enough to determine Gap-Formula-SAT, for an appropriate choice of the parameters α and β.

- Suppose F is UNSAT. Since $F(a) = 0$ for all a, we have $\sum_a S(a) \leq \alpha 2^n$, because $|S(a) - F(a)| \leq \alpha$.
- Suppose F has at least $\beta \cdot 2^n$ SAT assignments. Then, every SAT assignment to F adds at least $1 - \alpha$ to the sum, while each UNSAT assignment subtracts no more than α from the sum, in the worst case. So in this case,

$$\sum_a S(a) \geq (1-\alpha)\cdot\beta 2^n - \alpha\cdot(1-\beta)2^n = [(1-\alpha)\beta - \alpha(1-\beta)]\cdot 2^n.$$

Provided that $(1-\alpha)\beta - \alpha(1-\beta) > \alpha$, we can distinguish between the two cases using the value $\sum_a S(a)$. Manipulating the inequality, we find:

$$(1-\alpha)\beta - \alpha(1-\beta) > \alpha \;\Leftrightarrow\; \beta - \alpha > \alpha \;\Leftrightarrow\; \beta > 2\alpha \;\Leftrightarrow\; \beta/2 > \alpha.$$

Thus for any gap parameter $\beta \in (0,1)$, the above inequality holds for $\alpha := \beta/3$. (We could even let $\beta := 1/2^{n^{o(1)}}$, for example.)

This concludes the proof of Lemma 3.1.

A Generic Tradeoff.

We remark that the above approach is easily generalizable. Letting $t \ll n$ be a parameter, every formula of size $n^{3+\varepsilon}$ can be converted into a sum of the form

$$S(x) := \sum_{2^{\tilde{O}(t)}} \bigwedge_{\tilde{O}(t)} [\text{formula of } O(n^{3+\varepsilon}/t^2) \text{ size}].$$

Note that such a transformation is only meaningful for $t \ll n$: otherwise, the running time of converting the formula into $S(x)$ already exceeds 2^n. The above argument shows that if we can solve #SAT on formulas of $O(n^{3+\varepsilon}/t)$ size (in fact, a conjunction of $O(t)$ formulas of $n^{3+\varepsilon}/t^2$ size), in time $2^{n-\Omega(t/(\log n)^2)}$, then we obtain a nontrivial Gap-Formula-SAT algorithm.

Discussion: What Can Current Formula-SAT Algorithms Achieve?

Before moving on, let us briefly comment on how known #SAT algorithms based on shrinkage behave, and speculate on whether they can be generalized to achieve the kind of running time / formula size tradeoff that we seek.

Roughly speaking, all state-of-the-art algorithms work in the following way. First, they solve #SAT on all $O(n/\log^2 n)$-size formulas by brute force, building a $2^{o(n)}$-size look-up table for them. Next, they apply a concentration version of a shrinkage lemma to show that, by setting all possible assignments to some $n-k$ variables, a formula of size s shrinks to size approximately $k^2 \cdot s/n^2$, on all but $2^{n-\Omega(k)}$ assignments. When $k^2 \cdot s/n^2 < O(n/\log^2 n)$, the algorithm can consult the lookup table in the shrunken case, which means we need

$$s < \frac{n^3}{k^2 \log^2 n}$$

to use the lookup table efficiently, and obtain $2^{n-\Omega(k)}$ time. In summary, one might expect to obtain $2^{n-\Omega(k)}$ time for solving #SAT on formulas of size $n^3/(k^2 \log^2 n)$, for the relevant values of k.

However, for our desired running time in the hypothesis of Theorem 1.1, we require $k > n^{1-\varepsilon}$. In that case, the algorithm will only work for size

$$s < \frac{n^3}{n^{2-2\varepsilon} \log^2 n} = \frac{n^{1+2\varepsilon}}{\log^2 n}.$$

In principle, this is not necessarily a major issue: recall that we only need to solve #SAT on formulas which are *conjunctions* of $n^{1-\varepsilon}$ subformulas of size $n^{1+2\varepsilon}$. It appears plausible to us that a form of "simultaneous shrinkage", where we consider the shrinkage of $n^{1-\varepsilon}$ subformulas of size $n^{1+2\varepsilon}$ simultaneously under a common random restriction (analogous to "multi-switching" lemmas that work on a set of formulas [24, 23]), could be applied to obtain the necessary #SAT algorithm.

3.2 Formula Size Lower Bounds from Gap-Formula-SAT

Next, we show that the Gap-Formula-SAT algorithm derived in the previous subsection would be sufficient to establish the desired lower bound. Recall in the previous section, we showed that, under the #SAT algorithm hypothesis of Theorem 1.1, it follows that for every constant $\beta \in (0,1)$, we can solve Gap-Formula-SAT on formulas of size $n^{3+\varepsilon}$ in time $2^{n-\Omega(n^{1-\varepsilon})}$, in that we can distinguish UNSAT formulas from those satisfied by at least a $(1-\beta)$-fraction of possible assignments.

Chen and Williams [13] show there is a universal constant $\beta^\star > 0$ such that, for any class of circuits $\mathcal{C}$ that is closed under taking negations of the output, and variable projections (negating inputs and/or replacing inputs by constants), if there is a Gap-SAT algorithm running in $2^n/n^{\omega(1)}$ time for polynomial-size circuits C which are the AND of three circuits from $\mathcal{C}$, distinguishing UNSAT circuits from those with at least $\beta^\star 2^n$ satisfying assignments, then NEXP does not have polynomial-size circuits from $\mathcal{C}$.

Here, we will explain how to modify their argument, by combining other ideas from [56, 6, 13], to show that a faster Gap-Formula-SAT algorithm for formulas of size $O(n^{3+\varepsilon})$ actually implies that there is a function in $\mathsf{E}^{\mathsf{NP}} = \mathsf{TIME}[2^{O(n)}]^{\mathsf{SAT}}$ which does not have formulas of size $n^{3+\varepsilon}$. The main difficulty is that we have to be very careful to "respect" the formula size in the argument, so that it does not ever increase by more than a constant factor. We achieve this by (yet) another level of indirection, using the hypothesis E^{NP} has formulas of size $n^{3+\varepsilon}$ in multiple ways.

Indeed, we start by assuming that E^{NP} *does* have formulas of size $n^{3+\varepsilon}$. We will eventually show that every *unary* language $L \in \mathsf{NTIME}[2^n]$ (languages over the input alphabet $\{1\}$) can then be solved in nondeterministic time $o(2^n)$, contradicting the Nondeterministic Time Hierarchy Theorem (which holds for unary languages as well).

Reducing L to Gap-SAT.

In the following, let $k > 0$ be a sufficiently large universal constant. First, we show (drawing from prior work) that a non-trivial Gap-Circuit-Sat algorithm for poly(n)-size circuits would imply our desired lower bound. Next, we will show that a Gap-Formula-Sat algorithm suffices for the lower bound, even if the algorithm only works on $O(n^{3+\varepsilon})$-size formulas.

▶ **Lemma 3.4** ([56, 6, 13]). *If Gap-Circuit-Sat on circuits with $n + k\log n$ inputs and size $O(n^k \cdot n^{3+\varepsilon})$ can be solved in $2^n/n^{\omega(1)}$ time, then E^{NP} does not have DeMorgan formulas of size $O(n^{3+\varepsilon})$. (In fact,* **circuit** *lower bounds would follow.)*

Proof. Let $L \in \mathsf{NTIME}[2^n]$ be an arbitrary unary language. Our goal is to show that a nontrivial Gap-Circuit-Sat algorithm, combined with small circuits for E^{NP}, would allow us to nondeterministically decide L in $o(2^n)$ time, contradicting the Nondeterministic Time Hierarchy theorem [4].

Following prior work [56, 6, 13], we can choose a *succinct PCP* verifier V for L, with proofs of length $2^n \cdot n^k$, randomness $n + k\log n$, and for each choice of randomness, the verifier's $q = O(1)$ queries can be computed by a fixed n^k-size circuit C_n which takes in the $n + k\log n$ bits of randomness as input, and outputs q strings of length $n + k\log n$ corresponding to the indices of the proof bits for each query, along with sign bits (whether the outcome of the query should be negated, or not). The verifier accepts on a given random string if and only if the OR of the q queries is true, and the n^k-size circuit C_n implementing the verifier can be constructed in poly(n^k) time.

If E^{NP} has formulas of size $n^{3+\varepsilon}$, then for every n, if $1^n \in L$, there exists a family of formulas F_n of size $n^{3+\varepsilon}$ on $n + k\log n$ inputs such that the $2^n \cdot n^k$-bit truth table of F_n encodes a PCP π that the PCP verifier V accepts with probability 1. Indeed, there exists a $2^{O(n)}$-time algorithm with a SAT oracle (a.k.a. an E^{NP} algorithm) that on input $(1^n, i)$ outputs the i-th bit of such a proof π: this follows by a standard reduction from search-to-decision [56], augmented with exhaustive search over all random strings of length $n + k\log n$ to verify that the probability that V accepts π is 1.

We can simulate L on the input 1^n as follows. First, nondeterministically guess a formula F_n of size $n^{3+\varepsilon}$ that is intended to encode a PCP π_n for the PCP verifier V on 1^n. Next, we construct a n^k-size circuit C_n with $n + k \log n$ input bits encoding the verifier's query computation. Supposing the verifier makes $q = O(1)$ queries, we compose C_n with q copies of the formula F_n, and take the OR of the q copies of F_n. More precisely, letting the q output strings of C_n be $C_n^{(1)}(r), \ldots, C_n^{(q)}(r)$ be subcircuits of C_n such that $C_n^{(i)}(r)$ prints the $(n + k \log n)$-bit index of the i-th query on randomness r, and letting $S_n^{(i)}(r)$ be the sign bit for the i-th query, we construct the circuit

$$C_{\text{check}}(r) = \bigvee_{i=1}^{q} (F_n(C_n^{(i)}(r)) \oplus S_n^{(i)}(r)). \tag{1}$$

This circuit C_{check} has size at most $O(n^k \cdot n^{3+\varepsilon})$, and $C_{\text{check}}(r) = 1$ if and only if the verifier V on randomness r accepts with proof oracle F_n.

Since V is a PCP, we have two properties:

- If $1^n \in L$, then there is a choice of F_n of $n^{3+\varepsilon}$ size such that C_{check} is true on all inputs r. (Note such F_n exists, only assuming E^{NP} has $n^{3+\varepsilon}$ size formulas.)
- If $1^n \notin L$, then for every choice of F_n, C_{check} is false on at least a $1 - \beta$ fraction of its possible inputs r, where $\beta \in (0, 1)$ is determined by the soundness probability of the PCP protocol.

To line this up with our Gap-SAT formulation, we work with the negation of C_{check}: either $\neg C_{\text{check}}$ is unsatisfiable, or $\neg C_{\text{check}}$ is satisfiable on at least a $(1 - \beta)$-fraction of its possible assignments.

So now we are faced with the following task: given a circuit C_{check} with $n + k \log n$ inputs and $O(n^k \cdot n^{3+\varepsilon})$ size, and we wish to distinguish whether C_{check} is unsatisfiable on all possible inputs, or is satisfiable on at least a β-fraction. So far, everything is analogous to what one finds in earlier work showing how improving exhaustive search implies circuit lower bounds [56, 6]. If we can solve this task in $o(2^n)$ time, we will contradict the Nondeterministic Time Hierarchy Theorem. ◀

From Gap-SAT to Gap-Formula-SAT via PCP of Proximity and ECCs.

Our goal now is to show that, if E^{NP} has $n^{3+\varepsilon}$-size DeMorgan formulas, and Gap-Formula-Sat can be solved in $2^n/n^{\omega(1)}$ time on $O(n^{3+\varepsilon})$-size formulas for arbitrary constant-sized gaps, then we actually obtain the Gap-Circuit-Sat algorithm needed in the previous part.

We will apply ideas similar to that of the proof of Theorem 5 of Chen and Williams [13], which shows how to prove super-polynomial $\mathcal{C}$-circuit lower bounds for NEXP from a Gap-SAT algorithm for the conjunction (AND) of three $\mathcal{C}$-circuits that distinguishes the case where C is UNSAT from the case where C has a non-zero constant fraction of SAT assignments. (The following exposition is not entirely self-contained, but it should be quite friendly to a reader who has [13] open at their proof of Theorem 5.)

We work with a polynomial-time linear constant-rate error-correcting code $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^{cn}$ with a polynomial-time decoder $\mathsf{Dec} : \{0,1\}^{cn} \to \{0,1\}^n$, which can decode up to some relative distance $\delta_{ECC} > 0$ [43]. We use the ECC to modify the circuit C_{check} we want to solve Gap-SAT for, so that if C_{check} has a satisfying assignment, then the resulting circuit D has an assignment that is δ_{ECC}-far from any of its satisfying assignments. More precisely, as in [13], we define a circuit

$$D(y) := \neg C_{\text{check}}(\mathsf{Dec}(y)),$$

where Dec is the decoder for the aforementioned error-correcting code. Since Dec is a polynomial-time function, it has a circuit of size at most $O(n^k)$ for large enough k, and the circuit D can be implemented in size at most $O(n^{2k})$.

▷ Claim 3.5. Let $x \in \{0,1\}^n$ and let $y = \mathsf{Enc}(x)$. Then:

- If x is a satisfying assignment of C_{check}, then y is δ_{ECC}-far from any satisfying assignment of D. That is, one must flip more than a δ_{ECC}-fraction of bits of y in order to obtain a satisfying assignment to D.
- If x is a non-satisfying assignment of C_{check}, then y satisfies D.

Proof. Let x be such that $C_{\text{check}}(x) = 1$, and let $y = \mathsf{Enc}(x)$. For any string z that has relative distance at most δ_{ECC} from y, we have $\mathsf{Dec(z)} = x$ by the definition of an ECC with decoding distance δ_{ECC}. Therefore we also have $D(z) = \neg C_{\text{check}}(\mathsf{Dec(z)}) = \neg C_{\text{check}}(x) = 0$. It follows that y must be δ_{ECC}-far from any z such that $D(z) = 1$.

For the second item, observe that $D(y) = D(\mathsf{Enc}(x)) = \neg C_{\text{check}}(\mathsf{Dec}(\mathsf{Enc}(x))) = \neg C_{\text{check}}(x)$. So if x does not satisfy C_{check}, then y satisfies D. ◀

As in [13], we employ a *PCP of Proximity* (PCPP) for the Circuit-Eval problem [5]: given a circuit D on n inputs and $y \in \{0,1\}^n$, the Circuit-Eval problem asks whether $D(y) = 1$. The following lemma describes the important properties of this PCPP.

▶ **Lemma 3.6** (3-query PCP of Proximity for Circuit Evaluation with perfect completeness [13, Lemma 24])**.** *For any proximity parameter $\delta_{PCP} \in (0,1)$, there exists $s \in (0,1)$ such that there is a PCP of proximity system for Circuit-Eval with proximity δ_{PCP}, soundness s, randomness $r \leq O(\log n)$, and the following properties:*

- *Given the r random coins, a polynomial-time verifier V computes the OR on three queried bits (with some bits possibly negated), accepting if and only if the OR is true.*
- *Given a pair (D, y) such that D is a circuit, y is an input of length n, and $D(y) = 1$, a proof π can be constructed in $\text{poly}(|D| + |y|)$ time that makes the verifier V accept (D, y, π) with probability 1.*
- *Given (D, y), if y is δ_{PCP}-far from all satisfying assignments to D, then for all proofs π, the verifier V accepts (D, y, π) with probability at most s.*

In other words, the PCPP consists of an efficient verifier which is given a circuit D and input y, queries only three bits of the proof π, and tries to determine if $D(y) = 1$. If $D(y) = 1$ is true, then there is a proof π of this fact that always convinces the verifier, and can be efficiently constructed. If $D(y) = 0$ is false, and y is in fact *far in Hamming distance* from every SAT assignment to D, then the verifier will catch that $D(y) \neq 1$ with probability at least $1 - s$, regardless of the proof π.

We apply this PCPP to the circuit D defined above, with input $y = \mathsf{Enc}(x)$, and show how we can combine it with the algorithm of Lemma 3.1 to decide Gap-SAT for C. The PCPP has randomness $r \leq O(\log n)$, proximity parameter $\delta_{PCP} > 0$, and soundness $s < 1$, for fixed constants δ_{PCP} and s. Letting V be the verifier for the PCPP, as we vary over the $m := 2^r = \text{poly}(n)$ possible random strings, we obtain a set of 3-OR constraints on a given proof (y, π). Let $H_1, \ldots, H_m$ be the corresponding constraints. From the properties of the PCPP, we find:

- If $C_{\text{check}}(x) = 0$, then $D(y) = 1$, and there is a proof π such that all H_i are satisfied by the pair (y, π).
- If $C_{\text{check}}(x) = 1$, then y is δ_{ECC}-far from the set of all satisfying assignments to D. By setting $\delta_{PCP} < \delta_{ECC}$, at most an s fraction of the constraints $\{H_i\}$ are ever satisfied by (y, π).

In the case where C_{check} is unsatisfiable, there is a proof $\pi(x)$ for every $y = \mathsf{Enc}(x)$, such that the verifier V accepts $(D, y, \pi(x))$ with probability 1. For our setting, it is crucial that the proof function $\pi(x)$ can be computed in E^{NP} (so that it will also have $n^{3+\varepsilon}$-size DeMorgan formulas, by assumption).

To verify this, we go back to the original argument (before we began to invoke Chen and Williams [13]). Given an input (n, x, i) of length $n + O(\log n)$, where n is encoded in $O(\log n)$ bits, x is $n + O(\log n)$ bits, and i is $O(\log n)$ bits, here is an E^{NP} procedure (an algorithm running in $2^{O(n)}$ time with a SAT oracle) for constructing a proof $\pi(x)$ for the relevant circuit C_{check} defined in (1):

> First, use a SAT oracle to construct the lexicographically first formula F_n of size $n^{3+\varepsilon}$ which is a witness for our original PCP verifier for L (if $1^n \in L$, then F_n exists). Then, build the circuit C_{check} using F_n, evaluate $y = \mathsf{Enc}(x)$, and construct the proof $\pi(x)$ for (C_{check}, x) in polynomial time (which can be done by the construction of the PCP of Proximity). Finally, output the i-th bit of the proof $\pi(x)$.

Since this is an E^{NP} procedure with input of length $O(n)$, there is a formula of size $O(n^{3+\varepsilon})$ that, given (x, i), outputs the i-th bit of $\pi(x)$.

Following the notation of [13], we let T_i be the $O(n^{3+\varepsilon})$-size formula printing the i-th bit of $\pi(x)$, and let $T(x)$ be the concatenation of all $T_i(x)$. We let $G_i(x)$ be the constraint H_i (an OR of 3 literals) evaluated on the pair $(\mathsf{Enc}(x), T(x))$, and recall that there are only $m = \text{poly}(n)$ constraints $H_1, \ldots, H_m$. As our error-correcting code is linear, each bit of $\mathsf{Enc}(x)$ is a parity over some bits of x. Therefore, each G_i is an OR of 3 formulas, some of which are parities over a subset of the bits of x, and some of which are copies of $T_j(x)$ for some j. Observe (crucially) that, since PARITY has $O(n^2)$-size DeMorgan formulas, every formula G_i in fact has size only $O(n^{3+\varepsilon})$.

Finally, we compare the two cases when C_{check} is UNSAT, and when C_{check} has many satisfying assignments, and show that a non-trivial Gap-Formula-SAT algorithm for $O(n^{3+\varepsilon})$ size is sufficient for distinguishing these cases.

When $1^n \in L$ and we've guessed the "good" formula F_n of size $O(n^{3+\varepsilon})$, the constructed circuit C_{check} is UNSAT. In that case, as we argued above, there exists a polynomial-sized set of formulas $\{T_i\}$ that we can guess, each of size $O(n^{3+\varepsilon})$, so that the verifier V always outputs 1, over all choices of randomness. Therefore for all $i \in [m]$, every constraint formula $G_i(x)$ will be true, over all assignments x. It follows that, if we run a Gap-Formula-SAT algorithm on $\neg G_i$ for all $i = 1, \ldots, m$, it will always return an "UNSAT" answer.

When $1^n \notin L$, then no matter what formula F_n we guess to construct C_{check} with, the circuit C_{check} will be satisfied by at least a $(1-\beta)$-fraction of its possible assignments. Then, for all possible choices of the set of formulas $\{T_i\}$, over all assignments x, and all i, at most a $\beta + s$ fraction of pairs (x, i) are such that $G_i(x)$ is true. By averaging, there must be some $i^\star$ such that $G_{i^\star}(x)$ has at most a $\beta + s$ fraction of SAT assignments. Let $\beta > 0$ be sufficiently small, so that $(1 - \beta - s) \geq (1-s)/2$. Then, given a Gap-Formula-SAT algorithm that outputs "MANY-SAT" when at least a $(1-s)/2 > 0$ fraction of assignments are satisfying, this algorithm will return "MANY-SAT" on at least one $\neg G_i$.

Therefore, our Gap-Formula-SAT algorithm (which works for any constant fraction of SAT assignments) lets us determine whether or not 1^n is in L. After poly(n) bits of guessing, our nondeterministic algorithm for deciding L calls the Gap-Formula-SAT algorithm for poly(n) times, on formulas of size $O(n^{3+\varepsilon})$. Therefore the entire nondeterministic procedure runs in only $o(2^n)$ time, assuming the Gap-Formula-SAT algorithm runs in $2^n/n^{\omega(1)}$ time. Thus we can decide L in $\mathsf{NTIME}[o(2^n)]$, contradicting the Nondeterministic Time Hierarchy Theorem. This completes the proof.

Finally, observe that, even if our Gap-Formula-SAT algorithm was zero-error randomized, this would still suffice for constructing a nondeterministic algorithm for L.

Let us summarize this subsection with the tight connection between algorithms and lower bounds that we have proved:

▶ **Theorem 3.7.** *Let $\varepsilon > 0$. Suppose for every $\beta > 0$, there is a* Gap-Formula-SAT *algorithm for $O(n^{3+\varepsilon})$-size formulas that runs in $2^n/n^{\omega(1)}$ time, and distinguishes unsatisfiable formulas from those with at least a β-fraction of satisfying assignments. Then E^{NP} does not have $O(n^{3+\varepsilon})$-size formulas.*

3.3 Generalization to Other Circuit Classes

It is easy to see that the proof of Theorem 3.7 can be generalized to other circuit classes, beyond just DeMorgan formulas. The properties we used of DeMorgan formulas are that they are:

- closed under negation of their output,
- closed under variable projections (replacing variables with constants or their negations),
- closed under the OR of three copies, and
- PARITY has subcubic-size DeMorgan formulas—below the size bound of our DeMorgan formulas.

We can generalize the connection from Gap-Sat algorithms to lower bounds, as follows. Recall that Formula$[n^{2-\varepsilon}] \circ \mathcal{G}$ denotes the functions computable by DeMorgan formulas of size $n^{2-\varepsilon}$ with functions from the class $\mathcal{G}$ at their leaves [28].

Reminder of Theorem 1.2. *For every $\varepsilon > 0$ and every class of functions $\mathcal{G}$ such that:*
- *$\mathcal{G}$ is closed under variable projections,*
- *Formula$[n^{2-\varepsilon}] \circ \mathcal{G}$ contains PARITY, and*
- *the acceptance probability of a conjunction of $O(n)$ functions from $\mathcal{G}$, over n variables, can be additively approximated to within $1/2^{n^{1-\varepsilon/3}}$ in $2^{n-\Omega(n^{1-\varepsilon/3})}$ deterministic time,*

E^{NP} does not have Formula$[n^{2-\varepsilon}] \circ \mathcal{G}$ circuits.

We confine ourselves to a sketch of the proof of Theorem 1.2, as it closely mimics the proofs of previous subsections. First, in the proof of Lemma 3.1, we reduced the problem of computing Gap-SAT for a formula of size $n^{3+\varepsilon}$, to the problem of computing #SAT on a formula of size $n^{2-2\varepsilon}$. In particular, the reduction produced $2^{\tilde{O}(n^{1-\varepsilon}\log n)}$ instances of #SAT on formulas which are conjunctions of $\tilde{O}(n^{1-\varepsilon})$ formulas of $O(n^{1+3\varepsilon})$ size. This was accomplished by first applying Theorem 3.3 to decompose the $n^{3+\varepsilon}$-size formula into an $O(n^{2-2\varepsilon})$-size formula of $O(n^{1+3\varepsilon})$-size formulas, then applying Theorem 3.2 to reduce the $O(n^{2-2\varepsilon})$-size formula to a linear sum of $2^{\tilde{O}(n^{1-\varepsilon}\log n)}$ conjunctions on $\tilde{O}(n^{1-\varepsilon})$ variables.

Let C be a circuit from Formula$[n^{2-2\varepsilon}] \circ \mathcal{G}$. (For simplicity of exposition, we consider size $n^{2-2\varepsilon}$ instead of $n^{2-\varepsilon}$, to more closely model the situation in Lemma 3.1.) Performing exactly the same steps as above on the $n^{2-2\varepsilon}$-sized formula part of C, we can reduce C to a sum of $2^{\tilde{O}(n^{1-\varepsilon}\log n)}$ conjunctions of $\tilde{O}(n^{1-\varepsilon})$ functions from $\mathcal{G}$. By the same analysis as in Lemma 3.1, if we have a deterministic $2^{n-\Omega(n^{1-2\varepsilon/3})}$-time algorithm for approximating the acceptance probability of a conjunction of functions from $\mathcal{G}$ to within a gap of $1/2^{n^{1-2\varepsilon/3}}$, then we can determine Gap-Sat for the circuit C in time

$$2^{\tilde{O}(n^{1-\varepsilon}\log n)} \cdot 2^{n-\Omega(n^{1-2\varepsilon/3})} \leq 2^n/n^{\omega(1)}.$$

Next, assuming that $\mathcal{G}$ satisfies the hypotheses of Theorem 1.2, it follows that the circuit class Formula$[n^{2-\varepsilon}] \circ \mathcal{G}$ is closed under negation of the output (because DeMorgan formulas are), closed under the OR of three copies (because DeMorgan formulas are), closed under variable projections (by assumption on $\mathcal{G}$, and contains PARITY (by assumption). Therefore this class has all the closure properties necessary to apply the proof of Theorem 3.7 directly. Applying our algorithm for GAP-SAT on FORMULA$[n^{2-\varepsilon}]\circ\mathcal{G}$ circuits (obtained in the previous paragraph), we can conclude that E^{NP} does not have FORMULA$[n^{2-\varepsilon}]\circ\mathcal{G}$ circuits.

Here is one example of a canonical class that has high communication complexity, yet we can still prove circuit lower bounds for it.

Reminder of Corollary 1.3. *For all $\varepsilon > 0$, E^{NP} does not have* FORMULA$[n^{2-\varepsilon}]\circ\mathcal{G}$ *circuits, where $\mathcal{G}$ is the class of $n^{\varepsilon/3}$-degree polynomials over $\mathbb{F}_2$.*

Proof. Observe that FORMULA$[n^{2-\varepsilon}]\circ\mathcal{G}$ contains PARITY (a degree-one $\mathbb{F}_2$-polynomial), and that $\mathcal{G}$ is closed under variable projections. It remains to show that the algorithmic hypothesis of Theorem 1.2 is satisfied: namely, that we can estimate the acceptance probability of conjunctions of functions from $\mathcal{G}$ in $2^{n-\Omega(n^{1-\varepsilon/3})}$ time.

Building on the randomized algorithm of Lokshtanov, Paturi, Tamaki, Williams, and Yu [33], Chan and Williams [10] show that the #SAT problem for systems of m degree-d $\mathbb{F}_2$-polynomials in n variables can be solved in deterministic $2^{n-n/O(d)}\cdot\mathrm{poly}(m)$ time. Their algorithm works for all $d < o(\sqrt{n})$. Therefore we can set $d = n^{\varepsilon/3}$ and get a #SAT algorithm running in $2^{n-\Omega(n^{1-\varepsilon/3})}$ time, satisfying the algorithmic hypothesis of Theorem 1.2. ◀

Let us remark that we could have also applied known pseudorandom generators for degree-d $\mathbb{F}_2$-polynomials to obtain a good GAP-SAT algorithm, but we would have obtained a weaker degree bound. Namely, the best-known seed length for a PRG fooling degree-d $\mathbb{F}_2$-polynomials is $\Omega(2^d)$ [50], so enumerating over all seeds would cost at least $2^{\Omega(2^d)}$ time.

4 Uniform Depth Lower Bounds for SAT

4.1 Overview of the methodology

We now give an overview of how we implement "indirect diagonalization" to show that SAT does not have uniform NAND formulas of depth less than $c\log n$, for a value of $c > 3$ that we will specify later. Recall (Section 2) that we define NDepth $[c\log n]$ to be the class of languages that have uniform polylog-time NAND formulas of depth $(c+o(1))\log n$. A central component of our proof technique is a construction that allows us to simulate NDepth $[c\log n]$ very efficiently, using alternating machines with a low number of alternations. Following [54, 9], we call it a "Speedup Rule".

▶ **Lemma 4.1** (Speedup Rule for NAND Formulas)**.** *For every $d > 0$ and $0 \le x \le d$, we have*

$$\mathsf{NDepth}\,[d\log n] \subseteq (\exists n^{x/2})(\forall n)\mathsf{NDepth}\,[(d-x)\log n] \quad \text{and}$$
$$\mathsf{NDepth}\,[d\log n] \subseteq (\forall n^{x/2})(\exists n)\mathsf{NDepth}\,[(d-x)\log n]$$

The preliminaries (Section 2) cover the above complexity class notation in detail, but let us briefly remind the reader what it means. An inclusion like

$$\mathsf{NDepth}\,[d\log n] \subseteq (\exists n^{x/2})(\forall n)\mathsf{NDepth}\,[(d-x)\log n]$$

roughly means that we can evaluate a NAND formula of depth $d\log n$ on an input X of length n by first existentially guessing $n^{x/2+o(1)}$ bits Y, then universally choosing $n^{1+o(1)}$ bits Z, then evaluating a NAND formula of depth $(d-x)\log n$ on the input (X,Y,Z) (with some low-overhead computation). That is, NAND formulas of depth $d \log n$ can informally be simulated by "Σ_2-type" NAND formulas of depth $(d - x) \log n$, with $n^{x/2+o(1)}$ auxiliary bits of input.

To prove that SAT does not have NAND formulas of depth less than $c \log n$, we start by assuming that $\mathsf{SAT} \in \mathsf{NDepth}\,[c \log n]$, and aim to reach a contradiction. As SAT is NP-complete under polylog-time computable first-order projections [20, 48, 27], this inclusion extends to every language in $\mathsf{NTIME}[n]$ with negligible overhead. Since $\mathsf{NDepth}\,[c \log n]$ corresponds to NAND depth $(c+o(1)) \log n$ (and $\mathsf{NDepth}\,[c \log n]$ is closed under complement), we have:

$$\mathsf{SAT} \in \mathsf{NDepth}\,[c \log n] \Rightarrow \mathsf{NTIME}[n] \cup \mathsf{coNTIME}[n] \subseteq \mathsf{NDepth}\,[c \log n]\,.$$

Using this property, we can derive another important ingredient in the lower bound, the so-called Slowdown Rule, which allows us to reduce the number of quantifiers in an alternating computation, at the cost of an increased NAND depth:

▶ **Lemma 4.2** (Slowdown Rule). *Assume that SAT is in $\mathsf{NDepth}\,[c \log n]$ for some $c > 0$. Then, for every $a, b \geq 1$ and $d > 0$, we have:*

$$\ldots (Q\ n^a)(\neg Q\ n^b)\mathsf{NDepth}\,[d \log n] \subseteq \ldots (Q\ n^a)\mathsf{NDepth}\,[c \cdot \max(d/2, a, b) \log n]\,,$$

where $Q \in \{\exists, \forall\}$ and $\neg Q$ is the opposite quantifier.

On the other hand, we show that the assumption $\mathsf{NTIME}[n] \subseteq \mathsf{NDepth}\,[c \log n]$ allows us to lift the Nondeterministic Time Hierarchy Theorem to NAND formulas.

▶ **Theorem 4.3** (Conditional Depth Hierarchy Theorem). *For every $c > 0$, if $\mathsf{NTIME}[n] \subseteq \mathsf{NDepth}\,[c \log n]$, then for all $b > a \geq 1$,*

$$\mathsf{NDepth}\,[a \log n] \subsetneq \mathsf{NDepth}\,[b \log n]\,.$$

The core of our proof consists in repeatedly applying the rules of Lemmas 4.1 and 4.2 with suitable parameters, known as *alternation-trading* proofs, until we obtain a class containment that contradicts Theorem 4.3.

As a warm-up, let us apply the above rules to show that SAT does not have NAND formulas of depth less than $2.8284 \log n$, using a very simple alternation-trading proof. Assume $\mathsf{SAT} \in \mathsf{NAND\text{-}depth}[c \log n]$ for some $c > 0$. Then, we have

$$\begin{aligned} \mathsf{NDepth}\,[4 \log n] &\subseteq (\exists n)(\forall n)\mathsf{NDepth}\,[2 \log n] && \textit{(Speedup Rule with } x = 2\textit{)} \\ &\subseteq (\exists n)\mathsf{NDepth}\,[c \log n] && \textit{(Slowdown Rule)} \\ &\subseteq \mathsf{NDepth}\,\left[(c^2/2) \log n\right] && \textit{(Slowdown Rule)} \end{aligned}$$

For $c^2/2 < 4 \Leftrightarrow c < 2\sqrt{2} \simeq 2.8284\ldots$, this contradicts the Conditional Depth Hierarchy (Theorem 4.3). (Those familiar with the proof that SAT does not have algorithms running in $n^{\sqrt{2}-o(1)}$ time and $n^{o(1)}$ space [20] may be experiencing a feeling of *déjà vu*.)

The remainder of this section is organized as follows: Section 4.2 gives a proof of the above rules and of the hierarchy theorem, while Section 4.3 details the construction of alternation-trading proofs that extend this result to any $c < 4 \cos(\pi/7)$.

4.2 Speedups, Slowdowns and a Depth Hierarchy

Speedup and Slowdown

We first discuss the Speedup and Slowdown Rules.

▶ **Lemma 4.1** (Speedup Rule for NAND Formulas). *For every $d > 0$ and $0 \leq x \leq d$, we have*

$$\mathit{NDepth}\,[d \log n] \subseteq (\exists n^{x/2})(\forall n)\,\mathit{NDepth}\,[(d-x)\log n] \text{ and}$$
$$\mathit{NDepth}\,[d \log n] \subseteq (\forall n^{x/2})(\exists n)\,\mathit{NDepth}\,[(d-x)\log n]$$

Proof. We prove the statement for $\exists\forall$ computations, the other direction follows since $\mathsf{NDepth}\,[d \log n]$ is closed under complement (recall, we are allowed depth $(d + o(1))\log n$; it is easy to add an extra layer on the top of a NAND circuit which performs negation).

We view the input NAND formula as an alternating OR/AND formula ϕ, using the connection described in Section 2. The intuitive idea is to select, for each OR gate (starting from the output), one of its children which is intended to be true. Furthermore, for every choice of an "OR child" that we make, the size of the remaining formula that we have to evaluate drops by half, as we do not have to consider the subformula rooted at the other child. We use the non-deterministic bits from the existential quantifier to select these "OR children", for the OR gates among the top $x \log n$ levels of the formula. Then we use a sublinear number of bits from the universal quantifier to handle the remaining AND gates.

Let y denote the $n^{x/2+o(1)}$ bits received from the universal quantifier. Let us describe the procedure from the output gate (which is an OR). If the first bit of y is 0, the OR gate is replaced by its left child, otherwise it is replaced by its right child. In both cases, we reach an AND gate. We then apply this procedure recursively on the two OR gates below that AND gate, using the separately the two halves of the rest of y, until we reach depth $x/2 \log n$. The resulting formula ϕ' starts with $(x/2)\log n$ levels of AND gates, followed by smaller alternating OR/AND formulas ϕ_i of depth $(d - x + o(1))\log n$. Using the above observation, for a given x, there is a choice of y_x such that $\phi'(x) = 1$ if and only if $\phi(x) = 1$. The $\exists$ stage outputs the input x and the corresponding string y_x. Upon receiving (x, y), the $\forall$ stage universally guesses $\frac{x}{2}\log n$ nondeterministic bits z that represent an index i, and outputs x and $x \log n$ bits that encode a path from the root of ϕ to the root of ϕ_i. As ϕ' is the conjunction of the $n^{x/2}$ formulas ϕ_i, it evaluates to 1 on input x if and only if for every index $i \leq n^{x/2}$, $\phi_i(x) = 1$.

Finally, recall that the definition of alternating computation requires that there exists a *single* machine M' that runs the final stage of computation *for every* possible choice of nondeterministic bits: that is, the machine cannot depend on the nondeterministic bits. This is where our strong notion of uniformity comes in (from Definition 1.5), where the descriptor machine has random access to the input: giving M' access to (a few) bits of the input, allows us to adapt its behavior depending on the nondeterministic bits. If M is a machine that describes ϕ, we define the machine M', that on input a' of length $n' = n + x \log n$ for some n, sets $a' = ab$, where a has length n and b has length $x \log n$, and returns $M'(a', i, n') = M(a, bi, n)$. We use the random access to move $x \log n$ bits from the input to the position of the gate that we want to describe. ◀

Combining Lemma 4.1 and the simulation of $\mathsf{NDepth}\,[d \log n]$ by $\mathsf{TISP}\left[n^d, n^{o(1)}\right]$ machines (machines running in time n^d and space $n^{o(1)}$) gives us an efficient simulation of formulas by nondeterministic Turing machines that is faster than the deterministic simulation.

▶ **Corollary 4.4.** *For every $d \geq 1$, we have*

$$\mathit{NDepth}\,[d \log n] \subseteq \mathit{NTIME}[n^{d/2+o(1)}] \cap \mathit{coNTIME}[n^{d/2+o(1)}].$$

Proof. Applying Lemma 4.1 with $x = d$, we obtain

$$\mathsf{NDepth}\,[d \log n] \subseteq (\exists n^{d/2})(\forall n)\mathsf{NDepth}\,[0 \log n] \subseteq (\exists n^{d/2})(\forall n)\mathsf{TISP}\left[n^{o(1)}, n^{o(1)}\right].$$

A close inspection of the proof of Lemma 4.1 reveals that the $\forall$ quantifier here uses only $\frac{d}{2}\log n$ nondeterministic bits. Therefore, we can use exhaustive search over the $n^{d/2}$ strings of nondeterministic bits of length $\frac{d}{2}\log n$ to remove this quantifier. For every such string, we perform a $n^{o(1)}$-time computation, hence we get that $\mathsf{NDepth}\left[d\log n\right] \subseteq (\exists n^{d/2})\mathsf{TIME}[n^{d/2+o(1)}] = \mathsf{NTIME}[n^{d/2+o(1)}]$.

The inclusion in $\mathsf{coNTIME}[n^{d/2+o(1)}]$ follows from the closure of $\mathsf{NDepth}\left[d\log n\right]$ under complement. ◀

Next, we show that the assumption $\mathsf{SAT} \in \mathsf{NDepth}\left[c\log n\right]$ extends to every language in NP without significant overhead.

▶ **Lemma 4.5.** *If SAT $\in$ NDepth $[c\log n]$, then for every $d \geq 1$, NTIME$[n^d] \cup$ coNTIME$[n^d] \subseteq$ NDepth $[c\cdot d\log n]$.*

Later, this result will allow us to remove the quantifier from an alternating class, at the cost of an increased depth of the verifier.

A key idea for the proof of Lemma 4.5 is the fact that any language in $\mathsf{NTIME}[n]$ can be reduced to SAT via so-called *uniform poly-log time first-order projections*, where instances of size n can be mapped to formulas of size $n^{1+o(1)}$ [20, 48]. A first-order projection is a depth 0 circuit, i.e., each output bit is either a constant (0 or 1), an input x_i, or its negation $\neg x_i$. For an in-depth exposition of first-order projections, see for example [1, End of Sec. 3].

Proof of Lemma 4.5. We first prove that $\mathsf{SAT} \in \mathsf{NDepth}\left[c\log n\right]$ implies that $\mathsf{NTIME}[n] \subseteq \mathsf{NDepth}\left[c\log n\right]$.

Let $L \in \mathsf{NTIME}[n]$. As stated above, L is Karp-reducible to SAT via uniform poly-log time first-order projections, with instances of size n reduced to formulas of size $n^{1+o(1)}$. Now, notice that the composition of a NAND formula of depth d with a first-order projection is also a NAND formula of depth d: it only consists in relabeling leaves of the formula. Poly-log time uniformity is also preserved. Therefore, we can use $(c+o(1))\log n$-depth NAND formulas for SAT on $n^{1+o(1)}$ inputs to decide L by composing them with the first-order projection; the resulting formula has size $(c+o(1))\log(n^{1+o(1)}) = (c+o(1))\log n$, therefore $L \in \mathsf{NDepth}\left[c\log n\right]$.

We now use a padding argument to lift this result to $\mathsf{NTIME}[n^d]$. Let $d > 1$ and let $L \in \mathsf{NTIME}[n^d]$: the padded language $L' = \{x1^{|x|^d-|x|} \mid x \in L\}$ is in $\mathsf{NTIME}[n]$, and therefore $L' \in \mathsf{NDepth}\left[c\log n\right]$. We can then use the formula descriptor A for L' to build a machine B that describes formulas of depth $(c+o(1))\log\left(n^{d+o(1)}\right) = (c\cdot d+o(1))\log n$ for L as follows. On input (x, b, n), the machine B calls A on the input $(x1^{|x|^d-|x|}, b, n^d)$. If A outputs a gate type or a constant, B outputs the same information. If A outputs a literal x_i^*, B outputs the same x_i^* if $i \leq n$, and 1 otherwise. Passing x as input to the resulting formula is equivalent to passing $x1^{|x|^d-|x|}$ as input to the formula described by A. This construction shows that $L \in \mathsf{NDepth}\left[c\cdot d\log n\right]$. In summary, we have shown that

$$\forall d \geq 1, \mathsf{NTIME}[n^d] \subseteq \mathsf{NDepth}\left[c\cdot d\log n\right].$$

The result follows since $\mathsf{NDepth}\left[c\cdot d\log n\right]$ is closed under complement. ◀

We need to be careful when applying this result to alternating classes[6]: the $(i+1)$-th computation stage takes as input a string of length $m = n^{a_i}$, not n. Since the previous result

[6] See Definition 2.1 for the definition of alternating classes and of (a_i).

only holds for $\mathsf{NTIME}[n^d]$ when d is at least 1, we need to take into account the edge case where the running time of the $(i+1)$-th stage is sublinear in the size of its input. (This is the case below in the proof of Lemma 4.2.)

While the primary use of the Speedup Rule (Lemma 4.1) is to reduce the depth of the verifier at the cost of increasing the number of quantifiers, we show that we can combine it with the above lemma to efficiently remove quantifiers from an alternating NDepth computation.

▶ **Lemma 4.2** (Slowdown Rule). *Assume that SAT is in $\mathsf{NDepth}[c\log n]$ for some $c > 0$. Then, for every $a, b \geq 1$ and $d > 0$, we have:*

$$\ldots (Q\ n^a)(\neg Q\ n^b)\mathsf{NDepth}\,[d\log n] \subseteq \ldots (Q\ n^a)\mathsf{NDepth}\,[c \cdot \max(d/2, a, b)\log n]\,,$$

where $Q \in \{\exists, \forall\}$ and $\neg Q$ is the opposite quantifier.

Proof. Without loss of generality, assume that $Q = \forall$, and therefore $\neg Q = \exists$. In that case, for all $q, r > 0$, we have $(\exists n^q)\mathsf{NTIME}[n^r] \subseteq \mathsf{NTIME}[n^{\max(q,r)}]$. Applying Corollary 4.4 on the $\mathsf{NDepth}\,[d\log n]$ part, and applying the above observation, leads to the following:

$$\begin{aligned}\ldots(\forall n^a)(\exists n^b)\mathsf{NDepth}\,[d\log n] &\subseteq \ldots(\forall n^a)(\exists n^b)\mathsf{NTIME}[n^{d/2+o(1)}]\\ &\subseteq \ldots(\forall n^a)\mathsf{NTIME}[n^{\max(d/2,b)+o(1)}].\end{aligned}$$

Now, consider the $\mathsf{NTIME}[n^{\max(d/2,b)+o(1)}]$ language L that corresponds to the last stage of the alternating computation, and its corresponding nondeterministic machine M. This M takes as input a string of length $m = n^a$, and its runtime, as a function of m, is $m^{\max(d/2,b)/a+o(1)}$. We apply the Slowdown Rule (Lemma 4.2) to M. There are two possible cases:

- If $\max(d/2, b)/a < 1$, then we have

  $$L \in \mathsf{NTIME}[m^{\max(d/2,b)/a}] \subseteq \mathsf{NTIME}[m] \subseteq \mathsf{NDepth}[c\log m] \subseteq \mathsf{NDepth}\,[c \cdot a\log n]\,.$$

  Since $\max(d/2, b) < a$, we have $a = \max(d/2, a, b)$ and therefore $\mathsf{NDepth}\,[c \cdot a\log n] = \mathsf{NDepth}\,[c \cdot \max(d/2, a, b)\log n]$.
- Otherwise, $\max(d/2, b) \geq a$, and applying the Slowdown Rule yields

  $$L \in \mathsf{NTIME}[m^{\max(d/2,b)/a}] = \mathsf{NTIME}[n^{\max(d/2,b)}] \subseteq \mathsf{NDepth}\,[c \cdot \max(d/2, a, b)\log n]\,.$$

◀

Conditional Depth Hierarchy for NAND Formulas

We now prove the Conditional Depth hierarchy theorem, stated without proof in Section 4.1.

▶ **Theorem 4.3** (Conditional Depth Hierarchy Theorem). *For every $c > 0$, if $\mathsf{NTIME}[n] \subseteq \mathsf{NDepth}\,[c\log n]$, then for all $b > a \geq 1$,*

$$\mathsf{NDepth}\,[a\log n] \subsetneq \mathsf{NDepth}\,[b\log n]\,.$$

Proof. Assume that there exists c such that $\mathsf{NTIME}[n] \subseteq \mathsf{NDepth}\,[c\log n]$. In that case, the hypotheses of Lemma 4.5 are satisfied.

We prove the theorem by contradiction. Assuming there exists $1 \leq a < b$ such that $\mathsf{NDepth}\,[b\log n] \subseteq \mathsf{NDepth}\,[a\log n]$, we show this implies for *every* $d \geq a$ that $\mathsf{NDepth}\,[d\log n] \subseteq \mathsf{NDepth}\,[a\log n]$. Indeed, a padding argument shows that $\mathsf{NDepth}\,[r \cdot a\log n] \subseteq \mathsf{NDepth}\,[a\log n]$

implies $\mathsf{NDepth}\left[r^2 \cdot a \log n\right] \subseteq \mathsf{NDepth}\left[r \cdot a \log n\right] \subseteq \mathsf{NDepth}\left[a \log n\right]$ for any $r > 1$. Taking $r = b/a > 1$ and iterating this argument $t = \lceil \log_2 \log_r(d/a) \rceil$ times, we get

$$\mathsf{NDepth}\left[d \log n\right] \subseteq \mathsf{NDepth}\left[r^{2^t} \log n\right] \subseteq \mathsf{NDepth}\left[a \log n\right].$$

Now, assuming that there exists $c \geq 1/2$ such that $\mathsf{NTIME}[n] \subseteq \mathsf{NDepth}\left[c \log n\right]$, we have:

$$\begin{aligned}\mathsf{NTIME}[n^{2a}] &\subseteq \mathsf{NDepth}\left[2ca \log n\right] \\ &\subseteq \mathsf{NDepth}\left[a \log n\right] \\ &\subseteq \mathsf{NTIME}[n^{a+o(1)}].\end{aligned}$$

The first inclusion follows Lemma 4.5, whereas the second inclusion follows from the above argument, as $2c > 1$. The last inclusion uses the efficient simulation of NDepth by (nondeterministic) RAMs. The proof for $c < 1/2$ is similar, except that the second inclusion follows from $2c < 1$. We have proven $\mathsf{NTIME}[n^{2a}] = \mathsf{NTIME}[n^{a+o(1)}]$ for some $a \geq 1$, which contradicts the Nondeterministic Time Hierarchy Theorem. ◀

Note that this theorem relies heavily on the assumption that $\mathsf{SAT} \in \mathsf{NDepth}\left[c \log n\right]$, hence this theorem is *conditional*, but it is sufficient for our use case. It is an open problem to give an unconditional depth hierarchy theorem that separates $\mathsf{NDepth}\left[a \log n\right]$ from $\mathsf{NDepth}\left[b \log n\right]$ for any $a < b$.

4.3 The Structure of Good Proofs

In this section, we show how to construct alternation-trading proofs that reach a contradiction from the assumption $\mathsf{SAT} \in \mathsf{NDepth}\left[c \log n\right]$ for any $c < 4\cos(\pi/7)$. While these proofs might not be the shortest that yield a contradiction for a given c, they present a very simple inductive structure that makes them easier to analyze.

We introduce here the *proof annotation* notation of Williams [36], that succinctly describes an alternation-trading proof: let 1_x denote an application of the Speedup Rule with parameter x, and let 0 denote an application of the Slowdown Rule. The proof annotation of an alternation-trading proof is the string corresponding to the concatenation of the representations of each step. For example, the proof annotation corresponding to the short alternation-trading proof presented in Section 4.1 is $1_2 00$ (one application of the Speedup Rule with parameter $x = 2$, followed by two applications of the Slowdown Rule).

▶ **Observation 4.6.** *Consider an alternation-trading proof starting from*

$$\ldots (Q_k\ n^{a_k}) \mathit{NDepth}\left[d \log n\right]$$

and that never goes below k quantifiers and ends with the class

$$\ldots (Q_k\ n^{a_k}) \mathit{NDepth}\left[d' \log n\right].$$

Then we have $d' \geq c \cdot a_k$.

This is due to the presence of a_k, the exponent of the last quantifier, inside the max that defines the depth of the class obtained after an application of the Slowdown Rule (Lemma 4.2). In what follows, we aim to reach $d' = c \cdot a_k$, and want to apply this process inductively, i.e., the last line before $\ldots (Q_k\ n^{a_k}) \mathsf{NDepth}\left[d \log n\right]$ will be

$$\ldots (Q_{k+1}\ n^{a_{k+1}}) \mathsf{NDepth}\left[c \cdot a_{k+1} \log n\right],$$

which implies that $d = \frac{c^2}{2} a_{k+1}$. Not all values can be reduced to $c \cdot a_k$: a study of necessary and sufficient conditions will give us a lower and an upper bound on a_{k+1} in terms of c and a_k, from which we will derive an explicit construction of an optimal choice of values for $(a_k)_k$.

4.3.1 Reducing the Depth

The critical part is managing to reduce the depth of the verifier from $\left(\frac{c^2}{2} \cdot a_{k+1}\right) \log n$ to $ca_k \log n$, while preserving the last quantifier exponent to a_k. One way to do this is by applying a sequence of Speedup-Slowdown alternation. One application of Speedup then Slowdown proves the following:

$$\begin{aligned} \ldots (Q_k\ n^{a_k})\mathsf{NDepth}\left[d \log n\right] &\subseteq \ldots \left(Q_k\ n^{\max(a_k, x)}\right)(Q_{k+1}\ n)\mathsf{NDepth}\left[(d-2x)\log n\right] \text{ (Speedup with } x) \\ &\subseteq \ldots \left(Q_k\ n^{\max(a_k, x)}\right) \mathsf{NDepth}\left[c \cdot \max(a_k, x, (d-2x)/2, 1)\log n\right] \text{ (Slowdown)} \end{aligned}$$

Since we do not want to increase the exponent of the Q_k quantifier, we get the following constraint:

$$x \le a_k \tag{2}$$

Moreover, assuming that a_k is at least 1 allows us to reformulate the last class as

$$\ldots (Q_k\ n^{a_k})\,\mathsf{NDepth}\left[c \cdot \max(a_k, (d-2x)/2)\log n\right].$$

From here, we can get the following:

▶ **Lemma 4.7** ("Squiggle rule"). *Let a, d be positive real numbers such that $ac < d < \frac{2ca}{c-2}$. Then, the proof $(1_a0)^t$ for $t = \left\lceil \frac{d}{(4-c)a} \right\rceil + 1$ proves that*

$$\ldots (Q\ n^a)\mathit{NDepth}\left[d \log n\right] \subseteq \ldots (Q\ n^a)\mathit{NDepth}\left[ca \log n\right].$$

Proof. If $d/2 - a \le a$, or equivalently, $d \le 4a$, then we get the desired result using an application of the Speedup Rule with $x = d/2 - a$, followed by the Slowdown Rule.

We now assume that $d > 4a$, and show that we can get to $d' \le 4a$ in a finite number of steps. When $d \ge 4a$, applying the Speedup Rule with parameter $x = a$ followed by the Slowdown Rule transforms d is into $c \cdot (d/2 - a)$.

This is an improvement whenever d is such that:

$$d < \frac{2ca}{c-2} \tag{3}$$

Moreover, as long as $c < 4$, we have $\frac{2c}{c-2} > 4$, hence there is a gap between $4a$ and $\frac{2ca}{c-2}$ where we can get a non-trivial improvement.

The improvement is $d(1 - c/2) + ca$, which is at least $(4-c)a$, since we are in the case $d > 4a$. Therefore, as the improvement is bounded below by a constant independent of d, iterating this process starting from d reaches a value below $4a$ in at most $\left\lceil \frac{d}{(4-c)a} \right\rceil$ steps, after which one more step with parameter $x = d/2 - a$ yields the desired $ca \log n$ depth. ◀

4.3.2 Putting it all together

Lemma 4.7 shows that, in order to reach depth $ca_k \log n$ starting from depth $d \log n$, we need $d < \frac{2ca_k}{c-2}$. As we aim for an inductive structure with $d = \frac{c^2}{2} a_{k+1}$, we obtain:

$$a_{k+1} < \frac{4a_k}{c(c-2)} \tag{4}$$

Notice that $\frac{4}{c(c-2)} < 1$ whenever $c > 2\phi = 1 + \sqrt{5}$, where ϕ denotes the golden ratio.

Equation (4) captures most of the requirements for the parameters of an alternation-trading proof that yields a contradiction under the assumption $\mathsf{SAT} \in \mathsf{NDepth}\left[c \log n\right]$, which leads us to the following definition:

▶ **Definition 4.8.** *Let $c > 2\phi$. A sequence of real number $(a_k)_{k=0,\dots,t}$ is* well-behaved for c *if the following holds:*
1. *for every $k = 0, \dots, t, a_k \geq 1$,*
2. *$a_t = 1$,*
3. *for every $k = 0, \dots, t, ca_{k+1}/2 \geq a_k$, and*
4. *for every $k = 0, \dots, t-1, a_{k+1} < \frac{4a_k}{c(c-2)}$.*

Notice that constraint 4 above implies that $a_{k+1} < a_k$, since $\frac{4}{c(c-2)} < 1$ when $c > 2\phi$.

The following lemma gives a formal proof of the intuition that the conditions of Definition 4.8 are sufficient to obtain a contradiction using an alternation-trading proof.

▶ **Lemma 4.9.** *Let $c > 2\phi$, let $(a_k)_{k=0,\dots,t}$ be a sequence well-behaved for c, and let $d_0 = 2\left(1 + \sum_{k=0}^{t} a_k\right)$. Then there exists an alternation trading proof that shows*

$$\mathit{NDepth}\left[d_0 \log n\right] \subseteq \mathit{NDepth}\left[\frac{c^2}{2} a_0 \log n\right].$$

Proof. The proof starts by applying t speedups, with respective parameters $2a_1, 2a_2, \dots, 2a_t$. We obtain the following:

$$\begin{aligned}
\mathsf{NDepth}\left[d_0 \log n\right] &\subseteq (\exists n^{a_0})(\forall n)\mathsf{NDepth}\left[d_0 - 2_a 0 \log n\right] \\
&\subseteq (\exists n^{a_0})(\forall n^{a_1})(\exists n)\mathsf{NDepth}\left[d_0 - 2a_0 - 2a_1 \log n\right] \\
&\subseteq \vdots \\
&\subseteq (\exists n^{a_0}) \dots (Q_t n^{a_t})(Q_{t+1} n)\mathsf{NDepth}\left[d_0 - 2\sum_{k=0}^{t} a_k \log n\right] \\
&= (\exists n^{a_0}) \dots (Q_t n^{a_t})(Q_{t+1} n)\mathsf{NDepth}\left[2 \log n\right]
\end{aligned}$$

The last equality follows from the definition of d_0.

Then, applying the Slowdown Rule yields the class

$$(\exists n^{a_0}) \dots (Q_t n^{a_t})\mathsf{NDepth}\left[c \cdot a_t \log n\right].$$

We will now repeat t successive applications of the Slowdown Rule and Squiggle rule. Informally, it will give us the following inclusions:

$$\begin{aligned}
(\exists n^{a_0}) \dots (Q_t n^{a_t})\mathsf{NDepth}\left[c \cdot a_t \log n\right] &\subseteq (\exists n^{a_0}) \dots (Q_{t-1} n^{a_{t-1}})\mathsf{NDepth}\left[\frac{c^2}{2} \cdot a_t \log n\right] \quad \text{(Slowdown)} \\
&\subseteq (\exists n^{a_0}) \dots (Q_{t-1} n^{a_{t-1}})\mathsf{NDepth}\left[c \cdot a_{t-1} \log n\right] \quad \text{(Squiggle)} \\
&\vdots \\
&\subseteq (\exists n^{a_0})\mathsf{NDepth}\left[c \cdot a_0 \log n\right] \\
&\subseteq \mathsf{NDepth}\left[\frac{c^2}{2} a_0 \log n\right] \quad \text{(Slowdown)}
\end{aligned}$$

This can be shown formally by induction, using the fact that $(a_k)_k$ is well-behaved for c: during the i-th step ($i = 0, \dots, t-1$), we have the following properties:
1. before applying the Slowdown Rule, the depth of the verifier is $c \cdot a_{t-i} \log n$,
2. after applying the Slowdown Rule, the depth is $\frac{c^2}{2} a_{t-i} \log n$,
3. after applying the Squiggle rule, the depth is $c \cdot a_{t-i-1} \log n$.

At step $i = t$, we only apply the Slowdown Rule (we cannot apply the Squiggle Rule: there is no leading quantifier).

Property 1 is true for $i = 0$, and then holds by induction when Property 3 holds for $i - 1$. Property 2 holds whenever Property 1 holds, by definition of the Slowdown Rule (Lemma 4.2), and using the fact that $a_i \geq 1$ and $ca_i > a_{i-1}$. Finally, Property 3 hold when Property 2 holds: we can apply the Squiggle rule (Lemma 4.7) as a is well-behaved for c, which ensures that $i = 0, \dots, t-1, a_{i+1} < \frac{4a_i}{c(c-2)}$. ◀

Given $c < 4\cos(\pi/7)$, the last step is to construct a sequence of parameters that is well-behaved for c and such that $d_0 > \frac{c^2}{2}a_0$, which yields the desired contradiction. Using the insight gained in this section, we can construct such a sequence explicitly.

▶ **Lemma 4.10.** *Let $c \in (2\phi, 4\cos(\pi/7))$, and let $\tau = \frac{c(c-2)}{4}$. Then there exists an integer t such that, for $\varepsilon = 1/(t+1)$,*

- *the sequence $(a_k = r \cdot \tau^{t-k} - \varepsilon)_{k=0,\dots,t}$, where $r = 1 + \varepsilon$, is well-behaved for c,*
- *$d_0 = 2\left(1 + \sum_{k=0}^{t} a_k\right)$ is greater than $\frac{c^2}{2}a_0$.*

Proof. We first check that $(a_k)_k$ is well-behaved for c. First, as $r = 1 + \varepsilon$, we have $a_t = 1$, and since $\tau > 1$, we have $a_k \geq 1$ for every $k = 1, \dots, t$. Now, for every $k < t$, we have $a_k + \varepsilon = \tau \cdot (a_{k+1} + \varepsilon)$, which, combined with $\tau > 1$ implies that

$$a_{k+1} = a_k/\tau + \varepsilon(\tau^{-1} - 1) < \frac{a_k}{\tau} = \frac{4a_k}{c(c-2)}.$$

The constraint $ca_{k+1} \geq a_k$ can be rewritten as

$$a_{k+1}\left(\frac{c}{2} - \tau\right) \geq \varepsilon(\tau - 1).$$

As $\varepsilon = 1/(t+1)$ and τ is independent of t, the RHS of this inequality goes to 0 as t goes to infinity, and $(\frac{c}{2} - \tau) > 0$ when c is greater than 3 (which is the case here since $c > 2\phi > 3$). Therefore, there exists a t such that the inequality holds for every k, and the sequence $(a_k)_{k=1,\dots,t}$ is well-behaved for c.

We now show that there exists a t such that $d_0 > \frac{c^2}{2}a_0$. We have:

$$\begin{aligned}
d_0 &= 2\left(1 - (t+1)\varepsilon + \sum_{k=0}^{t} r\tau^k\right) \\
&= 2\left(1 - 1 + r\sum_{k=0}^{t} \tau^k\right) \\
&= 2r\frac{\tau^{t+1} - 1}{\tau - 1}
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
d_0 > \frac{c^2}{2}a_0 &\Leftrightarrow 2r\frac{\tau^{t+1} - 1}{\tau - 1} > \frac{c^2 a_0}{2} \\
&\Leftrightarrow r\frac{\tau\tau^t - 1}{\tau - 1} > \frac{c^2(r\tau^t - \varepsilon)}{4} \\
&\Leftrightarrow r\tau^t\left[\frac{c^2}{4} - \frac{\tau}{\tau - 1}\right] < \frac{-1}{\tau - 1} + \varepsilon\frac{c^2}{4} \\
&\Leftrightarrow \frac{c^2}{4}\tau - \frac{c^2}{4} - \tau < (r\tau^t)^{-1}\left[-1 + \frac{c^2(\tau - 1)}{4(t+1)}\right] \\
&\Leftrightarrow \frac{c^2}{4}\tau - \frac{c^2}{4} - \tau < \frac{-1}{(1+\varepsilon)\tau^t} + \frac{c^2(\tau - 1)}{4(t+1)(1+\varepsilon)\tau^t} \quad (*)
\end{aligned}$$

We can rewrite the LHS of the last inequality $(*)$ as follows:

$$\begin{aligned}\frac{c^2}{4}\tau - \frac{c^2}{4} - \tau &= \frac{c^3(c-2)}{16} - \frac{c^2}{4} - \frac{c(c-2)}{4} \\ &= \frac{1}{4}\left(\frac{c^4}{4} - \frac{c^3}{2} - 2c^2 + 2c\right) \\ &= \frac{c}{16}\left(c^3 - 2c^2 - 8c + 8\right)\end{aligned}$$

This allows us to rewrite the inequality $(*)$ as:

$$c^3 - 2c^2 - 8c + 8 < \frac{16}{c}\frac{-1}{(1+\varepsilon)\tau^t} + \frac{c^2(\tau-1)}{4(t+1)(1+\varepsilon)\tau^t} \tag{5}$$

The LHS of Inequality (5) does does not depend on t, while the RHS is negative and goes to 0 as t goes to infinity. Therefore, for every $\delta < 0$, there exists a t such that the RHS is greater than δ, and therefore Inequality (5) is satisfied as soon as we have:

$$c^3 - 2c^2 - 8c + 8 < \delta.$$

The polynomial $P = X^3 - 2X^2 - 8X + 8$is negative whenever X is between its 2nd largest root $r_2 \simeq 0.89$ and its largest root $r_1 = 4\cos(\pi/7)$: our parameter c satisfies this condition. Therefore, if we choose $\delta = P(c)/2$, there exists a t such that we have

$$P(c) < \delta < \frac{16}{c}\frac{-1}{(1+\varepsilon)\tau^t} + \frac{c^2(\tau-1)}{4(t+1)(1+\varepsilon)\tau^t},$$

i.e. Inequality (5) is satisfied, and $d_0 > \frac{c^2}{2}a_0$.

As the alternation-trading proof with parameters $(a_k)_k$ shows that

$$\mathsf{NDepth}\left[d_0 \log n\right] \subseteq \mathsf{NDepth}\left[\frac{c^2}{2}a_0 \log n\right],$$

we get a contradiction of Theorem 4.3 from the assumption $\mathsf{SAT} \in \mathsf{NDepth}\left[c \log n\right]$. ◀

Lemma 4.10 proves Theorem 1.4 for $c \in (2\phi, 4\cos(\pi/7))$. To extend this result to $c \leq 2\phi$, it suffices to notice that in that case, we have $\mathsf{NDepth}\left[c \log n\right] \subseteq \mathsf{NDepth}\left[(2\phi+\varepsilon)\log n\right]$, hence it reduces to the former case.

Optimality of the Results.

Finally, we comment briefly on the potential optimality of our results (within known techniques). In the case of SAT vs small-space algorithms, Buss and Williams [9] proved that the result $\mathsf{SAT} \notin \mathsf{TISP}\left[n^c, n^{o(1)}\right]$ for $c < 2\cos(\pi/7)$ is optimal for the alternation-trading proofs framework. This was later extended and simplified by Mudigonda and Williams [36], who showed lower bounds against randomized and quantum classes. As far as we can tell, it appears that their generalization covers our framework as well, therefore we cannot prove $\mathsf{SAT} \notin \mathsf{NDepth}\left[c \log n\right]$ for $c \geq 4\cos(\pi/7)$ without finding a new (and improved) way to speed up NDepth computation using alternations.

5 Conclusion

In this paper, we have considered two rather disparate approaches to proving stronger depth lower bounds: one for the non-uniform setting, and another for the uniform case. Let us highlight a few interesting steps for further work.

The Non-Uniform Setting.

Obviously the most pressing open problem is to design faster #SAT algorithms for DeMorgan formulas. Until now, researchers designing faster SAT algorithms for formulas and other weak classes have generally stated their time bounds in the form 2^{n-n^δ} where $\delta > 0$ is unspecified. Our work has shown that improving these δ factors, even for slightly-superquadratic formulas, would have interesting lower bound implications.

Another open problem is to consider other forms of circuit-analysis algorithms, and understand their implications. It is surprising (to us) that #SAT for formulas can be so powerful. The main intuition is that, by using low-degree approximate polynomials for formulas, the act of counting SAT assignments actually performs a non-trivial amount of the formula evaluation itself. Are there other representations of formulas which can be similarly exploited?

The Uniform Setting.

As mentioned in the introduction, the uniform lower bounds we have proved are "merely" *depth* lower bounds, which are weaker than size lower bounds, since the latter directly imply the former. However, Spira [44], Brent [8], and Bonet and Buss [7] have given a partial converse, that allows converting a $c\alpha \log n$ depth lower bounds into an n^c size lower bound, where $\alpha > 1$ depends on the computational basis. Unfortunately, for NAND formulas, the best known conversion theorems have $\alpha > 2$, making them useless for our purposes. For most families of formulas, the current best constants in the depth-to-size conversion theorems are not known to be tight.[7] So it might be possible to improve them, and use our result to prove nontrivial (and even super-cubic) size lower bounds.

The depth hierarchy theorem that we use in this work is conditional, and to our knowledge there is no unconditional equivalent. The diagonalization arguments used for uniform computational models are hard to translate to low-depth formulas, as there is no known efficient universal simulation of NAND gate formulas by a slightly larger NAND gate formula.

Finally, another immediate open problem is to lift our lower bounds for NAND gates to other computational bases. For example, one may allow AND/OR gates in any arrangement, or one might allow layers of XOR and ¬XOR (equality) gates, along with layers of OR/AND gates.

References

1 Eric Allender, José Balcázar, and Neil Immerman. A first-order isomorphism theorem. *SIAM Journal on Computing*, 26(2):557–567, 1997.

2 Eric Allender, Michal Koucký, Detlef Ronneburger, Sambuddha Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 295–302. IEEE Computer Society, 2001.

3 Alexander E. Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes. *Moscow Univ. Math. Bull.*, 42(1):63–66, 1987.

4 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[7] See Sergeev [42] (in Russian) for a recent exposition best upper and lower bounds known on the constants of multiple classes of formulas.

5 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. doi:10.1137/S0097539705446810.

6 Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *International Colloquium on Automata, Languages, and Programming*, pages 163–173. Springer, 2014.

7 Maria Luisa Bonet and Samuel R Buss. Size-depth tradeoffs for Boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

8 Richard P Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM)*, 21(2):201–206, 1974.

9 Samuel R Buss and Ryan Williams. Limits on alternation trading proofs for time–space lower bounds. *computational complexity*, 24(3):533–600, 2015.

10 Timothy M. Chan and R. Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

11 Ashok K Chandra, Dexter C Kozen, and Larry J Stockmeyer. Alternation. *Journal of the ACM (JACM)*, 28(1):114–133, 1981.

12 Lijie Chen, Zhenjian Lu, Xin Lyu, and Igor Carboni Oliveira. Majority vs. approximate linear sum and average-case complexity below NC^1. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198 of *LIPIcs*, pages 51:1–51:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.51.

13 Lijie Chen and R. Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via PCPs of proximity. In *34th Computational Complexity Conference, CCC*, volume 137 of *LIPIcs*, pages 19:1–19:43. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

14 Ruiwen Chen and Valentine Kabanets. Correlation bounds and #SAT algorithms for small linear-size circuits. *Theoretical Computer Science*, 654:2–10, 2016.

15 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24:333–392, 2015.

16 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small De Morgan formulas. *Algorithmica*, 76:68–87, 2016.

17 Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, and Robert Robere. KRW composition theorems via lifting. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 43–49. IEEE, 2020. doi:10.1109/FOCS46700.2020.00013.

18 Irit Dinur and Or Meir. Toward the KRW Composition Conjecture: Cubic Formula Lower Bounds via Communication Complexity. In *CCC 2016*, pages 3:1–3:51, 2016.

19 Lance Fortnow. Time-space tradeoffs for satisfiability. *J. Comput. Syst. Sci.*, 60(2):337–353, 2000. doi:10.1006/jcss.1999.1671.

20 Lance Fortnow, Richard J. Lipton, Dieter van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *J. ACM*, 52(6):835–865, 2005.

21 Dmitry Gavinsky, Or Meir, Omri Weinstein, and Avi Wigderson. Toward better formula lower bounds: The composition of a function and a universal relation. *SIAM J. Comput.*, 46(1):114–131, 2017. doi:10.1137/15M1018319.

22 Johan Håstad. The shrinkage exponent of DeMorgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. Preliminary version in FOCS'93.

23 Johan Håstad. On the correlation of Parity and small-depth circuits. *SIAM J. Comput.*, 43(5):1699–1708, 2014. doi:10.1137/120897432.

24 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC0. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 961–972. SIAM, 2012.

25 Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from shrinkage. *J. ACM*, 66(2):11:1–11:16, 2019. doi:10.1145/3230630.

26 Russell Impagliazzo and Noam Nisan. The effect of random restrictions on formula size. *Random Struct. Algorithms*, 4(2):121–134, 1993.

27 Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP, Proceedings, Part I*, pages 749–760, 2015. doi:10.1007/978-3-662-47672-7_61.

28 Valentine Kabanets, Sajin Koroth, Zhenjian Lu, Dimitrios Myrisiotis, and Igor C. Oliveira. Algorithms and lower bounds for de morgan formulas of low-communication leaf gates. *ACM Trans. Comput. Theory*, 13(4):23:1–23:37, 2021.

29 Valentine Kabanets, Sajin Koroth, Zhenjian Lu, Dimitrios Myrisiotis, and Igor Carboni Oliveira. Algorithms and lower bounds for de morgan formulas of low-communication leaf gates. In *35th Computational Complexity Conference, CCC*, volume 169 of *LIPIcs*, pages 15:1–15:41. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.15.

30 Ravi Kannan. Alternation and the power of nondeterminism. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 344–346. ACM, 1983. doi:10.1145/800061.808764.

31 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Comput. Complex.*, 5(3/4):191–204, 1995. doi:10.1007/BF01206317.

32 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for de Morgan formula size: Matching worst-case lower bound. *SIAM J. Comput.*, 46(1):37–57, 2017.

33 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2190–2202. SIAM, 2017.

34 William Finlay McColl. *Some results on circuit depth.* PhD thesis, University of Warwick, 1976. URL: http://wrap.warwick.ac.uk/59627/1/WRAP_THESIS_McColl_1977.pdf.

35 Ivan Mihajlin and Anastasia Sofronova. A better-than-3log(n) depth lower bound for De Morgan formulas with restrictions on top gates. In *37th Computational Complexity Conference, CCC*, volume 234 of *LIPIcs*, pages 13:1–13:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CCC.2022.13.

36 Abhijit S Mudigonda and R Ryan Williams. Time-space lower bounds for simulating proof systems with quantum and randomized verifiers. *arXiv preprint arXiv:2012.00330*, 2020.

37 Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1195887.

38 É. I. Nechiporuk. On a boolean function. *Dokl. Akad. Nauk SSSR*, 169:765–766, 1966.

39 Mike Paterson and Uri Zwick. Shrinkage of de Morgan formulae under restriction. *Random Struct. Algorithms*, 4(2):135–150, 1993.

40 Ben Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–569. SIAM, 2011. doi:10.1137/1.9781611973082.44.

41 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 183–192. IEEE, 2010.

42 Igor Sergeevich Sergeev. On a relation between the depth and complexity of monotone Boolean formulas. *Diskretnyi Analiz i Issledovanie Operatsii*, 26(4):108–120, 2019.

43 Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Information Theory*, 42(6):1723–1731, 1996. doi:10.1109/18.556668.

44 Philip Spira. On time-hardware complexity tradeoffs for Boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences, 1971*, pages 525–527, 1971.

45 Bella Abramovna Subbotovskaya. Realizations of linear functions by formulas using and, or, not. In *Soviet Mathematics Doklady*, volume 2, pages 110–112, 1961.

46 Avishay Tal. Shrinkage of De Morgan formulae by spectral techniques. In *FOCS 2014*, pages 551–560. IEEE, 2014.
47 Avishay Tal. #SAT algorithms from shrinkage. *Electron. Colloquium Comput. Complex.*, TR15-114, 2015. URL: `https://eccc.weizmann.ac.il/report/2015/114`, `arXiv:TR15-114`.
48 Dieter van Melkebeek. A survey of lower bounds for satisfiability and related problems. *Found. Trends Theor. Comput. Sci.*, 2(3):197–303, 2006. `doi:10.1561/0400000012`.
49 Dieter van Melkebeek and Ran Raz. A time lower bound for satisfiability. *Theor. Comput. Sci.*, 348(2-3):311–320, 2005. `doi:10.1016/j.tcs.2005.09.020`.
50 Emanuele Viola. The sum of D small-bias generators fools polynomials of degree D. *Comput. Complex.*, 18(2):209–217, 2009. `doi:10.1007/s00037-009-0273-5`.
51 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. `doi:10.1007/978-3-662-03927-4`.
52 R. Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *Theory of Computing*, 14(1):1–25, 2018. Preliminary version in STOC'14. `doi:10.4086/toc.2018.v014a017`.
53 Ryan Williams. Inductive time-space lower bounds for SAT and related problems. *Computational Complexity*, 15(4):433–470, 2006.
54 Ryan Williams. Time-space tradeoffs for counting NP solutions modulo integers. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 70–82. IEEE, 2007.
55 Ryan Williams. Alternation-trading proofs, linear programming, and lower bounds. *ACM Transactions on Computation Theory (TOCT)*, 5(2):1–49, 2013.
56 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.
57 Ryan Williams. Nonuniform ACC circuit lower bounds. *Journal of the ACM (JACM)*, 61(1):2, 2014.

A Andreev's Function in Low NAND Depth

In this section, we show that Andreev's function, known to require depth $(3 - o(1)) \log_2(n)$ for DeMorgan formulas, can be implemented by NAND formulas of $(3 + o(1)) \log_2(n)$ depth. This makes our depth lower bound against the Satisfiability problem all the more compelling.

Andreev's function is the composition of two functions: the "indexing function" or "storage address function", which takes as input a bitstring of length $n = 2^k$ and an index i encoded in binary on k bits and outputs x_i, the i-th bit of x, and the PARITY function. In particular, In Andreev's function, each bit of the index i in the indexing function is replaced by a PARITY on $2^k/k$ bits, so that the overall function takes $O(2^k)$ inputs.

We claim that the indexing function on bitstrings of length n has a NAND formula of depth $(1 + o(1)) \log_2(n)$, and the PARITY function on n variables has a NAND formula of depth $(2+o(1)) \log_2(n)$. It follows that Andreev's function has NAND depth $(3+o(1)) \log_2(n)$.

First, we observe the depth bound for PARITY. In fact, the natural recursive construction works. For $n = 2$, PARITY on x, y is equivalent to

$$(x \wedge \neg y) \vee (\neg x \wedge y)$$

and $\neg$PARITY is equivalent to

$$(x \wedge y) \vee (\neg x \wedge \neg y).$$

Since

$$(a \wedge b) \vee (c \wedge d) \equiv \mathsf{NAND}(\mathsf{NAND}(a, b), \mathsf{NAND}(c, d)),$$

there is an NAND formula of depth 2 for $n = 2$ variables. Let us inductively assume that PARITY and its negation on 2^k variables both have NAND formulas of depth $2k$. Then, PARITY on $n = 2^{k+1}$ variables can be expressed as

$$(X \wedge \neg Y) \vee (\neg X \wedge Y) \equiv \mathsf{NAND}(\mathsf{NAND}(X, \neg Y), \mathsf{NAND}(\neg X, Y)),$$

where X and Y denote parities on 2^k variables. This corresponds to a NAND formula of depth $2k + 2$. Similarly, $\neg$PARITY on 2^k variables has NAND depth $2k + 2$.

We now show that the indexing function has NAND depth close to $\log_2 n$.

▶ **Lemma A.1.** *For every $n = 2^k$, the indexing function over length-n bitstrings has NAND formulas of depth $\log_2(n) + O(\log \log n)$.*

Proof. We first show by induction over k a slightly different statement: for every $n = 2^k$, the indexing function over length-n bitstrings has a formula that is a NAND tree of depth $\log_2 n$ composed with n (arbitrary) formulas of size $k = O(\log n)$, each of these formulas having a single bit of x among their inputs (along with bits of i).

For $k = 1$, given $x \in \{0, 1\}^2$ and $i \in \{0, 1\}$, we have

$$\textsc{Index}(x, i) = (x_0 \wedge \neg i) \vee (x_1 \wedge i) = \mathsf{NAND}(\mathsf{NAND}(x_0, \neg i), \mathsf{NAND}(x_1, i))$$

, hence the indexing function over 2 inputs can be written as a NAND tree of depth 1 composed with formulas of size 1 (which happen to be NANDs in that case).

Now, assume that we have such a formula for some k. To build a formula for $k + 1$, we compose with a NAND gate the tree T_0 for indexing $\neg x_0, \ldots, \neg x_{2^k - 1}$ with $i_0, \ldots, i_{k-1}$ and the tree T_1 for indexing $\neg x_{2^k}, \ldots, \neg x_{2^{k+1}}$ with $i_0, \ldots, i_{k-1}$. If $i_0, \ldots, i_{k-1}$ encode the integer t, the output of this formula is $\mathsf{NAND}(\neg x_t, \neg x_{2^k + t}) = x_t \vee x_{2^k + t}$. By replacing the literals x_l^* in the formulas at the leaves of the tree by $x_l^* \wedge \neg i_k$ in T_0 and by $x_l^* \wedge i_k$ in T_1 for every l, we obtain a formula equivalent to

$$(x_t \wedge \neg i_k) \vee (x_{2^k + t} \wedge i_k),$$

which is equal to x_t if i_k is 0, and to $x_{2^k + t}$ otherwise: this is exactly x_i. Notice that, since each leaf formula contains a single x_l as input, this transformation adds at most one gate to each leaf formula, which has size at most $k + 1$. This concludes the induction.

To obtain the desired result, notice that an arbitrary formula of size s can be turned into an equivalent NAND formula of size $O(s)$, and using the result of Brent [8] and Spira [44], one can turn a NAND formula of size s' into an equivalent NAND formula of depth $O(\log s')$. Applying these transformations to the formulas of size $O(k)$ at the leaves of the NAND tree, we obtain a formula of depth $\log_2 n + O(\log k) = \log_2(n) + O(\log \log n)$ for the indexing function. ◀

Notice that this implies in particular that *any* boolean function on n variables has NAND formulas of depth $n + O(\log n)$, as any function can be written as indexing over its truth table. This result can be improved to $n + O(\log^* n)$, see [34, Theorem 3.6] for further details. This is similar to what can be achieved in the DeMorgan basis, and shows the expressiveness of NAND formulas.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il