Non-Uniform ACC Circuit Lower Bounds

Ryan Williams*
IBM Almaden Research Center

November 23, 2010

Abstract

The class ACC consists of circuit families with constant depth over unbounded fan-in AND, OR, NOT, and MOD_m gates, where $m > 1$ is an arbitrary constant. We prove:

- $\mathsf{NTIME}[2^n]$ does not have non-uniform ACC circuits of polynomial size. The size lower bound can be slightly strengthened to quasi-polynomials and other less natural functions.
- E^{NP}, the class of languages recognized in $2^{O(n)}$ time with an NP oracle, doesn't have non-uniform ACC circuits of $2^{n^{o(1)}}$ size. The lower bound gives an exponential size-depth tradeoff: for every d there is a $\delta > 0$ such that E^{NP} doesn't have depth-d ACC circuits of size 2^{n^δ}.

Previously, it was not known whether $\mathsf{EXP}^{\mathsf{NP}}$ had depth-3 polynomial size circuits made out of only MOD_6 gates. The high-level strategy is to design faster algorithms for the circuit satisfiability problem over ACC circuits, then prove that such algorithms entail the above lower bounds. The algorithm combines known properties of ACC with fast rectangular matrix multiplication and dynamic programming, while the second step requires a subtle strengthening of the author's prior work [STOC'10].

*Supported by the Josef Raviv Memorial Fellowship.

1 Introduction

Non-uniform computation allows the size of a program to grow with the sizes of inputs. A non-uniform computation can be naturally represented as an infinite family of *Boolean circuits*, one for each possible input length. A longstanding aim of complexity theory is to understand the power of non-uniform computation in relation to the usual uniform models which have fixed-size programs. One complication is that non-uniform computations can recognize *undecidable* languages by having a large enough circuit for each input length. Finding uniform computations that cannot be simulated by small non-uniform circuit families is an extremely difficult venture that is related to other major problems. For instance, $\mathsf{P} \neq \mathsf{NP}$ follows if one could provide an NP problem that cannot be solved by any circuit family where the size of the nth circuit is at most polynomial in n. Non-uniform lower bounds establish impossibility results for computation in the physical world: it could be that $\mathsf{P} \neq \mathsf{NP}$, yet NP-complete problems can still be efficiently solved using a "bloated" program with sufficiently many lines of code. Non-uniform circuit size lower bounds for NP would rule out this possibility. (However, it is currently possible that all of NP has circuits of size $6n$.)

In the early 1980's, researchers began to carefully study the power of non-uniform *low depth* circuits. Intuitively, such circuits correspond to extremely fast parallel computations. The initial hope was that if some functions in NP were proved to require large, restricted circuit families, then by gradually lifting the restrictions over time, superpolynomial size unrestricted lower bounds for NP could be attained, proving $\mathsf{P} \neq \mathsf{NP}$. Furst, Saxe, and Sipser [FSS81] and independently Ajtai [Ajt83] showed that functions such as the parity of n bits cannot be computed by polynomial size AC^0 circuits, i.e., polynomial size circuit families of constant depth over the usual basis of AND, OR, and NOT gates, where each AND and OR may have arbitrarily many inputs. Yao [Yao85] improved the lower bounds to exponential size, and Håstad [Hås86] proved essentially optimal AC^0 lower bounds for parity. Around the same time, Razborov [Raz85] proved superpolynomial lower bounds for solving clique with *monotone* circuits (i.e., general circuits without NOT gates), and the bound was improved to exponential size by Alon and Boppana [AB87]. However, it was later shown [Raz89] that the monotone techniques probably would not extend to general circuits.

Encouraged by the progress on AC^0, attention turned to lower bounds for what seemed to be minor generalizations. The most natural generalization was to grant AC^0 the parity function for free. Razborov [Raz87] proved an exponential lower bound for computing the majority of n bits with constant-depth circuits made up of AND, OR, NOT, and MOD_2 gates. (A MOD_m gate outputs 1 iff m divides the sum of its inputs.) Then Smolensky [Smo87] proved exponential lower bounds for computing MOD_q with constant-depth circuits made up of AND, OR, NOT, and MOD_p gates, for distinct primes p and q. Barrington [Bar89] suggested the next step would be to prove lower bounds for the class ACC, which consists of constant-depth circuit families over the basis AND, OR, NOT, and MOD_m for arbitrary constant $m > 1$.[1] It is here that progress on strong lower bounds began to falter (although there has been progress on further restricted cases, cf. the Preliminaries). While it was conjectured that the majority of n bits cannot have polynomial ACC circuits, strong ACC lower bounds remained elusive.

After some years of failing to prove a superpolynomial lower bound, the primary questions were weakened. Rather than trying to find simple functions that cannot be computed with weak circuits, perhaps we could rule out weak circuits for complicated functions. Could one prove that nondeterministic *exponential* time (NEXP) doesn't have polynomial size circuits? A series of papers starting with Babai, Fortnow, Nisan, and Wigderson [BFNW93, KvM99, IKW02] showed that even this sort of lower bound would imply derandomization results: in the case of NEXP lower bounds, it would imply that Merlin-Arthur games can

[1] The class is also called ACC^0 in the literature. However, as ACC^i is hardly studied at all, for any $i > 0$, at the present time it makes sense to drop the superscript.

be non-trivially simulated with nondeterministic algorithms. This indicated that proving good circuit lower bounds for NEXP would already require significantly new ideas.

In this paper, we address two frontier questions concerning non-uniform circuit complexity:

1. *Does nondeterministic $2^{O(n)}$ time have non-uniform polynomial size* ACC *circuits?*
   (Is $\mathsf{NTIME}[2^{O(n)}]$ in non-uniform ACC?)

2. *Does exponential time with an* NP *oracle have non-uniform polynomial size circuits?*
   (Is $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{P/poly}$?)

Over the years, these questions have turned into notorious and somewhat embarrassing open problems, because it seems so obvious that the answers should be no. It was open if $\mathsf{EXP}^{\mathsf{NP}}$ could be recognized with depth-3 polynomial size circuits made out of only MOD_6 gates.[2] We make headway on these frontiers, giving a strong *no* answer to the first question.

Theorem 1.1 $\mathsf{NTIME}[2^n]$ *does not have non-uniform* ACC *circuits of polynomial size.*

Stronger size lower bounds hold (e.g. quasi-polynomial size) but the results are not very clean; see Section 5.1 for details. For $\mathsf{EXP}^{\mathsf{NP}}$, we can prove an exponential lower bound.

Theorem 1.2 (Exponential Size-Depth Tradeoff) *For every d, there is a $\delta > 0$ and a language in E^{NP} that fails to have non-uniform* ACC *circuits of depth d and size 2^{n^δ}.*

Recall that the lowest complexity class for which we know exponential (unrestricted) circuit lower bounds is Δ_3^{EXP}, the third level of the exponential hierarchy [MVW99].

Extending the approach of this paper to settle the second frontier question may be difficult, but this prospect does not look as implausible as it did before. If polynomial unrestricted circuits could be simulated by subexponential ACC circuits, or if one could improve just a little on the running time of algorithms for the circuit satisfiability problem, the second question would be settled.

1.1 An Overview of the Proofs

Let us sketch how these new lower bounds are proved, giving a roadmap for the rest of the paper. In recent work [Wil10], the author suggested a research program for proving non-uniform circuit lower bounds for NEXP. It was shown that for many circuit classes $\mathcal{C}$, sufficiently faster satisfiability algorithms for $\mathcal{C}$-circuits would entail non-uniform lower bounds for $\mathcal{C}$-circuits. The objective of this paper is to carry out the proposed research program in the case of ACC circuits.

The proof of the lower bound for E^{NP} (Theorem 1.2) is a combination of complexity-theoretic ideas (time hierarchies, compression by circuits, the local checkability of computation) and algorithmic ideas (fast matrix multiplication, dynamic programming, table lookup).

1. First, we show that satisfiability algorithms for subexponential size n-input ACC circuits with running time $O(2^n/n^k)$ imply exponential size ACC lower bounds for E^{NP} (Theorem 3.2), where k is sufficiently large. The model of computation for the satisfiability algorithm is flexible; we may assume the multitape Turing model or a random access machine. This step considerably strengthens results of earlier work [Wil10] which could only show that an $o(2^{n/3})$ time algorithm for ACC circuit satisfiability implies lower bounds.

[2] Note that slightly larger classes such as MAEXP and $\mathsf{NEXP}^{\mathsf{NP}}$ are known to not have polynomial size circuits; see the Preliminaries.

The strategy is to prove that, if there is a faster algorithm for ACC circuit satisfiability, and there are subexponential ($2^{n^{o(1)}}$) size ACC circuits for E^{NP}, then every $L \in \mathsf{NTIME}[2^n]$ can be accepted by a nondeterministic algorithm in $O(2^n n^{10}/n^k)$ time. (Here, 10 is a substitute for a small universal constant.) When $k > 10$ this contradicts the nondeterministic time hierarchy theorem [SFM78, Zak83], so one of the assumptions must be false. The nondeterministic time hierarchy is fairly robust with respect to the machine model: for large enough k, $\mathsf{NTIME}_{TM}[2^n] \not\subseteq \mathsf{NTIME}_{RAM}[2^n/n^k]$ where NTIME_{TM} denotes nondeterministic multitape Turing machines and NTIME_{RAM} denotes nondeterministic RAMs in the usual logarithmic cost model (cf. Lemma 2.1). This implies it suffices for the underlying ACC SAT algorithm to work on a RAM (provided it runs in $O(2^n/n^k)$ time for large enough k).

Two known facts are required in the proof. First, there is a polynomial-time reduction from any $L \in \mathsf{NTIME}[2^n]$ to the NEXP-complete problem SUCCINCT 3SAT such that every instance x of length n (for sufficiently large n) is reduced to a (unrestricted, not ACC) circuit C_x of size $O(n^5)$ with at most $n + 5 \log n$ inputs (Fact 3.1). That is, the bit string obtained by evaluating C_x on its $O(2^n n^5)$ possible assignments (in lex order) encodes a 3CNF formula F_{C_x} that is satisfiable iff $x \in L$. Second, if E^{NP} is in subexponential ACC, then (given an input x) the lexicographically first satisfying assignment to the formula encoded by C_x can be described by an ACC circuit W of subexponential size (Fact 3.2). That is, the bit string obtained by evaluating W on all possible assignments encodes a satisfying assignment to the exponentially long F_{C_x}.

If C_x were an ACC circuit, then any L could be simulated in $O(2^n n^5/n^k)$ nondeterministic time, by nondeterministically guessing a subexponential ACC circuit W and constructing an ACC circuit satisfiability instance D built of C_x and W, where D is satisfiable if and only if W does not encode a satisfying assignment to F_{C_x} (as shown in the author's prior paper). The circuit D has at most $n + 5 \log n$ inputs and $2^{n^{o(1)}}$ size, so the assumed ACC satisfiability algorithm can handle D in $O(2^n n^5/n^k)$ time.

The above argument doesn't quite work, because we do not know how to produce a C_x that is an ACC circuit (indeed, it may not be possible). An ACC SAT algorithm will not work on D, because it contains a copy of an unrestricted C_x. However, assuming only P has subexponential ACC circuits, we show how to *guess and verify* an equivalent ACC circuit C'_x in nondeterministic $O(2^n n^{10}/n^k)$ time using a faster ACC satisfiability algorithm (Lemma 3.1). This new result makes it possible to prove lower bounds even with weak ACC satisfiability algorithms. Furthermore, this part of the proof does not use any specific properties of ACC, so it could potentially be applied for stronger lower bounds in the future.

2. Next we show how satisfiability of subexponential ACC circuits of depth d and n inputs can be determined in $2^{n-\Omega(n^\delta)}$ time, for a $\delta > 0$ that depends on d (Theorem 4.1). Given any such circuit C, replace it with C' which is an OR of 2^{n^δ} copies of C, where the first n^δ inputs of each copy are substituted with a variable assignment. This ACC circuit C' has $n - n^\delta$ inputs, $2^{O(n^\delta)}$ size, and C is satisfiable if and only if C' is. Applying a powerful result of Yao, Beigel-Tarui, and Allender-Gore (Lemma 4.1), C' can be replaced by an equivalent depth-2 circuit C'' of $2^{n^\delta 2^{O(d)}}$ size, which consists of an efficiently computable symmetric function at the output gate and AND gates below it. Setting $\delta \ll 1/2^{O(d)}$, and using a nearly optimal rectangular matrix multiplication algorithm due to Coppersmith (Lemma 4.3), C'' can be evaluated on all of its possible assignments in $2^{n-n^\delta}\mathrm{poly}(n)$ time (Lemma 4.2). Alternatively, this evaluation of C'' can also be done via simple dynamic programming. This concludes the sketch of the E^{NP} lower bound.

The only use of the full assumption "E^{NP} has ACC circuits" is in Fact 3.2. The lower bound for NEXP (Theorem 1.1) applies the result (which follows from work of Impagliazzo, Kabanets, and Wigderson [IKW02]) that if NEXP has polynomial size (unrestricted) circuits then satisfiable instances of SUCCINCT 3SAT already have polynomial size (unrestricted) circuits W encoding satisfying assignments (Theorem 5.1). But if P has ACC circuits, it is easy to see that these unrestricted circuits must have equivalent

ACC circuits as well (Lemma 5.1). This helps extend the E^{NP} lower bound to NEXP. However, the resulting size lower bound is not exponential: from $S(n)$-size circuits for NEXP one only obtains $S(S(S(n)^c)^c)^c$-size ACC circuits encoding satisfying assignments. This allows for some "half-exponential" type improvements in the size lower bounds against NEXP.

Perhaps the most interesting aspect of the proofs is that only the satisfiability algorithm for ACC circuits relies on specific properties of ACC. *Improved exponential algorithms for satisfiability are the only barrier to further progress on circuit lower bounds for* NEXP. In general, this paper weakens the algorithmic assumptions necessary to prove lower bounds, and strengthens the lower bounds obtained. Let $\mathcal{C}$ be a class of circuit families that is closed under composition (the composition of two circuit families from $\mathcal{C}$ is also a family in $\mathcal{C}$) and contains AC^0. Possible $\mathcal{C}$ include constant-depth threshold circuits, Boolean formulas, and unrestricted Boolean circuits. The arguments of Section 3 and Section 5 imply the following metatheorem.

Theorem 1.3 *There is a $k > 0$ such that, if satisfiability of $\mathcal{C}$-circuits with n variables and n^c size can be solved in $O(2^n/n^k)$ time for every c, then* $\mathsf{NTIME}[2^n]$ *doesn't have non-uniform polysize $\mathcal{C}$-circuits.*

2 Preliminaries

We presume the reader has background in circuit complexity and complexity theory in general. The textbook of Arora and Barak [AB09] covers all the necessary material; in particular, Chapter 14 gives an excellent summary of ACC and the frontiers in circuit complexity.

On the machine model. An important point about this paper is that the choice of uniform machine model is not crucial to the arguments. We show that if large classes have small non-uniform ACC circuits, then $\mathsf{NTIME}[2^n] \subseteq \mathsf{NTIME}[o(2^n)]$ (in fact, $\mathsf{NTIME}[2^n] \subseteq \mathsf{NTIME}[o(2^n/n^k)]$ for sufficiently large k), which is a contradiction in all computational models we are aware of. Moreover, Gurevich and Shelah proved that the nondeterministic machine models are tightly related in their time complexities. For example, let $\mathsf{NTIME}_{RTM}[t(n)]$ be the languages recognized by nondeterministic $t(n)$ time random-access Turing machines, and let $\mathsf{NTIME}_{TM}[t(n)]$ be the class for multitape Turing machines.

Theorem 2.1 (Gurevich and Shelah [GS89])
$\bigcup_{c>0} \mathsf{NTIME}_{RTM}[n \log^c n] = \bigcup_{c>0} \mathsf{NTIME}_{TM}[n \log^c n]$.

As a consequence, even if we showed $\mathsf{NTIME}_{TM}[2^n] \subseteq \mathsf{NTIME}_{RTM}[2^n/n^k]$ for sufficiently large k, we would still obtain the desired contradiction. (Note that such a result is *not* known for the deterministic setting.) A random access Turing machine can also simulate a log-cost random access machine with only constant factor overhead [PR81]. Hence in our proof by contradiction, we may assume that the source algorithm we're simulating is only a multitape TM, while the target algorithm has all the power we need to perform typical computations from the literature.

Notation. Inside of an algorithm description, the integer n refers to the length of the input to the algorithm. For a function $f : \mathbb{N} \to \mathbb{N}$, we use $\mathrm{poly}(f(n))$ to denote a growth rate of the form $cf(n)^c$ for a constant c.

The size of a circuit refers to the number of wires in it. However, since our attention shall be restricted to circuits with at least polynomially many gates, the distinction between the number of wires and gates does not matter. An *unrestricted* circuit has gate types AND/OR/NOT, and each gate has fan-in two. (That is, an unrestricted circuit is the generic variety used in the definition of P/poly.) All circuit size functions S considered in this paper are assumed to be monotone nondecreasing, i.e., $S(n+1) \geq S(n)$ for all n.

We say that a generic *circuit class* $\mathcal{C}$ is a collection of circuit families that (a) *contains* AC^0 (for every circuit family in AC^0, there is an equivalent circuit family in $\mathcal{C}$) and (b) *is closed under composition*: if $\{C_n\}$

and $\{D_n\}$ are families in $\mathcal{C}$, then for every c, the circuit family consisting of circuits which take n bits of input, feed them to $n^c + c$ copies of circuits from C_n, and feed those outputs to the inputs of D_{n^c+c}, are also circuit families in $\mathcal{C}$. Essentially all classes studied extensively in the literature (AC^0, ACC, TC^0, NC^1, NC^2, $\mathsf{P/poly}$, etc.) may be construed as circuit classes in this sense. For classes that allow for superpolynomial size circuits, the polynomial "$n^c + c$" in the above may be relaxed appropriately.

For a complexity class $\mathcal{C}$, the class i.o.-$\mathcal{C}$ consists of languages $L \subset \Sigma^\star$ such that there is a language $L' \in \mathcal{C}$ where $L \cap \Sigma^n = L' \cap \Sigma^n$ holds for infinitely many n.

When the expression "$O(1)$" appears inside of the time bound for a complexity class, this is shorthand for the union of all classes where the $O(1)$ is substituted by a fixed constant. For example, the class $\mathsf{TIME}[2^{n^{O(1)}}]$ is shorthand for $\bigcup_{c \geq 0} \mathsf{TIME}[2^{n^c}]$.

Other Prior Work. Kannan [Kan82] showed in 1982 that for any superpolynomial constructible function $S : \mathbb{N} \to \mathbb{N}$, the class $\mathsf{NTIME}[S(n)]^{\mathsf{NP}}$ does not have polynomial size circuits. Another somewhat small class known to not have unrestricted polynomial size circuits is MAEXP [BFT98]. Later it was shown that the MAEXP lower bound can be improved to *half-exponential* size functions f which satisfy $f(f(n)) \geq 2^n$ [MVW99]. Kabanets and Impagliazzo [KI04] proved that $\mathsf{NEXP}^{\mathsf{RP}}$ *either* doesn't have polynomial size Boolean circuits (over AND, OR, NOT), *or* it doesn't have polynomial size arithmetic circuits (over the integers, with addition and multiplication gates). Note that $\mathsf{NEXP}^{\mathsf{RP}} \subseteq \mathsf{MAEXP}$.

A line of work initiated by Yao [Yao90] has studied ways of representing ACC circuits by certain depth-two circuits which will play a critical role in this paper. Define a SYM^+ circuit to be a depth-two circuit which computes some symmetric function at the output gate, and computes ANDs of input variables on the second layer. Yao showed that every ACC circuit of s size can be represented by a probabilistic SYM^+ circuit of $s^{O(\log^c s)}$ size, where c depends on the depth, and the ANDs have $\mathrm{poly}(\log s)$ fan-in. Beigel and Tarui [BT94] showed how to remove the probabilistic condition. Allender and Gore [AG94] showed that every subexponential *uniform* ACC circuit family can be simulated by subexponential *uniform* SYM^+ circuits. This was applied to show that the Permanent does not have *uniform* ACC circuits of subexponential size. Later, Allender [All99] improved the Permanent lower bound to polynomial size uniform TC^0 circuits. However, these proofs require uniformity, and the difference between uniformity and non-uniformity may well be vast (e.g., it is clear that $\mathsf{P} \neq \mathsf{NEXP}$, but open whether $\mathsf{NEXP} \subseteq \mathsf{P/poly}$). Green *et al.* [GKRST95] showed that the symmetric function can be assumed to be the specific function which returns the *middle bit* of the sum of its inputs. This representation may also be used in the lower bounds of this paper.

There has also been substantial work on representing ACC in other interesting ways [BT88, AAD00, Han06, KH09] as well as many lower bounds in restricted cases [BST90, Thé94, YP94, KP94, BS95, Cau96, Gro98, GT00, CGPT06, CW09]. Significant work has gone into understanding the *constant degree hypothesis* [BST90] that a certain type of low-depth ACC circuit requires exponential size to compute the AND function. The hypothesis is still open.

All prior works on non-uniform ACC lower bounds attack the problem in a "bottom-up" way. (The exceptions are the uniform results mentioned above [AG94, All99].) Lower bounds have been proved for highly restricted circuits and these restrictions have been very gradually relaxed over time. In this paper, the strategy is "top-down": the goal is to find the smallest complexity classes for which it is still possible to prove superpolynomial ACC lower bounds. This is in line with the overall goal of eventually proving large circuit lower bounds for NP.

As mentioned before, this paper builds on the author's prior work which showed that mild improvements over exhaustive search can sometimes imply lower bounds. Let us briefly review the prior state-of-the-art for SAT algorithms. It is known that CNF satisfiability can be solved in $2^{n-\Omega(n/\ln(m/n))}\mathrm{poly}(m)$ time,

where m is the number of clauses and n is the number of variables [DH08, CIP06]. Recent work of Calabro, Impagliazzo, and Paturi has shown that AC^0 circuit satisfiability can be determined with a randomized algorithm in $2^{n-n^{1-o(1)}}$ time on circuits with $n^{1+o(1)}$ gates [CIP09]. Recently, Santhanam [San10] has applied ideas inspired by formula size lower bounds to show that for a fixed constant k, Boolean formula satisfiability can be determined in $O(2^{n-n/c^k})$ time on formulas of size cn. Unfortunately, these upper bounds are not yet strong enough to prove new circuit lower bounds.

How does this work get around the barriers? There are several well-known barriers to proving lower bounds, and any proof claiming a new lower bound should try to explain a bit about how it manages to work around them. Briefly, the approach of this paper circumvents the natural proofs barrier because of its use of diagonalization: a proof by contradiction is given which appeals to a time hierarchy theorem, so constructivity is violated. (Furthermore, to our knowledge there is little evidence that ACC contains pseudorandom function generators anyway, so it's not clear that natural proofs should be considered a barrier for ACC.) Informally, the approach circumvents relativization and algebrization because it relies on an efficient ACC satisfiability algorithm, which uses non-relativizing properties of ACC circuits (reduction to a SYM^+ circuit). In general, the approach of using SAT algorithms to prove lower bounds appears fruitful for circumventing oracle-based barriers, because all known improved satisfiability algorithms break down when oracles (or algebraic extensions thereof) are added to the instance. That is, significant improvements over exhaustive search necessarily exploit structure in instances that black-box methods cannot see.

3 A Strengthened Connection Between SAT Algorithms and Lower Bounds

In this section, we prove that if one can achieve a very minor improvement over exhaustive search in satisfying ACC circuits, then one can prove lower bounds for ACC. The required improvement is so minor that we are able to achieve it, in the sequel. However, let us stress upfront that all the results in this section hold equally well for other circuit classes as well: we only require basic properties of ACC that practically all robust circuit classes satisfy.

Define the ACC CIRCUIT SAT problem to be: *given an* ACC *circuit C, is there an assignment of its inputs that makes C evaluate to* 1*?* In recent prior work [Wil10], the author proved a relation between algorithms for ACC CIRCUIT SAT and lower bounds for ACC circuits:[3]

Theorem 3.1 ([Wil10]) *Let $s(n) = \omega(n^k)$ for every k. If* ACC CIRCUIT SAT *instances with n variables and n^c size can be solved in $O(2^{n/3}/s(n))$ time for every c, then E^{NP} does not have non-uniform* ACC *circuits of polynomial size.*

We shall sharpen this theorem. Let $S : \mathbb{N} \to \mathbb{N}$ be a monotone nondecreasing function such that $S(n) \geq n$. Let $\mathcal{C}$ be a circuit class as defined in the Preliminaries. ($\mathcal{C}$ can be ACC, TC^0, NC^1, P/poly, etc.) Define the $\mathcal{C}$-CIRCUIT SAT problem to be: *given a circuit C from class $\mathcal{C}$, is there an assignment of its inputs that makes C evaluate to* 1*?*

Theorem 3.2 *Let $S(n) \leq 2^{n/4}$. There is a $c > 0$ such that, if $\mathcal{C}$-*CIRCUIT SAT *instances with at most $n + c \log n$ variables, depth $2d + O(1)$, and $O(n\, S(2n) + S(3n))$ size can be solved in $O(2^n/n^c)$ time, then E^{NP} does not have non-uniform $\mathcal{C}$ circuits of depth d and $S(n)$ size.*

The constant c depends on the model of computation in which the SAT algorithm is implemented, but for all typical models, c is not large (less than 10). For us, the important corollary is this: if ACC satisfiability

[3] In fact a more general result for any circuit class was proved, which implies Theorem 3.1.

has a slightly faster algorithm on circuits that are mildly larger than $S(n)$, then E^{NP} does not have ACC circuits of $S(n)$ size. In what follows, we prove Theorem 3.2 only for ACC circuits, but the proof also works for any other circuit class. (The reader can verify that the only two properties of ACC used are that the class contains AC^0, and the class is closed under composition of circuit families.)

To understand the difficulty behind proving Theorem 3.2, let us recall the proof of Theorem 3.1 to see why it needed such a strong assumption. The generic proof idea in [Wil10] for results such as Theorem 3.1 is to derive a contradiction from assuming small circuits for E^{NP} and a faster algorithm for CIRCUIT SAT. In particular, it is shown that under the two assumptions, every language $L \in \mathsf{NTIME}[2^n]$ can be recognized in $\mathsf{NTIME}[o(2^n)]$, which is false by the nondeterministic time hierarchy theorem [SFM78, Zak83]. The contradiction is derived from several facts about circuits and satisfiability.

Define SUCCINCT 3SAT as the problem: *given a circuit C on n inputs, let F_C be the 2^n-bit instance of 3-SAT obtained by evaluating C on all of its possible inputs in lexicographical order. Is F_C satisfiable?*

That is, given a *compressed encoding* of a 3-CNF formula, the task is to determine if the underlying decompressed formula is satisfiable. For natural reasons, call F_C the *decompression* of C, and call C the *compression* of F_C. The SUCCINCT 3SAT problem is a canonical NEXP-complete problem [PY86].

Fact 3.1 *There is a constant $c > 0$ such that for every $L \in \mathsf{NTIME}[2^n]$, there is a reduction from L to* SUCCINCT 3SAT *which on input x of length n runs in $\mathrm{poly}(n)$ time and produces a circuit C_x with at most $n + c\log n$ inputs and $O(n^c)$ size, such that $x \in L$ if and only if the decompressed formula F_{C_x} of $2^n \cdot \mathrm{poly}(n)$ size is satisfiable.*

Fact 3.1 follows from several prior works concerned with the complexity of the Cook-Levin theorem [Tou01, FLvMV05]:

Theorem 3.3 (Tourlakis [Tou01], Fortnow *et al.* [FLvMV05]) *There is a $c > 0$ such that for all $L \in \mathsf{NTIME}[n]$, L reduces to 3SAT in $O(n(\log n)^c)$ time. Moreover there is an algorithm (with random access to its input) that, given an instance of L with length n and an integer $i \in [dn(\log n)^c]$ in binary (for some d depending on L), outputs the ith clause of the resulting 3SAT formula in $O((\log n)^c)$ time.*

In fact, the proofs in the above references build on even earlier work of Schnorr, Cook, Gurevich-Shelah, and Robson [Sch78, Coo88, GS89, Rob91]. In a nutshell, all of these proofs exploit the *locality of computation*: every nondeterministic computation running in linear time can be represented with a nondeterministic circuit of size $O(n \cdot \mathrm{poly}(\log n))$ which has a highly regular and efficiently computable structure. This circuit can be easily modeled as a 3-CNF formula using the Tseitin transformation that assigns a variable to each circuit wire and uses 3-CNF clauses to model the input-output relationships for each gate.

The value of c in Theorem 3.3 depends on the underlying computational model; typically one can take c to be at most 4. A standard padding argument (substituting 2^n in place of n) yields Fact 3.1. In more detail, given $L \in \mathsf{NTIME}[2^n]$, we apply Theorem 3.3 to the language $L' = \{x01^{2^{|x|}} \mid x \in L\}$, which is in $\mathsf{NTIME}[n]$. On an input x, this generates an equivalent 3SAT instance of length $O(2^{|x|}|x|^c)$. As it is easy to simulate random accesses to an input of the form $x01^{2^{|x|}}$ with a uniform $\mathrm{poly}(|x|)$ size circuit, one can simulate the $O((\log n)^c)$ time algorithm of Theorem 3.3 on L', with a uniform $\mathrm{poly}(|x|^c)$ size circuit.

Using Fact 3.1, one can then prove that every succinctly compressible satisfiable formula that is output by the SUCCINCT 3SAT reduction has some succinctly compressible *satisfying assignment*.

Fact 3.2 *If E^{NP} has ACC circuits of size $S(n)$, then there is a fixed constant c such that for every language $L \in \mathsf{NTIME}[2^n]$ and every $x \in L$ of length n, there is a circuit W_x of size at most $S(3n)$ with $k \leq n + c\log n$ inputs such that the variable assignment $z_i = W(i)$ for all $i = 1, \ldots, 2^k$ is a satisfying assignment for the formula F_{C_x}, where C_x is the circuit obtained by the reduction in Fact 3.1.*

Proof of Fact 3.2. Consider the E^{NP} machine:

> $N(x, i)$: *Compute the* SUCCINCT 3SAT *reduction from x to C_x in polynomial time. Decompress C_x, obtaining a formula F of $O(2^{|x|}|x|^c)$ size. Let k be the number of inputs to C_x. Binary search for the lexicographically smallest satisfying assignment A to F, by repeatedly querying:* given (F, A) where $|A| \leq 2^k$, is there an assignment $A' \leq A$ that satisfies F? *Then output the ith bit of A.*

Note the queries can be answered in NP, and N needs $O(2^k)$ queries to the oracle. By assumption, N has ACC circuits of size $S(n)$. It follows that for every $x \in L$ there is some satisfying assignment to F which is encoded by a circuit of size $S(|\langle x, i\rangle|) \leq S(3|x|)$, where $\langle \cdot, \cdot \rangle$ is a polynomial-time computable pairing function. □

Given these two facts, one can recognize any $L \in \mathsf{NTIME}[2^n]$ with a $o(2^n)$ nondeterministic algorithm (a contradiction), as follows. Given a string x of length n, compute the SUCCINCT 3SAT circuit C_x in polynomial time and nondeterministically *guess* a $S(3n)$ size circuit W. Now the goal is to check that W succinctly encodes a satisfying assignment for the underlying formula F_{C_x}. To verify this condition, the algorithm constructs a CIRCUIT SAT instance D. The circuit D has $n + c \log n$ inputs fed to $O(n)$ copies of C_x, so that when i is input to D, the copies altogether print the ith clause of the 3CNF formula F_{C_x}. These copies output three variable indices of length at most $n + c \log n$, along with sign bits (whether or not the variables are negated in the clause). Then D feeds each index to a copy of W, which prints a bit. Finally D compares the sign bits with the three bits printed by the copies of W, and outputs 0 iff the variable assignment encoded by W satisfies the ith clause. Observe D has $\text{poly}(n) + O(S(3n))$ size. Running a fast enough CIRCUIT SAT algorithm lets us determine the satisfiability of D in $o(2^n)$ time. Finally, this algorithm for L accepts x iff D is unsatisfiable. To see that this algorithm is correct, observe there is a size-$S(3n)$ circuit W such that D is an unsatisfiable circuit, if and only if there is such a W encoding a satisfying assignment for F_{C_x}, if and only if $x \in L$.

The above argument cannot be carried out directly to prove ACC circuit lower bounds from ACC CIRCUIT SAT algorithms, because of Fact 3.1. Given an instance x of L, the resulting circuit C_x produced in the reduction from L to SUCCINCT 3SAT can be constructed in polynomial time, however it looks hard (perhaps impossible) to show that this C_x can be assumed to be an ACC circuit. As C_x is a component of the circuit D, it follows that D itself would not be an ACC circuit, so an ACC CIRCUIT SAT algorithm would not seem to be useful for determining the satisfiability of D.

In the proof of Theorem 3.1 in the author's prior work [Wil10], this problem was fixed by settling for a weaker reduction from L to SUCCINCT SAT, which generates an AC^0 circuit C'_x with $3n + O(\log n)$ inputs rather than $n + O(\log n)$. Unfortunately this constant factor makes a huge difference: to quickly determine satisfiability of the resulting circuit D' in $o(2^n)$ time, a $2^{n/3}/n^{\omega(1)}$ time algorithm for ACC CIRCUIT SAT is needed, instead of a $2^n/n^{\omega(1)}$ algorithm. Algorithms of the former type are not known even for 3SAT; algorithms of the latter type are much more plentiful.

While it is unlikely that these C_x circuits can be implemented in ACC, note that we already *assume* that ACC is powerful in some sense: in a proof by contradiction, we may assume many functions have small ACC circuits! Since the function computed by C_x is computable in polynomial time, then even if we assume that only P has ACC circuits, there still *exists* a circuit C'_x which is ACC and equivalent to C_x, but it is from a non-uniform family, and therefore may be arbitrarily difficult to construct. However, we can use nondeterminism in the algorithm recognizing L in $\mathsf{NTIME}[o(2^n)]$, so at the very least we can *guess* this elusive C'_x. We also have a good algorithm for ACC CIRCUIT SAT at our disposal. By guessing two more

ACC circuits to help us, it turns out that we can always generate a correct ACC circuit C'_x that is equivalent to C_x in $o(2^n)$ time. We arrive at our main lemma:

Lemma 3.1 *There is a fixed $d > 0$ with the following property. Assume* P *has* ACC *circuits of depth d' and size at most $S(n)$. Further assume* ACC CIRCUIT SAT *on circuits with $n + c \log n$ inputs, depth $2d' + O(1)$, and at most $O(S(3n) + S(2n)n)$ size can be solved in $O(2^n/n^c)$ time, for sufficiently large $c > 2d$.*

Then for every $L \in \mathsf{NTIME}[2^n]$, there is a nondeterministic algorithm $\mathcal{A}$ such that:

- *$\mathcal{A}$ runs in $O(2^n/n^c + S(3n) \cdot \mathrm{poly}(n))$ time,*
- *for every x of length n, $\mathcal{A}(x)$ either prints* reject *or it prints an* ACC *circuit C'_x with $n + d \log n$ inputs, depth d', and $S(n + d \log n)$ size, such that $x \in L$ if and only if C'_x is the compression of a satisfiable 3-CNF formula of $2^n \cdot \mathrm{poly}(n)$ size, and*
- *there is always at least one computation path of $\mathcal{A}(x)$ that prints the circuit C'_x.*

That is, given an instance x, the algorithm $\mathcal{A}$ nondeterministically generates an equivalent SUCCINCT 3SAT instance C'_x which is an ACC circuit. Informally, $\mathcal{A}$ will guess and verify C'_x in three stages.

1. $\mathcal{A}$ guesses an ACC circuit D of depth d' and size less than $O(S(2n) \log n)$ which encodes the gate information of the circuit C_x which has $kn^d + k$ size. Given a gate index $j = 1, \ldots, kn^d + k$, D produces the gate type of j, as well as the indices of gates whose outputs are the inputs for gate j. The correctness of D can be verified in $O(n^d S(2n) \cdot \mathrm{poly}(\log S(2n)))$ time by simply producing the whole $kn^d + k$ size circuit described by D and comparing that with C_x.
2. $\mathcal{A}$ guesses an ACC circuit E of depth d' and $S(n + d \log n + O(1)) \leq S(2n)$ size which encodes the evaluation of C_x on every input i: given input i and a gate index $j = 1, \ldots, kn^d + k$, E produces the output of gate j in C_x evaluated on i. $\mathcal{A}$ verifies that E is correct, using the fact that D is correct. By constructing an appropriate ACC CIRCUIT SAT instance that checks for *all* inputs and *all* gates that the claimed inputs to that gate are consistent with the output of the gate, this verification takes $O(2^n/n^c)$ time (for c chosen to be greater than $2d$).
3. Then using the fact that E is correct, it is easy to verify C'_x is correct via a call to ACC CIRCUIT SAT in $O(2^n/n^c)$ time. $\mathcal{A}$ only needs to check if there is an i such that $C'_x(i) \neq E(i, j^\star)$, where $j^\star$ is the index of the output gate. (Alternatively, we could just print the circuit $E(\cdot, j^\star)$ as a valid ACC circuit that is equivalent to $C_x(\cdot)$.) If E is correct and no such i exists, then C'_x is also correct.

Proof of Lemma 3.1. We describe $\mathcal{A}$ in detail. On input x of length n, $\mathcal{A}$ guesses an ACC circuit C'_x of size $S(n + d \log n)$, and constructs the SUCCINCT 3SAT circuit C_x with $n + d \log n$ inputs and at most $kn^d + k$ size (of Fact 3.1) in polynomial time, for some fixed d that is independent of L. By Fact 3.1, $x \in L$ if and only if C_x is the compression of a satisfiable formula F_{C_x} of $O(2^n n^d)$ length. We must verify that C'_x and C_x compute exactly the same function, using only the algorithm for ACC CIRCUIT SAT.

Without loss of generality, the unrestricted circuit C_x above has gate types AND, OR, NOT, and INPUT, where every AND and OR has fan-in two. By definition an INPUT gate has no inputs, and the output value of an INPUT gate is the appropriate input bit itself. The gates are indexed by the numbers $1, \ldots, kn^d + k$, where the first $n + d \log n$ indices correspond to the $n + d \log n$ INPUT gates, and the $(kn^d + k)$th gate is the output gate.

Since the map $x \mapsto C_x$ is polynomial time computable, the following function f is polynomial-time computable:

Given x, and a gate index $j = 1, \ldots, kn^d + k$, $f(x, j)$ outputs the gate type (AND, OR, NOT, INPUT) of the jth gate in the circuit C_x. Furthermore, if the gate type is NOT, then f outputs the gate index j_1 in C_x whose output is the input to j; if the gate type is an AND or OR, then f outputs the two gate indices j_1 and j_2 in C_x whose outputs are the two inputs of j.

Consider the decision problem D_f: *given x, j, and $i = 1, \ldots, 2d \log n + O(1)$, decide if the ith bit of $f(x, j)$ is* 1. The problem D_f is solvable in polynomial time and hence has $O(S(n + d \log n + O(\log \log n)))$-size, d'-depth ACC circuits, by assumption.

Let $D(x, j)$ be an ACC circuit implementing the functionality of f. Note we may assume the size of D is $O(S(n + O(\log n)) \log n)$, by simply taking $2d \log n + O(1)$ copies of the $S(n + O(\log n))$-size circuit solving the decision problem D_f. (By convention, let us assume that when D is printing the gate information for an INPUT gate, it prints all-zeroes strings in place of j_1 and j_2, and when D is printing the information for a NOT gate, it prints all-zeroes in place of j_2.)

The nondeterministic algorithm $\mathcal{A}$ guesses D, and verifies that D is correct on the given input x in time

$$O(n^d S(n + O(\log n)) \cdot \text{poly}(\log S(n + O(\log n)))) \leq n^d \cdot S(2n) \cdot \text{poly}(\log S(2n)) \leq O(2^{2n/3}),$$

by evaluating $D(x, \cdot)$ on all possible $j = 1, \ldots, kn^d + k$, and checking that all outputs of D correspond with the relevant gates in C_x. If D does not output all the gates of C_x correctly, then $\mathcal{A}$ *rejects*.

Next, consider the problem:

Given x, an input i of $n + d \log n$ bits, and a gate index $j = 1, \ldots, kn^d + k$, output the bit value on the output wire of the jth gate when C_x is evaluated on i.

By assumption, this problem also has ACC circuits, since C_x can be constructed and evaluated on any input i in polynomial time. Let $E(x, i, j)$ be an ACC circuit of size $S(n + (n + d \log n) + d \log n + O(1)) \leq S(3n)$ and depth d' with this functionality.

Now $\mathcal{A}$ guesses E and wishes to verify its correctness on x. To do this, $\mathcal{A}$ constructs a circuit VALUE(i, j) built out of D and E, where i has $n + d \log n$ bits and $j = 1, \ldots, kn^d + k$. Intuitively, VALUE(i, j) will output 0 if and only if E produces a sensible output for the jth gate of C_x evaluated on input i.

First, VALUE(i, j) feeds j to the circuit $D(x, \cdot)$, producing gate indices j_1, j_2, and a gate type g. VALUE then computes $v_1 = E(x, i, j_1)$, $v_2 = E(x, i, j_2)$ and $v = E(x, i, j)$. (Depending on g, these j_1 and j_2 may be all-zeroes, but this does not matter to us.)

If $g =$ INPUT, then VALUE outputs 0 if and only if $j \in \{1, \ldots, n + d \log n\}$ and the jth bit of i equals v. This behavior can be implemented with an AC^0 circuit of $O(n \log S(n))$ size: an AND over all $2n$ choices of a bit from input i along with a bit v, of ORs of fan-in $\log S(n) + O(1)$.

If $g =$ NOT, then VALUE outputs 0 if and only if $v_1 = \neg v$.

If $g =$ AND, then VALUE outputs 0 if and only if $v_1 \wedge v_2 = v$.

If $g =$ OR, then VALUE outputs 0 if and only if $v_1 \vee v_2 = v$.

Note that each of the above three conditions can be implemented with a constant number of gates, given the values g, v_1, v_2, and v. It follows that VALUE can be implemented as an ACC circuit.

Since $\mathcal{A}$ has not rejected, D is correct, so we know that for all i, j, the gate types g and input connections j_1 and j_2 are correct. Therefore VALUE$(i, j) = 1$ if and only if E asserts that the output of gate j in $C_x(i)$ equals v, and E asserts the inputs to j have values v_1, v_2, but the gate type g dictates that the output of j

should be $\neg v$. It follows that VALUE is an unsatisfiable circuit if and only if E prints correct values for all gates in $C_x(i)$, over all i.

Therefore, by calling ACC circuit satisfiability on VALUE$(\cdot,\cdot)$, $\mathcal{A}$ determines whether E is correct. The algorithm $\mathcal{A}$ *rejects* if E is deemed incorrect. The circuit VALUE(i,j) has $n+2d\log n+O(1)$ inputs, depth $2d'+O(1)$, and $O(S(3n)+S(2n)\log S(2n)+n\log S(n)) \leq O(S(3n)+S(2n)n)$ size. By assumption, the assumed ACC satisfiability algorithm runs in $O(2^n/n^c)$ time for c chosen to be greater than $2d$.

After checking that E is a correct guess, the question of whether C'_x is equivalent to C_x can now be verified. (Alternatively, at this point we may simply print the circuit $E(\cdot, kn^d+k)$ as a valid circuit that is equivalent to $C_x(\cdot)$.) First note that if E is correct, then for all i, $C_x(i) = E(x,i,kn^d+k)$. Therefore it suffices to set up an ACC circuit EQUIV(i) which outputs 1 if and only if $C'_x(i) \neq E(x,i,kn^d+k)$, and determine if EQUIV is satisfiable using the algorithm for ACC CIRCUIT SAT. Since EQUIV(i) has $n+d\log n$ inputs, depth $d'+O(1)$, and size $O(S(n+O(\log n)))$, the circuit satisfiability call runs in $O(2^n/n^c)$ time by assumption. If EQUIV is satisfiable, then $\mathcal{A}$ *rejects*.

Finally, $\mathcal{A}$ prints its guessed circuit C'_x if the algorithm did not reject on any of the above steps. □

Remark 1 *The proof of the lemma does not really rely on specific properties of* ACC *at all. We only need that the underlying circuit class $\mathcal{C}$ contains* AC^0 *and is closed under composition of two circuit families. The same goes for the proof of Theorem 3.2 below.*

Remark 2 *In fact the lemma shows that, given any circuit C from a* P*-uniform family and a $\mathcal{C}$-circuit D, we can efficiently check if C is equivalent to D using nondeterminism (under the assumptions that* P *has $\mathcal{C}$-circuits and there are efficient $\mathcal{C}$-satisfiability algorithms).*

With Lemma 3.1 in hand, the proof of Theorem 3.2 closely follows the author's prior work (Theorem 3.1), except the circuit C'_x is substituted in place of C_x. Let us give the details, using the specific example of ACC in place of a generic circuit class $\mathcal{C}$.

Reminder of Theorem 3.2 *Let $S(n) \leq 2^{n/4}$. There is a $c>0$ such that, if $\mathcal{C}$-*CIRCUIT SAT *instances with at most $n+c\log n$ variables, depth $2d+O(1)$, and $O(n\,S(2n)+S(3n))$ size can be solved in $O(2^n/n^c)$ time, then* E^{NP} *does not have non-uniform $\mathcal{C}$ circuits of depth d and $S(n)$ size.*

Proof of Theorem 3.2. Suppose ACC CIRCUIT SAT instances with $n+c\log n$ variables, depth $2d+O(1)$, and $O(n\,S(2n)+S(3n))$ size can be solved in $O(2^n/n^c)$ time for a sufficiently large c. Further suppose that E^{NP} has non-uniform ACC circuits of depth d and $S(n)$ size. The goal is to show that $\mathsf{NTIME}[2^n] \subseteq \mathsf{NTIME}[o(2^n)]$, contradicting the nondeterministic time hierarchy [SFM78, Zak83].

Let $L \in \mathsf{NTIME}[2^n]$. We describe a fast nondeterministic algorithm $\mathcal{B}$ deciding L. As discussed earlier (Lemma 2.1), we may assume L has a multitape Turing machine implementation in $O(2^n)$ time, and we only need to simulate L on a RAM in $O(2^n/n^c)$ time for large enough c to obtain the contradiction.

On input x of length n, $\mathcal{B}$ first runs the nondeterministic algorithm $\mathcal{A}$ of Lemma 3.1. Using the ACC CIRCUIT SAT algorithm and the fact that P has ACC circuits, $\mathcal{A}$ runs in $O(2^n/n^c + S(3n)\cdot \mathrm{poly}(n)) \leq O(2^n/n^c)$ time, and for some computation path, $\mathcal{A}$ produces an ACC circuit C'_x of $S(n+c\log n)$ size and $n+c\log n$ inputs such that $x \in L$ if and only if C'_x is the compression of a satisfiable formula $F_{C'_x}$.

Then $\mathcal{B}$ nondeterministically guesses a $S(3n)$-size circuit W. By Fact 3.2, there exists such a W that encodes a satisfying assignment for $F_{C'_x}$ if and only if $x \in L$.

Next, $\mathcal{B}$ constructs an ACC CIRCUIT SAT instance D to verify that W is correct (just as in the proof of Theorem 3.1). The circuit D has $n+c\log n$ inputs fed to $O(n)$ copies of C'_x, so that when i is input to

D, the ith clause of the 3CNF formula $F_{C'_x}$ is printed on $O(n)$ bits of output. The $O(n)$ bits encode three variable indices along with sign bits for each variable. For the three variables, an assignment is computed for them by evaluating the indices on three copies of W. Finally, D compares the sign bits with the bits output by the copies of W, and outputs 0 iff the variable assignment encoded by W satisfies the ith clause.

Observe that D has $O(n\ S(2n) + S(3n))$ size, depth $2d + O(1)$, and $n + c \log n$ inputs. By assumption, the satisfiability of D can be determined in $O(2^n/n^c)$ time, hence $\mathcal{B}$ decides if $x \in L$ in $O(2^n/n^c)$ time. □

4 A Satisfiability Algorithm for ACC Circuits

Now we present an algorithm that determines the satisfiability of ACC circuits slightly faster than the 2^n runtime of exhaustive search. There are two components in the algorithm: a nice representation of ACC circuits, and a method for evaluating this representation quickly on all of its inputs. This method can be implemented using either fast rectangular matrix multiplication, or a dynamic programming approach.

It follows from the work of Yao [Yao90], Beigel and Tarui [BT94], and Allender and Gore [AG94] that, given any ACC circuit of size s, one can produce a $s^{O(\log^c s)}$ size SYM^+ circuit in $\text{poly}(s^{O(\log^c s)})$ time that has equivalent functionality, and very special properties. (For more background, see the Preliminaries.)

Lemma 4.1 *There is an algorithm and function $f : \mathbb{N} \to \mathbb{N}$ such that given an ACC circuit of depth d and size s, the algorithm outputs an equivalent SYM^+ circuit of $s^{O(\log^{f(d)} s)}$ size. The algorithm takes at most $s^{O(\log^{f(d)} s)}$ time.*

Furthermore, given the number of ANDs in the circuit that evaluate to 1, the symmetric function itself can be evaluated in $s^{O(\log^{f(d)} s)}$ time.

The function $f(d)$ is estimated to be no more than $2^{O(d)}$. Technically speaking, the above lemma is not explicitly proved in prior work, but Allender and Gore effectively show it: they prove that given a *uniform* ACC circuit (with an efficiently computable connection language), there is a similarly *uniform* SYM^+ circuit of the appropriate size. Their proof corresponds to an efficient, deterministic algorithm computing the transformation, and this algorithm works equally well if it is simply given any ACC circuit as input (not necessarily uniform). A proof of the lemma is included in Appendix A.

4.1 Rapid evaluation of an ACC circuit on all of its inputs

The other component of the ACC satisfiability algorithm is a method for rapidly evaluating a given SYM^+ circuit on all of its possible satisfying assignments:

Lemma 4.2 (Evaluation Lemma) *There is an algorithm that, given a SYM^+ circuit of size $s \leq 2^{.1n}$ and n inputs with a symmetric function that can be evaluated in $\text{poly}(s)$ time, runs in $(2^n + \text{poly}(s)) \cdot \text{poly}(n)$ time and prints a 2^n-bit vector V which is the truth table of the function represented by the given circuit. That is, $V[i] = 1$ iff the SYM^+ circuit outputs 1 on the ith variable assignment.*

That is, any SYM^+ circuit can be evaluated on all 2^n assignments in *polynomial amortized time per assignment*. Brute force search would take $2^n \cdot \text{poly}(s)$ time, but the algorithm manages to use roughly $2^n + \text{poly}(s)$ time instead.

Lemma 4.2 can be proved in two different ways; both are appealing for different reasons. The first proof uses a powerful primitive (fast matrix multiplication) that is common in theoretical computer science, and it is plausible that the matrix multiplication approach could be extended further. The second proof, using dynamic programming, has the benefit that it can be completely described with few technical details. For pedagogical purposes it is preferred.

Proof 1: Matrix multiplication. The first way to prove Lemma 4.2 is to use a fast rectangular matrix multiplication algorithm of Coppersmith, building on prior work of Schönhage [Sch81]. This algorithm works in the case where the "middle" dimension of the matrices is polynomially smaller than the other two. In this case, matrix multiplication can be done nearly optimally.[4]

Lemma 4.3 (Coppersmith [Cop82]) *For all sufficiently large N, multiplication of an $N \times N^{.1}$ matrix with an $N^{.1} \times N$ matrix can be done in $O(N^2 \log^2 N)$ arithmetic operations.*

More precisely, Coppersmith shows that there is a constant K such that one can multiply an $N \times N$ matrix with an $N \times N^{.1}$ matrix in $K \cdot N^2 \log^2 N$ operations with a bilinear algorithm, i.e., a depth-3 arithmetic circuit with additions at the top level, multiplications in the middle, and additions at the bottom level, where each input wire to an addition gate may also multiply the input by a scalar. From the duality of bilinear matrix multiplication algorithms [HM73], a bilinear algorithm for multiplying $N \times N$ and $N \times M$ directly implies a bilinear algorithm for multiplying $N \times M$ and $M \times N$. Furthermore, Coppersmith's algorithm is explicit, in that it can be executed on typical machine model (even a multitape TM) in $O(N^2 \cdot \text{poly}(\log N))$ time, on matrices over any field of $\text{poly}(\log N)$ elements. We give implementation details for his construction in Appendix B. For us, the relevant corollary is the following.

Corollary 4.1 *For all sufficiently large N, two 0-1 matrices of dimensions $N \times N^{.1}$ and $N^{.1} \times N$ can be multiplied over the integers in $O(N^2 \cdot \text{poly}(\log N))$ time.*

We arrive at our first proof of Lemma 4.2.

Proof of Lemma 4.2. Suppose we are given a SYM^+ circuit C'' of size $s'' \leq 2^{.1n}$. Partition the inputs of C'' into two sets A and B of size at most $n' = (n+1)/2$ each. Set up two matrices M_A and M_B of dimensions $2^{n'} \times s''$ and $s'' \times 2^{n'}$ (respectively). The rows of M_A are indexed by all possible assignments to the variables in set A, while the columns of M_A are indexed by the AND gates of C''. Similarly, the columns of M_B are indexed by variable assignments in B, while the rows of M_B are indexed by the ANDs of C''. Define:

$$M_A(i,j) = \begin{cases} 1 & \text{if the } i\text{th assignment of variables from } A \text{ does not force the } j\text{th AND to be } 0, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$M_B(j,k) = \begin{cases} 1 & \text{if the } k\text{th assignment of variables from } B \text{ does not force the } j\text{th AND to be } 0, \\ 0 & \text{otherwise.} \end{cases}$$

Note the preparation of M_A and M_B takes at most $2^{n/2} \cdot s'' \cdot \text{poly}(n) \leq O(2^{n/2+.2n})$ time.

Multiply M_A and M_B, yielding a matrix N. Note that $M_A(i,j) \cdot M_B(j,k) = 1$ iff the ith assignment in A and the kth assignment in B together set the jth AND of C'' to 1. (Given an assignment to all variables in A and B, the AND is forced to either 1 or 0.) Hence $N(i,k)$ equals the number of ANDs set to 1 by the ith assignment in A and the kth assignment in B. Therefore, C'' is satisfiable if and only if some entry of N makes the symmetric function of C'' output 1.

Since $s'' \leq 2^{.1n'}$, the fast rectangular matrix multiplication of Corollary 4.1 applies, and the multiplication of M_A and M_B can be done in $2^{2n'}\text{poly}(n)$ time.

[4] Curiously, later work on rectangular matrix multiplication from the 90's [Cop97, HP98] does not provide tight enough bounds: only $N^{2+\varepsilon}$ for all $\varepsilon > 0$, rather than $N^2 \log^2 N$. Note that a bound of $N^2 \cdot 2^{(\log N)^{o(1)}}$ already suffices for our application.

To decide whether some entry of N makes the symmetric function output 1, initialize a bit vector T of length $s'' + 1$, setting $T[i]$ to be the value of the symmetric function of C'' on each $i = 0, 1, \ldots, s''$. The construction of T takes $\text{poly}(s'')$ time, since the symmetric function can be evaluated in $\text{poly}(s'')$ time. Then for every pair $i, k \in \{1, \ldots 2^{n'}\}$, if $v[N(i,k)] = 1$ then stop and report *satisfiable*. If every pair has been examined without stopping, report *unsatisfiable*. The for-loop over all pairs can be implemented in $2^{2n'}\text{poly}(n) \leq 2^n\text{poly}(n)$ time by standard table lookup or by sorting the distinct elements of $N(i,k)$. □

Proof 2: Dynamic Programming. Since the above algorithm uses fast matrix multiplication, it is quite possibly a "galactic algorithm" (in the sense of Lipton [Lip10]) that would never be run on a physical computer, due to hidden constants. Moreover, the underlying matrix multiplication algorithm is rather technical and messy. The evaluation lemma can also be proved using simple dynamic programming, following a conversation with Andreas Björklund.

Proof of Lemma 4.2. Assume we are given a SYM^+ circuit C'' with a collection of s'' AND gates over some variables $\{x_1, \ldots, x_n\}$. Let $G_j \subseteq [n]$ be the set of variable indices that are input to the jth AND gate. Define a function $f : 2^{[n]} \to \mathbb{N}$, where $f(S)$ equals the number of $j = 1, \ldots, s''$ such that $S = G_j$. The function f can be prepared as a lookup table in $O(2^n + s'' \cdot \text{poly}(n)) \leq O(2^n)$ time, by building a table of 2^n entries which are initially zero, and for each of the s'' AND gates corresponding to a subset S, we increment the S-th entry in the table.

Now consider the function $g(T) = \sum_{S \subseteq T} f(S)$ defined on all $T \subseteq [n]$. (Typically, g is called the *zeta transform of* f.) Observe that $g(T)$ equals the number of AND gates set to 1 on the variable assignment obtained by setting $x_i = 1$ for $i \in T$, and $x_i = 0$ for $i \notin T$. Therefore the table of 2^n integers of size $O(\log s'')$ representing the function g is equivalent to the matrix N in the previous proof. Hence if we can compute g then we can evaluate C'' on all of its possible inputs.

It remains to show how to compute g efficiently. Given f, the function g can be computed in $O(2^n \cdot \text{poly}(n))$ time by a dynamic programming algorithm of Yates from 1937 (cf. [BHK09], Section 2.2). For $i = 0, \ldots, n$, define $g_i : 2^{[n]} \to \mathbb{N}$ by $g_0(T) = f(T)$, and

$$g_i(T) = \begin{cases} g_{i-1}(T) + g_{i-1}(T \setminus \{i\}) & \text{if } i \in T, \\ g_{i-1}(T) & \text{otherwise.} \end{cases}$$

It follows that each g_{i+1} can be obtained from g_i in $O(2^n\text{poly}(n))$ time. Induction shows that $g_i(T) = \sum_S f(S)$ where the sum is over all $S \subseteq T$ subject to the condition that $\{j \in S \mid j > i\} = \{j \in T \mid j > i\}$. When $i = n$, both of these sets are always empty, so it follows that $g_n = g$. □

The above description is suitable for random access machines, but the algorithm can also be implemented on a multitape Turing machine using standard ideas. (Strictly speaking, the multitape implementation is not necessary to prove ACC lower bounds, because Lemma 2.1 shows it suffices to have a fast random-access implementation of any $L \in \mathsf{NTIME}_{TM}[2^n]$. However, the extension to multitape may be useful for future work.) Details are in Appendix C.

4.2 The final algorithm

Given the evaluation lemma, the ACC satisfiability algorithm is relatively straightforward.

Theorem 4.1 *For every $d > 1$ there is an $\varepsilon \in (0,1)$ such that satisfiability of depth-d ACC circuits with n inputs and 2^{n^ε} size can be determined in $2^{n-\Omega(n^\delta)}$ time for some $\delta > \varepsilon$ that depends only on d.*

Proof. Let ℓ, ε be parameters to set later. Suppose we are given a depth-d ACC circuit C of $s = 2^{n^\varepsilon}$ size and n inputs. Make a circuit C' with $s \cdot 2^\ell$ size and $n - \ell$ inputs which is obtained by producing 2^ℓ copies of C, plugging in a different possible assignment to the first ℓ inputs of C in each copy, and taking the OR of these copies. Observe C' is a depth-$(d+1)$ ACC circuit, and C is satisfiable if and only if C' is satisfiable.

Applying the translation from ACC to SYM^+ (Lemma 4.1), a circuit C'' can be produced which is equivalent to C', where C'' consists of a symmetric gate connected to $s'' \leq s^{e(\ell^e \log^e s)}$ ANDs of variables, for some constant e that depends only on the depth d. Producing C'' from C' takes only $s^{O(\ell^e \log^e s)}$ steps. When $s = 2^{n^\varepsilon}$, $s'' \leq 2^{en^\varepsilon(\ell^e n^{\varepsilon e})}$. Set $\ell = n^{1/(2e)}$, and observe that $s'' \leq 2^{n^{2/3}}$ for all sufficiently large n and sufficiently small ε.

By the evaluation lemma (Lemma 4.2) and the fact that the symmetric function of C'' can be evaluated in $\mathrm{poly}(s'')$ time, C'' can be evaluated on all of its possible assignments in $O(2^{n-\ell} \cdot \mathrm{poly}(n)) \leq 2^{n-\Omega(n^{1/(2e)})}$ time, hence the satisfiability of C can be determined within this time. □

Two remarks. It is worth pointing out a couple more things about the algorithm. First, the algorithm can be generalized in multiple ways which may be useful in the future. Instead of taking an OR of all partial assignments to a small number of variables in C, one could instead take any constant number of ANDs and ORs of partial assignments, convert this to a SYM^+ circuit, then apply the evaluation lemma. This observation shows that any quantified Boolean formula with a constant number of quantifier blocks and a predicate described by an ACC circuit of subexponential size can also be solved faster than exhaustive search. Second, note that the algorithm does not give a faster way to solve satisfiability for the class SYM^+ itself, because in the algorithm we need that the OR of 2^ℓ circuits from the class is still a circuit in the class. Hence we cannot give lower bounds for SYM^+ at the present time.

5 ACC Lower Bounds

Combining the results of the previous two sections, non-uniform lower bounds for ACC can be proved.

Reminder of Theorem 1.2 *For every d, there is a $\delta > 0$ and a language in E^{NP} that fails to have non-uniform* ACC *circuits of depth d and size 2^{n^δ}.*

Proof. Theorem 4.1 states that for every d there is an $\varepsilon > 0$ so that satisfiability of depth-d ACC circuits with n inputs and $2^{O(n^\varepsilon)}$ size can be solved in $2^{n-\Omega(n^\delta)}$ time, for some $\delta > \varepsilon$. Theorem 3.2 says there is a $c > 0$ such that, if ACC CIRCUIT SAT instances with $n + c \log n$ variables, depth $2d + O(1)$, and at most $s = n\, 2^{O(n^\varepsilon)}$ size can be solved in $O(2^n/n^c)$ time, then E^{NP} does not have non-uniform ACC circuits of depth d and 2^{n^ε} size. The lower bound follows, as $2^{(n+c\log n)-\Omega((n+c\log n)^\delta)} \ll O(2^n/n^c)$ for every c. □

It follows that complete problems such as SMALLEST SUCCINCT 3SAT (*given a circuit C and integer i, output the ith bit of the smallest satisfying assignment to the formula F_C encoded by C*) require exponential ACC circuits. The E^{NP} lower bound can be "padded down" in a standard way to prove superpolynomial lower bounds for a class that is very close to P^{NP}.

Corollary 5.1 *For every d, $\mathsf{QuasiP}^{\mathsf{NP}} = \mathsf{TIME}[n^{\log^{O(1)} n}]^{\mathsf{NP}}$ does not have non-uniform* ACC *circuits of depth d and polynomial size.*

Proof. If there were a d such that $\mathsf{TIME}[2^{(\log n)^c}]^{\mathsf{NP}}$ had such circuits for every c, then by a padding argument (replacing n with $2^{n^{1/c}}$) it would follow that E^{NP} has depth-d size-$2^{O(n^{1/c})}$ circuits for every c, contradicting Theorem 1.2. □

Note it is known that $\mathsf{NTIME}[n^{\log^{O(1)} n}]^{\mathsf{NP}}$ does not have polynomial size (unrestricted) circuits [Kan82]. Superpolynomial ACC lower bounds for NEXP are also provable. First we need a theorem established in prior work: if NEXP has (unrestricted) polynomial size circuits, then every satisfiable formula output by the SUCCINCT 3SAT reduction in Fact 3.1 has some *satisfying assignment* that can be represented with a polynomial size unrestricted circuit.

More precisely, say that SUCCINCT 3SAT *has succinct satisfying assignments* if there is a fixed constant c such that for every language $L \in \mathsf{NTIME}[2^n]$ and every $x \in L$ of length n, there is a circuit W_x of $\mathrm{poly}(n)$ size with $k \leq n + c \log n$ inputs such that the variable assignment $z_i = W(i)$ for all $i = 1, \ldots, 2^k$ is a satisfying assignment for the formula F_{C_x}, where C_x is the circuit obtained by the SUCCINCT 3SAT reduction in Fact 3.1. Say that W_x *is a succinct satisfying assignment for* C_x.

Theorem 5.1 ([Wil10]) *Suppose* NEXP *has polynomial size circuits. Then* SUCCINCT 3SAT *has succinct satisfying assignments.*

Theorem 5.1 is not explicitly proved in the paper, however it follows immediately from another theorem. Say that NEXP has *universal witness circuits of polynomial size* if for every $L \in \mathsf{NEXP}$ and every correct exponential time verifier for L, there is a $c > 0$ such that for every $x \in L$, there is a circuit of size at most $|x|^c + c$ which encodes a witness for x that is accepted by the verifier. (For more formal definitions, see [Wil10].) The following directly implies Theorem 5.1:

Theorem 5.2 ([IKW02, Wil10]) *If* $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ *then every language in* NEXP *has universal witness circuits of polynomial size.*

The proof of Theorem 5.2 follows an argument by Impagliazzo, Kabanets, and Wigderson [IKW02]. The second ingredient is a simple folklore lemma.

Lemma 5.1 (Folklore) *Let* $\mathcal{C}$ *be any circuit class. If* P *has non-uniform* $\mathcal{C}$ *circuits of* $S(n)^{O(1)}$ *size, then there is a* $c > 0$ *such that every* $T(n)$*-size circuit family (uniform or not) has an equivalent* $S(n + O(T(n) \log T(n)))^c$*-size circuit family in* $\mathcal{C}$*.*

Proof. If P has non-uniform $S(n)^{O(1)}$-size $\mathcal{C}$ circuits, then for some $c > 0$, the CIRCUIT EVAL problem has $S(n)^c$-size circuits. (Recall the CIRCUIT EVAL problem is: *given an arbitrary Boolean circuit* C *and input* x*, evaluate* C *on* x *and output the answer.*) Let $\{D_n(\cdot, \cdot)\}$ be a $S(n)^c$-size circuit family for this problem. Now let $\{C_n\}$ be an arbitrary $T(n)$-size circuit family. To obtain an equivalent $\mathcal{C}$-circuit family $\{C'_n\}$ of $S(n + O(T(n) \log T(n)))^c$ size, define $C'_{|x|}(x) = D_{n_1}(C_{|x|}, x)$ for an appropriate length $n_1 \leq n + O(T(n) \log T(n))$. □

Note if $S(n)$ and $T(n)$ are polynomials, then $S(n + O(T(n) \log T(n)))^c$ is also polynomial.

Reminder of Theorem 1.1 $\mathsf{NTIME}[2^n]$ *does not have non-uniform* ACC *circuits of polynomial size.*

Proof. First, we claim that if $\mathsf{NTIME}[2^n]$ has polysize ACC circuits, then every language in NEXP has polysize ACC circuits. Let us sketch this implication, for completeness. If $\mathsf{NTIME}[2^n]$ has polysize ACC circuits, then the NEXP-complete problem SUCCINCT BOUNDED HALTING has polysize ACC circuits: *given a nondeterministic machine* N*, string* x*, and* t *written in binary, does* $N(x)$ *have an accepting computation path of length at most* t*?* The reduction from any $L \in \mathsf{NEXP}$ to SUCCINCT BOUNDED HALTING can be expressed with an AC^0 circuit of size $\mathrm{poly}(n, \log t)$. (Take any nondeterministic machine N with

running time 2^{n^k} that accepts L. Given an input x, the AC^0 circuit outputs the code of N as the first input of the SUCCINCT BOUNDED HALTING instance, x as the second input, and $2^{|x|^k}$ as the third input, written in binary. This only needs an AC^0 circuit that outputs 1 followed by $|x|^k - 1$ zeroes.) Hence every $L \in \mathsf{NEXP}$ can be recognized by an ACC circuit family of size n^ℓ, for some ℓ depending on L.

By Lemma 5.1 and Theorem 5.1, it follows that SUCCINCT 3SAT has succinct satisfying assignments that are polynomial size ACC circuits. We claim that a contradiction can be obtained by carefully examining the proof of Theorem 1.2 (the lower bound for E^{NP}). There, the only place requiring the full assumption "E^{NP} has non-uniform ACC circuits of size $S(n)$" is inside the proof of Theorem 3.2. In particular, the assumption is needed in Fact 3.2, where it is shown that for every satisfiable instance of SUCCINCT 3SAT, at least one of its satisfying assignments can be encoded in a size-$S(3n)$ ACC circuit. (The only other part of Theorem 3.2 where the assumption is applied is Lemma 3.1, but there it is only required that P has non-uniform ACC circuits.) But from the above, we already have that SUCCINCT 3SAT has succinct satisfying assignments which are ACC circuits.

Hence the ACC CIRCUIT SAT instance D constructed in Theorem 3.2 with the witness circuit W has size polynomial in its $n + c \log n$ inputs. Finally, the Circuit SAT algorithm of Theorem 4.1 can determine satisfiability of any $n + c \log n$ input, n^c size ACC circuit in $O(2^{n-\log^2 n})$ time, for *every* constant c. Therefore unsatisfiability of D can be determined in $O(2^n/n^c)$ time for every constant c, and the desired contradiction follows from the nondeterministic time hierarchy. □

It follows that problems complete under AC^0 reductions for NEXP such as SUCCINCT 3SAT (*given a circuit C, does it encode a satisfiable 3-CNF formula F_C?*) require superpolynomial size ACC circuits.

5.1 An Extension to "Half-Exponential" Type Bounds

The NEXP lower bounds can be extended a little by studying the proof of Theorem 5.2. However, the results are a bit ugly, so let us only sketch the arguments. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *sub-half-exponential* if for every k, $f(f(n^k)^k)^k \leq 2^{n^{o(1)}}$. The following was conjectured by Russell Impagliazzo (private communication), and can be proved by augmenting Theorem 5.2 with other known results.

Theorem 5.3 *Let $S(n)$ be any sub-half-exponential function such that $S(n) \geq n$ for all n. If $\mathsf{NTIME}[2^n]$ has $S(n)$ size circuits, then all languages in NEXP have universal witness circuits of size $O(S(S(n^c)^c)^c)$, for some c depending on the language.*

The proof goes along the lines of Theorem 5.2, but with $S(n)$ substituted in place of polynomials: we assume (a) NEXP does not have universal witness circuits of $S(S(n^c)^c)^c$ size for any c, (b) $\mathsf{NTIME}[2^n]$ does have $S(n)$ circuits, and derive a contradiction from the two. Assumption (a) implies that in time $O(2^n)$, one can nondeterministically guess and verify the truth table of a Boolean function on n bits that requires $S(S(n^c)^c)^c$ size circuits for every c, for infinitely many inputs. This is enough to partially derandomize $\mathsf{MATIME}[S(S(n^{O(1)})^{O(1)})^{O(1)}]$ for infinitely many input lengths, putting the class inside of i.o.-$\mathsf{NTIME}[2^n]/n$ [IKW02] (recall that $S(S(n^{O(1)})^{O(1)})^{O(1)} \leq 2^{n^{o(1)}}$). Assumption (b) implies that NEXP has $S(n^{O(1)})^{O(1)}$ circuits, hence $\mathsf{TIME}[2^{S(n^{O(1)})^{O(1)}}] \subseteq \mathsf{MATIME}[S(S(n^{O(1)})^{O(1)})^{O(1)}]$ [BFNW93, MVW99]. It also follows from assumption (b) that i.o.-$\mathsf{NTIME}[2^n]/n$ has $O(S(n))$ size circuits on infinitely many input lengths. Putting these containments together, it follows that $\mathsf{TIME}[2^{S(n)^{O(1)}}]$ has $O(S(n))$ size circuits on infinitely many input lengths. This is false by direct diagonalization: for all large enough n there is a function f on n variables with circuit complexity greater than $S(n)^2$, and the lexicographically first f can be found in $2^{O(S(n)^3)}$ time and simulated on a given input.

Combining Theorem 5.3 and Lemma 5.1, we immediately obtain the following implication between ACC circuits for $\mathsf{NTIME}[2^n]$ and ACC circuits which encode witnesses for NEXP.

Corollary 5.2 *If* $\mathsf{NTIME}[2^n]$ *has* $S(n)$*-size* ACC *circuits, then every language in* NEXP *has universal witness* ACC *circuits of* $S(S(S(n^c)^c)^c)^c$ *for some* c *depending on the language.*

One extra application of S comes from Theorem 5.3 which produces universal witness circuits; the other comes from Lemma 5.1 which converts those circuits to ACC. Define $f : \mathbb{N} \to \mathbb{N}$ to be *sub-third-exponential* if for every k, $f(f(f(n^k)^k)^k)^k \leq 2^{n^{o(1)}}$. Examples of sub-third-exponential functions are $f(n) = n^{\text{poly}(\log n)}$ and $f(n) = 2^{2^{\text{poly}(\log \log n)}}$.

Theorem 5.4 $\mathsf{NTIME}[2^n]$ *does not have sub-third-exponential size* ACC *circuits.*

The argument is the same as Theorem 1.1, except we apply Corollary 5.2: if $\mathsf{NTIME}[2^n]$ has such circuits, then Corollary 5.2 says that NEXP has universal witness circuits which are ACC and have subexponential size. This implies that SUCCINCT 3SAT instances have subexponential size ACC circuits that encode their satisfying assignments, which is enough to establish the contradiction in Theorem 1.1.

Theorem 5.5 *Let* $g : \mathbb{N} \to \mathbb{N}$ *have the property that there is a sub-third-exponential function* f *satisfying* $g(f(n)) \geq 2^n$*. Then* $\mathsf{NTIME}[g(n)]$ *does not have polynomial size* ACC *circuits.*

If such circuits did exist, then by padding, $\mathsf{NTIME}[2^n] \subseteq \mathsf{NTIME}[g(f(n))]$ would have ACC circuits of size $f(n)^{O(1)}$ for some sub-third-exponential f, contradicting Theorem 5.4. (Raising f to a constant power is still a sub-third-exponential function.) It follows that the polynomial size lower bound can be extended down to grotesque classes such as $\mathsf{NTIME}[2^{2^{2^{\sqrt{\log \log n}}}}] \subsetneq \mathsf{NTIME}[2^n]$, since $f(n) = 2^{(\log n)^{\log \log n}} = 2^{2^{(\log \log n)^2}}$ is sub-third-exponential, and $g(f(n)) \geq 2^n$ for functions like $g(n) = 2^{2^{2^{\sqrt{\log \log n}}}}$.

Finally, it is also straightforward to extend the lower bounds to polysize ACC circuits of slightly non-constant depth, as the ACC SAT algorithm still beats exhaustive search on polynomial size circuits of depth $o(\log \log n)$. The details can be worked out by perusing Theorem 4.1.

6 Conclusion

This paper demonstrates that the research program of proving circuit lower bounds via satisfiability algorithms is a viable one. Further work will surely improve the results. Three natural next steps are: replace ACC with TC^0 circuits in the lower bounds, or replace NEXP with EXP, or extend the exponential lower bounds from E^{NP} to NEXP.

The results of Section 3 and Lemma 5.1 show that one only has to find a very minor improvement in algorithms for TC^0 satisfiability in order to establish non-uniform TC^0 lower bounds for NEXP. The author sees no serious impediment to the existence of such an algorithm; he can only report that the algorithms tried so far do not work. The evaluation lemma for SYM^+ circuits is key to the ACC SAT algorithm, and it would be very interesting to find similar lemmas for TC^0 or NC^1. It is plausible that the characterization of NC^1 as bounded-width branching programs [Bar89] could be applied to prove an analogous evaluation lemma for Boolean formulas, which would lead to nontrivial depth lower bounds for NEXP.

It should be possible to extend the superpolynomial lower bound for ACC down to the class $\mathsf{QuasiNP} = \mathsf{NTIME}[n^{\log^{O(1)} n}]$. This paper comes fairly close to proving this result. The only step missing is a proof of the implication: "if QuasiNP has polynomial-size ACC circuits, then there are polynomial-size ACC circuits that encode witnesses to QuasiNP languages." A couple lemmas rely only on P having non-uniform ACC circuits, so they could be potentially applied in proofs of even stronger lower bounds. At any rate, the prospects for future circuit lower bounds look very promising.

Acknowledgments. I am grateful to Virginia Vassilevska Williams for reading initial drafts of this work and reassuring me that they made sense. I also thank Miki Ajtai, Andreas Björklund, Ron Fagin, and Russell Impagliazzo for helpful discussions, and Eric Allender, Sanjeev Arora, Luke Friedman, Rahul Santhanam, and Fengming Wang for helpful comments on earlier drafts. I especially thank Andreas for suggesting the zeta transform as an alternative proof for Lemma 4.2.

References

[AAD00] M. Agrawal, E. Allender, and S. Datta. On TC0, AC0, and arithmetic circuits. *J. Computer and System Sciences* 60(2):395–421, 2000.

[Ajt83] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic* 24:1–148, 1983.

[All99] E. Allender. The permanent requires large uniform threshold circuits. *Chicago J. Theor. Comput. Sci.*, 1999.

[AB87] N. Alon and R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica* 7(1):1–22, 1987.

[AB09] S. Arora and B. Barak. *Computational Complexity – a modern approach*. Cambridge University Press, 2009.

[AG94] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM J. Comput.* 23(5):1026–1049, 1994.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential simulations unless EXPTIME has publishable proofs. *Computational Complexity* 3:307–318, 1993.

[Bar89] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. *J. Computer and System Sciences* 38:150–164, 1989. See also STOC'86.

[BT88] D. A. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC1. *JACM* 35:941–952, 1988.

[BS95] D. A. Mix Barrington and H. Straubing. Superlinear lower bounds for bounded-width branching programs. *J. Computer and System Sciences* 50:374–381, 1995.

[BST90] D. A. Mix Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation* 89:109–132, 1990.

[BT94] R. Beigel and J. Tarui. On ACC. *J. Computational Complexity* 4:350–366, 1994. See also FOCS'91.

[BHK09] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.* 39(2):546–563, 2009.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing separations. In *Proc. IEEE Conference on Computational Complexity*, 8–12, 1998.

[CIP06] C. Calabro, R. Impagliazzo, and R. Paturi. A duality between clause width and clause density for SAT. In *Proc. IEEE Conf. on Computational Complexity*, 252–260, 2006.

[CIP09] C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *Proc. International Workshop on Parameterized and Exact Computation*, 2009.

[CKY89] J. F. Canny, E. Kaltofen, and L. Yagati. Solving systems of non-linear equations faster. *Proc. ACM-SIGSAM International Symposium on Symbolic and Algebraic computation*, 121–128, 1989.

[Cau96] H. Caussinus. A note on a theorem of Barrington, Straubing, and Thérien. *Information Processing Letters* 58(1):31–33, 1996.

[CGPT06] A. Chattopadhyay, N. Goyal, P. Pudlák, and D. Thérien. Lower bounds for circuits with MODm gates. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 709–718, 2006.

[CW09] A. Chattopadhyay and A. Wigderson. Linear systems over composite moduli. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 43–52, 2009.

[Coo88] S. A. Cook. Short propositional formulas represent nondeterministic computations. *Inf. Process. Lett.* 26(5):269–270, 1988.

[Cop82] D. Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.* 11(3):467–471, 1982.

[Cop97] D. Coppersmith. Rectangular matrix multiplication revisited. *J. Complexity* 13(1):42–49, 1997.

[DH08] E. Dantsin and E. A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren and T. Walsh (eds.), 341–362, 2008.

[FLvMV05] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *JACM* 52(6):835–865, 2005.

[FSS81] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *Mathematical Systems Theory* 17:13–27, 1984. Also in FOCS'81.

[GO94] I. Gohberg and V. Olshevsky. Fast algorithms with preprocessing for matrix-vector multiplication problems. *J. Complexity* 10(4):411–427, 1994.

[GS89] Y. Gurevich and S. Shelah. Nearly linear time. *Logic at Botik '89*, Springer-Verlag LNCS 363, 108–118, 1989.

[Gree95] F. Green. Lower bounds for depth-three circuits with equals and mod-gates. In *Proceedings of STACS*, Springer LNCS 900:71–82, 1995.

[GKRST95] F. Green, J. Köbler, K. W. Regan, T. Schwentick, and J. Torán. The power of the middle bit of a #P function. *J. Computer and System Sciences* 50(3):456–467, 1995.

[Gro98] V. Grolmusz. A lower bound for depth-3 circuits with MOD m gates. *Information Processing Letters* 67(2):87–90, 1998.

[GT00] V. Grolmusz and G. Tardos. Lower bounds for (MODp-MODm) circuits. *SIAM J. Comput.* 29(4):1209–1222, 2000.

[Han06] K. A. Hansen. Constant width planar computation characterizes ACC0. *Theory of Computing Systems* 39(1):79–92, 2006.

[Hås86] J. Håstad. Almost optimal lower bounds for small depth circuits. *Advances in Computing Research* 5:143–170, 1989. See also STOC'86.

[HM73] J. Hopcroft and J. Musinski. Duality applied to the complexity of matrix multiplication and other bilinear forms. *SIAM Journal on Computing* 2(3):159–1973, 1973.

[HP98] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity* 14(2):257–299, 1998.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time versus probabilistic polynomial time. *J. Computer and System Sciences* 65(4):672–694, 2002.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity* 13(1-2):1–46, 2004.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control* 55(1-3):40–56, 1982.

[KVVY93] R. Kannan, H. Venkateswaran, V. Vinay, A. C.-C. Yao. A circuit-based proof of Toda's theorem. *Information and Computation* 104(2):271–276, 1993.

[KvM99] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Computing* 31(5):1501–1526, 2002. Also in STOC'99.

[KH09] M. Koucky and K. A. Hansen. A new characterization of ACC0 and probabilistic CC0. *Computational Complexity* 19(2):211–234, 2010.

[KP94] M. Krause and P. Pudlák. On the computational power of depth 2 circuits with threshold and modulo gates. In *ACM Symposium on Theory of Computing*, 48–57, 1994.

[Lip10] R. J. Lipton. Galactic Algorithms. *Gödel's Lost Letter and P $\neq$ NP*, 2010. http://rjlipton.wordpress.com/2010/10/23/galactic-algorithms/

[MVW99] P. B. Miltersen, N. V. Vinodchandran, and O. Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. *Proc. COCOON*, Springer LNCS 1627:210–220, 1999.

[Pan84] V. Y. Pan. How to multiply matrices faster. *Springer-Verlag Lecture Notes in Computer Science* 179, 1984.

[PY86] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control* 71:181–185, 1986.

[PR81] W. J. Paul and R. Reischuk. On Time versus Space II. *J. Comput. Syst. Sci.* 22(3):312–327, 1981.

[Raz85] A. A. Razborov. Lower bounds on the monotone complexity of some boolean functions. *Doklady Akademii Nauk SSSR* 281(4):798–801, 1985. Translated in *Soviet Math. Dokl.* 31(2):354–357.

[Raz87] A. A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Matematicheskie Zametki* 41(4):598–607, 1987. Translation in *Mathematical Notes of Academy of Sciences USSR* 41(4):333–338, 1987.

[Raz89] A. A. Razborov. On the method of approximations. *Proc. ACM Symposium on Theory of Computing*, 169–176, 1989.

[Rob91] J. M. Robson. An $O(T \log T)$ Reduction from RAM Computations to Satisfiability. *Theor. Comput. Sci.* 82(1):141–149, 1991.

[San10] R. Santhanam. Fighting perebor: new and improved algorithms for formula and QBF satisfiability. *Proc. IEEE Symposium on Foundations of Computer Science*, to appear, 2010.

[Sch78] C.-P. Schnorr. Satisfiability is quasilinear complete in NQL. *J. ACM* 25(1):136–145, 1978.

[Sch81] A. Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.* 10(3):434–455, 1981.

[SFM78] J. Seiferas, M. J. Fischer, and A. Meyer. Separating nondeterministic time complexity Classes. *JACM* 25:146–167, 1978.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. *Proc. ACM Symposium on Theory of Computing*, 77–82, 1987.

[Thé94] D. Thérien. Circuits constructed with MODq gates cannot compute AND in sublinear size. *Computational Complexity* 4:383–388, 1994.

[Tou01] I. Tourlakis. Time-space tradeoffs for SAT on nonuniform machines. *J. Computer and System Sciences* 63(2):268–287, 2001.

[VV86] L. G. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.* 47(3):85–93, 1986.

[Wil10] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proc. ACM Symposium on Theory of Computing*, 231–240, 2010.

[Yao85] A. C.-C. Yao. Separating the polynomial-time hierarchy by oracles. In *Proc. IEEE Symposium on the Foundations of Computer Science*, 1–10, 1985.

[Yao90] A. C.-C. Yao. On ACC and threshold circuits. In *Proc. IEEE Symposium on Foundations of Computer Science*, 619–627, 1990.

[YP94] P. Y. Yan and I. Parberry. Exponential size lower bounds for some depth three circuits. *Information and Computation* 112:117–130, 1994.

[Zak83] S. Zak. A Turing machine hierarchy. *Theoretical Computer Science* 26:327–333, 1983.

A Appendix: Proof of Lemma 4.1

Reminder of Lemma 4.1 *There is an algorithm and function $f : \mathbb{N} \to \mathbb{N}$ such that given an* ACC *circuit of depth $d = O(1)$ and size s, the algorithm outputs an equivalent* SYM^+ *circuit of $s^{O(\log^{f(d)} s)}$ size. The algorithm takes at most $s^{O(\log^{f(d)} s)}$ time.*

Furthermore, given the number of ANDs in the circuit that evaluate to 1*, the symmetric function itself can be evaluated in $s^{O(\log^{f(d)} s)}$ time.*

There is absolutely nothing new in the proof below. The algorithm is described as a series of $s^{O(\log^{f(d)} s)}$ time transformations, closely following Allender and Gore [AG94] in the appropriate places. We just need to point out that the relevant transformations are still efficiently computable when the algorithm is given an arbitrary input circuit.

Proof. Let C be the given circuit. Note that since C has depth $d = O(1)$ and will remain constant depth throughout the algorithm, we may always assume at any point in the algorithm that C is a tree (i.e., all gates have fan-out 1).

Transformation 1. Let s be the size of C. We transform C into a probabilistic circuit C' that has $\text{poly}(\log s)$ probabilistic inputs, such that C' has constant depth, $s^{O(1)}$ size, C' has no OR or MOD_m gates for any composite m, and C' has AND gates of fan-in at most $\text{poly}(\log s)$. (The circuit C' is said to accept an input x if it outputs 1 on the majority of settings to the probabilistic inputs.)

First, note that one can replace NOT gates by MOD_m gates (for any m), and one can replace the AND and OR gates by fixed-depth probabilistic circuits with having only MOD_m gates and AND gates of $\text{poly}(\log s)$ fan-in. In fact, all of the AND and OR gates can share the *same* set of $\text{poly}(\log s)$ probabilistic inputs. This is a standard trick that goes back to Valiant and Vazirani [VV86] (and an alternative proof of Toda's theorem [KVVY93]) that can be performed in $s^{O(1)}$ time.

The MOD_m gates for composite m are eliminated as follows. Let $p_1^{e_1} \cdots p_k^{e_k}$ be the factorization of m. Since m divides a number x iff $p_i^{e_i}$ divides x for all i, every MOD_m gate can be replaced by an AND of $\text{MOD}_{p_i^{e_i}}$ gates. (Also, since m is assumed to be a fixed constant independent of the size of C, the factorization of m can be computed in $O(1)$ time!) Observe that p^e divides a number x iff for all $i = 0, \ldots, e-1$, p divides $\binom{x}{p^i}$. Using this fact, a MOD_{p^e} gate can be replaced with an constant-fan-in AND of MOD_p gates of constant-fan-in ANDs, as follows. A MOD_{p^e} gate with t inputs is replaced with an AND of fan-in e, where the inputs are MOD_p gates. For all $i = 0, \ldots, e-1$, the ith MOD_p gate has fan-in $\binom{t}{p^i}$, one for every subset of the t inputs that has cardinality p^i. For all $j = 1, \ldots, \binom{t}{p^i}$, the ith MOD_p gate has its jth input connected to an AND of the p^i-subset of t inputs corresponding to integer j. All of this can be computed within $t^{O(p^e)}$ time, and hence $s^{O(1)}$ time.

Transformation 2. We have a probabilistic circuit C' with $\text{poly}(\log s)$ probabilistic inputs, constant depth, $s^{O(1)}$ size, no OR or MOD_m gates for any composite m, and AND gates of fan-in at most $\text{poly}(\log s)$. Now we produce a C'' with no probabilistic inputs and all of the above properties except that the output gate is now a MAJORITY gate (which outputs the majority value of its inputs). This is easy to do, by enumerating through all possible values of the $\text{poly}(\log s)$ inputs, making a new copy of C' for every valuation, and taking the MAJORITY of all these copies. Certainly the new circuit C'' has size $s^{O(\text{poly}(\log s))}$ and the transformation can be performed in this much time.

Transformation 3. Now we have a circuit C'' has size $s^{O(\text{poly}(\log s))}$ size, a MAJORITY gate at the output, no OR or MOD_m gates for any composite m, and AND gates of fan-in at most $\text{poly}(\log s)$. We produce another constant-depth C''' where all these polylog fan-in AND gates are at the bottom: no MOD_p gates are below them in C'''.

Take any AND gate g with $f = \text{poly}(\log s)$ fan-in. Without loss of generality, g has MOD_p gates $h_1, \ldots, h_f$ as input for some fixed prime p, by inserting "dummy" MOD_p gates in the appropriate places, and all MOD_p gates have the same fan-in $f' \leq s^{O(\text{poly}(\log s))}$, by inserting "dummy" zeroes in the inputs. We want to show that this AND of MOD_p gates can be rewritten as a MOD_p of ANDs.

Let x_{ij} represent the jth input to the MOD_p gate h_i. Allender and Gore [AG94] show that this AND of MOD_p can be rewritten as:

$$\bigwedge_{i=1}^{f} \left[\!\!\left[\sum_j x_{ij} \equiv 0 \bmod p \right]\!\!\right] = \sum_{k=1}^{f} (p-1)^{k-1} \sum_{\{i_1,\ldots,i_k\}\subseteq[f]} \sum_{\substack{\langle j_{1,1},\ldots,j_{1,p-1}\rangle\in[f']^{p-1} \\ \cdots \\ \langle j_{k,1},\ldots,j_{k,p-1}\rangle\in[f']^{p-1}}} \prod_{t=1}^{k} \prod_{\ell=1}^{p-1} x_{i_t\, j_{i_t,\ell}} \bmod p, \quad (1)$$

where $[\![P]\!] = 1$ if the predicate P is true, and 0 otherwise.

The right-hand side can be represented with a MOD_p gate with fan-in at most

$$O\left(\sum_{k=1}^{f} \binom{f}{k} (f')^{k(p-1)}\right) \leq s^{O(\text{poly}(\log s))},$$

which is connected to ANDs of fan-in at most $f(p-1)$. The transformation takes $s^{O(\text{poly}(\log s))}$ time.

Transformation 4. We have a C''' of $s^{O(\text{poly}(\log s))}$ size with AND gates of polylog fan-in connected to the inputs, a MAJORITY at the output, and MOD_{p_i} gates in between, where p_i is a prime dividing $m = O(1)$. We now show how to express C''' as a symmetric function of $s^{O(\text{poly}(\log s))}$ AND gates, completing the proof.

To do this, we prove that if you have a circuit D which has a symmetric function at the output, ANDs at the bottom, and depth-d subcircuits of MOD_{p_i}'s in between, then this can be turned into an equivalent D' with quasi-polynomial size, a symmetric function at the top, ANDs at the bottom, and depth-$(d-1)$ subcircuits of MOD_{p_i}'s. That is, the topmost layer of MOD_{p_i}'s can be "consumed" by choosing a different symmetric function.

We may assume without loss of generality that all f gates with input to the symmetric function $F : [f] \to \{0,1\}$ are MOD_p gates, for a fixed prime p, and all of the MOD_p gates have the same fan-in f' (by adding dummy wires and gates where necessary). Let x_{ij} be the jth input to the ith MOD_p gate. Note that the function we want to simulate is $H(x_{1,1},\ldots,x_{f,f'}) = F(\sum_{i=1}^{f} \text{MOD}_p(x_{i,1},\ldots,x_{i,f'}))$. We will replace H with a symmetric function F' of ANDs of polylogarithmic fan-in. Then, applying Transformation 3 to these ANDs, the resulting circuit can be converted into one which has the ANDs at the bottom and only a quasi-polynomial increase in size.

Define

$$G(x_{1,1},\ldots,x_{f,f'}) := F\left(\sum_{i=1}^{f} \text{MOD}_p(x_{i,1},\ldots,x_{i,f'}) \bmod p^k\right)$$

where k is the smallest integer exceeding $\log_p f$. Then $p^k > f$, so it is clear that $G(x_{1,1},\ldots,x_{f,f'}) = H(x_{1,1},\ldots,x_{f,f'})$ when all $x_{i,j}$ are in $\{0,1\}$. We shall show how to implement G as a symmetric function F' of ANDs.

We use the *modulus amplifying polynomials* of Beigel and Tarui [BT94]. Define

$$P_k(x) = (-1)^k (x-1)^k \left(\sum_{i=0}^{k-1} \binom{k+i-1}{i} x^i\right) + 1.$$

This polynomial has the property that for all $x \geq 0$ and $p \geq 1$,

$$x = 0 \bmod p \Longrightarrow P_k(x) = 0 \bmod p^k,$$

$$x = 1 \bmod p \Longrightarrow P_k(x) = 1 \bmod p^k.$$

Defining $Q_k(x) = 1 - P_k(x^{p-1})$ and appealing to Fermat's little theorem, it follows that $Q_k(x) = 1 \bmod p^k$ if p divides x, and is equal to $0 \bmod p^k$ otherwise. Therefore $Q_k(\sum_{i=1}^{f'} y_i) = \text{MOD}_p(y_1, \ldots, y_{f'}) \bmod p^k$, and

$$G(x_{1,1}, \ldots, x_{f,f'}) = F\left(\sum_{i=1}^{f} Q_k\left(\sum_{j=1}^{f'} x_{i,j}\right) \bmod p^k\right).$$

Note that each $Q_k(\sum_{j=1}^{f'} x_{i,j})$ is a symmetric multivariate polynomial of degree at most $k(p-1)$. Hence Q_k can be expanded into a sum of at most $(f \cdot f')^{O(k(p-1))} \leq s^{O(\text{poly}(\log s))}$ terms. Each term is a product of $\text{poly}(\log s)$ variables and a coefficient c that is represented in $O(k \log k) \leq \text{poly}(\log s)$ bits and easily computed. The product of variables can be directly represented by an AND. Multiplication by the coefficient c can be simulated by taking the sum of c copies of the relevant monomials (ANDs).

Therefore the sum of all f of these sums of monomials can be efficiently expressed as a single sum modulo p^k of $s^{O(\text{poly}(\log s))}$ AND gates, where each AND has fan-in $k(p-1) \leq \text{poly}(\log s)$. Finally, we take the symmetric function F' to be: compute the sum v of the outputs of all the AND gates created, then output $F(v \bmod p^k)$. Observe that a symmetric function composed with a sum modulo p^k is still a symmetric function.

In summary, for any constant depth circuit, all the above transformations take at most quasi-polynomial time, increase the circuit size by only a quasi-polynomial amount, and the transformations are applied at most a quasi-polynomial number of times. (Transformation 4 is applied a constant number of times.) Moreover, the symmetric function generated at the end of the process takes no more time to evaluate than the time it takes to build the SYM^+ circuit. In more detail, the final symmetric function has the form

$$F(v) = \text{MAJORITY}((\cdots((v \bmod p_1^{k_1}) \bmod p_2^{k_2}) \cdots \bmod p_{d'}^{k_{d'}}),$$

for some constant d' that depends on the depth d and initial modulus m (both are constants). Here, MAJORITY outputs the high-order bit of its input, and each $p_i^{k_i}$ is at most a constant factor larger than the size of the final circuit. □

B Appendix: Coppersmith's algorithm

Recall we are studying the following algorithm of Coppersmith:

Lemma B.1 (Coppersmith [Cop82]) *For all sufficiently large N, multiplication of an $N \times N^{.1}$ matrix with an $N^{.1} \times N$ matrix can be done in $O(N^2 \log^2 N)$ arithmetic operations.*

Prima facie, it could be that Coppersmith's algorithm is non-uniform, making it difficult to apply. For the sake of completeness, here we verify using standard ideas that Coppersmith's algorithm can indeed be implemented to run (even on a multitape TM) in $O(N^2 \cdot \text{poly}(\log N))$ time, on matrices over any field of $\text{poly}(N)$ elements. (As we work with 0-1 matrices A'' and B'' in our application, it suffices for us to work over a prime field of $\text{poly}(N)$ elements.) We focus on the implementation details of his algorithm, without going very far into its correctness. The algorithm relies on some of the older tools from the matrix multiplication literature. More background on these tools can be found in the highly readable reference [Pan84].

Coppersmith's algorithm follows a paradigm introduced by Schönhage [Sch81]. For example, suppose we wish to multiply two matrices A'' and B''. First we *preprocess* A'' and B'' in some efficient way; in our

first example, we devise highly structured matrices A, A', B, B' so that $A'' \cdot B'' = A' \cdot A \cdot B \cdot B'$. The matrices A and B are sparse "partial" matrices with particular structure in their nonzeroes, and A' and B' are explicit matrices of scalar constants which are independent of A'' and B''. Next, we recursively apply a constant-sized matrix multiplication algorithm to multiply A and B essentially optimally. (Recall that Strassen's algorithm has an analogous form; such algorithms are known to be efficiently implementable on a multitape TM.) Finally, we *postprocess* the resulting product C to obtain our desired product $A'' \cdot B''$; in the first example, this means computing $A' \cdot C \cdot B'$. Using the explicit structure of A' and B', these matrix products are also done nearly optimally. Our aim is to verify that each step of this process can be efficiently computed, for Coppersmith's full matrix multiplication algorithm.

Coppersmith begins with A'' of dimensions $2^{4M/5} \times \binom{M}{4M/5}2^{4M/5}$ and B'' of dimensions $\binom{M}{4M/5}2^{4M/5} \times 2^{M/5}$ where $M \approx \log N$, and obtains an $O(5^M \text{poly}(M))$ algorithm for their multiplication. Later, he symmetrizes the construction to get the algorithm for the desired dimensions. In this first construction, the structured matrices A' and B' have dimensions $2^{4M/5} \times 2^M$ and $2^M \times 2^{M/5}$, respectively. Coppersmith needs that all $2^{4M/5} \times 2^{4M/5}$ submatrices of A' and $2^{M/5} \times 2^{M/5}$ submatrices of B' are non-singular. Following Schönhage, this can be accomplished by picking A' and B' to be rectangular Vandermonde matrices. More precisely, the i, j entry of A' is $(\alpha_j)^{i-1}$, where $\alpha_1, \alpha_2, \ldots$ are distinct elements of the field; B' is defined analogously. Such matrices have the additional advantages that they can be succinctly described (with 2^M field elements), and linear algebra with them can be done very efficiently, as described below.

The matrices A and B have dimensions $2^M \times 3^M$ and $3^M \times 2^M$, respectively. Although these dimensions are large, the matrices are stored in a sparse representation, and they have structure in their nonzeroes. In more detail, A has only $O(5^M)$ nonzeroes, B has only $O(4^M)$ nonzeroes, and there is an optimal algorithm for multiplying 2×3 (with 5 nonzeroes) and 3×2 matrices (with 4 nonzeroes) that can be recursively applied to multiply A and B optimally, in $O(5^M \cdot \text{poly}(M))$ operations. (In particular, the 2×3 and 3×2 matrix multiplication is an "approximate" algorithm, which can be recursively applied to larger matrices using $O(M)$-degree univariate polynomials over the field; operations on such polynomials increase the overall time by only a $\text{poly}(M)$ factor.) These A and B are constructed by multiplying each of the $\binom{M}{4M/5}2^{4M/5}$ columns in A'' and $\binom{M}{4M/5}2^{4M/5}$ rows in B'' by inverses of Vandermonde matrices and their transposes (the inverses of appropriate $2^{4M/5} \times 2^{4M/5}$ submatrices of A' and $2^{M/5} \times 2^{M/5}$ submatrices of B', respectively). Due to the structure of inverse Vandermonde matrices and their transposes, $n \times n$ matrices of this form can be multiplied with n-vectors in $O(n \cdot \text{poly}(\log n))$ operations with explicit algorithms (for references, cf. [CKY89, GO94]).[5] Hence the inverse of a submatrix of A' can be multiplied with an arbitrary vector in $O(2^{4M/5} \cdot \text{poly}(M))$ operations. It follows that constructing A and B takes only $O(\binom{M}{4M/5}2^{4M/5} \cdot 2^{4M/5} \cdot \text{poly}(M))$ time. Since $5^M \approx \binom{M}{4M/5}4^{4M/5}$ (within $\text{poly}(M)$ factors), this quantity is $O(5^M \cdot \text{poly}(M))$.

By construction (using an efficient correspondence between columns of A'' and columns of A' with $2^{4M/5}$ nonzeroes), we have $A'' \cdot B'' = A' \cdot (A \cdot B) \cdot B'$. After A and B are constructed, the constant-sized algorithm for 2×3 and 3×2 mentioned above can be applied in the usual recursive way to multiply the sparse A and B in $O(5^M \cdot \text{poly}(M))$ time; call this matrix Z. Then using the Vandermonde structure of A' and B', the product $Z' = A' \cdot Z$ can be done in $o(5^M \cdot \text{poly}(M))$ operations, and the final product $Z' \cdot B'$ can be done in $o(5^M \cdot \text{poly}(M))$ operations. All in all, we have an algorithm for multiplying matrices of dimensions $2^{4M/5} \times \binom{M}{4M/5}2^{4M/5}$ and $\binom{M}{4M/5}2^{4M/5} \times 2^{M/5}$ that is explicit and uses $O(5^M \cdot \text{poly}(M))$

[5] In general, operations on Vandermonde matrices, their transposes, their inverses, and the transposes of inverses can be reduced to fast multipoint computations on univariate polynomials. For example, multiplying an $n \times n$ Vandermonde matrix with a vector is equivalent to evaluating a polynomial (with coefficients given by the vector) on the n elements that comprise the Vandermonde matrix, which takes $O(n \log n)$ operations. This translates to $O(n \cdot \text{poly}(\log n))$ time on multitape TMs over small fields.

operations. Call this ALGORITHM 1. Observe ALGORITHM 1 also works when the entries of A'' and B'' are themselves matrices over the field. (The running time will surely increase in proportion to the sizes of the underlying matrices, but the bound on the number of *operations on the entries* remains the same.)

We can extract more algorithms from the above construction by exploiting the symmetries of bilinear algorithms. The underlying 2×3 and 3×2 matrix multiplication algorithm with 5 products is a bilinear algorithm, meaning that it can be expressed in the so-called trilinear form

$$\sum_{ijk} A_{ik}B_{kj}C_{ji} + p(x) = \sum_{\ell=1}^{5}(\sum_{ij} \alpha_{ij}A_{ij}) \cdot (\sum_{ij} \beta_{ij}B_{ij}) \cdot (\sum_{ij} \gamma_{ij}C_{ij}) \tag{2}$$

where α_{ij}, β_{ij}, and γ_{ij} are constant-degree polynomials in x over the field, and $p(x)$ is a polynomial with constant coefficient 0. Such an algorithm can be converted into one with no polynomials and minimal extra overhead (as described in Coppersmith's paper). Typically one thinks of A_{ik} and B_{kj} as entries in the input matrices, and C_{ji} as indeterminates, so the LHS of (2) corresponds to a polynomial whose C_{ji} coefficient is the ij entry of the matrix product. Note the *transpose* of the third matrix C corresponds to the final matrix product. The RHS corresponds to the special matrix multiplication algorithm with only 5 products. For example, Strassen's famous 7-multiplication algorithm can be expressed in the form of (2) as follows:

$$\begin{aligned}\sum_{i,j,k=0,1} A_{ik}B_{kj}C_{ji} \quad = \quad & (A_{00}+A_{11})(B_{00}+B_{11})(C_{00}+C_{11}) \\ & +(A_{10}+A_{11})B_{00}(C_{01}-C_{11}) + A_{00}(B_{01}-B_{11})(C_{10}+C_{11}) \\ & +(A_{10}-A_{00})(B_{00}+B_{01})C_{11} + (A_{00}+A_{01})B_{11}(C_{10}-C_{00}) \\ & +A_{11}(B_{10}-B_{00})(C_{00}+C_{01}) + (A_{01}-A_{11})(B_{10}+B_{11})C_{00}.\end{aligned} \tag{3}$$

The LHS of (2) and (3) represents the trace of the product of three matrices A, B, and C (where the ij entry of matrix X is X_{ij}). It is well known that every bilinear algorithm naturally expresses multiple algorithms through this trace representation. Since

$$tr(ABC) = tr(BCA) = tr(CAB) = tr((ABC)^T) = tr((BCA)^T) = tr((CAB)^T),$$

if we think of A as a symbolic matrix and consider (2), we obtain a new algorithm for computing a matrix A when given B and C. Similarly, we get an algorithm for computing a B when given A and C, and analogous statements hold for computing A^T, B^T, and C^T. So the aforementioned algorithm for multiplying a sparse 2×3 and sparse 3×2 yields several other algorithms. In particular (the case of computing B^T from A^T and C) we obtain an algorithm for computing 4 entries in a 3×2 matrix which is the product of a 3×2 matrix (with 5 nonzeroes) and a 2×2 matrix.

Using the identity $tr(ABC) = tr((BCA)^T) = tr(A^TC^TB^T)$, we can treat B^T as symbolic and let A^T and C^T correspond to input matrices in (2). Applying the resulting algorithm recursively, a very similar preprocessing and postprocessing can be used to multiply $\binom{M}{4M/5}2^{4M/5} \times 2^{4M/5}$ and $2^{4M/5} \times 2^{M/5}$ matrices using an algorithm that runs in $O(5^M \cdot \text{poly}(M))$ time over a small field.

In more detail, recall in ALGORITHM 1 the matrices A'' and B'' were decomposed to satisfy $A'' \cdot B'' = A' \cdot A \cdot B \cdot B'$. The trace identity tells us

$$tr(A''B'' \cdot C) = tr(A'A \cdot BB' \cdot C) = tr(B \cdot B'CA' \cdot A) = tr(A^T \cdot (A')^T C^T (B')^T \cdot B^T).$$

This suggests the following algorithm for multiplying $\binom{M}{4M/5}2^{4M/5} \times 2^{4M/5}$ and $2^{4M/5} \times 2^{M/5}$ matrices. Given A'' and C'' of the appropriate dimensions, preprocess C'' into the $2^M \times 2^M$ matrix $D = B' \cdot (C'')^T \cdot A'$,

and use A' as before to preprocess A'' into a sparse $3^M \times 2^M$ matrix A^T having $\binom{M}{4M/5}4^{4M/5} \approx 5^M$ nonzeroes. Both steps can be done efficiently using the Vandermonde structure of A' and B'. Next, multiply A^T and D^T, following the bilinear algorithm for computing a 3×2 (with 4 nonzeroes) from a 3×2 (with 5 nonzeroes) and a 2×2, in $O(5^M \text{poly}(M))$ time. The multiplication results in a $3^M \times 2^M$ matrix B with $O(4^M)$ nonzeroes, which can be efficiently transformed to the output matrix using the inverses of submatrices of B'. (This postprocessing step is analogous to the preprocessing of B'' in ALGORITHM 1.) Notice we have analogous preprocessing, multiplication, and postprocessing steps, albeit the steps are "out of order" from before. (Before, multiplication of the result matrix C by A' and B' occurred in postprocessing; now it occurs in preprocessing, as C is now part of the input.) Call this construction ALGORITHM 2.

Next, we may "tensorize" the two algorithms in a standard way. This consists of dividing the input matrices into blocks, executing ALGORITHM 1 on the blocks themselves, and calling ALGORITHM 2 when the product of two blocks is needed. As both of these algorithms are explicit and efficient, their "tensorization" is also explicit and efficient. ALGORITHM 1 multiplies $2^{4M/5} \times \binom{M}{4M/5}2^{4M/5}$ and $\binom{M}{4M/5}2^{4M/5} \times 2^{M/5}$ matrices, and ALGORITHM 2 multiplies $\binom{M}{4M/5}2^{4M/5} \times 2^{4M/5}$ and $2^{4M/5} \times 2^{M/5}$. Hence their tensorization multiplies matrices of dimensions

$$\left(2^{4M/5} \cdot \binom{M}{4M/5}2^{4M/5}\right) \times \left(\binom{M}{4M/5}2^{4M/5} \cdot 2^{4M/5}\right) \text{ and } \left(\binom{M}{4M/5}2^{4M/5} \cdot 2^{4M/5}\right) \times \left(2^{M/5} \cdot 2^{M/5}\right),$$

and the algorithm runs in $O(5^{2M} \cdot \text{poly}(M))$ time. Since $\binom{M}{4M/5}4^{4M/5} \approx 5^M$, this means we are multiplying $5^M \times 5^M$ and $5^M \times 2^{2M/5}$ in $O(5^{2M}\text{poly}(M))$ time. Call this ALGORITHM 3. This is the algorithm obtained by Coppersmith.

Finally, using the symmetry of ALGORITHM 3 itself, we can obtain an algorithm for multiplying a $5^M \times 2^{2M/5}$ matrix with a $2^{2M/5} \times 5^M$ matrix in $O(5^{2M}\text{poly}(M))$ time. ALGORITHM 3 is also a bilinear algorithm that can be interpreted as an efficient way to compute $tr(ABC)$ where A is $5^M \times 5^M$, B is $5^M \times 2^{2M/5}$, and C is $2^{2M/5} \times 5^M$. In the above version of ALGORITHM 3, we have treated A and B as input, and C as symbolic. Treating B and C as input yields an algorithm for multiplying $5^M \times 2^{2M/5}$ and $2^{2M/5} \times 5^M$ in $O(5^{2M}\text{poly}(M))$ time. This algorithm also has a preprocessing step, a product of partial matrices, then a postprocessing step, which involve multiplications with Vandermonde-style matrices, their transposes, their inverses, and their inverse transposes. The important point is that this transformation does not fundamentally change the algorithm: just as ALGORITHM 2 is a "reordering" of ALGORITHM 1, this transformation of ALGORITHM 3 only reorganizes these efficiently computable operations. It follows the final algorithm will also be efficiently computable. (Of course, it is possible in principle to describe this algorithm directly as a preprocessing-multiplication-postprocessing procedure, but it is quite messy.) Let $N = 5^M$. We have arrived at the following.

Corollary B.1 *For all sufficiently large N, two 0-1 matrices of dimensions $N \times N^{.1}$ and $N^{.1} \times N$ can be multiplied over the integers in $O(N^2 \cdot \text{poly}(\log N))$ time.*

C Appendix: The dynamic programming algorithm on a multitape TM

Here we show how Proof 2 of the evaluation lemma (Lemma 4.2) can be implemented efficiently on a multitape Turing machine. The main idea is to use efficient sorting on a multitape TM:

Lemma C.1 (Schnorr [Sch78]) *Any list of ℓ items can be sorted in $O(\ell \cdot \text{poly}(\log \ell))$ time on a multitape Turing machine, provided that a comparison of two items can be done in $O(\text{poly}(\log \ell))$ time.*

For example, the Merge Sort algorithm can be seen to satisfy this property.

We start by initializing a 2^n table representing the initial function f in the proof. Pass over a tape holding information on the AND gates of the SYM^+ circuit. For each AND gate, determine its set of inputs S, and append S to the end of a tape, call it L. Then, treating each S as an n-bit string, sort the list of sets on this tape in lexicographical order using Lemma C.1. The tape has s'' elements on it, so this takes at most $s'' \cdot \text{poly}(n)$ time. Let F be another tape which will hold the 2^n table representing f. Pass along tape L from left to right. Let T be the current set being examined on tape L, and let S be the previous set examined. (If T is the first set on the tape, then let S be the empty set.) If $T > S$, then compute the total number of subsets N between T and S in lex order in $\text{poly}(n)$ time, via subtraction. For N times, record on tape F that the value of f is 0 on all subsets between T and S, and record that $f(T) = 1$ on tape F. If $T = S$, then increment the current value of $f(S)$ on tape F (the tape head of F is sitting on the current value of $f(S)$, since the previous set was S). This process produces a table for f on tape F, in $O(2^n\text{poly}(n))$ time.

Each phase of the dynamic program constructs a new function g_i based on g_{i-1}. Suppose g_{i-1} is written on a tape as a collection of pairs $(T, g_{i-1}(T))$ for every $T \subseteq [n]$, where T is represented as an n-bit string. Let T_i be the ith bit of T, and let T_{-i} be the $(n-1)$-bit string obtained by removing the ith bit of T. To compute g_i, first sort all 2^n pairs $(T, g_{i-1}(T))$ according to the first key T, using the following ordering $<_i$:

$T <_i T'$ iff $(T_i < T'_i) \vee ((T_i = T'_i) \wedge (T_{-i} < T'_{-i}$ in lex order)).

That is, the ordering $<_i$ gives precedence to the ith bit. The sort can be done in $O(2^n\text{poly}(n))$ time on a multitape Turing machine using Lemma C.1. Put a marker $\#$ on the sorted list to separate those pairs with T's that don't contain i, from those which do contain i. Note that exactly 2^{n-1} pairs are on the left side of $\#$, and 2^{n-1} pairs are on right side. Also note that the Jth set on the left-side list is exactly the Jth set on the right-side list, except in the ith position of the bit string (the left-side set has a 0, right-side has a 1).

Prepare another tape G which will hold g_i. Copy all data on the sorted list over on G. Move the tape head on G back to $\#$, and move the tape head on the sorted list to the leftmost square. Then move the tape head right on the sorted list, concurrently with moving the tape head right on G. Note that, as we scan over the pair $(T - \{i\}, g_{i-1}(T - \{i\}))$ on the sorted list, we scan over $(T, g_{i-1}(T))$ on G. Hence we can just add the current value on the sorted list to the current value on G. These are both $O(\text{poly}(n))$-bit values, so the addition takes $O(\text{poly}(n))$ time. When the tape head reaches the end of G, the tape G will now contain g_i. (We can remove the marker $\#$ in a final pass, if need be.)

Finally, we note that the evaluation of the symmetric function on all entries in the table of $g = g_n$ can also be done efficiently. Sort the entries of g_n in increasing order, and remove duplicates, in $O(2^n\text{poly}(n))$ time. The resulting list has $O(s'')$ items, each of which can be evaluated in $\text{poly}(s'')$ time.