# Appendix B

# Web addendum: $\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$

Very recently, Williams resolved "Frontier 2.1" from Chapter 14 of our book, by showing that $\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$. This exciting result uses many previous results described in the book, and suggests an approach for further lower bounds, and so we believe many instructors would want to teach it. We will add a writeup of the result to a future printing (giving a proof for the Yao-Beigel-Tarui Theorem currently stated in Chapter 14, and giving the proof of Williams's lower bound in Chapter 22). In the meantime we provide this web addendum in the hope it will be helpful for instructors and students. In Section B.1 we present Williams's Theorem, and in Section B.2 we prove the Yao-Beigel-Tarui Theorem that Williams uses in his result.

## B.1 $\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$

We recall the class $\mathbf{ACC}^0(m_1, m_2, ..., m_k)$ (Definition 14.3) consists of all functions computed by polynomial size circuits of constant depth with AND, OR, NOT, and $MOD_{m_1}, \ldots, MOD_{m_k}$ gates, where a $MOD_m$ gate outputs 0 on inputs $x_1, \ldots, x_k$ if $\sum_{i=1}^{k} x_i = 0 \pmod m$ and outputs 1 otherwise. The class $\mathbf{ACC}^0$ consists of the union of $\mathbf{ACC}^0(m_1, \ldots, m_k)$ for every finite $k$-tuple $(m_1, \ldots, m_k)$. While strong lower bounds were shown in Chapter 14 for $\mathbf{ACC}^0(p)$ for any prime $p$, very little was known about $\mathbf{ACC}^0$ or even $\mathbf{ACC}^0(6) = \mathbf{ACC}^0(2, 3)$. Allender and Gore [AG94] gave lower bounds for a uniform variant of $\mathbf{ACC}^0$ (see Chapter 14 notes). But nothing non-trivial was known for the non-uniform class.

**Theorem B.1** *(Williams, 2010)*
$\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$

The proof uses many of the results presented in this book, including the structural properties of $\mathbf{ACC}^0$ (Theorem 14.11, which itself uses the proof of Toda's Theorem 17.14), the non-deterministic time hierarchy theorem (Theorem 3.2), and via the "easy witness" method of Lemma 20.20 also uses $\mathbf{IP} = \mathbf{PSPACE}$ (Theorem 8.19) and derandomization results (Theorem 20.7). The philosophy behind the proof is somewhat dual to the natural proofs paradigm. While natural proofs say that certain classes of lower bounds against a class $\mathcal{C}$ imply a meta-algorithm that can distinguish the truth table of a circuit in $\mathcal{C}$ from a random function, Williams's proof gives a fairly general way to translate a different meta-algorithm for $\mathcal{C}$ (a slightly non-trivial circuit satisfiability algorithm) into a lower bound showing that $\mathbf{NEXP} \not\subseteq \mathcal{C}$. That said, Williams's proof techniques, which include clever usages of diagonalization and proof by contradiction, do *not* fall under the natural proof paradigm (though we do not know if they can be "naturalizable"). Indeed, it needs to be investigated if Williams's paradigm could extend to showing lower bounds higher complexity

classes such as $\mathbf{TC}^0$, $\mathbf{NC}1$ (and indeed maybe even $\mathbf{P}_{/\mathbf{poly}}$) which are widely conjectured to contain pseudorandom functions, and hence cannot be separated from higher classes via natural proofs.

### B.1.1 Non trivial satisfiability algorithm for $\mathrm{ACC}^0$ circuits

First, we explain the property of $\mathbf{ACC}^0$ (not shared by any higher class, as far as we know) that allows this lower bound: the circuit satisfiability problem for $\mathbf{ACC}^0$ circuits is solvable in time that is slightly better than trivial.

**Theorem B.2** *For every depth $d$ there is a $\delta = \delta(d) > 0$ and an algorithm $A$ with the following property. Given an instance of* CKT-SAT *which is a depth-$d$ $\mathbf{ACC}^0$ circuit with $n$-bit input and up to $2^{n^\delta}$ gates, the algorithm solves it in $2^{n-n^\delta}$ time.* ♢

The proof relies on the following characterization of $\mathbf{ACC}^0$ circuits, which was stated without proof in slightly different terms in Theorem 14.11 and proven below in Section B.2 below. We define a SYM+ circuit of degree $d$ and size $s$ to be a pair $(P, \Theta)$ such that $P$ is an $n$-variable degree-$d$ multilinear polynomial over the integers with coefficients of magnitude at most $s$, and $\Theta$ is a function from $\mathbb{Z}$ to $\{0,1\}$. (In Theorem 14.11 we described this as depth 3 circuit with a symmetric gate on top and AND gates of fanin at most $s$ below it.) A SYM+ circuit $(P, \Theta)$ computes a function $f : \{0,1\}^n \to \{0,1\}$ if $f(x) = \Theta(P(x))$ for every $x \in \{0,1\}^n$. The polynomial $P$ is represented as the list of non-zero coefficients of its monomials, and the function $\Theta$ is represented as the list of values on which it outputs 1. (Note that for every $x \in \{0,1\}^n$, $|P(x)| \leq s \cdot n^d$, and so this list need not be longer than $O(s \cdot n^d)$.)

**Theorem B.3** *([Yao90, BT91]) There is an algorithm that given any $\mathbf{ACC}^0$ $n$-variable circuit $C$ of depth $d$ and size $s$ runs in $s^{\mathrm{polylog}(s)}$ and outputs a SYM+ circuit $(P, \Theta)$ of degree at most* polylog$(s)$ *and size at most $s^{\mathrm{polylog}(s)}$ such that $\Theta(P(x)) = C(x)$ for every $x \in \{0,1\}^n$. (The implicit exponents in the* polylog *notation depends on $d$.)* ♢

Proof of Theorem B.2 from Theorem B.3: . Let $d \in \mathbb{N}$, and we choose $\delta$ as some function of $d$ to be specified later. We're given an $\mathbf{ACC}^0$ circuit $C$ of $n$ inputs and size $s \leq 2^{n^\delta}$ and need to decide in $2^{n-n^\delta}$ whether or not $C(x) = 0$ for every $x \in \{0,1\}^n$. Let $C'$ be the circuit on $n' = n - 2n^\delta$ inputs such that $C'(x_1, \ldots, x_{n'}) = \vee_{x_{n'+1},\ldots,x_n} C(x_1, \ldots, x_n)$. Clearly $C'$ is a depth $d+1$ $\mathbf{ACC}^0$ circuit of at most $s' \leq 2^{2n^\delta}$ size such that $C'$ is satisfiable if and only if $C$ is. We will evaluate $C'$ on *all* of its $2^{n'} = 2^{n-2n^\delta}$ inputs and check if one of the outputs happens to be 1. While naively doing these evaluations would cost $s' \cdot 2^{n'} \gg 2^n$, it turns out that for $\mathbf{ACC}^0$ it's possible to *amortize* the work and perform all $2^{n'}$ evaluations at a cost of $(2^{n'} + s'^{\,\mathrm{polylog}(s')})\,\mathrm{poly}(n)$ which will be smaller than $2^{n-n^\delta}$ for our setting of parameters.

Specifically, we choose $\delta$ to be a small enough function of $d$ so that Theorem B.3's transformation of $C'$ to $C''$ of the form $C'' = \Theta(P(x))$ takes time less than, say, $2^{\sqrt{n}}$. We then use the following useful observation on evaluating polynomials:

**Lemma B.4** *There is an algorithm that given an integer multilinear polynomial $P(x_1, \ldots, x'_n) = \sum_{y \in \{0,1\}^{n'}} \hat{P}(y) \prod_{i=1}^n x_i^{y_i}$ in the form of the array $\hat{P} : \{0,1\}^{n'} \to \mathbb{Z}$, outputs the evaluations of $P$ on all inputs in $\{0,1\}^{n'}$ in time $2^{n'}\,\mathrm{poly}(n', \log s)$ where $s = \max_{y \in \{0,1\}^{n'}} |\hat{P}(y)|$.* ♢

Lemma B.4 is proven via a simple but clever dynamic programming algorithm, somewhat reminiscent of the Fast Fourier Transform, see Exercise B.1. It immediately concludes the proof of Theorem B.2 since we can compute the polynomial $P$ in time $2^{\sqrt{n}}$, and then evaluate all its values and apply $\Theta$ to them in time $2^{n'}\,\mathrm{poly}(n) \ll 2^{n-n^\delta}$. ■

## B.1.2 Proof of Theorem B.1

We now turn to proving Theorem B.1. We will assume that **NEXP** $\subseteq$ **ACC**$^0$ and show that **NTIME**$(2^n/n^{10}) \subseteq$ **NTIME**$(2^{n-n^\delta})$ for some $\delta > 0$, which contradicts the non-deterministic hierarchy theorem (Theorem 3.2). We will use the fact that every language $L$ in **NTIME**$(2^n/n^{10})$ reduces to SUCCINCT-SAT$_n$, the problem in which one has to determine the satisfiability of a 3CNF Boolean formula $\varphi$ with $2^n$ clauses and $2^n$ variables that is described in *succinct form* using a circuit of size poly$(n)$. This is a circuit on $n$ inputs that, given $j$ in the set $\{0,1\}^n$ (which we identify with the numbers in $[2^n]$), outputs the description of the $j^{th}$ clause of $\varphi$: indices $\mathrm{var}_1(j), \mathrm{var}_2(j), \mathrm{var}_3(j)$ of the three variables in clause $j$, and three bits $\mathrm{neg}_1(j), \mathrm{neg}_2(j), \mathrm{neg}_3(j)$ that are 1 iff the corresponding variable appears negated in the clause. Thus the $j$th clause can be thought of as

$$\bigvee_{i=1,2,3} X_{\mathrm{var}_i(j)} \oplus \mathrm{neg}_i(j), \tag{1}$$

where $\oplus$ denotes parity as usual. For every $L \in$ **NTIME**$(2^n/n^{10})$, given any input $x$ for $L$ one can write in $n^5$ time a circuit $C_x$ of size $n^5$ such that the instance of SUCCINCT-SAT$_n$ represented by $C_x$ is satisfiable iff $x \in L$. See Problem 6.18 in Chapter 6 and the discussion in Section 5.4 and also Problem B.4. (This reduction uses the Cook-Levin proof of NP-completeness of SAT; the fact that **NTIME**$(2^n/n^{10})$ can be reduced to satisfiability of instances of size $2^n$ uses the efficient simulation of Turing Machines using oblivious Turing Machines in Chapter 1.) We will denote the formula encoded by $C$ as $\varphi(C)$.

Before we describe the algorithm we record two useful implications of our assumptions that **NEXP** $\subseteq$ **ACC**$^0$. First, one immediate corollary of **NEXP** $\subseteq$ **ACC**$^0$ is that **P** $\subseteq$ **ACC**$^0$. In particular, the CIRCUIT-EVAL problem of evaluating a Boolean circuit on a given input is in **ACC**$^0$, which implies that there is some absolute constants $c, d_0$ such that for every Boolean circuit $C$ of size $s$ there exists an equivalent **ACC**$^0$ circuit $C'$ of size at most $s^c$ and depth at most $d_0$. (The circuit $C'$ can be obtained by hardwiring the constants corresponding to the description of $C$ into the **ACC**$^0$ circuit for CIRCUT-EVAL.) Of course, this does not necessarily mean that there is an efficient way to find $C'$ given $C$, but nonetheless the observation above will be very useful for us below.

We say that SUCCINCT-SAT$_n$ has *succinct witnesses* if the satisfying assignment is also representable as a small circuit. In other words, there is some constant $c > 0$ such that for every circuit $C$ that is a YES instance for SUCCINCT-SAT$_n$ there exists a circuit $D$ of size at most $n^c$ that encodes the satisfying assignment for the formula encoded by $C$. That is, $D$ has $n$ inputs, and the assignment $D(0^n), \cdots, D(1^n)$ satisfies $\varphi(C)$. It turns out that under our assumption SUCCINCT-SAT$_n$ has succinct witnesses:

**Lemma B.5** *If* **NEXP** $\subseteq$ **P**$_{/\mathbf{poly}}$ *then* SUCCINCT-SAT$_n$ *has succinct witnesses.* ♢

Proof: This is implicit in the proof of Lemma 20.20 which showed that under this assumption **NEXP** = **EXP**— the **EXP** algorithm actually worked by searching for an assignment encoded by a small circuit. We briefly recall the proof here. If this was false we would get an infinite sequence of circuits $\{C_n\}$ that encode satisfiable formulas without a succinct satisfying assignment. Then one can guess these assignments to be used as a hard functions to obtain a Nisan-Wigderson style derandomization of **MA** in **NEXP**, but since under these assumption **MA** = **EXP** we'd be able to use this to diagonalize against circuits of any fixed polynomial size. ■

We now turn to the algorithm. We'll choose $\delta = \delta(10d_0)$ as in the statement of Theorem B.2 where $d_0$ is the constant above such that every Boolean circuit has an equivalent depth-$d_0$ **ACC**$^0$ circuit. We are given an $n$-bit input Boolean circuit $C$, and need to decide in non-deterministic $2^{n-n^\delta}$-time whether or not the Formula $\varphi(C)$ is satisfiable. Under our assumption that **NEXP** $\subseteq$ **ACC**$^0$ for every input $C$, $\varphi(C)$ is satisfiable if and only if there are two $n$-input **ACC**$^0$ circuits $C'$ and $D'$ of poly$(n)$ size and depth $d_0$ such that **(i)** $C'$ is equivalent to $C$ and **(ii)** $D'$ encodes a satisfying assignment to $\varphi(C) = \varphi(C')$. Thus our algorithm will simply non-deterministically guess $C'$ and $D'$ and then check the conditions **(i)** and **(ii)** in $2^{n-n^\delta}$ poly$(n)$-time.

The condition **(ii)** can be checked in time $2^{n-n^\delta}$ using the algorithm of Theorem B.2 since it amounts to checking whether the following $\mathbf{ACC}^0$ circuit $G$ outputs 1 on all possible inputs $j \in \{0,1\}^n$:

$$G(j) = \bigvee_{i=1,2,3} D'(\mathrm{var}_i(j)) \oplus \mathrm{neg}_i(j). \quad (2)$$

Note that $G$ is an $\mathbf{ACC}^0$ circuit of depth smaller than $2d_0 + 10$ since the clause description $\mathrm{var}_i(j), \mathrm{neg}_i(j)$ for $i = 1, 2, 3$ is computed by the depth-$d_0$ $\mathbf{ACC}^0$ circuit $C'$, and the candidate assignment is computed by the depth-$d_0$ $\mathbf{ACC}^0$ circuit $D'$.

Thus all we need to show is how to check condition **(i)**— that $C'$ is equivalent to $C$. The algorithm of Theorem B.2 does not directly apply since $C$ is a general Boolean (not necessarily $\mathbf{ACC}^0$) circuit. But we can still verify equivalence using non-determinism. The idea is that for every wire $i$ of $C$ we guess an $\mathbf{ACC}^0$ circuit $C'_i$ of depth $d_0$ that is supposed to be equivalent to the value on the $i^{th}$ wire of $C$. That is, for every input $x \in \{0,1\}^n$, $C'_i(x)$ outputs the same value as the value on the $i^{th}$ wire of $C$ when invoked input $x$. (The circuit $C'$ is equal to the circuit corresponding to the output wire of $C$.)

If the $i^{th}$ wire in $C$ is the result of the AND of the $j^{th}$ and $k^{th}$ wire then for every $x \in \{0,1\}^n$ it holds that $C_i(x) = C_j(x) AND C_k(x)$ for all $x$. Similar condition holds for OR and NOT gates, as well as wires that are connected to the inputs. Now consider the $\mathbf{ACC}^0$ circuit $E$ that outputs the AND of all those conditions for all wires $i$ of $C$. Since all the circuits $C_i$ are of depth $d_0$, $E$ has depth less than, say, $d_0 + 100$, and size polynomial in the size of $C$. Moreover, if $E(x) = 1$ for every $x$, then by it's easy to prove that for every $i$ the circuit $C_i$ is indeed equivalent to the $i^{th}$ wire of $C$. (This is just like the reduction from CKT-SAT to 3SAT of Lemma 6.11.) Thus by running the satisfiability algorithm of Theorem B.2 on $1 - E$ one can verify condition **(ii)** in time $2^{n-n^\delta}$, completing the description of our SUCCINCT-SAT$_n$ algorithm. The overall (non-deterministic) running time of the algorithm is $2^{n-n^\delta}$ poly$(n)$. ■

**Remark B.6**

The proof above does not really require a full-fledged satisfiability algorithm for $\mathbf{ACC}^0$. It's enough to come up with an *approximate satisfiability* algorithm that distinguishes between circuits that are unsatisfiable and those that have, say, at least 1/10 fraction of satisfying assignments. The main idea is that one can use a highly efficient variant of the scaled up **PCP** Theorem (Theorem 11.8) to ensure we can derive a contradiction to the non-deterministic hierarchy theorem even via an algorithm that on input a circuit $C$ finds an assignment that *approximately* satisfies the formula $\varphi(C)$. This allows to use circuits $C', D'$ in the proof that only agree with the circuits $C, D$ on some large fraction of the inputs. Of course such an approximate satisfiability algorithm is easy to achieve in *probabilistic* polynomial time, and thus Williams's result is another instance of the connection between derandomization and lower bounds explored in Chapter 20.

## B.2 Simulating $\mathrm{ACC}^0$ circuits by low degree polynomials: Proof of Theorem B.3

The proof of Theorem B.3 uses the ideas of the proof of Toda's Theorem (Theorem 17.14). The intuitive link is that Toda's proof can be seen as a transformation for exponential-size constant depth circuits, since **PH** languages have such circuits that are DC-uniform. Here we will apply similar transformations on polynomial-size circuits.

Recall that Toda's Theorem is proven in two steps. First we showed a randomized reduction from **PH** to $\oplus$**P** (Theorem 17.17). Using the characterization of **PH** as constant depth DC uniform circuits, one can interpret this probabilistic reduction as a way of transforming an **AC**0 circuit to a depth three circuit with majority gate at the output level and $MOD_2$ gates (the majority gate corresponds to the fact that this is a probabilistic reduction,

note that the same transformation is used in the proof of Theorem 14.4— the Razborov-Smolensky lower bound for $\mathbf{ACC}^0(q)$ ). The second step in Toda's theorem is to transform this randomized reduction to $\oplus\mathbf{P}$ into a deterministic reduction to $\#\mathbf{P}$ (Lemma 17.22)— this can be thought of as a way of removing the $MOD_2$ gates and reducing the depth of the circuit to two at the expense of changing the majority function to a different symmetric function (i.e., a function that only depends on the number of its inputs that are one). We are going to prove Theorem B.3 by applying the first transformation to the $\mathbf{ACC}^0$ circuit to replace the OR and AND gates by small addition modulo-2 and low fan-in multiplication and then repeatedly applying the second transformation to "peel off" one by one the layers of the circuit until we are left with a $\mathsf{SYM}+$ circuit. We now turn to presenting the actual proof.

Let $C$ be a size-$s$ depth-$d$ $\mathbf{ACC}^0$ circuit $C$, and for concreteness we restrict to the case that $C$ only uses $MOD_2$, $MOD_3$ and OR gates and the constant 1. (We leave the relatively straightforward extension to the case of general modula gates as Problem B.2; we omitted the NOT gate as it can be replaced with a $MOD_2$ gate and similarly OR and $MOD_2$ can replace the AND gate.) The proof will consist of applying several transformations to $C$, where in each step we modify $C$ to an equivalent circuit $C'$ that is "simpler" in some specific sense.

**Step 0: preliminary tidying up of the circuit.** We can assume that $C$ has the following properties without loss of generality, as it can be easily modified to satisfy them at a polynomial cost in the size and constant factor in the depth:

1. It is *layered* with gates at layer $i$ only depending on inputs from gates at layer $i-1$, where layer 1 is the bottom-most layer depending on only the inputs. Moreover, in every layer there is only one type of gates. (This can be ensured by using fan-in one OR, $MOD_2$ or $MOD_3$ gates as "dummy gates" that just copy the input.)

2. It is a *tree* in the sense that every gate has fan-out at most 1. (This can be ensured by duplicating gates at polynomial cost since the circuit has constant depth.)

We will maintain these invariants throughout the proof, applying the above transformations whenever we modify the circuit.

**Step 1: Reducing the fan-in of OR gates; adding symmetric gate at the output** It turns out that if we are willing to add a symmetric function on top, we can use the following claim to ensure that the fan-in of all OR gates is at most polylogarithmic:

Claim: There is a set $\mathcal{D}$ of $\mathbf{ACC}^0$ circuits with the same gateset as $C$ and of size at most $s^2$ and depth at most $3d$ such that every OR gate in $D \in \mathcal{D}$ has fan-in at most $O(\log^2 s)$ and for every input $x \in \{0,1\}^n$, $\Pr_{D \in_R \mathcal{D}}[D(x) = C(x)] > 0.9$. Moreover $|\mathcal{D}| \leq s^{O(\log^2 s)}$.

Proof of Claim: The proof uses the Valiant-Vazirani Lemma (Lemma 17.19), that shows that given an OR gate over $2^k$ inputs of the form $OR(x_1, \ldots, x_{2^k})$, if we pick $h$ to be a pairwise independent hash function from $[2^k]$ to $\{0,1\}$ then for any nonzero $x \in \{0,1\}^{2^k}$, with probability at least $1/(10k)$ it will hold that $\sum_{i:h(i)=1} x_i \pmod 2 \neq 0$. Thus if we pick $t = 100k \log s$ such hash functions $h_1, \ldots, h_t$ independently then with probability at least $1 - 1/(10s)$ the OR of $x_1, \ldots, x_{2^k}$ will equal $OR_{j=1}^t(\sum_{i:h_j(i)=1} x_i \pmod 2)$, enabling us to take a union bound over all the at most $s$ OR gates in $C$. Because we used the union bound we can reuse the same set of $t$ hash functions for all the gates, and thus the set of circuits has cardinality at most $|\mathcal{H}|^t$ where $\mathcal{H}$ is the cardinality of the set of pairwise independent hash functions on $2^k$ inputs (we can assume $2^k \leq 2s$ since the fan-in of every OR gate is at most $s$). As noted in Section 8.2.2, there exists such a collection of cardinality at most $2^{2k} = O(s^2)$, and thus the total number of circuits will be $s^{O(\log^2 s)}$. ■

An immediate corollary of the claim is that, writing $\mathcal{D} = \{D_1, \ldots, D_{s'}\}$, for every $x \in \{0,1\}^n$, $C(x) = MAJ(D_1(x), \ldots, D_{s'}(x))$ where $MAJ$ is the majority function.

**Step 2: Moving to arithmetic gates and pushing multiplications down.** We will now transform $C$ so that all the gates will operate over the *integers*, meaning they take an integer (not necessarily 0/1) as input and output an integer as well. (Note that the inputs to the entire circuits are still 0/1 as before.) The output gate of the circuit $C'$ consists of an addition gate (which outputs an integer), followed by some function $\Theta : \mathbb{Z} \to \{0,1\}$ such that the final output $C(x) = \Theta(C'(x))$ for every $x$. Thus the output gate can be viewed as a symmetric gate.

First, we use $OR(x_1, \ldots, x_k) = 1 - x_1 \cdots x_k$ to replace OR gates with multiplication and addition gates over the integers. (Note that the fan-in of the multiplication gates will be at most $O(\log^2 s)$.) Next we use the equation $MOD_p(x_1, \ldots, x_k) = (\sum_{i=1}^{k} x_i)^{p-1} \pmod p$ (implied by Fermat's Little Theorem) to replace every $MOD_p$ gate in $C$ with $k$ inputs with one addition gate and $p-2$ multiplication gates (taking integer inputs) followed by a "$MOD_p$ conversion" gate that given an integer input $z$ outputs $z \pmod p$.

We can also push all multiplication gates to the bottom of the circuit using the distributive rule (i.e., $(a+b)(c+d) = ac + bc + ad + cd$) and the equation $(x_1 \pmod p) \cdots (x_k \pmod p) = (x_1 \cdots x_k) \pmod p$. Thus all multiplication gates will only involve input variables. Note that pushing the multiplication gates down can increase their fan-in from $\log^2 s$ to at most $\log^{O(d)} s$. By adding "dummy gates" we will keep the invariant that every layer of the circuit only has one type of gate (addition, $\mod 2$, $\mod 3$, or multiplication).

The bottom line is that we get a circuit $C'$ of $O(d)$ depth and size at most $s^{\mathrm{polylog}(s)}$ over the integers involving only addition, $\mod 2$ and $\mod 3$ conversion gates, and multiplication gates. Every layer in the circuit has only one type of gate, with multiplication at the very bottom and with fan-in at most $\mathrm{polylog}(s)$. The output gate of the circuit $C'$ is an addition gate (which outputs an integer), and there is some function $\Theta : \mathbb{Z} \to \{0,1\}$ such that $C(x) = \Theta(C'(x))$ for every $x$. (The function $\Theta$ can be represented by a table of $s^{\mathrm{polylog}(s)}$ size, as that is the maximum magnitude of a value that $C'$ can output on an input in $\{0,1\}^n$.)

**Step 3: Peeling off layers.** We now show how we can take the uppermost layer of the circuit $C'$ and remove it (at the cost of somewhat increasing its size and changing the function $\Theta$), we repeat this again and again until there are no more $\mod 2$ or $\mod 3$ gates left, and hence all we are left is with a circuit with one addition gate at the top, and multiplication gates of $\mathrm{polylog}(s)$ fan-in at the bottom - a $\mathrm{polylog}(s)$-degree multivariate polynomial over the integers. (Using the equation $x = x^2$ for $x \in \{0,1\}$, we can assume this polynomial is multilinear without loss of generality.)

To do this "layer removal" operation, suppose that the top layer of $C'$ has $\mod 3$ gates (the case of $\mod 2$ gates is symmetric, and addition gates can just be collapsed into the output addition gate anyway). Thus, the output gate of $C'$ has the form $\sum_{i=1}^{k}(y_i \pmod 3)$ where $y_i$ is the value fed into the $i^{th}$ $\mod 3$ gate of the top layer. (Note that the output gate outputs an integer, not necessarily in $\{0,1\}$, and that integer is fed into the function $\Theta$ to obtain a 0/1 answer.) We will use a family of univariate polynomials over the integers $\{P_t\}$ such that $P_t$ has degree at most $t^{10}$ (with coefficients of size at most $10^t$) and such that $P_t(y) \pmod{3^t} = y \pmod 3$. (These are known as Modulus amplifying polynomials, see also Lemma 17.22 and Problem B.3.) Taking $t > \log_3 k$ we see that $\sum_{i=1}^{k}(y_i \pmod 3) = \sum_{i=1}^{k}(P_t(y_i) \pmod{3^t}) = (\sum_{i=1}^{k} P_t(y_i)) \pmod{3^t}$ (where the latter equality holds since the sum on the left hand side is smaller than $3^t$. Thus, by changing the function $\Theta(x)$ to $\Theta'(x) = \Theta(x \pmod{3^t})$, and by pushing down the multiplication gates involved in computing $P_t$ (at a cost of $\mathrm{poly}(t) = \mathrm{polylog}(s)$ to the bottom fan-in), we get a circuit of the same form but with depth reduced by one.

Repeating Step 3 again and again we end up with a SYM+ circuit, one can also verify that all our transformations can be carried out in time polynomial in the size of the output circuit. ■

# Exercises

**B.1** (Yates 1937 via Williams 2010) Let $P : \mathbb{Z}^n \to \mathbb{Z}$ be a multilinear polynomial over the integers with coefficients of size at most $s$. That is, $P(x_1, \ldots, x_n) = \sum_{y \in \{0,1\}^n} \hat{P}(y) \prod_{j=1}^{n} x_j^{y_j}$.

(a) Let $P_0(x) = \hat{P}(x)$ and $P_i(x) = \sum_{y_1,\ldots,y_i \in \{0,1\}^i} \hat{P}(y_1, \ldots, y_i, x_{i+1}, \ldots, x_n) \prod_{j=1}^{i} x_j^{y_j}$. Note that $P_n(x) = P(x)$. Prove that for every $x \in \{0,1\}^n$,

$$P_i(x) = P_{i-1}(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + x_i P_{i-1}(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \ .$$

(b) Give a $2^n \, \mathrm{poly}(n, s)$-time algorithm that given the length $2^n$ array $\hat{P}(0^n), \cdots, \hat{P}(1^n)$ outputs the array $P(0^n), \cdots, P(1^n)$. (Hint: use dynamic programming)

(c) Show that one can implement the above algorithm on a multi-tape Turing machine. (Hint: sort the arrays under a different order between after each one of the $n$ stages of the algorithm.)

**B.2** (a) Extend the proof of Theorem B.3 to the case where the $\mathbf{ACC}^0$ circuit uses the gates $MOD_{p_1}, \ldots, MOD_{p_k}$ where $k$ is some constant and the $p_i$'s are distinct primes.

(b) Extend the proof of Theorem B.3 to the general case where the $\mathbf{ACC}^0$ circuit uses the gates $MOD_{m_1}, \ldots, MOD_{m_k}$ where $k$ is some constant and the $m_i$'s can prime, prime powers, or composites.

**B.3** Define the polynomials $S_i(x)$ as follows: $S_1(x) = 3x^2 - 2x^3$ and $S_i(x) = S_1(S_{i-1}(x))$.

(a) Prove that for every $x, i, m$ if $x \pmod m \in \{0,1\}$ then $P_i(x) \pmod{m^{2^t}} = x \pmod m$.

(b) Use these polynomials to construct the collection of polynomials $\{P_t\}$ needed in the proof of Theorem B.3. (Hint: use Fermat's Little Theorem.)

**B.4** Prove that there is some constant $c$ such that for every $L$ in $\mathbf{NTIME}(2^n/n^c)$, there is a polynomial-time computable function $f$ such that $x \in L$ iff $f(x) \in \text{SUCCINCT-SAT}_{|x|}$.