Barriers in Computational Complexity

Tair Akhmejanov

May 1, 2013

Introduction

The first barrier result in complexity theory was published in 1975 by Baker, Gill, and Solovay when they showed that there exist oracles A, B such that $\mathtt{P}^A = \mathtt{NP}^A$ and $\mathtt{P}^B \neq \mathtt{NP}^B$. Therefore, if a candidate proof of $\mathtt{P} \neq \mathtt{NP}$ remains valid when the machines are given access to an oracle, then in particular it remains valid when the machines have access to A. Then this same proof of $\mathtt{P} \neq \mathtt{NP}$ also gives $\mathtt{P}^A \neq \mathtt{NP}^A$, but we know that this is false. The same reasoning holds with the possibility of proving $\mathtt{P} = \mathtt{NP}$. Thus, the ideas in a potential proof resolving the P vs. NP question must be affected by the introduction of oracles to the machine model. At the time, most of the techniques in complexity theory, if not all, were simulation and diagonalization arguments borrowed from logic, all of which are insensitive to the introduction of oracles. Thus, analyzing the complexity of computability will require techniques more powerful than those of classical computability theory, so the slogan became that, "we must analyze computation rather than merely simulate it."

Here we explore two more recent barriers, the first of which is a barrier to proving circuit lower bounds. The circuit model was introduced as a potential research plan to analyze computation more closely, thereby avoiding the relativization issue. But after many years of research, there were no significant lower bounds for languages in classes such as NP, and the few known lower bounds were for restricted models of circuits. In 1996 Razborov and Rudich showed that this research program also has its own intrinsic barrier, offering a potential explanation as to why such little progress had been made. As with the message of the relativization barrier, this does not necessarily mean that the circuit model should be completely abandoned, but that perhaps we need to be even more clever. A careful examination of the barrier may even indicate how to proceed.

The second barrier concerns the idea of treating a Boolean formula as an arithmetic expression—an idea that proved fruitful in establishing results about interactive proofs. These results proved statements that are known to have contrary relativization just like $\mathtt{P} \overset{?}{=} \mathtt{NP}$. Furthermore, circuit lower bounds were discovered that avoid both the relativization barrier and the natural proofs barrier, so it was natural to wonder whether the same techniques could be adapted to resolve P vs. NP, or at least some other important open questions. In 2008 Aaronson and Wigderson introduced the algebrization barrier precisely to quantify the usefulness of this method. They established that resolving P vs. NP, as well as many other potentially easier open problems, will require non-algebrizing techniques.

The structure of the essay is as follows. There are three main sections: 1) Natural Proofs, 2) Algebrization, and 3) Circumventing the Barriers. In the third section, we briefly explore Ryan Williams's recently proposed research program for proving circuit lower bounds via slightly faster algorithms for NP problems.

1 Natural Proofs

The work of Razborov and Rudich examines the common characteristics of (non-monotone) circuit lower bound proofs and identifies a framework that they all share. We will not be concerned with results about monotone circuit here. Formally, they define a natural property and argue that every known circuit lower bound proof identifies such a property. Their main message is that proving a much stronger lower bound with the same strategy would identify a stronger natural property, one that could be used to break a pseudorandom generator. Since the existence of pseudorandom generators is a plausible conjecture, it is doubtful that we can use the common framework of all known lower bound proofs to prove a stronger lower bound.

In order to appreciate the significance of their result, it is essential to understand the proofs of some circuit lower bounds. We begin by examining one of the first lower bounds proved and analyze the proof to motivate the definition of a natural property. Then we introduce the notions of a pseudorandom generator and show how to construct from it a pseudorandom function generator. Technically speaking, Razborov and Rudich prove that a strong enough natural property contradicts the security of a pseudorandom function generator constructed from a pseudorandom generator, which contradicts the security of a generator itself. Finally, we explore one other circuit lower bound to illustrate the method for determining that a lower bound proof indeed identifies a natural property.

The first lower bound we will explore is $\mathsf{Parity} \notin \mathtt{AC}^0$. Initially, researchers focused on proving lower bounds for simple functions and very limited models of circuits. The hope was to gradually increase the intricacy of the function while dropping certain restrictions to the circuit models. This early result of Furst, Saxe, and Sipser [FSS] (and independently of Ajtai [Ajt]) says that the parity function of summing the binary inputs mod 2 does not have constant-depth polynomial size circuits with AND, OR, NOT gates of unbounded fan-in.

1.1 $\mathsf{Parity} \notin \mathtt{AC}^0$

Although this proof will be a lot of hard work, it will serve as a good example of the types of arguments used in the field. Here we follow the original proof of [FSS], even though more refined proofs were given by Yao [Yao1], and later by Hastad [Has1] using Hastad switching. Actually, from the perspective of natural proofs the proofs are arguably "the same". We begin with some definitions.

Definition 1. The class $\mathtt{NC}^\mathtt{i}$ is the class of languages that are decided by polynomial size circuits of depth $O(\log^i(n))$ with AND, OR gates of fan-in 2 and NOT gates. Here n is the number of bits in the binary input. The class $\mathtt{AC}^\mathtt{i}$ is defined analogously except that the gates are allowed unbounded fan-in. A circuit family $\{C_n\}$ is an $\mathtt{NC}^\mathtt{i}(\mathtt{AC}^\mathtt{i})$ circuit family if C_n is of the respective form for every n.

Any unbounded fan-in gate with polynomially many inputs can be simulated using a depth $O(\log n)$ tree of fan-in 2 gates, so we have the trivial string of inclusions,

$$\mathtt{NC}^0 \subseteq \mathtt{AC}^0 \subseteq \mathtt{NC}^1 \subseteq \mathtt{AC}^1 \subseteq \mathtt{NC}^2 \subseteq \mathtt{AC}^2 \subseteq \cdots .$$

When talking about circuits we will refer to the "top" as the level of the inputs and the "bottom" as the level of the output. This standard varies in the literature.

Definition 2. Let Parity be the family of Boolean functions

$$\left\{ f_n(x_1, \ldots, x_n) = \sum_{i=0}^{n} x_i \mod 2 \;\middle|\; n \in \mathbb{N} \right\}.$$

One can also view Parity as a language such that a Boolean string $x_1 \cdots x_n = x \in \mathsf{Parity}$ if and only if $f_n(x_1, \ldots, x_n) = 1$.

We would like to show that $\mathsf{Parity} \notin \mathtt{AC}^0$. First, observe that an $\mathtt{AC}^0$ circuit can be converted into the following form: 1) all NOT gates occur at the top, and 2) the levels of the circuits alternate between only AND gates and only OR gates. The resulting circuit is still polynomial size because the conversion consists of pushing negations to the top using DeMorgan's laws and mostly adding trivial AND and OR gates. Therefore, for the purposes of proving a lower bound we may assume that $\mathtt{AC}^0$ circuit families are of this form. In fact, $\mathtt{AC}^\mathtt{i}$ stands for alternating class, and this alternation property allows us to give a recursive definition for such circuits.

Definition 3 ([FSS])**.** For every n let $X_n = \{x_1, \ldots, x_n\}$ be *variables* and $\overline{X}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$ be *literals* ($\bar{x}_i$ represents the negation of x_i). Define a 0-circuit to be a literal, which can be thought of as computing a projection function $f(x_1, \ldots, x_n) = x_j$ or the negation of such a function.

Recursively define an *i-circuit* to be a non-empty collection of $i-1$-circuits without repetition, which we call the *members* of the i-circuit. If i is odd then an i-circuit is an $\vee$-circuit and computes the $\vee$(OR) function of the functions computed by its members. If i is even then an i-circuit is an $\wedge$-circuit and computes the $\wedge$(AND) function of the functions computed by its members. Denote by C_f the function that the circuit C computes. There are two constant circuits 0 and 1 that compute the constant functions.

Therefore, an i-circuit in some larger circuit C of depth $d > i$ can be thought of as a subcircuit of C. We will sometimes refer to these subcircuits as gates, but the recursive definition should be kept in mind. A circuit by this definition always has the first level consisting of $\vee$-circuits with literals as inputs, and the second level consisting of $\wedge$-circuits with inputs the $\vee$-circuits of the first level. It is implied that the collection of members of any i-circuit is at most polynomial because we will be considering circuits of polynomial size.

Definition 4. The *depth* of an i-circuit is i. The *size* of a circuit C is the number of circuits belonging to C where *belonging to* is the transitive closure of the member relation. In other words, the size of a circuit C is the number of its members plus the sum of the number of their members and so forth until depth-0 circuits are reached.

The key ingredient to the proof will be to consider what happens to the circuit and the function it computes when some of the variables are set to constants.

Definition 5. A *restriction* to the input variables is a function $\rho : X_n \to \{0, 1, *\}$, which can be extended to all of $\overline{X}_n$ by

$$\rho(\overline{x}_j) = \begin{cases} 1 - \rho(x_j) & \text{if } \rho(x_j) = 0 \text{ or } 1, \\ \rho(\overline{x}_j) = * & \text{otherwise.} \end{cases}$$

Here $*$ represents that the variable remains a free variable. For a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ and restrictions ρ that assigns k $*$'s, we have the restricted Boolean function $f^\rho(x_1, \ldots, x_n) = f(\rho(x_1), \ldots, \rho(x_n)) : \{0, 1\}^k \to \{0, 1\}$ where the k occurrences of $*$ are replaced by the appropriate free variable.

We observe a simple yet important property of Parity.

Lemma 6. *Fix an arbitrary n. If f_n computes* Parity *on n inputs, then for any restriction ρ assigning k $*$'s, f_n^ρ computes* Parity *or its negation on k inputs.*

Restrictions can also be applied to circuits in a natural way. We define recursively that a restriction ρ *forces* an $\vee$-circuit to be 1 (0) if it forces any (all) of its members to be 1 (0). Define the dual statement for $\wedge$-circuits. If a circuit C is not forced, then the forced members of C are irrelevant, so we define C^ρ to be the collection $\{B^\rho \mid B \text{ is a member of C that is not forced}\}$. We state another obvious observation.

Lemma 7. *Given a restriction ρ, if a circuit C computes f then C^ρ computes f^ρ.*

Let $f = \{f_n\}$ be the Parity function. We will use the above two lemmas in the following way: if a family of d-circuits $\{C_n\}$ computes $f = \{f_n\}$, then for any restriction ρ, C_n^ρ computes f_n^ρ, which is still the Parity function or its negation on some k_n variables.

First note that there cannot be a family of 1-circuits that computes f because for any n, f_n depends on all of its inputs, so a single $\vee$ gate cannot compute f_n. Not as obviously, any family of 2-circuits that computes f must have size at least 2^{n-1}. To see this note that any Boolean function can be represented as an AND of ORs or an OR of ANDs, which are the conjunctive normal form and the disjunctive normal form respectively. The function f_n has 2^{n-1} terms in both of these normal forms. By examining what are called *max-terms* and *min-terms* of Boolean assignments the CNF and DNF formulae can be seen to be optimal for Parity. This was originally proved by Lupanov in [Lup].

We are now ready to prove the desired theorem with the following outline. To show that for any depth d there does not exist a polynomial size family of d-circuits solving f, assume the contrary. Then there exists a minimal d such that there is a polynomial size family of d-circuits computing f. By the previous paragraph $d \geq 3$. We would like to swap the first level of $\vee$-circuits with the second level of $\wedge$-circuits, resulting in an initial level of $\wedge$-circuits followed by a level of $\vee$-circuits instead. Since $d \geq 3$, there would be two adjacent levels of $\vee$-circuits that could be merged into one single level. We can then adjust the circuit to begin with a level of $\vee$-circuits again by using the distributive laws. If this transformation is accomplished so that the resulting circuit family still computes f and still has polynomial size, then the new circuit family contradicts the minimality of d.

In general, an OR of ANDs can always be written as an AND of ORs, but doing a few examples reveals that the obvious use of the distributive law could produce a formula that is exponentially larger than the one started with.

Example 8. Interpreted as a circuit, the right hand side has much greater size.

$$\begin{aligned}&(a \vee b \vee c) \wedge (x \vee y \vee z) = \\ &(a \wedge x) \vee (a \wedge y) \vee (a \wedge z) \vee (b \wedge x) \vee (b \wedge y) \vee (b \wedge z) \vee (c \wedge x) \vee (c \wedge y) \vee (c \wedge z)\end{aligned}$$

With this method, although one level shorter, the new circuit becomes exponentially large. To get around this issue, given a C_n computing f_n, we apply a restriction to the inputs that forces enough, but not too many, subcircuits to become constant, so that the size of the new circuit does not increase too much, yet the entire circuit is not forced. By the two lemmas, the resulting circuit computes f on some fewer number of inputs that are the free variables of the restriction. These circuits can be used to create a new polynomial size family of circuits computing f. The existence of a restriction that forces just the right number of subcircuits constant will be established by the probabilistic method: we define a distribution and show that an appropriate restriction exists with positive probability.

Theorem 9 ([FSS]). Parity $\notin$ $\mathtt{AC}^0$.

Proof. Let f denote the Parity function. To get a contradiction assume that f does indeed have $\mathtt{AC}^0$ circuits. Then there exists a smallest constant d such that there exists a family of d-circuits $\{C_n\}$ with the size of each $C_n \leq n^k$ for some constant k not depending on n. As mentioned before, $d \geq 3$. We use $\{C_n\}$ to obtain a polynomial size family of $d-1$-circuits that also compute f, contradicting the minimality of d.

The proof has three steps. 1) Show that there exists a restriction such that the 1-circuits of C_n have constant size. From these circuits we form a polynomial size family $\{D_n\}$ with constant size 1-circuits computing f. 2) Show that there is a restriction such that the 2-circuits of D_n have constant size. From these we form a polynomial size family $\{E_n\}$ with constant size 2-circuits computing f. 3) Flip the 1-circuits and 2-circuits of E_n using the trivial method, which can only increase the size by a constant factor. Merge adjacent

$\vee$-circuits, and DeMorgan the circuit back to standard form to create a final polynomial size family $\{F_n\}$ of $d-1$-circuits computing f.

We will show that for every large enough n there exists with positive probability a restriction ρ such that

- C_n^ρ computes f on $m \geq \frac{\sqrt{n}}{2}$ variables and
- the size of any 1-circuit of C_n^ρ is bounded by a constant independent of n.

Define the distribution on restrictions $\rho : X_n \to \{0,1,*\}$ that assigns independently for each i the probabilities

- $\Pr[\rho(x_i) = *] = \frac{1}{\sqrt{n}}$
- $\Pr[\rho(x_i) = 0] = \Pr[\rho(x_i) = 1] = \frac{1}{2} - \frac{1}{2\sqrt{n}}$.

Denote by ρ *fails* the event that ρ assigns less than $\frac{\sqrt{n}}{2}$ variables to $*$ or the size of some 1-circuit is greater than the constant c, which will be selected later to achieve our desired result. Therefore, a crude upper bound is

$$\Pr[\rho \textit{ fails}] \leq \Pr[\rho \text{ assigns } < \frac{\sqrt{n}}{2} *' s] + n^k \cdot \Pr[\text{a given 1-circuit in } C_n^\rho \text{ has size } > c].$$

Using Chebyshev's inequality, we can bound the first term by $O(\frac{1}{\sqrt{n}})$. For the second term take any 1-circuit B of C_n and consider B^ρ. There are two cases: B is wide, meaning that it has $\geq c \ln n$ inputs, or B is narrow, meaning that it has $< c \ln n$ inputs.

(a) B is wide. Intuitively, we can bound this case because there are enough variables so that at least one of them will be assigned 1 with good probability. For B^ρ to even be able to have some sort of size, B must not be forced by ρ, so we have

$$\begin{aligned} Pr[B^\rho \text{ has size} > c] &\leq Pr[B \text{ is not forced}] \\ &= Pr[\text{ no input of } B \text{ is assigned to } 1] \\ &\leq \left(\frac{1 - 1/\sqrt{n}}{2}\right)^{c \ln n} \\ &\leq \left(\frac{3}{4}\right)^{c \ln n} \qquad \text{for say } n \geq 9 \\ &= n^{c \ln(3/4)}. \end{aligned}$$

This last expression is $o(n^{-c/4})$ because $n^{c/4} \cdot n^{c \ln(3/4)} = n^{c \cdot (1/4 + \ln(3/4))}$ tends to 0 because $1/4 + \ln(3/4) < 0$.

(b) B is narrow. In contrast to the first case, this case can be bounded because intuitively it corresponds to so few variables that the size is constant with high probability. We have

$$
\begin{aligned}
Pr[B^\rho \text{ has size } > c] &\leq Pr[\rho \text{ assigns } \geq c *' s] \\
&= \binom{cln}{c}\left(\frac{1}{\sqrt{n}}\right)^c\left(\frac{1-1/\sqrt{n}}{2}\right)^{cln-c} + \cdots + \binom{cln}{cln}\left(\frac{1}{\sqrt{n}}\right)^{cln} \\
&\leq (cln)^c\left(\frac{1}{\sqrt{n}}\right)^c\left(\frac{1-1/\sqrt{n}}{2}\right)^{cln-c} + \cdots + (cln)^{cln}\left(\frac{1}{\sqrt{n}}\right)^{cln} \\
&\leq (cln)^c\left(\frac{1}{\sqrt{n}}\right)^c\left[\left(\frac{1-1/\sqrt{n}}{2}\right)^{cln-c} + \cdots + (cln)^{cln-c}\left(\frac{1}{\sqrt{n}}\right)^{cln-c}\right] \\
&\leq \binom{c\ln n}{c}\left(\frac{1}{\sqrt{n}}\right)^c \text{ for large enough } n \\
&\leq (c\ln n)^c n^{-c/2},
\end{aligned}
$$

where we have used a common bound on the binomial coefficient. Observe that $(c\ln n)^c n^{-c/2} = o(n^{-c/4})$ because $(c\ln n)^c n^{-c/2} n^{c/4} = c^c \frac{\ln n}{n^{c/4}}$ tends to 0.

Combining (a) and (b) gives Pr[a given 1-circuit in C_n^ρ has size $> c$] $\leq o(n^{-c/4})$. Setting $c = 4k$, where recall that k is the power of the polynomial representing the size of the circuit family, gives $o(n^{-c/4}) = o(n^{-k})$. Therefore,

$$
\begin{aligned}
\Pr[\rho \textit{ fails}] &\leq \Pr[\rho \text{ assigns } < \frac{\sqrt{n}}{2} *' s] + n^k \cdot \Pr[\text{a given 1-circuit in } C_n^\rho \text{ has size } > c] \\
&\leq O(n^{-1/2}) + n^k \cdot o(n^{-k}) \\
&= o(1).
\end{aligned}
$$

Once n is large enough this probability is bounded strictly away from 1, so that for every n large enough there exists a ρ that does not fail.

For such an n and restriction ρ the resulting circuit C_n^ρ computes f on $m \geq \sqrt{n}/2$ variables, has constant size 1-circuits, and has size at most n^k, which is still polynomial in $m \geq \sqrt{n}/2$. (If C_n^ρ initially computes $\neg f$, adjust with a NOT gate and propagate the NOT to the input level.) From these circuits we construct a family of polynomial size d-circuits $D_1, D_2, \ldots$ such that each 1-circuit has constant size c. Although the initial family of restricted circuits C_n^ρ may not hit every input size, we can adjust by assigning 0's to an appropriate number of inputs in some C_n^ρ for large enough n to get the desired D_i *for every* i.

Now we reason similarly to show that there exist restrictions to D_n that make the 2-circuits constant size. Consider the same distribution on restrictions ρ as above. We will show that for every large enough n a restriction chosen according to the distribution has a positive probability that

- D_n^ρ computes f on $m \geq \sqrt{n}/2$ inputs and
- the size of any 2-circuit of D_n^ρ is bounded by a constant independent of n (but may depend on c, the constant chosen in the first part).

In this case, we say that ρ *fails* if ρ assigns fewer than $\sqrt{n}/2$ $*$'s or some 2-circuit B^ρ depends on $\geq b_c$ inputs, where b_c will be a constant that we will specify later to fit our needs. Note that we are counting the number of inputs that B^ρ depends on and not it's size because if a 2-circuit depends on b_c inputs then it's size can be at most $b_c \cdot 2^{b_c}$. Therefore, a constant bound on the number of inputs B^ρ depends on is sufficient to give a constant bound on its size.

As in the first part, the probability of the former happening is bounded by $O(n^{-1/2})$. For the second part we prove by induction that

Claim 10. *For every constant c there exists a constant b_c such that if B is a 2-circuit with 1-circuits of size $\leq c$ then $Pr[B^\rho$ depends on $\geq b_c$ inputs$] \leq o(n^{-k})$.*

Base case: For $c = 1$ we have that every 1-circuit that is a member of B has only one input, so all of the 1-circuits are unnecessary. Therefore, we can view B as the $\wedge$ of the inputs to the 1-circuits rather than the 1-circuits. Now we can argue dually to the first part of the proof to establish the existence of b_1, i.e. the only reasoning that changes is in the wide case where now B is not forced if and only if no input of B is assigned a 0.

Inductive step: Assume that b_{c-1} exists. Call two 1-circuits *disjoint* if they have no common inputs. Let $b = 2k \cdot 4^c$. Again, consider a wide case where B has $\geq b \ln n$ disjoint 1-circuits and a narrow case with $< b \ln n$ disjoint 1-circuits. The constant b was chosen as a convenient delineation of "wide" versus "narrow" so that the bounds in the wide case work out as desired.

(a) B is wide. As before in this case, we show that the probability that a member of B is not forced is small.

$$\begin{aligned}
Pr[B \text{ is not forced}] &= Pr[\text{none of } B\text{'s 1-circuits are forced to } 0] \\
&\leq Pr[\text{1-circuit of size } c \text{ not forced to } 0]^{|\{B\text{'s members}\}|} \\
&\leq Pr[\text{1-circuit of size } c \text{ not forced to } 0]^{b \ln n} \\
&\leq \left[1 - \left(\frac{1 - 1/\sqrt{n}}{2}\right)^c\right]^{b \ln n} \\
&\leq \left(1 - \frac{1}{4^c}\right)^{b \ln n} \qquad \text{for } n \geq 4 \\
&= n^{b \ln\left(1 - 4^{-c}\right)} \\
&\leq n^{-b\left(4^{-c}\right)} \\
&= o(n^{-k}).
\end{aligned}$$

(b) B is narrow. Let $\{A_i\}$ be the 1-circuit members of B. Consider a maximal collection $\mathcal{B}$ of disjoint A_i i.e. adding any other A_j to $\mathcal{B}$ implies that some two 1-circuits in $\mathcal{B}$ share at least one common input. Let H be the set of inputs occurring in $\mathcal{B}$. We have that $\mid H \mid < c \cdot b \ln n$ because B is narrow and each $A \in \mathcal{B}$ has size at most c. Note that for every i, A_i has at least one input variable that is in H because if it did not then A_i could be added to our collection $\mathcal{B}$, contradicting its maximality. Say that H *hits* every member of B.

Now consider B^ρ for some restriction ρ and let h be the set of variables in H that are assigned $*$. Let $\rho_1, \rho_2, \dots, \rho_{2^h}$ be the 2^h restrictions obtained by "extending" ρ so that every variable in h is assigned a 0 or a 1, i.e. ρ_1 assigns all of the variables in h to 0's, ρ_2 assigns all but one of the variables in h to 0's, and so on. Note that for each ρ_i the only $*$'ed inputs to members of B^{ρ_i} may occur outside of H, that is in a member of B not in $\mathcal{B}$.

Therefore, B^ρ depends on the inputs h, as well as some more inputs in each B^{ρ_i}. We would like to bound the probability that B^ρ depends on $> 4k + 2^h \cdot b_{c-1}$ inputs.

We claim that each member of B^{ρ_i} has size $\leq c-1$. This is the case because H hits every member of B: given a member A of B

- if the original restriction ρ assigns a 1 or a 0 to any of the inputs of A, then A has size $\leq c-1$.
- Otherwise, ρ assigned all $*$'s to the inputs of A. But since H hits every A, there is at least one variable in h that is an input of A. Therefore, ρ_i sets this input to a 1 or a 0, reducing the size of A to $\leq c-1$.

Now by the inductive hypothesis conclude that for every i

$$Pr[B^{\rho_i} \text{ depends on } > b_{c-1} \text{ inputs}] < o(n^{-k}). \tag{1}$$

When does B^ρ depend on more than $4k + 2^h \cdot b_{c-1}$ inputs? This occurs if $\mid h \mid > 4k$ or for some i, $B^{\rho_i} > b_{c-1}$. Bound the probability that $\mid h \mid > 4k$ by a similar method as before:

$$\begin{aligned} Pr[\rho \text{ assigns } \geq 4k * \text{'s to } H] &\leq \binom{cb \ln n}{4k} \left(\frac{1}{\sqrt{n}}\right)^{4k} && (2)\\ &\leq (cb \ln n)^{4k} n^{-2k} && (3)\\ &= (c2k4^c \ln n)^{4k} n^{-2k} && (4)\\ &= o(n^{-k}). && (5) \end{aligned}$$

Since $2^h \leq 2^{4k}$ whenever $h \leq 4k$, we can set $b_c = 4k + 2^{4k} \cdot b_{c-1}$. Combining (1) and (5) we have that

$$\begin{aligned} Pr[B^\rho \text{ depends on } < b_c \text{ inputs}] &\leq Pr[\mid h \mid > 4k] + 2^h \cdot Pr[B^{\rho_i} \text{ depends on } > b_{c-1} \text{ inputs}] \\ &\leq o(n^{-k}) + 2^{4k} o(n^{-k}) \\ &= o(n^{-k}). \end{aligned}$$

Thus, we have proved the claim and can conclude that for large enough n there exists a ρ that does not fail. Given a circuit D_n from our previous family, if n is large enough, there exists a restriction ρ such that D_n^ρ computes f on at least $m \geq \sqrt{n}/2$ inputs and has constant sized 2-circuits. As before, we can make a polynomial size family $E_1, E_2, \ldots$ that computes f such that each E_i has constant size 2-circuits.

Finally, modify each E_n by switching every 2-circuit that is an $\wedge$ of $\vee$'s to a circuit that is an $\vee$ of $\wedge$'s using the distributive law. The size of the resulting circuit increases by at most a constant factor. Now there are two adjacent levels of $\vee$-circuits because $d \geq 3$, which are merged together. We have a circuit with a first level consisting of $\wedge$-circuits, so we use DeMorgan's laws to exchange the $\wedge$ and $\vee$ levels to obtain a $d-1$-circuit of appropriate form that computes f. The resulting family is still polynomial size, contradicting the minimality of d. □

An exponential lower bound was later proved by Yao [Yao1]. Then Hastad identified the main idea of these proofs, now called Hastad switching, to give an optimal bound [Has1]. Hastad switching is an important concept in its own right, but the proofs of these tighter bounds still rely on the crucial property that an $\mathtt{AC}^0$ circuit can be put into an alternating form. Although the proof just given appears to rely on technical details, and a non-constructive existence argument to derive a contradiction, we will see that it adheres to the Razborov, Rudich general framework.

1.2 Natural Property

By examining lower bound proofs such as the one proved above, Razborov and Rudich [RR] discovered a common characteristic: the proofs all identify a property of a candidate Boolean function and show that circuits of a given complexity class could not possibly decide a function with this property. Therefore, the proofs do not rely on all of the details concerning the candidate function, but rather on the details of a function property that many other Boolean functions might have as well. Even more surprisingly, in every lower bound proof thus far, the property also satisfies two criteria, and they call a property satisfying this criteria a natural property. Following [RR], we give the definitions and argue that the $\mathtt{AC}^0$ proof fits their framework.

Let F_n denote the set of all Boolean functions $f : \{0,1\}^n \to \{0,1\}$. A function $f : \{0,1\}^n \to \{0,1\}$ is completely specified by its truth table, a 2^n long string such that the

ith bit represents the value f takes on the ith lexicographic string in $\{0,1\}^n$. From this specification note that $\mid F_n \mid = 2^{2^n}$. If we provide as input to an algorithm a Boolean function f then we can imagine the input encoded as a truth table, so that the input has size 2^n.

Definition 11 ([RR]). A collection of subsets $\{C_n \mid C_n \subseteq F_n\}_{n\in\mathbb{N}}$ is a *property of functions*. The property $\{C_n\}$ is *useful against* $\mathtt{P}_{/\mathtt{Poly}}$ if the following holds.

(a) Let $f_1, f_2, \ldots$ be a sequence of functions such that $f_n \in C_n$. Then for every k there exists an N such that for every $n > N$, f_n cannot be computed by circuits of size n^k.

A property of functions is a *natural property* if there exists a collection of subsets $\{C_n^* \mid C_n^* \subseteq C_n\}$ such that

(b) given some $f : \{0,1\}^n \to \{0,1\}$ there exists an algorithm running in $\text{poly}(2^n) = 2^{O(n)}$ time that decides whether or not $f \in C_n^*$, and

(c) there exists a constant c such that $\frac{|C_n^*|}{|F_n|} \geq \frac{1}{2^{cn}}$ for every large enough n.

More generally, if the algorithm in (b) for deciding the predicate $f \in C^*$ is in complexity class Γ, then we say that C_n is Γ-natural. Furthermore, for a complexity class Λ if instead of (a) we have that for any sequence $f_1, f_2, \ldots$ with $f_n \in C_n$ that $f_1, f_2, \ldots = f \notin \Lambda$, then C_n is useful against Λ.

Criterion (a) gives a way for showing that a function has superpolynomial circuit complexity. If (a) holds for some property $\{C_n\}$, then for any family of Boolean functions $f = f_1, f_2, \ldots$ such that each f_n has property C_n, $f \notin \mathtt{P}_{/\mathtt{Poly}}$. Hence, C_n is *useful against* $\mathtt{P}_{/\mathtt{Poly}}$ because if we can succeed in identifying a family of Boolean functions each of which has the useful property then we have identified a family with superpolynomial circuit complexity.

Criterion (b) says that there exists a polynomial size circuit that given a Boolean function in the form of a truth table decides whether f has the property C_n^*, i.e. deciding the predicate $f \in C_n$ is in $\mathtt{P}_{/\mathtt{Poly}}$. If (b) holds we say that C_n has *constructvity.* In other words, if a useful property $\{C_n\}$ is natural, then it comes with an efficient built in algorithm.

The next requirement (c) says that $\mid C_n^* \mid \geq \frac{2^{2^n}}{2^{cn}} = 2^{2^n - cn}$, which says that the subsets C_n^* are large. Alternatively, it is convenient to interpret this as saying that a randomly chosen function $f \in F_n$ has a non-negligible probability of having property C_n^*. If (c) holds we say that C_n has *largeness.* The necessity of the distinction between C_n and C_n^* will become evident when we examine a second lower bound.

Up until this point the definitions given have been entirely rigorous. More loosely, we say that a lower bound proof is $\mathtt{P}_{/\mathtt{Poly}}$-*natural against* $\mathtt{P}_{/\mathtt{Poly}}$ if it identifies a $\mathtt{P}_{/\mathtt{Poly}}$-natural property useful against $\mathtt{P}_{/\mathtt{Poly}}$. This is not an entirely rigorous statement because it is not always clear whether a proof explicitly defines a natural property or, if it does, whether it is necessary in the proof at all. In particular, we will examine a natural proof where it is a non-trivial task to identify $\{C_n^*\}$. However, with practice and enough experience one can usually tell when a proof is "natural". On the other hand, it is difficult to see which potential arguments may withstand "being naturalized", other than by trying some choices of $\{C_n^*\}$ to see if they fail.

Razborov and Rudich claim that all circuit lower bound proofs have gone according to the following scheme ($\star$):

- identify some property of a problem f,
- show that *any* function with this property cannot be computed by circuits from the class Γ,
- thereby conclude that $f \notin \Gamma$.

Is this the case for our proof that $\mathsf{Parity} \notin \mathtt{AC}^0$? It may seem that our proof was specific to the Parity function, but in reality we worked exclusively with the circuit model after we established a basic fact about Parity. Recall that the main property of $\mathsf{Parity} = \{f_n\}$ that we utilized in the proof is that, for each f_n and any ρ leaving at least $\frac{\sqrt{n}}{2}$ free variables, f_n^ρ does not become the constant function, i.e. is not trivialized. Let $\{C_n\}$ be this property. It was not at all straight forward to show that $\{C_n\}$ is useful against $\mathtt{AC}^0$, but despite all of the details, $\{C_n\}$ allows us to reuse the restricted circuit associated to f_n^ρ to construct a new family of circuits. By exploiting the alternating properties of $\mathtt{AC}^0$ we eventually obtain a contradiction. Thus, the proof given above is precisely the proof that the property $\{C_n\}$ is useful against $\mathtt{AC}^0$ because it shows that for a sequence $f = f_1, f_2, \ldots$ such that $f_i \in C_i$ for all i, $f \notin \mathtt{AC}^0$.

The claim that a lower bound result uses the scheme ($\star$) can be viewed as generalizing the result to any function that has the property $\{C_n\}$ useful against $\mathtt{AC}^0$. In fact, Furst, Saxe, and Sipser realized this (perhaps not as generally as Razborov and Rudich) by providing a corollary to their main result of $\mathsf{Parity} \notin \mathtt{AC}^0$.

Corollary 12. *Given any $p \geq 2$ and $a < p$ the Boolean function*

$$f(x) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} x_i \equiv a \mod p \\ 0 & \text{otherwise.} \end{cases}$$

is not in $\mathtt{AC}^0$.

In contrast to our proof by contradiction, the subsequent proof given by Hastad in [Has1] establishes the usefulness of $\{C_n\}$ against $\mathtt{AC}^0$ more directly. Using Hastad switching it can be shown by the probabilistic method that any $\mathtt{AC}^0$ circuit has a restriction ρ that trivializes the circuit, so it could not have possibly decided a function for which no such trivializing

ρ exists. See [AB, Chp 14], [Has1], [Has2] for this proof. The argument focuses even closer on the alternating properties of $\mathtt{AC}^0$ and quantifies more precisely the probabilities of commuting a row of ORs with a row of ANDs under random restrictions. Thus, the essence of the proofs is the same, except that the stronger proofs reveal the useful property more directly. We gave the historically earlier proof to emphasize that at first it may not be obvious that the argument adheres to this framework

Not surprisingly, demonstrating the usefulness of a property $\{C_n\}$ against a complexity class Γ has more to do with studying the properties of Γ rather than the candidate function. Proving naturality, on the other hand, deals with the functions rather than the complexity class. In examining the definition of a natural property useful against $\mathtt{P}_{/\mathtt{Poly}}$ we see that the scheme $(\star)$ only uses condition (a): we only need the usefulness against some circuit class Γ for the scheme to succeed in proving a lower bound, and could care less about any other conditions that the property may satisfy. Razborov and Rudich's main insight is that all lower bound proofs not only use $(\star)$, but in doing so identify a property that is also natural, i.e. the useful property also has conditions (b) and (c), which turn out to be very important in relation to pseudorandom generators.

This leaves the question of whether the property that we used in our proof is natural. To show this we need to identify a collection of subsets $\{C_n^* \subseteq C_n\}$. In this case, setting $C_n^* = C_n$ suffices. To see constructivity consider the following circuit D that takes as input a truth table for a function f_n and decides $f_n \overset{?}{\in} C_n$. There are $\binom{n}{\sqrt{n}/2} \cdot 2^{n-\sqrt{n}/2} = 2^{O(n)}$ restrictions that leave $\sqrt{n}/2$ free variables. Fix a restriction ρ. We construct a subcircuit E_ρ that is true if and only if f_n^ρ is nontrivial. The truth table of f_n^ρ consists of $2^{\sqrt{n}/2}$ bits of the 2^n long truth table of f_n. Take the appropriate inputs $z_1, \ldots, z_{2^{\sqrt{n}/2}}$ from the inputs to D and wire them into E_ρ. Then let E_ρ be the OR of $z_i \oplus z_j$ for every $i \neq j$ (where $\oplus$ is exclusive-or). There are $\binom{2^{\sqrt{n}/2}}{2} = poly(2^n)$ pairs, so the size of E_ρ is polynomial in 2^n and the depth is 3. Let D output the AND of all of the E_ρ's. Since there are $poly(2^n)$ ρ's, all of D has size $poly(2^n)$ and depth 4, so deciding the predicate $f_n \in C_n^*$ is in $\mathtt{AC}^0$.

To see largeness intuitively, let f be a random function. Then a restriction ρ that leaves some variables free when applied to f produces a random function, but there are only two constant functions. More formally, let ρ be a restriction that leaves $m = \sqrt{n}/2$ free variables. Consider how many functions f_n this ρ can trivialize. The truth table of f_n^ρ consists of 2^m bits all of which would have to be equal. The remaining $2^n - 2^m$ bits of the original truth table of f_n could have been anything. Therefore, a single ρ can trivialize $2 \cdot (2^n - 2^m)$ function f_n. Since there are $\binom{n}{m}2^{n-m}$ restrictions ρ, there are $\binom{n}{m}2^{n-m} \cdot 2 \cdot (2^n - 2^m)$ functions that are trivialized by some ρ. Therefore,

$$2^{2^n} - \binom{n}{m}2^{n-m} \cdot 2 \cdot (2^n - 2^m) = 2^{2^n} - 2^{O(n)}$$

functions f_n have property C_n^*. This gives $\frac{|C_n^*|}{|F_n|} = \frac{2^{2^n} - 2^{O(n)}}{2^{2^n}}$, which is certainly greater than $\frac{1}{2^{c \cdot n}}$ for some constant c.

To summarize this is an $\texttt{AC}^0$-natural proof against $\texttt{AC}^0$. Showing the naturality of C_n was not particularly difficult, but we will see another lower bound example where this process is not as straight forward. This second example will be a $\texttt{NC}^2$-natural proof. First we prove the main result regarding pseudorandom generators.

1.3 Pseudorandom Generators

Razborov and Rudich's main result is that a natural property against $\texttt{P}_{/\texttt{poly}}$ contradicts the existence of pseudorandom generators. It turns out that this proof relies on the ideas in [GGM] of constructing a pseudorandom *function* generator from a pseudorandom generator. Therefore, we first give the construction of a pseudorandom function generator from a pseudorandom generator following the original method of [GGM]. This proof illustrates the main ideas necessary to establish the main result. We begin by defining pseudorandom generators according to Blum and Micali [BM].

Definition 13 ([BM])**.** A family of Boolean functions $G_n : \{0,1\}^n \to \{0,1\}^{2n}$ is a *pseudorandom generator* (PRG) if for every polynomial $Q(n)$ and any probabilistic polynomial time algorithm A

$$\mid p^A_{2n} - p^A_{G_n} \mid \leq \frac{1}{Q(n)}$$

for every n large enough, where p^A_{2n} is the probability that A outputs 1 when given a random input from the uniform distribution on string of length $2n$ and $p^A_{G_n}$ is the probability that A outputs 1 when given an input produced by applying G_n to a random input from the uniform distribution on strings of length n.

A PRG can be thought of as taking a truly random string and producing a longer string of length $l(n)$ that is random enough to fool polynomial time probabilistic Turing machine. The function $l(n)$ ($2n$ in our definition) is referred to as the *stretch* of G.

There is a stronger definition in relation to the non-uniform circuit model that quantifies the hardness of a pseudorandom generator. This definition is usually used in the context of hardness-randomness tradeoffs and derandomization. We will need this particular formulation in the proof of the main theorem.

Definition 14 ([BM])**.** The *hardness* of a generator $G_n : \{0,1\}^n \to \{0,1\}^{2n}$, denoted $H(G_n)$, is the smallest $S(n)$ such that there exists a family $\{C_n\}$ of circuits of size at most $S(n)$ such that

$$\mid Pr_{x\in\{0,1\}^n}[C(G_n(x)) = 1] - Pr_{y\in\{0,1\}^{2n}}[C(y) = 1] \mid \geq \frac{1}{S(n)}.$$

Alternatively, $H(G_n)$ can be viewed as the largest $S(n)$ such that for all circuit families $\{C_n\}$ of circuits of size at most $S(n)$,

$$\mid Pr_{x\in\{0,1\}^n}[C(G_n(x)) = 1] - Pr_{y\in\{0,1\}^{2n}}[C(y) = 1] \mid < \frac{1}{S(n)}$$

There are many different ways to define pseudorandom generators. In these two definitions we have considered probabilistic algorithms, as well as the circuit model. Another possible definition provides a whole set of polynomially many strings as input to the probabilistic Turing machine. Yao showed in [Yao2] that this definition is equivalent with the one we give.

Under a hardness assumption for the discrete logarithm problem Blum and Micali showed in [BM] how to construct a pseudorandom generator. Then Goldreich, Goldwasser, and Micali [GGM] introduced pseudorandom function generators and showed how to construct them from pseudorandom generators. We follow their proof in the next section, after providing their definitions.

Let H_n be the set of functions $f : \{0,1\}^n \to \{0,1\}^n$ so that $\mid H_n \mid = 2^{n\cdot 2^n}$, and let $H = \{H_n\}$.

Definition 15. Let $F = \{F_n \subseteq H_n\}$ be a collection of subsets of functions. Let A be a probabilistic polynomial-time algorithm that on input 1^n and access to an oracle computing $f : \{0,1\}^n \to \{0,1\}^n$ outputs a 1 or a 0. We say that F *passes test* A if for every polynomial $Q(n)$ and every large enough n

$$\mid p_n^F - p_n^H \mid \leq \frac{1}{Q(n)},$$

where p_n^F denotes the probability that A^f on input 1^n outputs 1 for a randomly chosen $f \in F_n$ and p_n^H denotes the probability that A^f on input 1^n outputs 1 for a randomly chosen $f \in H_n$. Note that the probabilities are taken over the random choices of the function and the internal randomness of the algorithm A.

Intuitively, we can think of the test A as being given some function $f : \{0,1\}^n \to \{0,1\}^n$, being allowed to ask about polynomially many values of f (perhaps determined probabilistically), while doing some other probabilistic computation, and finally outputting its verdict as to whether f came from F_n or H_n.

Definition 16. We say that a collection F is a *pseudorandom collection of functions* if it passes all probabilistic polynomial-time tests.

We would like to be able to construct the functions in a pseudorandom collection efficiently and for the functions to be themselves efficiently computable. In particular, let each function in F_n be indexed by a n-bit string x, so we can denote each function as f_x for some $x \in \{0,1\}^n$.

Definition 17 ([GGM]). Let F be a pseudorandom collection. A *pseudorandom function generator* is a polynomial time algorithm that on input $x \in \{0,1\}^n$, $y \in \{0,1\}^n$ outputs $f_x(y)$ for an $f_x \in F_n$.

We show that such a construction is possible assuming the existence of a pseudorandom generator of stretch $2n$.

1.3.1 Construction of Pseudorandom Function Generator

Let $G_n : \{0,1\}^n \to \{0,1\}^{2n}$ be a pseudorandom generator. We can think of n as fixed and suppress the indexing. Let $G^0, G^1 : \{0,1\}^n \to \{0,1\}^n$ be the "first-half" and "second-half" functions such that if $G(x) = y_1 \cdots y_n y_{n+1} \cdots y_{2n}$ then $G^0(x) = y_1 \cdots y_n$ and $G^1(x) = y_{n+1} \cdots y_{2n}$. Consider an n-bit string $b = b_n \cdots b_1$. Define

$$G^b(x) = G^{b_n}(G^{b_{n-1}}(\cdots(G^{b_1}(x))\cdots)).$$

Let $f_x(b) = G^b(x)$. See figure 1 for an intuitive understanding of this construction.

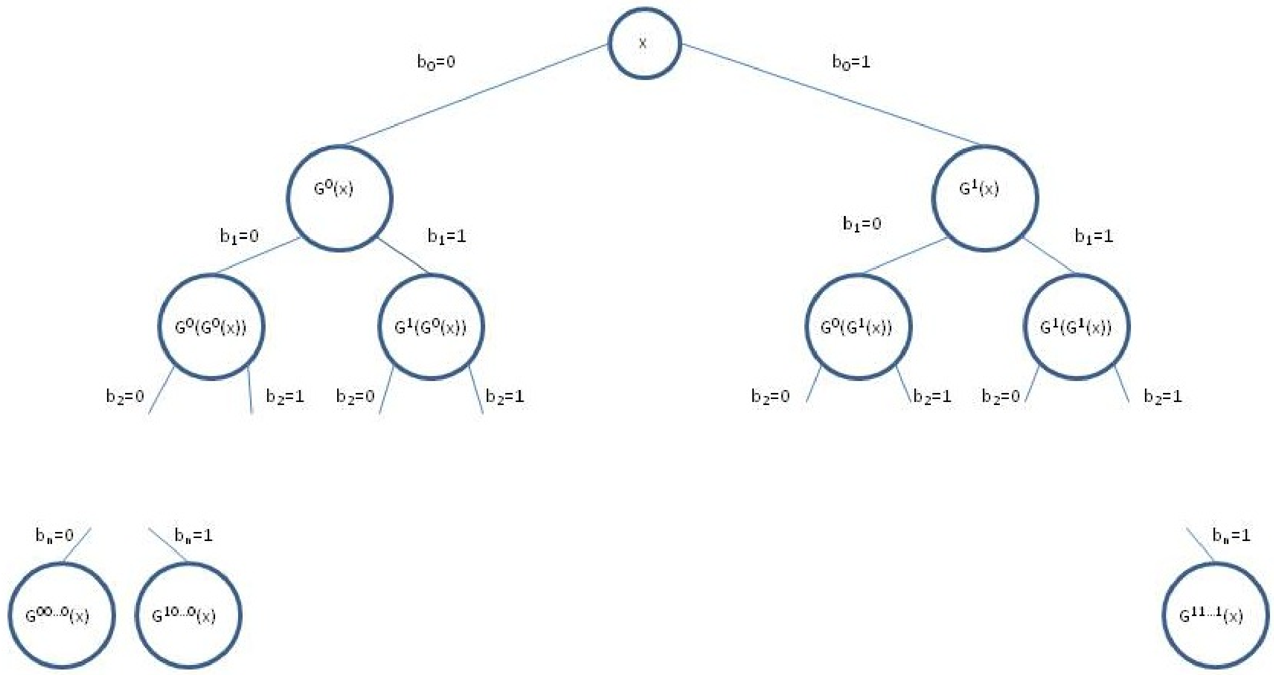

Figure 1: Tree representing the pseudorandom collection of functions as in [GGM]

In the tree diagram, one may think of x as an input token and the string b as the trail one needs to follow to reach the correct leaf that holds the output value. The roles of x and b are such that we view the token x as the seed in our random function generator and we view the trail b as the input to the function. In particular, for any $x, b \in \{0,1\}^n$ $f_x(b) = G^b(x)$ is computable in $n \cdot p(n)$ time if G is computable in $p(n)$ time. Thus, if the subsets $\{f_x\}_{x\in\{0,1\}^n} = F_n$ form a pseudorandom collection, we have a pseudorandom function generator.

Theorem 18 ([GGM]). *Assuming that G is a pseudorandom generator, the collection $F_n = \{f_x \mid x \in \{0,1\}^n\}$ constructed above is a pseudorandom collection.*

Proof. The proof is by contradiction. Suppose that there exists a probabilistic polynomial-time algorithm T and a polynomial $p(n)$ such that

$$\mid p_n^F - p_n^H \mid > \frac{1}{p(n)}$$

for infinitely many n. Here p_n^F is the probability T outputs 1 on input 1^n given an oracle computing a randomly selected $f_x \in F_n$, and p_n^H is the probability T outputs 1 on input 1^n given an oracle computing a randomly selected $f \in H_n$. Using this distinguisher algorithm for functions we will create a distinguishing algorithm for distributions. The algorithm A_T will distinguish the uniform distribution on strings of length $2n$ from the distribution on strings of length $2n$ created by G, thereby contradicting G's pseudorandomness. Let $Q(n)$ be the polynomially many queries that T makes on input 1^n. Consider T's computation when its queries are answered by the probabilistic algorithms M_i defined bellow.
M_i answers one of T's queries $y = y_1 y_2 \cdots y_n$ according to the following procedure:

- **if** y is the first occurrence with an initial segment of $y_1 y_2 \cdots y_i$
  then randomly select an $r \in \{0,1\}^n$, store the tuple $(y_1, \ldots, y_i, r)$, and output $G^{y_n \cdots y_{i+1}}(r)$.
- **else** look up $(y_1, \ldots, y_i, r)$ and output $G^{y_n \cdots y_{i+1}}(r)$.

Extend the meaning of G^y to $G^y(r) = r$ when y is the empty string, so that M_n is well-defined. We can imagine the M_i's as starting their computation of G^y a little late by filling in the ith level of the tree in the diagram with random values and proceeding from there. The i determines the different levels of lateness. The purpose of the definition becomes clear when considering M_0 and M_n. In particular, M_0 when queried by $y = y_1 \cdots y_n$ returns $G^y(r)$ for a random r, but this is precisely $f_r(y)$ for a random r, i.e. a function from our collection $\{f_x \mid x \in \{0,1\}^n\}$ chosen at random. On the other hand, M_n when queried on y chooses a random string $r \in \{0,1\}^n$ and outputs r. The machine M_n answers queries as a random function uniformly selected from H_n.

Let p_n^i be the probability that T outputs a 1 on input 1^n with M_i answering its queries. In particular, $p_n^0 = p_n^F$ and $p_n^n = p_n^H$ as we just reasoned. The probabilistic polynomial-time algorithm A_T is defined as follows. A_T is given a set S_n of $Q(n)$ strings each of length $2n$ (recall T makes $Q(n)$ queries on input 1^n). On input $\langle 1^n, S_n \rangle$, A_T does the following:

(a) chooses uniformly at random a value $i \in \{0, \ldots, n-1\}$,

(b) gives T the input 1^n and responds to T's queries as follows:
on query $y = y_1 \cdots y_n$

- **if** y is the first occurrence with an initial segment of $y_1 y_2 \cdots y_i$
then A_T picks the next $s_0 s_1 = s \in S_n$, stores the tuples $(y_1, \ldots, y_i, 0, s_0), (y_1, \ldots, y_i, 1, s_1)$ and outputs $G^{y_n \cdots y_{i+2}}(s_0)$ if $y_{i+1} = 0$ or outputs $G^{y_n \cdots y_{i+2}}(s_1)$ if $y_{i+1} = 1$,
- **else** A_T looks up $(y_1, \ldots, y_i, y_{i+1}, u)$ and returns $G^{y_{i+2}}(u)$.

The key observation is that if S_n is a set of strings selected randomly from the uniform distribution on strings of length $2n$ then A_T simulates T with responses given by M_{i+1}. On the other hand, if S_n is a set of strings generated by G then A_T simulated T with responses given by M_i. To see this note that if A_T randomly selects i then it is filling in the tree at level $i+1$ with pairs of strings obtained by breaking up a string s of length $2n$. If s is randomly selected according to the uniform distribution then both s_0, s_1 are uniformly random, so the $i+1$th level will be assigned uniformly randomly. On the other hand, if s is generated by G then the $i+1$th level will be filled in with pairs $s_0 = G^0(r)$, $s_1 = G^1(r)$ where r was chosen uniformly from $\{0,1\}^n$, but this is the same as if we filled in the ith level with uniformly chosen r's.

Therefore,

- If the set S_n is a randomly selected subset of the length $2n$ strings generated by G then the probability A_T outputs 1 is
$$\sum_{i=0}^{n-1} \frac{1}{n} \cdot p_n^i.$$
- If the set S_n is a randomly selected subset of all strings of length $2n$ then the probability A_T outputs 1 is
$$\sum_{i=0}^{n-1} \frac{1}{n} \cdot p_n^{i+1}.$$

The difference in these probabilities is

$$\frac{1}{n} \mid p_n^0 - p_n^n \mid = \frac{1}{n} \mid p_n^F - p_n^H \mid > \frac{1}{np(n)}$$

for infinitely many n. Therefore, the output of A_T non-negligibly differs for infinitely many n when given a subset of the uniform distribution as opposed to a subset of the distribution created by G. □

The ideas of this proof will play a key role in the next section.

1.4 Main Result

The proof that a $\mathtt{P}_{/\mathtt{poly}}$-natural property useful against $\mathtt{P}_{/\mathtt{poly}}$ contradicts the existence of pseudorandom generators is very similar to the proof just given. In particular, an almost identical pseudorandom function generator is constructed. The natural property is then used as a test to distinguish between these functions and truly random functions. Using similar ideas to the previous proof, this distinguisher of distributions on functions is used to distinguish distributions on strings. Recall the definition of the hardness of a generator.

Definition 19. The *hardness* of a generator $G_n : \{0,1\}^n \to \{0,1\}^{2n}$ denoted $H(G_n)$ is the smallest $S(n)$ such that there exists a family $\{C_n\}$ of circuits of size at most $S(n)$ such that

$$\mid Pr_{x\in\{0,1\}^n}[C(G_n(x)) = 1] - Pr_{y\in\{0,1\}^{2n}}[C(y) = 1] \mid \geq \frac{1}{S(n)}$$

It is a widely believed conjecture that there exist generators of hardness 2^{n^ϵ} for some fixed constant ϵ. This would be impossible if there exists a $\mathtt{P}_{/\mathtt{poly}}$-natural proof against $\mathtt{P}_{/\mathtt{poly}}$.

Theorem 20. *[RR] If there is a lower bound proof that is $\mathtt{P}_{/\mathtt{Poly}}$ natural against $\mathtt{P}_{/\mathtt{Poly}}$ then for every generator $G_n : \{0,1\}^n \to \{0,1\}^{2n}$, $H(G_n) \leq 2^{n^{o(1)}}$.*

Proof. To get a contradiction suppose that there is such a proof that identifies a natural property C_k and $C_k^* \subseteq C_k$ is the subset having largeness and constructivity. Without loss of generality we may assume that $C_k = C_k^*$ to begin with. Recall the definition:

- Constructivity: Deciding $f \in C_k$ can be done in $\mathtt{P}_{/\mathtt{Poly}}$, so we can abuse notation to have C_k also denote a polynomial size circuit deciding membership in the subset C_k given the truth table of a Boolean function on k inputs.

- Largeness: At least a $\frac{1}{2^{ck}} = \frac{1}{2^{O(k)}}$ fraction of all functions on k inputs have property C_k for some constant c and large enough k. Or in the alternative interpretation, C_k accepts at least $\frac{1}{2^{ck}}$ inputs out of all possible Boolean functions on k inputs.

Let $G_n : \{0,1\}^n \to \{0,1\}^{2n}$ be a pseudorandom generator. We will use C_k to construct an algorithm that distinguishes between the strings produced by G_n and all strings uniformly distributed. Ultimately, we would like to prove that for any $\epsilon > 0$ there exists a distinguisher D_n of size at most $S_n = 2^{O(n^\epsilon)}$ such that for infinitely many n

$$\mid Pr_{x\in\{0,1\}^n}[D_n(G_n(x)) = 1] - Pr_{y\in\{0,1\}^{2n}}[D_n(y)] \mid > 1/2^{O(n^\epsilon)}.$$

In other words, for every ϵ circuits of size $2^{O(n^\epsilon)}$ are already big enough to tell apart the outputs of the generator from truly random strings with nontrivial probability. Therefore the hardness of the generator is at most $2^{O(n^\epsilon)}$ for every ϵ, implying $H(G_n) \leq 2^{n^{o(1)}}$.

Since C_k is a property of functions and not strings, we first show that C_k can be used to distinguish functions that are constructed from G_n from all functions uniformly distributed.

Let $\epsilon > 0$ and fix $k = \lfloor n^\epsilon \rfloor$. Define the pseudorandom function generator $x \in \{0,1\}^n \mapsto f_x \in F_k$ where $f_x : \{0,1\}^k \to \{0,1\}$ such that on any input $y \in \{0,1\}^k$

$$f_x(y) = \text{ the first bit of } G^{y_k}(\cdots(G^{y_2}(G^{y_1}(x)))\cdots).$$

This is exactly as our previous construction except that the seeds are length n, the inputs are length $k = n^\epsilon$, and the output is just the first bit. Therefore, instead of a full binary tree of depth n as in figure 1, we have a full binary tree of depth k. For any $x \in \{0,1\}^n$, $y \in \{0,1\}^k$ $f_x(y)$ can be computed in polynomial time as before. Therefore, fixing any x, f_x has polynomial size circuits. Since the natural property C_k is useful against $\mathtt{P}_{/\mathtt{Poly}}$, $f_x \notin C_k$ for any x when k is large enough. Interpreting C_k as a circuit of size $2^{O(n)}$, we have that for large enough k, $C_k(f_x) = 0$. On the other hand, by largeness $C_k(f) = 1$ for at least $\frac{1}{2^{ck}}$ Boolean functions f on k inputs for large enough k. This gives us a statistical test: there exists K such that for all $k > K$

$$\mid Pr_{f:\{0,1\}^k \to \{0,1\}}[C_k(f) = 1] - Pr_{x \in \{0,1\}^{k^{1/\epsilon}}}[C_k(f_x) = 1] \mid > \frac{1}{2^{O(k)}} \tag{6}$$

Note that in contrast to the formulation in the previous proof, where the statistical tests were Turing machines given oracle access to functions, this formulation consists of statistical tests that are circuits provided with the entire truth table of a function as input.

We would like to use the family C_k to construct a family of circuits D_n that distinguishes the distribution of strings. We cannot use the same exact strategy as in the previous proof because there we were allowed to construct a probabilistic algorithm. However, we may do something similar to defining M_i's, algorithms that fill in level i of the binary tree with random strings and begin computing G^y from there. In this case however, we do not fill in all levels after a depth i, but rather fill in all levels after a depth i along with some nodes in level $i-1$.

Consider the full binary tree T of depth k, which has $2^{k+1} - 1$ total nodes, 2^k leaf nodes denoted L, and $2^k - 1$ internal (non-leaf) nodes. We number all of the internal nodes $v_1, v_2, \ldots, v_{2^k-1}$ such that v_i a child of $v_j \implies i < j$ (See figure 2 for an example with $k = 4$).

Given a subset $\{v_1, \ldots, v_i\}$, let T_i be the union of maximal subtrees in T consisting of the nodes $\{v_1, \ldots, v_i\} \cup L$. Given a leaf $l \in L$ let $v_i(l)$ be the root of the maximal subtree in T_i containing l and $d(i,l)$ the distance from $v_i(l)$ to l. In our example, if $i = 10$ then for a leaf l on the left half of the tree we have $v_{10}(l) \in \{v_9, v_{10}\}$ and $d(10,l) = 2$, while for a leaf l' on the right half of the tree we have $v_{10}(l') \in \{v_5, v_6, v_7, v_8\}$ and $d(10,l') = 1$.

Note that a string y of length k is equivalent to a leaf node l because it specifies the path taken to reach l. We define a set of functions $F_{i,n}$ for every $i \in \{0, \ldots, 2^k - 1\}$. Let $F_{i,n}$ be the set of functions f that for each subtree in T_i, f fills in the root node

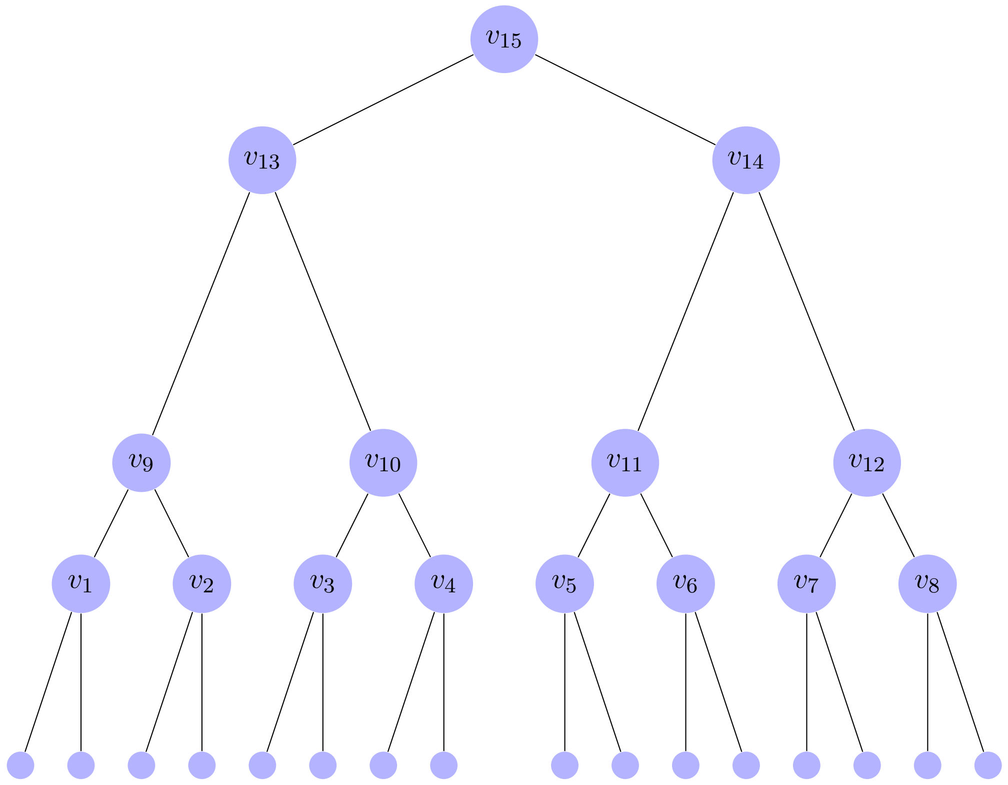

Figure 2: Example of tree numbering with k=4

with a string $x_{v_i} \in \{0,1\}^n$, and given input $y \in \{0,1\}^k$ corresponding to $l \in L$ computes $G^{y_n}(\dots(G^{y_{n-d(i,l)+1}}(x_{v_i(l)}))\dots)$. The idea is the same as before, each $f \in F_{i,n}$ fills in the root nodes of the subtrees up to index i with some set of strings from $\{0,1\}^n$ and picks up the computation of G^y from there.

Choosing a function f_i randomly from $F_{i,n}$ amounts to filling in the $x_{v_i(l)}$ randomly. Selecting randomly a function $f \in F_{0,n}$ is equivalent to selecting a random function $h : \{0,1\}^k \to \{0,1\}$. On the other hand, selecting a random function from $F_{2^k-1,n}$ is the same as randomly selecting an $x \in \{0,1\}^n$ to get f_x. Under this notation 6 can be written as

$$\mid Pr_{f \in F_{0,n}}[C_k(f) = 1] - Pr_{f \in F_{2^k-1,n}}[C_k(f) = 1] \mid > \frac{1}{2^{O(k)}},$$

so there exists a constant α and there exists a K such that $\forall k \geq K$

$$\mid Pr_{f \in F_{0,n}}[C_k(f) = 1] - Pr_{f \in F_{2^k-1,n}}[C_k(f) = 1] \mid > \frac{1}{2^{\alpha k}}.$$

Therefore, for all $k \geq K$ there exists some i such that

$$\mid Pr_{f \in F_{i,n}}[C_k(f) = 1] - Pr_{f \in F_{i+1,n}}[C_k(f) = 1] \mid > \frac{1}{2^{(\alpha+1)(k)}}.$$

To see this let $p_{i,n} = Pr_{f \in F_{i,n}}[C_k(f) = 1]$. Intuitively, if we can't tell apart any two $F_{i,n}$, $F_{i+1,n}$ with sufficient probability, then we would not have been able to tell apart $F_{0,n}$ and $F_{2^k-1,n}$ to begin with. So suppose the contrary that $\mid p_{i,n} - p_{i+1,n} \mid \leq \frac{1}{2^{(\alpha+1)k}}$ for all i infinitely often. Then for large enough k

$$\begin{aligned}
\frac{1}{2^{\alpha k}} &< \mid p_{0,n} - p_{2^k-1,n} \mid \\
&= \mid p_{0,n} - p_{1,n} + p_{1,n} - p_{2,n} + p_{2,n} \cdots - p_{2^k-2,n} + p_{2^k-2,n} - p_{2^k-1,n} \mid \\
&\leq \mid p_{0,n} - p_{1,n} \mid + \mid p_{1,n} - p_{2,n} \mid + \cdots + \mid p_{2^k-2,n} - p_{2^k-1,n} \mid \\
&\leq (2^k - 1)\frac{1}{2^{(\alpha+1)k}} \\
&\leq \frac{1}{2^{\alpha k}} - \frac{1}{2^{(\alpha+1)k}},
\end{aligned}$$

a contradiction.

Now consider T_{i+1} and note that v_{i+1} is a root node of a maximal subtree in T_{i+1}. Suppose that we fix the root node values x_v for every root node v in T_{i+1} except for v_{i+1}. Then this assignment is one string short of specifying a function $f \in F_{i+1,n}$. Let $F'_{i+1,n}$ be the new distribution that corresponds to selecting a final string $x_{v_{i+1}}$ to define a function $f \in F_{i+1,n}$. The same partial assignment is also two strings short of completely specifying a function $f \in F_{i,n}$ i.e. the children v' and v'' of v_{i+1} do not have strings $x_{v'}$ and $x_{v''}$

assigned to them yet. Consider the new distribution $F'_{i,n}$ such that choosing an $f \in F'_{i,n}$ corresponds to choosing these final two strings. Then we still have

$$| Pr_{f \in F'_{i,n}}[C_k(f) = 1] - Pr_{f \in F'_{i+1,n}}[C_k(f) = 1] | > \frac{1}{2^{O(k)}}.$$

This provides a statistical test D_n that distinguishes $G(x_{v_{i+1}}) \in \{0,1\}^{2n}$ from $x_{v'}x_{v''} \in \{0,1\}^{2n}$. Given a string $x \in \{0,1\}^{2n}$ the circuit D_n has hard wired part of the truth table that is deducible from the assignments of the root nodes of T_{i+1} other than v_{i+1} for appropriate i. Given $x = x_0x_1$ it fills in the missing portion of the truth table by assigning $x_{v'} = x_0$ and $x_{v''} = x_1$ and computing the missing values, which can certainly be done in $2^{O(n)}$ time. The truth table values are then fed into C_k and D_n outputs the same value as C_k. Therefore, we have a circuit family D_n such that

$$| Pr_{x \in \{0,1\}^n}[D_n(G_n(x)) = 1] - Pr_{y \in \{0,1\}^{2n}}[D_n(y) = 1] | > \frac{1}{2^{O(k)}} = \frac{1}{2^{O(n^\epsilon)}} \geq \frac{1}{| D_n |}.$$

Recall that epsilon was arbitrary, so the result follows. □

Therefore, proving a lower bound against $\mathtt{P_{/poly}}$ by identifying a natural property, produces an algorithm that can tell apart pseudorandom generated strings from random strings. Taking the contrapositive, if there exists a pseudorandom generator of hardness 2^{n^ϵ} for any constant ϵ, then there cannot be a $\mathtt{P_{/poly}}$ natural proof against $\mathtt{P_{/poly}}$. This result can be generalized to complexity classes Γ and Λ. If there exists pseudorandom generators computable in complexity class Λ that are secure against Γ algorithms, then there is no Γ natural proof against Λ.

1.5 $\mathsf{Mod_q} \notin \mathtt{ACC^0[p]}$

In this section, we examine a lower bound proof that is not obviously natural. Let $\mathsf{Mod_n}$ be the Boolean function that outputs 0 if and only if the sum of the inputs is congruent to $0 \mod n$.

Definition 21. The class $\mathtt{ACC^i[p]}$ is the class of languages that are decided by polynomial size circuits of depth $O(log^i(n))$ with AND, OR, Mod$_p$ gates of unbounded fan-in along with NOT gates.

So an $\mathtt{ACC^0[p]}$ circuit is just an $\mathtt{AC^0}$ circuit that is allowed gates computing $\mathsf{Mod_p}$ in one step. Building off the work of Razborov [Raz], Smolensky [Smo] proved that the $\mathsf{Mod_q}$ function cannot be computed by $\mathtt{ACC^0[p]}$ circuits where q and p are distinct primes. We give a proof of the special case where $q = 2$ and $p = 3$ following the proof given in [AB]. Note that when $q = 2$, $\mathsf{Mod_q}$ is just the Parity function from before.

The proof fits the Razborov, Rudich scheme for lower bounds: identify a property of your function, and show that a function with this property cannot have $\mathtt{ACC^0[p]}$ circuits. We prove the following where (b) is the property identification step, and (a) is the usefulness step.

(a) Any $\mathtt{ACC}^0[3]$ circuit can be "approximated" by a "low" degree polynomial over $\mathbb{F}_3$.

(b) Mod_2 cannot be approximated by a "low" degree polynomial over $\mathbb{F}_3$.

A polynomial $p(x_1, \ldots, x_n) : \mathbb{F}_3^n \to \mathbb{F}_3$ is said to approximate a Boolean function $f(x_1, \ldots, x_n) : \{0,1\}^n \to \{0,1\}$ if it agrees with the value of f on many Boolean inputs. We quantify "approximate" and "low" in the following lemma to prove (a).

Lemma 22. *If C is a $\mathtt{ACC}^0[3]$ circuit of depth d and size S, then for any α there is a polynomial f over $\mathbb{F}_3$ of degree $(2\alpha)^d$ such that f agree with C on at least $1 - \frac{S}{2^\alpha}$ fraction of the inputs.*

Proof. The proof is by induction on the depth d. If $d = 0$ there are no gates other than the inputs, so the output is just one of the input gates x_i, which can be approximated by the polynomial x_i. Suppose that $d > 0$ and assume that for every $\mathtt{ACC}^0[3]$ circuit C of depth $d-1$ there exists an approximating polynomial $\widetilde{g}$ of degree at most $(2\alpha)^{d-1}$ that agrees with C on at least $1- \mid C \mid /2^\alpha$ fraction of the inputs.

For every gate type we provide an arithmetic expression that either exactly evaluates to the output of the gate or does so for a large fraction of the inputs. Each gate has as input some wires $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ that may themselves be outputs to previous gates.

- NOT gate: Given a not gate with input $f_1(x_1, \ldots, x_n)$ by the inductive hypothesis we have an approximating polynomial $\widetilde{f_1}(x_1, \ldots, x_n)$. Set the approximating polynomial for the NOT gate to be $1 - \widetilde{f_1}(x_1, \ldots, x_n)$. Whenever $\widetilde{f_1}(x_1, \ldots, x_n) = f_1(x_1, \ldots, x_n)$, our new approximating polynomial introduces no new error and the degree stays the same.

- MOD$_3$ gate: Given a MOD$_3$ gate with inputs $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ approximate the output with the polynomial $\left(\sum_{i=1}^k \widetilde{f_i}\right)^2$. By Fermat's Little Theorem if the sum of the $\widetilde{f_i}$'s is not divisible by 3 then the polynomial evaluates to $1 \mod 3$ as desired, otherwise it evaluates to $0 \mod 3$. So again, this step is responsible for no new error that did not come from the $\widetilde{f_i}$'s. The resulting degree is at most double the degree of the largest degree among the $\widetilde{f_i}$'s, so by induction the resulting degree is $2 \cdot (2\alpha)^{d-1} \leq (2\alpha)^d$.

- OR gate: Given an OR gate with inputs $f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ the obvious attempt would be to set the approximating polynomial to $1 - \prod_{i=1}^k (1 - \widetilde{f_i})$. This is the standard way to "arithmetize" the OR function, so that the output is also Boolean. The product term acts as an indication of whether any one of the $\widetilde{f_i}$'s is 1, in which case the expression evaluates to 1. However, this would increase the degree by a factor of k, which in the worst case can be as large as S, the size of the entire circuit. The solution is to introduce some error to approximate the OR gate.

The output of the OR gate is 1 if at least one of $f_1(x_1,\dots,x_n),\dots,f_k(x_1,\dots,x_n)$ is 1. Consider the f_i's as a vector $f=(f_1,\dots,f_k)$. Assuming one $f_i=1$ then f is a non-zero vector in $\mathbb{F}_2^k$. Therefore, $\dim(f)+\dim(f^\perp)=k \implies \dim(f^\perp)=k-1$, so we have

$$|\{v : v\cdot f = 0 \mod 2, v\in\mathbb{F}_2^k\}| = 2^{k-1}.$$

Therefore, $\Pr_{v\in\mathbf{F}_2^k}[v\cdot f=0]=\frac{1}{2}$. A random vector v is equivalent to a random subset $I_v\subseteq\{1,\dots,k\}$ and $v\cdot f=\sum_{i\in I_v} f_i \mod 2$. So with probability 1/2 over the random choices of subsets $I\subseteq\{1,\dots,k\}$, $\sum_{i\in I} f_i \not\equiv 0 \mod 2$. If $\sum_{i\in I} f_i\not\equiv 0 \mod 2$ holds then so does $\sum_{i\in I} f_i\not\equiv 0 \mod 3$. In fact, this might happen even more often, so over the choices of subsets I we have $\sum_{i\in I} f_i\not\equiv 0 \mod 3$ with probability at least 1/2.

Randomly choose α such subsets $I_1,\dots,I_\alpha\subseteq\{1,\dots,k\}$ and set $\widetilde{g}_j=(\sum_{i\in I_j}\widetilde{f}_i)^2$. The reason for squaring the summations is that we can only guarantee with probability at least 1/2 that the summation is nonzero mod 3, so $\sum_{i\in I_j}\widetilde{f}_i\in\{\pm 1\}$. Squaring this value always gives a 1, the desired Boolean value in this case. Now we arithmetize the OR function on the $\widetilde{g}_j$'s in the standard way to get the approximating polynomial $\widetilde{g}=1-\prod_{j=1}^{\alpha}(1-\widetilde{g}_j)$. If for a fixed x every $\widetilde{f}_i=0$ then $\widetilde{g}=0$. If for a fixed x at least one $\widetilde{f}_i=1$ then $\Pr[\widetilde{g}\neq 1]=\Pr[\widetilde{g}_j=0\forall j]\leq\frac{1}{2^\alpha}$. By the probabilistic method there exists a choice of subsets $I_1,\dots,I_\alpha$ such that the probability that $\widetilde{g}\neq 1$ over the choices x is less than or equal to $\frac{1}{2^\alpha}$. We use these specific α subsets to construct $\widetilde{g}$, so that $\widetilde{g}$ differs from $\mathrm{OR}(\widetilde{f}_1,\dots,\widetilde{f}_k)$ on at most $\frac{1}{2^\alpha}$ inputs.

If the degrees of the $\widetilde{f}_i$ are at most $(2\alpha)^{d-1}$ then the degree of $\widetilde{g}$ is at most $\alpha\cdot 2\cdot(2\alpha)^{d-1}=(2\alpha)^d$, as desired.

- AND gate: Given an AND gate with inputs $f_1,\dots,f_k$, simply note that by DeMorgan's laws $AND(f_1,\dots,f_k)=\neg OR(\neg f_1,\dots,\neg f_k)$, so we can carry out the combined procedure of the NOT gate and the OR gate described above.

Each gate introduces an error of at most $\frac{1}{2^\alpha}$, so the resulting approximating polynomial does not agree with C on at most at most $\frac{S}{2^\alpha}$ fraction of the inputs. Note also that successive errors might cancel out to correct themselves, so $\frac{S}{2^\alpha}$ is really the worst case total error. $\square$

Given a circuit C by plugging in $\alpha=\frac{1}{2}n^{1/(2d)}$ there exists an approximating polynomial $\widetilde{g}$ of degree $\sqrt{n}$ such that $\widetilde{g}$ agrees with C on at least $1-S/2^{n^{1/(2d)}/2}$ fraction of the inputs. Now we prove part (b) that says if MOD_2 can be approximated by a degree $\sqrt{n}$ polynomial over $\mathbb{F}_3$ then they agree on at most $\frac{49}{50}$ fraction of the inputs. This will give us

$$\begin{aligned} 1 - S/2^{n^{1/(2d)}/2} &\leq 49/50 \\ \implies (1/50)2^{n^{1/(2d)}/2} &\leq S. \end{aligned}$$

Lemma 23. *If $f : \mathbb{F}_3^n \to \mathbb{F}_3$ is a degree $\sqrt{n}$ polynomial over $\mathbb{F}_3$ that agrees with MOD_2 on the inputs $G \subseteq \{0,1\}^n$ then $\mid G \mid < \frac{49}{50}2^n$.*

Proof. First, change the variables so that 0(FALSE) is represented by $1 \mod 3$ and 1(TRUE) is represented by $-1 \mod 3$. This allows MOD_2 to be conveniently represented by $x_1x_2\cdots x_n$. With this change of variables a polynomial $g(x_1,\ldots,x_n)$ becomes a polynomial $g(y_1,\ldots,y_n)$ with $y_i = 1 + x_i$, so the degree does not change. Let f be a polynomial of degree $\sqrt{n}$ that agrees with MOD_2 agree on $G \subseteq \{0,1\}^n$. Let g be the change of variable polynomial so that g and MOD_2 agree on a set $G' \subseteq \{\pm 1\}^n$ of equal cardinality.

Consider the set of functions $h : G' \to \mathbb{F}_3$ denoted $F_{G'}$. Then $\mid F_{G'} \mid = 3^{|G'|}$. Showing that $3^{|G'|} \leq 3^{\frac{49}{50}2^n}$ would imply $\mid G \mid = \mid G' \mid \leq \frac{49}{50}2^n$. Since the inputs to functions in $F_{G'}$ are from $\{\pm 1\}$, $x_i^2 = 1$ for any variable. Therefore, for any function $h \in F_{G'}$ the degree of any individual variable is at most 1 (h is *multilinear*). Write any h as a linear combination of monomials $a_I \prod_{i \in I} x_i$ for $I \subseteq \{1,\ldots,n\}$. Consider h in the form $h = h_1 + h_2$ where h_2 consists of all monomials with $\mid I \mid \leq n/2$, i.e. the portion of h with degree $\leq n/2$, and h_1 with monomials of degree $> n/2$. Then h_1 can be written as

$$\begin{aligned} h_1 &= (x_1 \cdots x_n)(x_1 \cdots x_n) \prod_{i \in I} x_i \\ &= x_1 \cdots x_n \prod_{i \in \overline{I}} x_i \\ &= x_1 \cdots x_n \overline{h}_1 \end{aligned}$$

where $\overline{h}_1$ has degree $\leq n/2$. Therefore, any $h \in F_{G'}$ can be written as $h = x_1 \cdots x_n \overline{h}_1 + h_2$ where both $\overline{h}_1$ and h_2 have degree $\leq n/2$. But $x_1 \cdots x_n$ and g agree on G', so $x_1 \cdots x_n$ can be replaced by g, so that every h can be written as $g\overline{h}_1 + h_2$ with the total degree of $h \leq \sqrt{n} + n/2$. The number of multilinear functions with degree $\leq \sqrt{n} + n/2$ is at most 3 raised to the number of all possible monomials, which is

$$\sum_{i=0}^{\sqrt{n}+n/2} \binom{n}{i} \leq \frac{49}{50}2^n$$

where the bound is established using usual bounds on the tail distribution of binomial coefficients. Therefore, $\mid F_{G'} \mid \leq 3^{\frac{49}{50}2^n}$ as desired. $\square$

Theorem 24. $\mathsf{MOD}_2 \notin \mathtt{ACC}^0[3]$.

Proof. Let C_n be a depth d $\mathtt{ACC}^0[3]$ circuit of size $S(n)$ that solves MOD_2. Then there exists a degree $\sqrt{n}$ polynomial approximating C_n that agrees with C_n on at least a $1-S/2^{n^{1/(2d)}/2}$ fraction of the inputs. Since this polynomial is a polynomial of degree $\sqrt{n}$, it agrees with MOD_2 on at most a $49/50$ fraction of the input. Therefore, $S(n) \geq (1/50)2^{n^{1/(2d)}/2}$. $\square$

1.6 Naturalizing $\mathsf{MOD}_2 \notin \mathtt{ACC}^0[3]$

Razborov and Rudich claim that the proof just given is a natural proof, so we examine their argument. Let C_n be the property such that $f \in C_n \iff f$ cannot be approximated by a degree $\sqrt{n}$ polynomial p over $\mathbb{F}_3$ such that f and p agree on more than a $\frac{49}{50}$ fraction of the inputs. The second lemma above shows that MOD_2 has this property. The first lemma shows that C_n is useful against $\mathtt{ACC}^0[3]$ by plugging in $\alpha = \frac{1}{2}n^{1/(2d)}$.

We would like to find a subset $C_n^* \subseteq C_n$ that has constructivity and largeness. Consider the obvious choice of $C_n^* = C_n$. By a counting argument, C_n^* has largeness because most Boolean functions indeed cannot be approximated well by a low degree polynomial over $\mathbb{F}_3$. On the other hand, it is not known how to decide in $\mathtt{P}_{/\mathtt{Poly}}$ whether f can be approximated by a low degree polynomial, given the truth table of f. Razborov and Rudich leave this as an open problem of its own interest.

We would like to examine the proof of the second lemma to determine a more specific C_n^* such that constructivity holds. Recall that the actual property used in the proof is the fact that any multilinear polynomial h can be written as $x_1 \cdots x_n h_1 + h_2$ where h_1, h_2 both have degree less than or equal to $n/2$. Let $C_n^* \subseteq C_n$ be the property $f \in C_n^* \iff$ every multilinear h can be written as $\overline{f}h_1 + h_2$ where $\overline{f}$ is the unique multilinear representation of f. Note that C_n^* is indeed a subset of C_n because if $f \in C_n^*$ then the proof of the second lemma proceeds in exactly the same way, so that $f \in C_n$.

Now we have constructivity: Let V be the vector space of all multilinear polynomials. Let L be the vector space of polynomials with degree less than $n/2$ and let H be the complement vector space of polynomials consisting of monomials with degree greater than $n/2$. Then $V = L \oplus H$ and for n odd $\dim(L) = \dim(H)$. Determining whether $f \in C_n^*$ is equivalent to determining whether the map $\pi_f : L \to L \oplus H$ composed with the projection $p : L \oplus H \to H$ is one-to-one where π_f is the map $l \mapsto \overline{f}l$. In other words, given an $l \in L$, multiply by $\overline{f}$ and see what remains in H. Therefore, if L denotes a basis for L then the property is equivalent to the statement $\overline{f}L + L$ has full rank. Given the truth table of f the matrix for this linear map is easily computable and determining the rank of a matrix can be done in $\mathtt{NC}^2$.

We can check that $p \circ \pi_{\mathsf{MOD}_2}$ is indeed one-to-one. If $n = 3$ then

$$\begin{aligned} 1 &\mapsto x_1x_2x_3, \\ x_1 &\mapsto x_2x_3, \\ x_2 &\mapsto x_1x_3, \\ x_3 &\mapsto x_1x_2 \end{aligned}$$

and it is easy to see that this gives a one-to-one linear map if we fix $\{1, x_1, x_2, x_3\}$, $\{x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3\}$ as basis vectors of L and H respectively.

However, it is unknown whether C_n^* has largeness. Therefore, although we have restricted C_n^* enough to be able to prove constructivity, we have lost largeness. Define a new $C_n^* \subseteq C_n$ to be the property $f \in C_n^* \iff \dim(\overline{f}L + L) \geq (1/2 + \epsilon)2^n$ for some fixed $\epsilon > 0$. When $\epsilon = 1/2$ this property is the same as the previous property, but now C_n^* is larger for $\epsilon < 1/2$. Furthermore, for any fixed ϵ constructivity is preserved because $f \in C_n^*$ can again be determined by calculating the rank of the matrix.

Note that $C_n^* \subseteq C_n$ still holds for a fixed ϵ. Let G' be as in the proof, set $\{\pm 1\} \backslash G' = W$ and $w = | W |$. Then in the proof we showed that $3^{|G'|} = 3^{2^n - w} \leq 3^{\frac{49}{50}2^n}$. In this case, not all 2^n functions $h : G' \to \mathbb{F}_3$ can be written as $\overline{f}h_1 + h_2$, but $3^{2^n(1/2+\epsilon)-w}$ still can. Then the counting argument still applies to get a sufficient bound.

Define C_n^* with $\epsilon = 1/4$. Then C_n^* has largeness. We claim that either $f \in C_n^*$ or $x_1 \oplus \cdots \oplus x_n \oplus f \in C_n^*$ from which largeness follows. Note that $\overline{x_1 \oplus \cdots \oplus x_n \oplus f} = x_1 \cdots x_n \overline{f}$. If $\dim(\overline{f}L + L) \geq \frac{3}{4}2^n$ then $f \in C_n^*$. Otherwise, $\dim(\overline{f}L + L) < \frac{3}{4}2^n$, and we show that $\dim(x_1 \cdots x_n \overline{f}L + L) \geq \frac{3}{4}2^n$. Note $(\overline{f})^2 = 1$ implying that mulitplication by $\overline{f}$ is an automorphism of $L \oplus H$. Therefore,

$$\begin{aligned} \dim\big((x_1 \cdots x_n \overline{f}L + L)/L\big) &= \dim\big((x_1 \cdots x_n L + \overline{f}L)/\overline{f}L\big) \\ &\geq \dim\big((x_1 \cdots x_n L + \overline{f}L + L)/(\overline{f}L + L)\big) \\ &= \dim\big((L + H)/(\overline{f}L + L)\big) \\ &\geq \frac{1}{4}2^n, \end{aligned}$$

so $x_1 \oplus \cdots \oplus x_n \oplus f \in C_n^*$ as desired. Thus, this is a $\mathtt{NC}^2$-natural proof against $\mathtt{ACC}^0[3]$.

There is a general pattern in lower bound proofs that adhere to the Razborov, Rudich framework. Firstly, proving that a property C_n is useful involves working with the specific circuit model one is proving a lower bound against. In the $\mathtt{ACC}^0$ lower bound this was a fairly easy step, requiring a not too clever arithmetization of gates. On the other hand, this was the main step in the $\mathtt{AC}^0$ lower bound.

The second step then has to do with showing that a particular function does indeed have the property C_n. The level of difficulty of this second step determines how difficult it may be to track down a subproperty C_n^* that is natural. In our first example of an $\mathtt{AC}^0$ lower bound, this step was the easy step. Thus, showing that this property is natural was straightforward. For the second lower bound this was the more subtle step. Thus, finding a natural subproperty required a more thoughtful analysis of the proof.

The subset C_n^* arose from examining closely the proof that $f \in C_n$. The analysis determined the crucial property C_n^* that establishes $f \in C_n^* \subseteq C_n$, revealing that the main idea was linear algebra and a counting argument. This is the reason for the distinction between C_n and C_n^* in Razborov and Rudich's original definition. In particular, we had to adjust the subproperty C_n^* in both directions, first to get constructivity and then to re-obtain largeness. Thus, naturalizing a proof requires searching for the C_n^* that is not too general to make it difficult for an algorithm to exist and not too specific so that too few functions actually have the property.

2 Algebrization

In this section we focus on a more recent complexity barrier from [AW] called *algebrization.* The main motivations for this barrier are some results regarding interactive proof protocols that are non-relativizing. Furthermore, there are some circuit lower bounds (e.g. $\mathtt{MA}_{\mathtt{EXP}} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$) that are both non-relativizing and non-natural, avoiding the relativization barrier and the natural proof barrier simultaneously. The question arose of whether the ideas that established these results were sufficient to prove stronger lower bounds.

The main idea in these non-relativizing proofs is to transform a Boolean formula into an arithmetic expression. We have already seen this idea in Smolensky's proof that $\mathsf{MOD_q} \notin \mathtt{ACC}^0[\mathtt{p}]$ where a circuit is approximated by a low degree polynomial. The following results give further evidence that this approach is worthwhile in complexity theory. The algebrization barrier attempts to understand the extent of the technique's power and its limitations.

We begin by proving what was historically the first significant interactive proof result of the early 1990s: every language in the polynomial hierarchy (PH) has an interactive proof protocol (IP). Almost all of the necessary ideas are established in this proof to fully characterize the power of IP as IP = PSPACE. This was a surprising result because it was known that there exists an oracle that separates IP from coNP [FS]. Therefore, PH $\subseteq$ IP is non-relativizing, and since PSPACE contains all of PH, IP = PSPACE is a non-relativizing result as well.

We then examine the main ideas of these proofs to motivate the definition of an algebraic extension oracle and what it means for an inclusion or separation to algebrize or not. Under these notions the interactive proof results algebrize, but many other open questions in complexity theory do not.

2.1 PH $\subseteq$ IP and PSPACE $\subseteq$ IP

One possible way to view the definition of interactive proof protocols is as a generalization of the fact that the class NP consists of languages with short proofs that a polynomial-time computation can verify. One way to generalize this notion is to allow the verifier to be probabilistic and to sometimes make mistakes. The complexity class according to this definition is denoted `MA` and was introduced by Babai in [Ba2].

Definition 25. [Ba2] The class of languages `MA` is defined as follows.
A language $L \in \texttt{MA} \iff$ there exists a probabilistic polynomial-time verifier V and a polynomial p such that

- If $x \in L$ then there exists a string $y\{0,1\}^{p(|x|)}$ such that $V(x, y)$ accepts with probability greater than 2/3.
- If $x \notin L$ then there exists no string $y \in \{0,1\}^{p(|x|)}$ such that $V(x, y)$ accepts with probability greater than 1/3.

The letters M and A in `MA` stand for Merlin and Arthur. Merlin is thought to be an all powerful prover and Arthur the probabilist polynomial-time verifier. On an input x, Merlin gives Arthur the certificate y, which Arthur can check with some certainty whether y proves that $x \in L$. The first bullet point in the definition asserts that for a good x there exists a Merlin who convinces Arthur that $x \in L$ with good probability. The second bullet point asserts that for a bad x there is no Merlin who can trick Arthur into believing $x \in L$ with good probability. If the probability is changed to 1 in the first bullet point of the definition, then this is referred to as perfect completeness. The class `MA` is commonly referred to as the class of languages with publishable proofs that can be statistically verified at any later time [Ba1].

In the `MA` scenario Merlin publishes a proof that Arthur can verify with no further interaction. We can generalize this even further by allowing Merlin and Arthur to interact. Define the class `MAMA` similarly to `MA` except now Merlin gives Arthur a string y_1, Arthur does some probabilistic computation at the end of which he asks Merlin a query q_1 (with Arthur's coin tosses appended), which Merlin responds to with the string y_2. Finally Arthur does some more probabilistic computation before outputting 0 or 1. Merlin's response y_2 can depend on the input x, Arthur's query q_1 and Arthur's random coin tosses r_1. Formally we can think of Merlin's second answer as a function f that takes input x, q_1, r_1 and outputs a string y_2. If there are even more rounds we can think of Merlin as a function that in round n takes as input $(x, q_1, r_1, q_2, r_2, \ldots, q_{n-1}, r_{n-1})$ and outputs y_n.

We will work with the analogous model introduced independently in [GMR], in which Arthur does not have to share his random coin tosses with Merlin. It turns out that the two definitions are equally powerful [GS]. When the number of rounds is polynomial the class is denoted `IP` and defined as follows.

Definition 26. A *verifier* is a probabilistic polynomial-time Turing machine that has a designated communication tape. A *prover* P is a map f_P taking finite sequences $x, q_1, q_2, \ldots, q_n$ where $x, q_i \in \{0,1\}^*$ for all i to some string $y_n \in \{0,1\}^*$. A verifier V and prover P interact by both receiving an input x, and V probabilistically computing in polynomial-time a string q_1 written on the communication tape. Then P replaces q_1 with $f_P(x, q_1)$. Again V computes and replaces $f_P(x, q_1)$ with a new string q_2 to which P responds with $f_P(x, q_1, q_2)$. These interactions continue until V outputs a 1 or a 0.

The language L has an *interactive proof protocol* (V, P) if V is a probabilistic polynomial time verifier and P a prover such that

- if $x \in L$, P makes V accept with probability greater than 2/3 in polynomially many rounds
- if $x \notin L$ no prover Q makes V accept with probability greater than 1/3 in polynomially many rounds.

A language $L \in \mathtt{IP}$ if L has an interactive proof protocol.

Note that the responses y_i are assumed to be polynomially long, so that V has time to read them. In interactive proof variants where the verifier is allowed exponential probabilistic time these responses can be exponentially long. Now we prove $\mathtt{PH} \subseteq \mathtt{IP}$ according to Lund, Fortnow, Karloff, and Nisan [LFKN]. In fact, this protocol is sometimes called the LFKN protocol in the literature.

Recall Toda's theorem that $\mathtt{PH} \subseteq \mathtt{P}^{\sharp\mathtt{P}}$, so that if $\mathtt{P}^{\sharp\mathtt{P}} \subseteq \mathtt{IP}$ then $\mathtt{PH} \subseteq \mathtt{IP}$. Define a 0-1 matrix to be an integer matrix with entries either 0 or 1. Recall Valiant's theorem that $\sharp\mathsf{Perm}$, the problem of computing the permanent of a 0-1 matrix, is $\sharp\mathsf{P}$-complete. If $\sharp\mathsf{Perm}$ has an interactive proof system then so does any other language in $L \in \mathtt{PH} \subseteq \mathtt{P}^{\sharp\mathtt{P}}$.

Lemma 27. *Let $L = \{\langle M, p\rangle \mid M$ is a 0-1 matrix $, p = perm(M)\}$. If L has an interactive proof protocol then so does every language in $\mathtt{P}^{\sharp\mathtt{P}}$.*

Proof. The class $\mathtt{P}^{\sharp\mathtt{P}}$ is equivalent to polynomial time with a $\sharp\mathsf{P}$-complete oracle, so we may take $\sharp\mathsf{Perm}$ as the oracle. For any language $L \in \mathtt{P}^{\sharp\mathtt{P}}$ the verifier V runs the polynomial time $\mathtt{P}^{\sharp\mathtt{P}}$ algorithm, but every time a query is to be made V runs the interactive protocol for $\sharp\mathsf{Perm}$. □

We will show that $\sharp\mathsf{Perm}$ does indeed have an interactive protocol. Note that for an $n \times n$ 0-1 matrix M $perm(M)$ cannot exceed $n!$. Therefore, $perm(M)$ coincides with $perm(M)$ mod p for a prime $p \in (n!, 2 \cdot n!)$. The protocol is based on computations over $\mathbb{Z}_p$ for such a prime p. In general, denote the entries of some matrix M by m_{ij} so that $M = (m_{ij})$.

Theorem 28. *[LFKN]* $\sharp\mathsf{Perm} \in \mathtt{IP}$.

Proof. The interactive proof protocol begins with V and P receiving the pair $\langle M, a\rangle$ where M is an $n \times n$ 0-1 matrix and a is an integer. First V and P need to establish the field $\mathbb{Z}_p$ to work over, so P selects a prime $p \in (n!, 2 \cdot n!)$ and sends it to V. The language of Primes has short certificates, so V can request certificates from P to verify that p is indeed prime. If P fails to prove that p is a prime, V rejects. Now V and P are ready for the main portion of the protocol.

During the interaction V will maintain a list of pairs $\{\langle M_1, a_1\rangle, \dots, \langle M_k, a_k\rangle\}$ where for a $r \leq n$ each M_i is an $r \times r$ matrix and a_i are integers supposedly representing perm(M_i). However, V cannot be sure of the validity of the a_i if P is malicious. The point of the protocol is to have V reduce the dimension of the matrices in the list until there remains just one pair $\langle M, a\rangle$ where M is a 1×1 matrix, at which point V can check on its own whether perm$(M) = a$. There are two stages: 1) if there is one pair in the list, the list will expand, but the sizes of the new matrices will be one less than the previous matrix, and 2) if there is more than one matrix, then the list will shrink to one pair with a matrix the same size as all of the matrices previously in the list. Repeating steps 1 and 2 n times will leave just one pair with a 1×1 matrix.

Consider step 1, so that the list consists of just one pair $\langle M, a\rangle$ where M is $r \times r$. Then V constructs the minors $M_i = M_{1i}$ $1 \leq i \leq r$ and requests the values perm(M_i) from P, receiving a_i. Note that perm$(M) = \sum_{i=1}^{r} m_{1i}$perm$(M_i)$, so V checks that these new values a_i that supposedly represent perm(M_i) are consistent with the previous value of perm(M). If $a \neq \sum_{i=1}^{r} m_{1i}a_i$ then V rejects. Otherwise, V updates the list to $\{\langle M_1, a_1\rangle, \langle M_2, a_2\rangle, \dots, \langle M_r, a_r\rangle\}$.

Now consider step 2 where the list consists of more than one pairs. Let the first two pairs be $\langle M_1, a_1\rangle$ and $\langle M_2, a_2\rangle$. Consider the polynomial $f(x) =$perm$(M_1 - x(M_1 - M_2))$ where M_1, M_2 are $r \times r$ matrices. The polynomial f has at most degree $r \leq n$, so V requests the $r+1$ coefficients that supposedly represent f. Denote the polynomial constructed from the coefficients received from P by g. Note that $f(0) =$perm(M_1) and $f(1) =$perm(M_2), so V can again check for consistency. If $g(0) \neq a_1$ or $g(1) \neq a_2$ then V rejects. Otherwise, V chooses randomly an $\alpha \in \mathbb{Z}_p$ and sends α to P. V then updates the list by replacing $\langle M_1, a_1\rangle$ and $\langle M_2, a_2\rangle$ by $\langle M_1 - \alpha(M_1 - M_2), g(\alpha)\rangle$.

The key realization is that if P attempts to lie by giving an incorrect g then $f \neq g$. If $g(0) \neq a_1$ or $g(1) \neq a_2$ then P is caught immediately. Otherwise, the probability that perm$(M_1 - \alpha(M_1 - M_2)) = g(\alpha)$ for a randomly chosen α is very small because $f \neq g$. This inconsistency has a high probability of being revealed in the final step when V has a single pair $\langle M_1, a_1\rangle$ and accepts if and only if perm$(M_1) = a_1$.

See Algorithm 1 for a formal description (we omit the portion regarding the selection of the prime p). It is helpful to view the description from the point of view of the verifier, assuming that the verifier has access to some unknown prover. The prover may be honest or malicious and we shall examine both cases with the formal statement of the algorithm in mind.

Recall that according to the definition of IP there must exist an honest prover P such that for any $\langle M, a\rangle \in \mathsf{Perm}$, P convinces V to accept with probability greater than 2/3. It is easy to see that such a prover exists because if P answers every query truthfully then V will always accept with probability 1. The second part of the definition requires that no prover P can convince V to accept an incorrect pair with probability greater than 1/3.

So suppose that $\langle M, a\rangle \notin \mathsf{Perm}$ and consider how a malicious P could convince V that $\text{perm}(M) = a$. Since the input $\langle M, a\rangle \notin \mathsf{Perm}$, the protocol begins with $\text{perm}(M) \neq a$. The invariant that the list contains at least one pair $\langle M_i, a_i\rangle$ such that $\text{perm}(M_i) \neq a_i$ is maintained with high probability throughout the entire protocol. Call such a pair a bad pair.

Consider the first expansion step. Since $a \neq \text{perm}(M) = \sum m_{1i}\text{perm}(M_i)$, if the responses pass the test ($a = \sum m_{1i}a_i$) then

$$\sum m_{1i}a_i = a \neq \text{perm}(M) = \sum m_{1i}\text{perm}(M_i)$$

so $\text{perm}(M_i) \neq a_i$ for at least one pair $\langle M_i, a_i\rangle$. Of course if the responses don't pass the test, V immediately rejects, so P's only hope for V to accept is to lie about some a_i.

Now consider the first list contraction step. There is some bad pair in this list, so without loss of generality suppose one of $\langle M_1, a_1\rangle$, $\langle M_2, a_2\rangle$ is a bad pair. Consider what happens when $\langle M_1, a_1\rangle, \langle M_2, a_2\rangle$ are replaced in a contraction step. The prover P has to provide V with the coefficients of $f(x) = \text{perm}(M_1 - x(M_1 - M_2))$. If the resulting polynomial g constructed from the received coefficients passes both of the tests $g(0) = a_1$ and $g(1) = a_2$, then either

$$f(0) = \text{perm}(M_1) \neq a_1 = g(0) \text{ or } f(1) = \text{perm}(M_2) \neq a_2 = g(1).$$

In other words, f and g disagree on at least one of these values, so they are two distinct $\leq n$ degree polynomials that can agree on at most n values. Then for the randomly chosen $\alpha \in \mathbb{Z}_p$, $\text{perm}(M_1 - \alpha(M_1 - M_2)) \neq g(\alpha)$ with high probability, i.e.

$$\Pr[f(\alpha) = g(\alpha)] \leq n/p \leq n/n!.$$

Of course if g does not pass the two tests, then V immediately rejects, so P has to lie to have any hope of V accepting.

These tests occur for every expansion step and pair contraction step. If by some small chance V picks an α such that $f(\alpha) = g(\alpha)$, then P no longer has to lie (P is off the hook) and can pass all the remaining tests by telling the truth. Since there are $(n-1)+(n-2)+\ldots 2+1$ shrink steps, the number of tests affecting the probability is at most quadratic. At each step, P has a small chance of absolving the original lie. If this never happens then the lie will be revealed in the final step when V checks whether $M = a$. Therefore, $\Pr[V accepts] \leq n^3/p \leq n^3/n!$.

□

Summarizing informally, a malicious prover P who desires to convince V of the false statement $\text{perm}(M) = a$ can be viewed as beginning with a lie. Then every step of the protocol forces P to stay consistent with its initial lie with a small chance of getting away with the lie.

Algorithm 1 Interactive Proof Protocol for Perm

```
Input $\langle M, a\rangle$ where $M$ is an $n \times n$ 0-1 matrix and $a$ is an integer.
$k \leftarrow n$
$list \leftarrow \{\langle M, a\rangle\}$
while $k > 1$ do
    if $|\ list\ |= 1$ then //list has just one item, denote $list = \{\langle M, a\rangle\}$
        Construct $M_1 = M_{11}, M_2 = M_{12}, \ldots, M_k = M_{1k}$ and query $P$ for $a_1, \ldots, a_k$.
        if $\sum_{i=1}^{k} m_{1i}a_i \neq a$ then //new permanent values not consistent with the old value
            reject
        else
            $list \leftarrow \{\langle M_1, a_1\rangle, \ldots, \langle M_k, a_k\rangle\}$
            $k \leftarrow k - 1$
        end if
    else
        query $P$ for coefficients of the polynomial f(x)=perm($M_1 - x(M_1 - M_2)$).
        if $f(0) \neq a_1$ or $f(1) \neq a_2$ then //$f$ not consistent with old values
            reject
        else
            Select at random $\alpha \in \mathbf{F}_p$.
            Send $\alpha$ to $P$.
            $list = \{\langle M_1, a_1\rangle, \ldots, \langle M_{k-r}, a_{k-r}\rangle\} \leftarrow \{\langle M_1 - \alpha(M_1 - M_2), f(\alpha)\rangle, \langle M_3, a_3\rangle, \ldots, \langle M_{k-r+1}, a_{k-r+1}\rangle\}$
        end if
    end if
end while
//list has just one item, denote $list = \{\langle M, a\rangle\}$ and $M$ is $1 \times 1$ matrix
if $M = a$ then
    Accept
else
    Reject
end if
```

One of the main ideas of this protocol is that two distinct low degree polynomials have a small probability of agreeing on some value. This idea was combined in [BF] with an arithmetization of Boolean formulas to create a similar protocol for the number of satisfying assignment of a CNF formula, bypassing the need for Valiant's result of the $\sharp$P-completeness of the permanent. Finally, Shamir [Sham] proved that PSPACE = IP by arithmetizing a quantified Boolean formula with a small degree polynomial.

First note the more standard direction: IP $\subseteq$ PSPACE because a PSPACE machine can enumerate recursively over all possible coin flips of V and polynomial long responses of P giving an algorithm similar to the PSPACE algorithm for TQBF.

Lemma 29. IP $\subseteq$ PSPACE

We omit the details and aim to prove the less elementary direction PSPACE $\subseteq$ IP as in [Sham]. Showing that the PSPACE-complete language TQBF has an interactive protocol will suffice.

Definition 30. A *quantified Boolean formula* is a formula consisting of Boolean variables x_i and negations $\overline{x}_i$ with operations $\wedge$, $\vee$ and quantifiers $\forall x_i$ and $\exists x_i$. A quantified Boolean formula is *closed* if every variable appears in the scope of some quantifier.

A closed quantified Boolean formula is either true or false. Recall the standard result that the problem TQBF of deciding whether a (well-formed) closed quantified Boolean formula is true is PSPACE-complete. Note that a common definition of TQBF requires that all of the quantifiers appear as a prefix (i.e. $Q_1x_1 \cdots Q_kx_kq(x_1, \ldots, x_k)$ where q is quantifier free), but in our definition we allow the quantifiers and logical operations to appear in any order. Under both definitions TQBF is PSPACE-complete.

Consider the natural way to algebrize a quantified Boolean formula.

- Replace every x_i with a variable z_i and every $\overline{x}_i$ by $1 - z_i$ where the z_i are variables ranging over the integers.
- Replace every $\wedge$ by multiplication $\cdot$ and every $\vee$ with addition $+$.
- Replace every $\forall x_i$ with $\prod_{z_i \in \{0,1\}}$ and every $\exists x_i$ with $\sum_{z_i \in \{0,1\}}$.

For example consider the closed quantified Boolean formulas

(a) $\forall x_1 \exists x_2 (x_1 \wedge x_2)$

(b) $\forall x_1 \exists x_2 (x_1 \vee x_2)$

(c) $\forall x_1 \exists x_2 \left[(x_1 \vee x_2) \wedge \forall x_3 (x_2 \wedge x_3) \vee x_1 \vee \overline{x}_3\right]$

where (a) is false and $(b), (c)$ are true. The corresponding arithmetic expressions are

(a) $\prod_{z_1 \in \{0,1\}} \sum_{z_2 \in \{0,1\}} (z_1 z_2)$

(b) $\prod_{z_1\in\{0,1\}}\sum_{z_2\in\{0,1\}}(z_1+z_2)$

(c) $\prod_{z_1\in\{0,1\}}\sum_{z_2\in\{0,1\}}(z_1+z_2)\left(\prod_{z_3\in\{0,1\}}(z_2z_3+z_1+1-z_3)\right)$,

which evaluate to an integer. It is not difficult to see that a closed quantified Boolean formula is true if and only if the corresponding arithmetic expression evaluates to a nonzero value. For our examples, (a) evaluates to 0, (b) to 3 and (c) to 10. This value can be as large as $O(2^{2^n})$ by noting that the maximum value of the arithmetic form of an $\wedge$ of two subexpressions is the product of the maximum values of the two subexpressions. Similarly, the the maximum value of an $\vee$ is the sum of the maximums of the subexpressions. An existential quantifier can at most double the value of the subexpression and a universal quantifier can at most square it. The worst case value occurs if there are n universal quantifiers. In particular, for a given formula there exists a polynomial size prime p such that the arithmetic expression is not 0 mod p if and only if the expression is nonzero over the integers. If not, by Chinese remaindering the expression would be 0 modulo the product of all such prime, which exceeds 2^{2^n}.

The idea of the protocol is very similar to the permanent protocol of simplifying expressions at each step and selecting random values to plug into a polynomial. At every step the verifier eliminates the leftmost $\prod_{z_i\in\{0,1\}}$ or $\sum_{z_1\in\{0,1\}}$ symbol, and the arithmetic expression is viewed as a polynomial $f(z_i)$ over the variable z_i. The verifier receives a polynomial supposedly representing $f(z_i)$ from the prover, performs a check, and chooses a random number to plug into the polynomial. The process is then repeated with the next leftmost occurrence of a product or summation sign, thereby simplifying the arithmetic expression by one variable.

We would like the degree of the polynomials to remain small, but for an expression such as $\prod_{z_1\in\{0,1\}}\prod_{z_2\in\{0,1\}}\cdots\prod_{z_n\in\{0,1\}}(z_1+\cdots+z_n)$ the corresponding polynomial $p(z_1) = \prod_{z_2\in\{0,1\}}\cdots\prod_{z_n\in\{0,1\}}(z_1+\cdots+z_n)$ has degree 2^{n-1}. A polynomial time verifier cannot handle expressions that are this large, but luckily there is a trick to get around this problem.

Definition 31. A quantified Boolean formula is *simple* if the occurrence of every variable is separated from its point of quantification by at most one universal quantifier.

All three of our examples are simple quantified Boolean formulae. Example (c) is simple because

- the first occurrence of x_1 is not separated from its point of quantification by any universal quantifiers,
- the first occurrence of x_2 is not separated from its point of quantification by any quantifier at all,
- the second occurrences of x_1 and x_2 are separated only by $\forall x_3$,

- both x_3 and $\overline{x}_3$ are not separated from their point of quantification by any quantifiers.

To contrast, the expression $\forall x_1 \exists x_2 \forall x_3 \left[(x_1 \vee x_2) \forall x_4 \left(x_1 \wedge x_3 \wedge x_4\right)\right]$ is not simple because the second occurrence of x_1 is separated by $\forall x_4$ and $\forall x_3$.

Lemma 32. *Any quantified Boolean formula of size n can be transformed to a simple quantified Boolean formula of size polynomial in n.*

Proof. Rename every variable x_i as x_i^0. For every occurrence of x_i^0 following a universal quantifier create an existentially quantified alias x_i^j and insert the expression $x_i^j = x_i^{j-1}$ into the formula. For example, consider $\exists x_1 \forall x_2 \exists x_3 \forall x_4 q(x_1, x_2, x_3, x_4)$ where q is a quantifier free Boolean formula. Then the transformation gives

$$\begin{aligned}
&\exists x_1^0 \forall x_2^0 \\
&\left[\exists x_1^1 (x_1^0 = x_1^1)\right] \wedge \exists x_3^0 \forall x_4^0 \\
&\left[\exists x_1^2 \exists x_2^1 \exists x_3^1 (x_1^1 = x_1^2) \wedge (x_2^0 = x_2^1) \wedge (x_3^0 = x_3^1)\right] \wedge q(x_1^2, x_2^1, x_3^1, x_4^0).
\end{aligned}$$

(The expression $x_i = x_j$ is merely shorthand for $(x_i \wedge x_j) \vee (\overline{x}_i \wedge \overline{x}_j)$.) Each extra x_i^j allows x_i to hop one more universal quantifier, while preserving the information of the formula with the clause $x_i^{j-1} = x_i^j$. The resulting quantified Boolean formula is simple because by definition any x_i^j will have to pass at most only one universal quantifier to reach its point of quantification. The number of resulting variables is at most quadratic, as each of the n variables can be separated by at most n universal quantifiers. $\square$

Therefore, any closed quantified Boolean formula can be transformed to this form in polynomial time, so deciding whether a simple quantified Boolean formula is true or false is PSPACE-complete. Thus we may restrict ourselves to considering simple quantified Boolean formulas, which allows the degree of the resulting polynomials to remain small.

Lemma 33. *If a quantified Boolean formula B is simple then the degree of $q(z_i)$, the polynomial obtained by removing the leftmost $\prod_{z_i \in \{0,1\}}$ or $\sum_{z_i \in \{0,1\}}$ in the arithmetic expression B, grows at most linearly in the size of B.*

Proof. The degree of z_i of a quantifier free expression containing z_i is at most the size of B. Any summations over other variables do not increase the degree of z_i and any product can only double the degree of z_i. Since B is simple, there can be only one such product, so the degree is at most linear in the size of B. $\square$

We are finally ready to give the description of the protocol. On input a closed simple quantified Boolean formula B of size n, V arithmetizes B to get A. As in the previous protocol, P initially sends a prime p to V along with a primality certificate. All arithmetic will be done mod p, and in particular P can always find such a p so that $A \not\equiv 0 \mod p$ if and only if $A \neq 0$. If the certificate failed, V rejects. Alternatively, V could itself select a prime p, relying on the fact that $A \neq 0$ if and only if $A \not\equiv 0 \mod p$ for most primes p. Assume that a prime has been established. The main portion of the protocol is better understood with an example.

Example 34. Let

$$B = \forall x_1 \exists x_2 \left[(x_1 \vee x_2) \wedge \forall x_3 \left((x_2 \wedge x_3) \vee x_1 \vee \overline{x}_3\right)\right] \quad \text{with arithmetization}$$

$$A = \prod_{z_1 \in \{0,1\}} \sum_{z_2 \in \{0,1\}} (z_1 + z_2) \left(\prod_{z_3 \in \{0,1\}} (z_2 z_3 + z_1 + 1 - z_3) \right).$$

Suppose that the prover is truthful and sends the correct value $a = 10$. Then V requests the polynomial for

$$A'(z_1) = \sum_{z_2 \in \{0,1\}} (z_1 + z_2) \left(\prod_{z_3 \in \{0,1\}} (z_2 z_3 + z_1 + 1 - z_3) \right)$$

and receives $q_1(z_1)$. Then V checks that $q_1(0)q_1(1) = a$ because this would have to be the case if q_1 actually represents $A'(z_1)$. Since P was truthful, this test passes. Then V randomly selects say $r_1 = 2$ and sends $r_1 = 2$ to P. Then A is updated to

$$A = A'(2) = \sum_{z_2} (2 + z_2) \left(\prod_{z_3} (z_2 z_3 + 3 - z_3) \right) \text{ which has value 39, and } a \text{ is updated to}$$

$a = q_1(2)$ (which is also 39 if P is truthful).

Now we repeat, so A' is updated to

$$A'(z_2) = (2 + z_2) \left(\prod_{z_3} (z_2 z_3 + 3 - z_3) \right)$$

of which V requests a polynomial representation from P, receiving a new $q_2(z_2)$. Then V checks that $a = q_2(0) + q_2(1)$. Again, this test passes if P is truthful. Now V randomly selects say $r_2 = 3$, sends $r_2 = 3$ to P, and updates

$$A = A'(3) = 5 \left(\prod_{z_3} (3z_3 + 3 - z_3) \right) = 5 \left(\prod_{z_3} (2z_3 + 3) \right) \text{ which has value 75.}$$

At this point, we need to be a bit more careful. If we request a polynomial representing $5\,(2z_3 + 3)$ and $q_3(z_3)$ is the truthful answer then $q_3(0)q_3(1) = 375$, 5 times the value of 75. So we merely account for the leading value 5 by setting

$$A'(z_3) = 2z_3 + 3 \text{ and}$$
$$a = q_2(3)/5.$$

(We can account for the sum of a leading constant in a similar way.) Now V requests the polynomial representation of $A'(z_3)$, receives a new $q_3(z_3)$, and again checks $q_3(0)q_3(1) = a$. Finally, V selects say $r_3 = 5$ and checks on its own that $q_3(5) = A'(5) = 13$. See Algorithm 2 for a rigorous description

Theorem 35. *[Sham]* $\mathsf{TQBF} \in \mathtt{IP}$*.*

Proof. There are two things to check:

(a) If B is a true quantified Boolean formula then the honest prover P convinces V with probability at least 2/3.

(b) If B is a false quantified Boolean formula then no prover Q convinces V with probability greater than 1/3.

If a prover P always responds truthfully to V's queries then according to the protocol V will always accept. On the other hand, if the arithmetization A of a false B is claimed to be some nonzero value a by a malicious prover P, then P will have to give an incorrect polynomial $q_1(z_1)$ to support the false initial claim. Because V performs one of the checks $q_i(0) + q_i(1) = a$ or $q_i(0)q_i(1) = a$ at every step, P will have to continue to lie with high probability. Just as in the permanent protocol, if p is exponential in the degree of q, then q can agree with at most $\deg(q)$ values with the actual polynomial, and by the lemma $\deg(q)$ is at most linear in the size of B. Therefore, the probability that P gets away with the lie is low for every step. Since there are at most n steps, the lie will be exposed in the final step with high probability. $\square$

Gathering the previous results we have the theorem.

Theorem 36. $\mathtt{PSPACE} = \mathtt{IP}$*.*

The interactive proof results came as a surprise because it was known that there exists an oracle A such that $\mathtt{coNP}^A \not\subseteq \mathtt{IP}^A$. Since $\mathtt{coNP} \subseteq \mathtt{PH} \subseteq \mathtt{P}^{\sharp\mathtt{P}}$ and $\mathtt{coNP} \subseteq \mathtt{PH} \subseteq \mathtt{PSPACE}$, both of the inclusions $\mathtt{P}^{\sharp\mathtt{P}} \subseteq \mathtt{IP}$ and $\mathtt{PSPACE} \subseteq \mathtt{IP}$ are non-relativizing. Therefore, we actually know that $\mathtt{PSPACE} = \mathtt{IP}$, yet there exists a relativized world where these two classes are not equal.

Using virtually the same ideas, along with a combinatorial lemma, Babai, Fortnow, and Lund proved that $\mathtt{MIP} = \mathtt{NEXP}$ in [BFL] i.e. the set of languages with multiple prover interactive protocols is exactly the set of languages decided by nondeterministic exponential-time Turing machines.

2.2 Algebrizing Interactive Proof Results

In this section we show that the non-relativizing results of the previous section *algebraically relativize*, or *algebrize* for short. Working off of the observation that the previous proof relies

Algorithm 2 Interactive Proof Protocol for TQBF

```
Input $B$ a closed simple quantified Boolean formula.
$A \leftarrow$ the arithmetization of $B$.
Receive $a$ from $P$ the supposed value of $A \mod p$.
$A = a_1 + A_1$ or $A = a_1 A_1$ where $a_1$ is a constant.
//Say $a_1$ nontrivial if $a_1 \neq 0$ when $A = a_1 + A_1$ or $a_1 \neq 1$ when $A = a_1 A_1$
while $A_1$ nonempty do
    if $a_1$ nontrivial then //have to adjust for the constant $a_1$
        if $A = a_1 + A_1$ then
            $a \leftarrow a - a_1 \mod p$
            $A \leftarrow A_1$
        end if
        if $A = a_1 A_1$ then
            if $a_1 = 0$ then //trying to divide by 0
                Stop and accept if and only if $a = 0$.
            else
                $a \leftarrow a/a_1 \mod p$
                $A \leftarrow A_1$
            end if
        end if
    else
        $A' \leftarrow A_1$ with leftmost $\prod$ or $\sum$ eliminated.
        Receive polynomial $q(z_i)$ describing $A'$ from $P$.
        if $A_1$ has leftmost $\sum$ then
            Check $q(0) + q(1) = a$. Reject if not.
        end if
        if $A_1$ has leftmost $\prod$ then
            Check $q(0)q(1) = a$. Reject if not.
        end if
        Choose randomly $r_i \in \mathbb{Z}_p$ and send it to $P$.
        $A \leftarrow A'(r_i)$.
        $a = q(r_i)$.
    end if
end while
if $a = a_1$ then
    accept
else
    reject
end if
```

on treating a Boolean formula as an arithmetic expression, Aaronson and Wigderson [AW] defined a notion of an oracle that is not only a collection of Boolean functions, but a collection of polynomials extending Boolean functions. The polynomials are over finite fields, but agree with the Boolean functions on Boolean inputs. With this definition the notion of an algebrizing inclusion or separation is similar to the notion of relativization, but with some differences.

Definition 37. An *oracle* A is a collection of Boolean functions $\{A_n : \{0,1\}^n \to \{0,1\} \mid n \in \mathbb{N}\}$. For a complexity class $\mathtt{C}$ denote by $\mathtt{C}^A$ the class of languages decidable by a $\mathtt{C}$ machine that can query A_n for any n. Sometimes we will refer to the oracle A as a language, meaning that a string $x \in \{0,1\}^n$ is in the language A if and only if $A_n(x) = 1$.

For certain complexity classes it is controversial whether to allow the machine to make superpolynomial queries to the oracle. This trouble usually arises when considering space bounded complexity classes such as `PSAPCE` [For]. To avoid confusion let $\mathtt{C}^{A[poly]}$ denote that the $\mathtt{C}$ machine can only make queries of polynomial length. Now we define the main idea of extending an oracle to a low degree polynomial that agrees with the oracle on Boolean inputs.

Definition 38 ([AW]). Let $A_n : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then a polynomial $\widetilde{A}_n^{\mathbb{F}} : \mathbb{F}^n \to \mathbb{F}$ is an *extension of* A_n *over* $\mathbb{F}$ if $A_n(x_1, \ldots, x_n) = \widetilde{A}_n^{\mathbb{F}}(x_1, \ldots, x_n)$ whenever $(x_1, \ldots, x_n) \in \{0,1\}^n$. Let $A = \{A_n\}$ be an oracle. An *oracle extension* $\widetilde{A}$ of A is a collection of polynomials comprised of an extension $\widetilde{A}_n^{\mathbb{F}}$ of A_n for every $n \in \mathbb{N}$ and every finite field $\mathbb{F}$ such that

- there exists a constant c such that $mdeg(\widetilde{A}_n^{\mathbb{F}}) \leq c$ for every $n, \mathbb{F}$.

Here $mdeg(\widetilde{A}_n^{\mathbb{F}})$ is the maximum multi-degree of the polynomial $\widetilde{A}_n^{\mathbb{F}}$, i.e. the greatest degree of any individual variable x_i occurring in the polynomial. For a complexity class $\mathtt{C}$ denote by $\mathtt{C}^{\widetilde{A}}$ the class of languages decidable by a $\mathtt{C}$ machine that can query $\widetilde{A}_n^{\mathbb{F}}$ for any n and $\mathbb{F}$.

It is important to note that when a Turing machine has access to $\widetilde{A}$ the queries it is allowed to make are comprised of a binary representation of a finite field together with an finite sequence of integers $(z_1, \ldots, z_k)$ and the address i of the bit of the value $\widetilde{A}_n^{\mathbb{F}}(z_1, \ldots, z_n)$ that is to be returned by the oracle. We will often refer to $\widetilde{A}$ as a collection of polynomials, one for every n and $\mathbb{F}$, but we must keep in mind the accessing mechanism that is meant by the notation $M^{\widetilde{A}}$ for some Turing machine M. Of course a Turing machine M can obtain the entire output $\widetilde{A}_n^{\mathbb{F}}(z_1, \ldots, z_n)$ by querying every address i, and in the description of algorithms we will say things like "M queries $\widetilde{A}$ for the value $\widetilde{A}_n^{\mathbb{F}}(z_1, \ldots, z_n)$". Now we define the algebraic extension analogues of relativizing inclusions and separations.

Definition 39 ([AW]).

(a) A complexity class inclusion $\mathtt{C} \subseteq \mathtt{D}$ *algebrizes* if $\mathtt{C}^{A} \subseteq \mathtt{D}^{\widetilde{A}}$ for all oracles A and oracle extensions $\widetilde{A}$.

(b) A complexity class separation $\mathtt{C} \not\subseteq \mathtt{D}$ *algebrizes* if $\mathtt{C}^{\widetilde{A}} \not\subseteq \mathtt{D}^{A}$ for all oracles A and oracle extensions $\widetilde{A}$.

If (a) does not hold for an inclusion then we say that it is a *non-algebrizing* inclusion. Similarly a separation is *non-algebrizing* if (b) does not hold.

The most obvious difference from the usual definition of relativization is the asymmetry. For inclusions only the superset complexity class has access to the extension polynomials, whereas for separations only the subset complexity class has access to the extension polynomials. Note that for proving that a result is non-algebrizing this definition is stronger: if we prove that a result does not algebrize under this definition then it does not algebrize under the definition where both complexity classes have access to the extension polynomials (e.g. if for an inclusion there exists an oracle A such that $\mathtt{C}^{A} \not\subseteq \mathtt{D}^{\widetilde{A}}$, then $\mathtt{C}^{\widetilde{A}} \not\subseteq \mathtt{D}^{\widetilde{A}}$ as well since $\mathtt{C}^{A} \subseteq \mathtt{C}^{\widetilde{A}}$).

On the other hand, this definition is weaker for proving that an existing result algebrizes (e.g. for an inclusion we need only show $\mathtt{C}^{A} \subseteq \mathtt{D}^{\widetilde{A}}$ rather than $\mathtt{C}^{\widetilde{A}} \subseteq \mathtt{D}^{\widetilde{A}}$). The reason Aaronson and Wigderson give for choosing this definition is that, "under the definition where both complexity classes are given the extension polynomials we do not know how to prove that existing results algebrize" [AW]. We will briefly examine this issue after proving that $\mathtt{P}^{\sharp\mathtt{P}} \subseteq \mathtt{IP}$ and $\mathtt{PSPACE} \subseteq \mathtt{IP}$ algebrize under the definition given above.

Consider proving $\mathtt{P}^{\sharp\mathtt{P}^{A}} \subseteq \mathtt{IP}^{\widetilde{A}}$. Given any oracle A we would like to (1) define a $\mathtt{P}^{\sharp\mathtt{P}^{A}}$-hard language L and (2) show that $\mathsf{L} \in \mathtt{IP}^{\widetilde{A}}$. By $\mathtt{IP}^{\widetilde{A}}$ we mean the class of languages that have interactive protocols where the probabilistic polynomial-time verifier V has access to the oracle $\widetilde{A}$. We begin by defining the appropriate language L as in [AW].

Definition 40. Let a *formula circuit over* $\mathbb{F}$ be a circuit with

- $+$, $\times$ internal gates
- x_i, $1 - x_i$ input nodes and constant nodes $\alpha \in \mathbb{F}$.

The *size* of a formula circuit is the number of gates. Such a circuit represents a polynomial in the natural way by propagating through the gates and applying the operation specified by the gate. Let $\sharp\mathsf{FSAT}_{L,\mathbb{F}}$ be the following problem. On input a formula circuit of size at most L represented by polynomial p over $\mathbb{F}$, compute

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} p(x_1, \ldots, x_n).$$

Let $\sharp\mathsf{FSAT}$ be the same problem with input $\langle L, \mathbb{F}, p\rangle$, i.e. the size and finite field are provided as input. Note that the size of a $\langle L, \mathbb{F}, p\rangle$ input instance is taken to be $n = L \cdot \log \mid \mathbb{F} \mid$ ($\log \mid \mathbb{F} \mid$ bits to specify $\mathbb{F}$, and at most L gates each of which requiring at most $\log \mid \mathbb{F} \mid$ bits to specify if it is a constant node).

This problem corresponds to the classical $\sharp\mathsf{P}$-complete problem $\sharp\mathsf{CircuitSAT}$ of counting the number of satisfying assignments of a circuit. Indeed, if the input circuit is a classical circuit rather than a formula circuit, then the summation represents the number of satisfying assignments. Now for an oracle A and extension $\widetilde{A}$ let $\sharp\mathsf{FSAT}^{\widetilde{A}}$ be the same problem except that the formula circuit is allowed to have $\widetilde{A}$ gates, i.e. gates with arbitrary fan-in k that on input $a_k, \ldots, a_k \in \mathbb{F}^k$ output $\widetilde{A}(a_1, \ldots, a_k)$. If p is the polynomial representing such a formula circuit of size L then $deg(p) \leq L^2 mdeg(\widetilde{A})$ where recall that $mdeg(\widetilde{A}_n) \leq c = mdeg(\widetilde{A})$ for some constant c irrespective of n.

This is the language L that we will use to establish (1) and (2).

Lemma 41. *For any oracle A and any extension $\widetilde{A}$ the problem $\sharp\mathsf{FSAT}^{\widetilde{A}}$ is $\mathsf{P}^{\sharp\mathsf{P}^A}$-hard under randomized reductions.*

Proof. Consider the $\mathsf{P}^{\sharp\mathsf{P}^A}$-complete problem $\sharp\mathsf{CircuitSAT}^A$of computing

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} C^A(x_1, \ldots, x_n),$$

where C^A is a classical circuit containing A gates. This is simply the relativized version of $\sharp\mathsf{CircuitSAT}$, the $\mathsf{P}^{\sharp\mathsf{P}}$-complete problem mentioned above. We will reduce this problem to $\sharp\mathsf{FSAT}^{\widetilde{A}}$. Given a circuit C^A construct a polynomial p as follows. For each circuit input x_a there will be a variable x_a and for each gate g there will be a variable x_g. The polynomial p is the product of the following arithmetic expressions that enforce the correct propagation of values through the circuit. Suppose an AND gate g computes the AND of gates i and j. Then $x_g = x_i \wedge x_j$ must hold, which we arithmetize as

$$x_i x_j x_g + (1 - x_j x_j)(1 - x_g).$$

Similarly, suppose an OR gate g computes the OR of gates i and j. Then we arithmetize $x_g = x_i \vee x_j$ as

$$(1 - x_i)(1 - x_j)(1 - x_g) + x_i(1 - x_j)x_g + (1 - x_i)x_j x_g + x_i x_j x_g.$$

A NOT gate g with input i can be represented by $(x_i \vee x_g) \wedge (\overline{x}_i \vee \overline{x}_g)$, which can be arithmetized by the methods above by inserting $(1 - x_i)$ and $(1 - x_g)$ for $\overline{x}_i$ and $\overline{x}_g$ respectively. Alternatively, NOT gates can be handled at the arithmetization step of the subsequent gate by inserting $1 - x_i$ as input. Finally, for an oracle gate g arithmetize the constraint $x_g = A_k(x_{i_1}, \ldots, x_{i_k})$ with

$$x_g \widetilde{A}_k^{\mathbb{F}}(x_{i_1}, \ldots, x_{i_k}) + (1 - x_g)(1 - \widetilde{A}_k^{\mathbb{F}}(x_{i_1}, \ldots, x_{i_k}))$$

where we want the order of $\mathbb{F}$ to be large enough as not to affect the sum over Boolean inputs. To do this we need to select a prime $q > 2^n$ for the order of $\mathbb{F}$, which can be done in randomized polynomial time.

If the resulting polynomial is $p(x_1, \ldots, x_n, y_1, \ldots, y_l)$, then

$$\sum_{x_1,\ldots,x_n,y_1,\ldots,y_l \in \{0,1\}} p(x_1, \ldots, x_n, y_1, \ldots, y_l) = \sum_{x_1,\ldots,x_n \in \{0,1\}} C^A(x_1, \ldots, x_n).$$

□

This proof is essentially the arithmetized version of the proof reducing a circuit to a Boolean formula. Some complexity texts prove the NP-completeness of 3SAT directly, while others first prove that CircuitSAT is NP-complete and then reduce CircuitSAT to 3SAT. The second approaches requires a reduction that encodes the correct functioning of each gate with a Boolean formula. We have performed the same encoding here except in the context of arithmetic formulas. In this setting we are able to simulate the oracle with an oracle extension, which allows us to adapt the interactive proof protocols that we have seen before to the problem $\sharp\mathsf{FSAT}^{\widetilde{A}}$.

Proposition 42. *If we restrict to $\sharp\mathsf{FSAT}^{\widetilde{A}}$ instances with $\mid \mathbb{F} \mid \geq 3L^3 mdeg(\widetilde{A})$ then $\sharp\mathsf{FSAT}^{\widetilde{A}} \in \mathtt{IP}^{\widetilde{\mathtt{A}}}$.*

Proof. Given an input instance $\langle L, \mathbb{F}, p \rangle$ we would like an interactive protocol verifying the value given by the prover for $\sum_{x_1,\ldots,x_n \in \{0,1\}} p(x_1, \ldots, x_n)$. The protocol is the same as we have seen before. The prover P sends the value a. The verifier V then requests P to send coefficients for the polynomial

$$p_1(x) = \sum_{x_2,\ldots,x_n \in \{0,1\}} p(x, x_2, \ldots, x_n)$$

and receives the polynomial $q_1(x)$. The verifier checks that $q_1(0) + q_1(1) = a$ (if not V rejects), picks a random $r_1 \in \mathbb{F}$, sends r_1 to P, and requests the coefficients for the polynomial

$$p_2(x) = \sum_{x_3,\ldots,x_n \in \{0,1\}} p(r_1, x, x_3 \ldots, x_n).$$

The prover sends a new polynomial $q_2(x)$ and V checks that $q_1(r_1) = q_2(0) + q_2(1)$ (if not V rejects). This process continues until V requests the coefficients for the polynomial

$$p_n(x) = p(r_1, r_2, \ldots, r_{n-1}, x)$$

and receives $q_n(x)$. As usual V checks $q_{n-1}(r_{n-1}) = q_n(0) + q_n(1)$ and rejects if the test fails. Then V picks a final randomly chosen $r_n \in \mathbb{F}$ and at this point V can evaluate

$p_n(r_n) = p(r_1, \ldots, r_n)$ on its own using the formula circuit representation of p, along with its oracle extension $\widetilde{A}$. If $q_n(r_n) = p_n(r_n)$ then V accepts and rejects otherwise.

The proof of completeness is the same as we have seen before. Any truthful prover will always be able to back up the claims. On the other hand, if P gives an initial value a that is incorrect, P can get away with this lie at each step i with probability at most $\frac{deg(p)}{|\mathbb{F}|}$. Since there are n steps, by the union bound a malicious prover P can successfully trick V with probability at most

$$\frac{n \cdot deg(p)}{\mid \mathbb{F} \mid} \leq \frac{L \cdot L^2 mdeg(\widetilde{A})}{3L^3 mdeg(\widetilde{A})} = \frac{1}{3}.$$

□

Theorem 43 ([AW]). *For any oracle A and extension $\widetilde{A}$, $\mathtt{P}^{\sharp\mathtt{P}^A} \subseteq \mathtt{IP}^{\widetilde{A}}$.*

Proof. The previous proposition gives an $\mathtt{IP}^{\widetilde{A}}$ protocol for $\sharp\mathsf{FSAT}^{\widetilde{A}}$ when $\mid \mathbb{F} \mid > L^3 mdeg(\widetilde{A})$. Such instances are $\mathtt{P}^{\sharp\mathtt{P}}$-hard under randomized reductions because of the previous lemma where we can choose with high probability a prime $q \geq 2^n > L^3 mdeg(\widetilde{A})$ for all large enough n. A randomized reduction suffices because V can be programmed to select a prime q in randomized polynomial time that will not affect the sum in the protocol with high probability. □

Alternatively, one could combine the lemma and proposition into a single protocol for the $\mathtt{P}^{\sharp\mathtt{P}^A}$-complete problem of computing $\sum_{x_1,\ldots,x_n \in \{0,1\}} C^A(x_1, \ldots, x_n)$. On input C^A the verifier converts the circuit to a polynomial representation containing $\widetilde{A}$ polynomial expressions, and sends this to the prover. The prover then responds with a large enough prime q along with a certificate of primality. They then proceed with the protocol given in the proposition. One final thing to note is that we could have even allowed $mdeg(\widetilde{A})$ to be polynomially large.

Showing that $\mathtt{PSPACE} \subseteq \mathtt{IP}$ algebrizes is done similarly so we briefly sketch the proof. Recall that the problem TQBF defined above is $\mathtt{PSPACE}$-complete. By examining the proof of this standard result it is easy to see that for an oracle A, TQBF^A is $\mathtt{PSPACE}^A$-complete where TQBF^A is the problem of determining whether a closed quantified Boolean formula, possibly containing $A(x_{i_1}, \ldots, x_{i_k})$ predicates, is true or not. Here k is a number that is at most polynomial in the number of inputs. Let $\widetilde{A}$ be any extension of A. Using Shamir's method for arithmetizing a quantified Boolean formula, TQBF^A can be reduced to an arithmetized version of the problem that contains $\widetilde{A}_h^{\mathbb{F}}$ polynomials. This step corresponds to the above lemma.

The arithmetized version of a problem instance is an arithmetic expression containing $\widetilde{A}$'s where every variable x_i occurs in either a $\prod_{x_i \in \{0,1\}}$ expression or a $\sum_{x_i \in \{0,1\}}$ expression i.e. every variable is "arithmetically quantified". As before, the order of the field is established either probabilistically by the verifier, or given by the prover with a certificate.

Then the verifier and prover can carry out Shamir's interactive protocol for $\mathtt{PSPACE}$ with the only difference being in the last step when the verifier evaluates the final expression using the $\widetilde{A}$ oracle. Taken together, this procedure shows that the $\mathtt{PSPACE}^A$-complete language TQBF^A is in $\mathtt{IP}^{\widetilde{A}}$ for any $A, \widetilde{A}$.

Theorem 44. *For any oracle A and extension $\widetilde{A}$, $\mathtt{PSPACE}^{A[poly]} \subseteq \mathtt{IP}^{\widetilde{A}}$.*

2.3 $\mathtt{P}^{\sharp\mathtt{P}^{\widetilde{A}}} \subseteq \mathtt{IP}^{\widetilde{A}}$

Now we briefly consider a naive attempt at proving that $\mathtt{P}^{\sharp\mathtt{P}} \subseteq \mathtt{IP}$ algebrizes under the altered symmetrical definition of an algebrizing inclusion. Using the methods we have already seen, consider trying to prove for any oracle A and extension $\widetilde{A}$ that $\mathtt{P}^{\sharp\mathtt{P}^{\widetilde{A}}} \subseteq \mathtt{IP}^{\widetilde{A}}$. The first step in proving $\mathtt{P}^{\sharp\mathtt{P}^A} \subseteq \mathtt{IP}^{\widetilde{A}}$ was to identify the $\mathtt{P}^{\sharp\mathtt{P}^A}$-complete language $\sharp\mathsf{CircuitSAT}^A$. The analogous choice for a $\mathtt{P}^{\sharp\mathtt{P}^{\widetilde{A}}}$-complete problem is $\sharp\mathsf{CircuitSAT}^{\widetilde{A}}$. Consider an instance of this problem $C^{\widetilde{A}}$ where C is a classical circuit operating on Boolean values. Here it is important to note that $\widetilde{A}$ represents an oracle that is queried by providing input $x_{i_1}, \dots, x_{i_k}, j$ where j is the requested bit of the value $\widetilde{A}_k^{\mathbb{F}}(x_{i_1}, \dots, x_{i_k})$. We can arithmetize the regular gates of C by the usual methods, but it is unclear how to arithmetize the $\widetilde{A}$ oracle gates by only using $\widetilde{A}$ polynomial expressions. The issue is that the $\widetilde{A}$ oracle gates do not respect the arithmetic structure of the $\widetilde{A}$ polynomial expressions because they pick out a particular bit of the output. To summarize in the language of the lemma, it is unclear whether it is possible to show that $\sharp\mathsf{FSAT}^{\widetilde{A}}$ is $\mathtt{P}^{\sharp\mathtt{P}^{\widetilde{A}}}$-complete.

However, if we view $\widetilde{A}$ as just some Boolean oracle (as it is from C's point of view), then we can use the exact same method as before to show that $\mathtt{P}^{\sharp\mathtt{P}^{\widetilde{A}}} \subseteq \mathtt{IP}^{\widetilde{\widetilde{A}}}$ where $\widetilde{\widetilde{A}}$ is an extension of the Boolean oracle $\widetilde{A}$. Thus, we once again get an asymmetrical inclusion. This is just a small bit of reasoning to suggest that perhaps it is the correct definition to stipulate that an inclusion $\mathtt{C} \subseteq \mathtt{D}$ algebrizes if $\mathtt{C}^A \subseteq \mathtt{D}^{\widetilde{A}}$ for all A, $\widetilde{A}$.

Aaronson and Wigderson note the relevance of considering a hierarchy of algebrizations $A, \widetilde{A}, \widetilde{\widetilde{A}}, \widetilde{\widetilde{\widetilde{A}}}, \dots$ where $\widetilde{A}$, a low degree polynomial extension of A, is itself viewed as a Boolean oracle that is then extended by a low degree polynomial extension $\widetilde{\widetilde{A}}$, and so on. Consider the transitivity property of relativization. If two inclusions $\mathtt{C} \subseteq \mathtt{D}$ and $\mathtt{D} \subseteq \mathtt{E}$ relativize, then $\mathtt{C} \subseteq \mathtt{E}$ relativizes. Aaronson and Wigderson mention that they do not know whether the analogous transtivity holds for algebrization. However, it is true that if $\mathtt{C}^A \subseteq \mathtt{D}^{\widetilde{A}}$ and $\mathtt{D}^A \subseteq \mathtt{E}^{\widetilde{A}}$ for all A, $\widetilde{A}$, then $\mathtt{C}^A \subseteq \mathtt{E}^{\widetilde{\widetilde{A}}}$ for all A, $\widetilde{\widetilde{A}}$.

Under this notation an inclusion $\mathtt{C} \subseteq \mathtt{D}$ is *double-algebrizing* if $\mathtt{C}^A \subseteq \mathtt{D}^{\widetilde{\widetilde{A}}}$ for all A, $\widetilde{A}$, *triple-algebrizing* if $\mathtt{C}^A \subseteq \mathtt{D}^{\widetilde{\widetilde{\widetilde{A}}}}$ for all A, $\widetilde{\widetilde{\widetilde{A}}}$ and so forth. In general, a k-algebrizing result is $k+1$-algebrizing for the same reason that a relativizing result algebrizes. On the other hand, a $k+1$-algebrizing result is not necessarily k-algebrizing, so there is a hierarchy of algebrizations. Finally, they prove that $\texttt{P}$ vs. $\texttt{NP}$ is non-k-algebrizing for any k, but mention that double-algebrizing techniques may be sufficient to resolve $\texttt{P}$ vs. $\texttt{RP}$ or $\texttt{NEXP}$ vs. $\texttt{P}_{/\texttt{poly}}$.

2.4 $\texttt{P}$ vs. $\texttt{NP}$ is non-algebrizing

In this section we prove by the arguments of [AW] that resolving the $\texttt{P}$ vs. $\texttt{NP}$ question will require "non-algebrizing techniques." To do so we would like to prove two things:

(a) there exists an oracle A and oracle extension $\widetilde{A}$ such that $\texttt{NP}^{\widetilde{A}} \subseteq \texttt{P}^{A}$ and

(b) there exists an oracle B and oracle extension $\widetilde{B}$ such that $\texttt{NP}^{B} \not\subseteq \texttt{P}^{\widetilde{B}}$.

We begin by proving (a) in a similar way to the usual result that $\texttt{NP}^{A} \subseteq \texttt{P}^{A}$ when A is a $\texttt{PSPACE}$-complete language. To do so we need a few facts about multilinear extensions of Boolean functions. Let $\mathbb{F}$ be a finite field. An extension $\widetilde{A}_n^{\mathbb{F}}$ of a Boolean function A_n is multilinear if $mdeg(\widetilde{A}_n^{\mathbb{F}}) \leq 1$ and multiquadratic if $mdeg(\widetilde{A}_n^{\mathbb{F}}) \leq 2$. For a Boolean point $b = b_1, \ldots, b_n$ define

$$\delta_b(x_1, \ldots, x_n) = \prod_{\{i|b_i=1\}} x_i \prod_{\{i|b_i=0\}} (1 - x_i).$$

These functions evaluate to 1 only on their respective point b and 0 elsewhere. They form a basis for all multilinear functions $f : \mathbb{F}^n \to \mathbb{F}$.

Lemma 45. *Let $f : \mathbb{F}^n \to \mathbb{F}$ be a multilinear function. Then f can be written uniquely as*

$$f(x) = \sum_{b \in \{0,1\}^n} f_b \delta_b(x)$$

where $f_b \in \mathbb{F}$. For any Boolean point b, $f(b) = f_b$.

Proof. This follows from linear algebra when considering the dual space of the appropriate vector space. □

Using this lemma we may identify any multilinear polynomial with the coefficients f_b.

Lemma 46. *Let $A_n : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then A_n has a unique multilinear extension $\widetilde{A}_n^{\mathbb{F}} : \mathbb{F}^n \to \mathbb{F}$.*

Proof. Consider the multilinear extension polynomial

$$\widetilde{A}_n^{\mathbb{F}}(x) = \sum_{b \in \{0,1\}^n} A(b) \delta_b(x),$$

which agrees with A_n on Boolean inputs and is unique by the previous lemma. □

Babai, Fortnow and Lund [BFL] noticed that the multilinear extension of a $\texttt{PSPACE}$ language A is also in $\texttt{PSPACE}$.

Proposition 47. *[BFL] Let A be the characteristic function of a $\texttt{PSPACE}$ language and $\widetilde{A}$ the unique multilinear extension of A over a field $\mathbb{F}$. Then $\widetilde{A} \in \texttt{PSPACE}$.*

Proof. Note that $\texttt{PSPACE}^A = \texttt{PSPACE}$, so it suffices to show that $\widetilde{A} \in \texttt{PSPACE}^A$. We will construct an alternating polynomial-time Turing machine with access to A that computes $\widetilde{A}$. Recalling the standard result that an alternating Turing machine can be simulated by a $\texttt{PSPACE}$ machine this will prove that $\widetilde{A} \in \texttt{PSPACE}^A$.

On input $x_1, \dots, x_n$ let the alternating Turing machine M existentially guess the value $z = \widetilde{A}_n^{\mathbb{F}}(x_1, \dots, x_n)$ and guess the linear polynomial $p_1(y) = \widetilde{A}_n^{\mathbb{F}}(y, x_2, \dots, x_n)$. Then universally choose $z_1 \in \{0,1\}$ and existentially guess the linear polynomial $p_2(y) = \widetilde{A}_n^{\mathbb{F}}(z_1, y, x_3, \dots, x_n)$ and so forth until $z_1, \dots, z_n$ have been chosen. Now M uses the oracle A to verify that $A_n(z_1, \dots, z_n) = 1$. □

Therefore, for a $\texttt{PSPACE}$-complete language A, the multilinear extension $\widetilde{A}$ is in $\texttt{PSPACE}$. In particular, $\widetilde{A}$ is also $\texttt{PSPACE}$-complete. With this proposition the proof of (a) follows similarly to the usual $\texttt{NP}^A = \texttt{P}^A$ proof.

Theorem 48. *There exists an oracle A and extension $\widetilde{A}$ such that $\texttt{NP}^{\widetilde{A}} \subseteq \texttt{P}^A$.*

Proof. Let A be the characteristic function of a $\texttt{PSPACE}$-complete language, so that $\texttt{P}^A = \texttt{PSPACE}$. Then by the proposition $\widetilde{A} \in \texttt{PSPACE}$ which implies that $\texttt{NP}^{\widetilde{A}} \subseteq \texttt{PSPACE}$. Therefore, $\texttt{NP}^{\widetilde{A}} \subseteq \texttt{PSPACE} = \texttt{P}^A$ as desired. (Since $\widetilde{A}$ is $\texttt{PSPACE}$-complete, we actually have the equality $\texttt{NP}^{\widetilde{A}} = \texttt{P}^A$.) □

The same exact proof yields the following theorem.

Theorem 49. *[AW] There exists an oracle A and an extension $\widetilde{A}$ such that $\texttt{PSPACE}^{\widetilde{A}[poly]} \subseteq \texttt{P}^A$.*

Furthermore, since $\widetilde{A}$ is $\texttt{PSAPCE}$-complete, the unique multilinear extension $\widetilde{\widetilde{A}}$ of the binary representation $\widetilde{A}$ is also $\texttt{PSPACE}$-complete. Therefore, $\texttt{NP}^{\widetilde{\widetilde{A}}} = \texttt{PSPACE} = \texttt{P}^A$ as well. In fact, this holds for any k-extension, so a potential proof that $\texttt{NP} \not\subseteq \texttt{P}$ will require non-k-algebrizing techniques for all k as we alluded to above.

Now we turn to proving (b), which will be similar to the proof of the standard result. Recall that the oracle B such that $\texttt{NP}^B \not\subseteq \texttt{P}^B$ is defined in stages. For any oracle B let

$$L_B = \cup_{n \in \mathbb{N}} \left\{1^n : \exists w \in \{0,1\}^n \text{ s.t. } w \in B\right\},$$

so that $L_B \in \texttt{NP}^B$. As in the classical case, we will show that $L_B \notin \texttt{P}^{\widetilde{\texttt{B}}}$. However, now the extension $\widetilde{B}$ needs to be constructed so that the $\texttt{P}^{\widetilde{B}}$ machine cannot exploit the extra algebraic structure of $\widetilde{B}$ to decide L_B.

Recall the classical strategy: the existence of B is established by considering at stage i the ith oracle machine M_i in an enumeration of polynomial-time machines with oracle access to B. Fixing an appropriate n_i, if $M_i(1^{n_i})$ accepts then define $x \notin B$ for all $x \in \{0,1\}^{n_i}$. In this case $1^{n_i} \notin L_B$, but M_i accepts 1^{n_i}. Otherwise, if $M_i(1^{n_i})$ rejects then define $x \notin B$ for every x that the computation $M_i(1^{n_i})$ queried. Having set n_i so that $M_i(1^{n_i})$ does not have time to query every $x \in \{0,1\}^{n_i}$, there exists a $y \in \{0,1\}^{n_i}$ for which we can declare $y \in B$. In this case $1^{n_i} \in L_B$, but M_i rejects 1^{n_i}.

From the perspective of M_i if it ever receives a YES answer on a query of length n_i then it accepts. To avoid this we construct B such that the oracle answers NO on M_i's queries of length n_i and M_i does not have time to ask about all of them. Thus once M_i's time is up it has only seen NO answers, so it does not know for certain whether there truly is no string of length n_i in B or whether it just didn't have time to ask about the right string. With whatever guess that M_i makes we can construct B such that the opposite is true.

In our algebrization setting M_i can query $\widetilde{B}^{\mathbb{F}}_{n_i}$ for any finite field $\mathbb{F}$. In this case we will construct B and $\widetilde{B}$ so that once M_i's time is up it does not know for certain whether $\widetilde{B}^{\mathbb{F}}_{n_i}$ is the actually the identically zero polynomial for every $\mathbb{F}$ or whether it just didn't have time to query the right value. If M_i accepts, then this case is easy: we define B to contain no strings of length n_i as before and just make $\widetilde{B}^{\mathbb{F}}_{n_i}$ the zero polynomial for all $\mathbb{F}$.

If M_i rejects, then we need to select a single Boolean string of length n_i to be in B, such that there exist extension polynomials that all evaluate to 1 on this string and 0 on all of the points that M_i had time to query. Then the extension polynomials will be genuine extensions of B. To rephrase, our goal is to make sure that we can construct extension polynomials such that there exists a single Boolean input on which all of the polynomials evaluate to 1 and evaluate to 0 on all of the inputs that M_i queried. A multilinear extension will not suffice for this purpose, but a multiquadratic extension will. We first establish some facts about multiquadratic extensions.

Lemma 50. *Let $\mathbb{F}$ be a field and $y_1, \ldots, y_t \in \mathbb{F}^n$. Then there exists a multilinear polynomial $f : \mathbb{F}^n \to \mathbb{F}$ such that*

- *$f(y_i) = 0$ for all i, and*
- *$f(w) = 1$ for at least $2^n - t$ points $w \in \{0,1\}^n$.*

Proof. Write an arbitrary $f : \{0,1\}^n \to \{0,1\}$ as $f(x) = \sum_{b \in \{0,1\}^n} f_b \delta_b(x)$ with f_b not yet specified. Then $f(y_i) = 0$ give t linear equations over $\mathbb{F}$ for the 2^n variables f_b. Therefore, by linear algebra there exists a solution with at least $2^n - t$ of the f_b's equal to 1. $\square$

It would be nice if such multilinear polynomials were sufficient for our purposes, but they are not. The problem is that on the remaining t Boolean points the multilinear polynomial f may not be Boolean, so it would not be a true Boolean extension. We fix this problem by multiplying two multilinear polynomials together to form a single multiquadratic polynomial.

Lemma 51. *Let $\mathbb{F}$ be a field and $y_1, \ldots, y_t \in \mathbb{F}^n$. Then for at least $2^n - t$ points $w \in \{0,1\}^n$ there exists a multiquadratic polynomial $g : \mathbb{F}^n \to \mathbb{F}$ such that*

- *$g(y_i) = 0$ for all i,*
- *$g(w) = 1$,*
- *$g(w') = 0$ for all $w' \in \{0,1\}^n$ such that $w' \neq w$.*

Proof. Take f from the previous lemma and choose a Boolean w such that $f(w) = 1$. Then setting $g(x) = f(x)\delta_w(x)$ gives a multiquadratic polynomial with the desired properties. □

We would like a similar result stating that there exist $g_{\mathbb{F}}$ for every $\mathbb{F}$ in some collection of fields $\mathcal{F}$. The $\mathcal{Y}_{\mathbb{F}}$'s in this proposition will represent the points on which M_i queries the extension polynomials.

Proposition 52. *Let $\mathcal{F}$ be a collection of fields and $\mathcal{Y}_{\mathbb{F}} \subseteq \mathbb{F}^n$. Let $s = \sum_{\mathbb{F} \in \mathcal{F}} \mid \mathcal{Y}_{\mathbb{F}} \mid$. Then for $2^n - s$ Boolean points w there exist multiquadratic polynomials $g_{\mathbb{F}} : \mathbb{F}^n \to \mathbb{F}$, one for every $\mathbb{F} \in \mathcal{F}$, such that*

- *$g_{\mathbb{F}}(y) = 0$ for all $y \in \mathcal{Y}_{\mathbb{F}}$ and $\mathbb{F} \in \mathcal{F}$,*
- *$g_{\mathbb{F}}(w) = 1$ for all $\mathbb{F} \in \mathcal{F}$,*
- *$g_{\mathbb{F}}(w') = 0$ for all Boolean $w' \neq w$ and $\mathbb{F} \in \mathcal{F}$.*

Proof. This is immediate from the previous lemma since each $\mathbb{F}$ can prevent the result from holding for at most $\mid \mathcal{Y}_{\mathbb{F}} \mid$ points. □

Now we are ready to prove the separation relative to an extension oracle following the outline given above.

Theorem 53. *[AW] There exists an oracle B and a multiquadratic extension $\widetilde{B}$ of B such that $\mathtt{NP}^B \not\subseteq \mathtt{P}^{\widetilde{B}}$.*

Proof. For any oracle B the language $L_B = \cup_{n \in \mathbb{N}} \{1^n : \exists w \in \{0,1\}^n \text{ s.t. } w \in B\}$ is in $\mathtt{NP}^B$. We will define an oracle B along with a multiquadratic extension $\widetilde{B}$ such that $L_B \notin \mathtt{P}^{\widetilde{B}}$. Consider the notation $L_B(1^n) = 1$ if $1^n \in L_B$ and $L_B(1^n) = 0$ otherwise. Adopt a similar notation for Turing machines.

Let $M_1, M_2, \ldots$ be an enumeration of $(n^{\log n})$-time deterministic Turing machines with oracle access to $\widetilde{B}$. The oracle B and extension $\widetilde{B}$ are defined in stages based on M_i's computation on 1^{n_i}. At any stage i it has been determined for a finite number of strings x whether $x \in B$, and for every $j < i$ there exists an n_j such that $L_B(1^{n_j}) \neq M_j(1^{n_j})$. Let

S_j be the set of indices n such that M_j on input 1^{n_j} has queried a $\widetilde{B}^{\mathbb{F}}_{n_j}$. Let $T_i = \cup_{j<i} S_j$. Then for any $n \in T_i$ every $\widetilde{B}^{\mathbb{F}}_n$ is already determined and will not change for any other stage $\geq i$.

Let n_i be the least index n such that $n \notin T_i$ and $2^n > n^{\log n}$. At stage i continue the construction of B and $\widetilde{B}$ by observing M_i's computation on input 1^{n_i}.

(a) If M_i ever makes a query to a $\widetilde{B}^{\mathbb{F}}_n$ for a $n \in T_i$ then respond consistently because $\widetilde{B}^{\mathbb{F}}_n$ is already completely determined by the previous stages.

(b) If M_i makes a query to a $\widetilde{B}^{\mathbb{F}}_n$ for an $n \notin T_i$ then return 0 for that query.

At the end of M_i's computation, let S_i be set of indices n such that M_i makes a query to some $\widetilde{B}^{\mathbb{F}}_n$. For the new indices $n \in S_i \setminus T_i$ such that $n \neq n_i$ define for all $\mathbb{F}$, $B^{\mathbb{F}}_n = 0$ to be the identically-zero polynomial. We now have two cases.

(a) If M_i accepts then set $\widetilde{B}^{\mathbb{F}}_{n_i}$ equal to the identically-zero polynomial.

(b) If M_i rejects then let $\mathcal{Y}_{\mathbb{F}}$ be the set of points $y \in \mathbb{F}^{n_i}$ that M_i queries $\widetilde{B}^{\mathbb{F}}_{n_i}$ on. Since $\sum \mid \mathcal{Y}_{\mathbb{F}} \mid < n_i^{\log n_i}$, by the proposition there exists a Boolean point w such that there exist multiquadratic polynomials $g_{\mathbb{F}} : \mathbb{F}^{n_i} \to \mathbb{F}$ such that

   - $g_{\mathbb{F}}(y) = 0$ for all $y \in \mathcal{Y}_{\mathbb{F}}$ for all $\mathbb{F}$,
   - $g_{\mathbb{F}}(w) = 1$,
   - $g_{\mathbb{F}}(w') = 0$ for all Boolean $w' \neq w$.

   Setting $\widetilde{B}^{\mathbb{F}}_{n_i} = g_{\mathbb{F}}$ for every $\mathbb{F}$ and $B(w) = 1$ implies that $L(1^{n_i}) = 1$, yet M_i rejects 1^{n_i}.

$\square$

We have shown two things: (1) that the non-relativizing interactive proof results algebrize, and (2) that resolving the P vs. NP question will require non-algebrizing techniques. In addition to (1), there are a number of other true complexity statements that are non-relativizing, yet are algebrizing. For example, one can show that NEXP $\subseteq$ MIP algebrizes by defining a convenient NEXP-complete language and following the details of the original proof.

There are also circuit lower bound results that avoid both the relativization barrier and natural proofs barrier. One such result that avoids both of the previous barriers is $\mathtt{MA}_{\mathtt{EXP}} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$ where $\mathtt{MA}_{\mathtt{EXP}}$ is the class of languages with a one round Merlin, Arthur protocol where the verifier Arthur is allowed exponential-time. It can be shown that this separation algebrizes, so although this result avoids the previous two barriers it is still subject to algebrization.

The second portion (2) says that current techniques that can avoid relativization and natural proofs are not enough to resolve $\mathtt{P}$ vs. $\mathtt{NP}$. In fact, many other open questions are non-algebrizing. It can be shown that there exists an oracle A and extension $\widetilde{A}$ such that $\mathtt{NP}^A \not\subseteq \mathtt{BPP}^{\widetilde{A}}$, and an oracle and extension such that $\mathtt{NEXP}^{\widetilde{A}} \subseteq \mathtt{P}_{/\mathtt{Poly}}{}^A$. There also exist $A, \widetilde{A}$ such that $\mathtt{RP}^A \not\subseteq \mathtt{P}^{\widetilde{A}}$ where $\mathtt{RP}$ is the randomized class with one-sided error. (Of course, our previous result that there exist oracles $A, \widetilde{A}$ such that $\mathtt{NP}^{\widetilde{A}} \subseteq \mathtt{P}^A$ handles the other direction, since $\mathtt{RP}^{\widetilde{A}} \subseteq \mathtt{NP}^{\widetilde{A}}$.) Therefore, even resolving $\mathtt{RP}$ vs. $\mathtt{P}$ will require non-algebrizing techniques. As mentioned above, it is not known whether there exist oracles $A, \widetilde{\widetilde{A}}$ such that $\mathtt{RP}^A \not\subseteq \mathtt{P}^{\widetilde{\widetilde{A}}}$, so although this problem is not solvable by the techniques of arithmetization, it may be solvable by techniques of double-algebrization.

3 Avoiding the Barriers

In this section we briefly touch on the main ideas of the recent results of Ryan Williams [WIL10], [WIL11]. We prove a central result that outlines this strategy for avoiding complexity barriers. The main idea lies in proving that upper bounds imply lower bounds by contradicting hierarchy theorems. In fact, this strategy can already be seen in proofs of much more elementary results in complexity theory. To illustrate, recall the following standard theorems regarding the conditional collapse of the polynomial hierarchy.

Theorem 54. *If* $\mathtt{P} = \mathtt{NP}$*, then* $\mathtt{P} = \sum_2$.

Theorem 55. *If* $\mathtt{EXP} \subseteq \mathtt{P}_{/\mathtt{poly}}$*, then* $\mathtt{EXP} = \sum_2$.

Here $\sum_2$ is the second level of the polynomial hierarchy. These theorems are then used to prove the following conditional lower bound.

Corollary 56. *If* $\mathtt{P} = \mathtt{NP}$ *then* $\mathtt{EXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$.

In other words, $\mathtt{P} = \mathtt{NP}$ and $\mathtt{EXP} \subseteq \mathtt{P}_{/\mathtt{poly}}$ cannot simultaneously hold. The proof is by contradiction: assume to the contrary that $\mathtt{EXP} \subseteq \mathtt{P}_{/\mathtt{poly}}$, so that $\mathtt{EXP} = \sum_2$. Then $\mathtt{P} = \mathtt{NP}$ implies that $\mathtt{P} = \sum_2 = \mathtt{EXP}$, but this contradicts the time hierarchy theorem.

Therefore, the upper bound $\mathtt{NP} \subseteq \mathtt{P}$ implies the lower bound $\mathtt{EXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$. Or equivalently a polynomial-time algorithm for an $\mathtt{NP}$-complete problem implies the lower bound $\mathtt{EXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$. The main idea behind Ryan Williams's work is that we do not have to assume an upper bound as strong as $\mathtt{NP} \subseteq \mathtt{P}$, but that even a slight improvement to the run-time of the trivial algorithm for an $\mathtt{NP}$-complete problem will be sufficient to prove weaker lower bounds such as $\mathtt{NEXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$.

Consider the trivial algorithm for solving deterministically a problem in $\mathtt{NP}$. If $L \in \mathtt{NP}$ has witnesses of length n^k, then enumerating over every witness and running the polynomial time verifier gives an algorithm with running time $O(2^{n^k} \cdot poly(n))$. For the concrete example of $\mathsf{CircuitSAT}$ this algorithm runs in $O(2^n \cdot n^k)$ time where the problem instance circuit has n inputs and n^k gates. However, for many $\mathtt{NP}$-complete problems there have been minor improvements to the trivial brute-force algorithm [WIL10], [CIP], [DH]. It turns out that a sufficiently strong, but not unreasonable, improvement to the correct problem yields lower bounds.

Our aim will be to prove the following theorem.

Theorem 57. *[WIL10] Suppose that there exists a superpolynomial function $s(n)$ such that* $\mathsf{CircuitSAT}$ *on circuits with n variables and n^k gates can be solved in $2^n \cdot poly(n^k)/s(n)$ time by a deterministic algorithm for all k. Then* $\mathtt{NEXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$.

Our argument will derive a contradiction to the nondeterministic-time hierarchy theorem. Recall the standard formulation.

Theorem 58. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be time-constructible functions such that $f(n+1) = o(g(n))$. Then* $\mathtt{NDTIME(f(n))} \subsetneq \mathtt{NDTIME(g(n))}$.

To accomplish our goal we need the notion of a witness circuit, which was studied by Impagliazzo, Kabanets, and Wigderson [IKW] in the setting of hardness-randomness tradeoffs (to prove $\mathtt{NEXP} \subseteq \mathtt{P}_{/\mathtt{poly}} \iff \mathtt{NEXP} = \mathtt{MA}$). The definitions we will need are as follows.

Definition 59. Let $L \in \mathtt{NTIME(t(n))}$ for some function t. A polynomial time algorithm V is *verifier* for L if

$$x \in L \iff \exists y\,(\mid y \mid \leq t(\mid x \mid)) \text{ such that } V(x, y) = 1.$$

Definition 60. A language $L \in \mathtt{NTIME(t(n))}$ has $S(n)$-size *witness circuits* if for every polynomial time verifier V for L there exists a size $S(n)$ circuit family $\{C_n\}$ such that

$$x \in L \iff V(x, w(x)) = 1$$

where $w(x)$ is the concatenation of the outputs $C_{|x|+|z|}(\langle x, z\rangle)$ evaluated over $z \in \{0,1\}^{\lceil \log(t(n)) \rceil + 1}$ in lexicographic order.

The input x can be thought of as being fixed and $C_{|x|+|z|}(\langle x, \mathbf{z}\rangle)$ producing the witness string $w(x)$ as $\mathbf{z}$ ranges lexicographically over $\{0,1\}^{\lceil \log(t(n)) \rceil + 1}$. Thus, if the circuit family $\{C_n\}$ has small size, every input x that is a member of L has a witness that is easily encoded. We quote, but do not prove, a result that follows from [IKW] and proved in [WIL10]. The proof uses previous work of [BFNW] on hardness-randomness relations.

Lemma 61. *If* $\mathtt{NEXP} \subseteq \mathtt{P}_{/\mathtt{poly}}$ *then every language in* $\mathtt{NEXP}$ *has polynomial size witness circuits.*

We will use the contrapositive of this by first proving that NEXP does not have polynomial size witness circuits if CircuitSAT has a faster algorithm. This will prove that NEXP $\not\subseteq$ P$_{/\mathtt{poly}}$. Consider the following lemma that quantifies, a bit more carefully than usual, the exponentially padded version of the Cook-Levin reduction of languages to 3SAT, along with a standard fact about converting an algorithm into a circuit.

Lemma 62. *Every language $L \in \mathtt{NTIME}[2^n]$ can be reduced to* 3SAT *instances of $b \cdot 2^n \cdot n^4$ size. In particular, there exists an algorithm that on input x and an index $i \in [b \cdot 2^n \cdot n^4]$ outputs the ith clause of the 3SAT formula corresponding to x in $O(n^4)$ time.*

Lemma 63. *An algorithm running in $t(n)$-time can be simulated by a circuit family of size $t(n)^2$.*

Now we are ready to prove the main result.

Theorem 64. *Let $s(n)$ be a superpolynomial function and let c, k be constants. Let $a(n)$ be a monotonically increasing, unbounded function such that*

- $\frac{s(n)}{(n^k+n^8)^c} \geq \Omega(n^4 \cdot a(n))$ *and*
- $n^k \leq O(\frac{2^n}{n \cdot a(n)})$.

Suppose that CircuitSAT *on n variables and m gates can be solved deterministically in $O(\frac{2^n \cdot m^c}{s(n)})$ time for some constant c. Then* $\mathtt{NTIME}[2^n]$ *does not have n^k-sized witness circuits.*

This statement can be proved in greater generality by replacing n^k with any function $T(n)$ that adheres to the hypothesis, but we choose n^k for concreteness because it will suffice for our purposes.

Proof. First note that such a function $a(n)$ does indeed exists for any choice of c, k. Suppose to the contrary that every language in $\mathtt{NTIME}(2^n)$ has witness circuits of size n^k. We will show that this contradicts the nondeterministic-time hierarchy theorem.

Let $L \in \mathtt{NTIME}(2^n)$. Then by lemma 62, L can be reduced to 3SAT instances of size $b \cdot 2^n \cdot n^4$ for a constant b. Define $V(x, y)$ to be the polynomial time verifier that on input x reduces x to the 3SAT instance ϕ_x of size at most $b \cdot 2^n \cdot n^4$, plugs in y_i for the ith variable in ϕ_x, and returns the value that the formula evaluates to. Since L has n^k-size witness circuits, for all $x \in L$ there exists a witness string y of length at most $b \cdot 2^n \cdot n^4$ such that there exists a circuit C^x that encodes y. Note that C^x takes as input strings of length $l = \log(b \cdot 2^n \cdot n^4)$ so that the concatenation of $C^x(z)$ over $z \in \{0,1\}^l$ equals y.

We give a nondeterministic algorithm N for deciding L. On an input x, nondeterministically guess the witness circuit C^x. Because of the size of C^x this takes $n^k \cdot \log(n^k) \leq O(\frac{2^n \cdot \log(n^k)}{a(n) \cdot n}) \leq O(\frac{2^n}{a(n)})$ nondeterministic bits by the second bullet point in the assumptions.

Construct a circuit D with the help of C^x. Recall that given an integer $i \in [b \cdot 2^n \cdot n^4]$ the ith clause of ϕ_x can be computed in $O(n^4)$ time, so by lemma 63 there is a circuit E of size $O(n^8)$ that takes l bits representing the integer i and outputs the binary representation of three indices in $[b \cdot 2^n \cdot n^4]$ representing the three variables in the ith clause of ϕ_x. Call these representations z_1, z_2, z_3 that encode say the variables $y_{j_1}, y_{j_2}, y_{j_3}$. It also outputs three bits b_1, b_2, b_3 such that b_k is true if y_{j_k} is negated in the clause. For example, if the ith clause is $y_{j_1} \vee y_{j_2} \vee \overline{y}_{j_3}$ then $b_1 = 0, b_2 = 0, b_3 = 1$. Thus, E takes as input l bits and outputs $3l+3$ bits. The circuit D consists of the circuit E followed by feeding the $3l$ output bits of E representing z_1, z_2, z_3 each into a copy of C^x. Recall C^x takes l input bits and each z_i is l bits long. Define the wires $a_1 = C^x(z_1), a_2 = C^x(z_2), a_3 = C^x(z_3)$. Then D outputs $\neg\left[(a_1 \oplus b_1) \vee (a_2 \oplus b_2) \vee (a_3 \oplus b_3)\right]$. See figure 3 for an illustration of D.

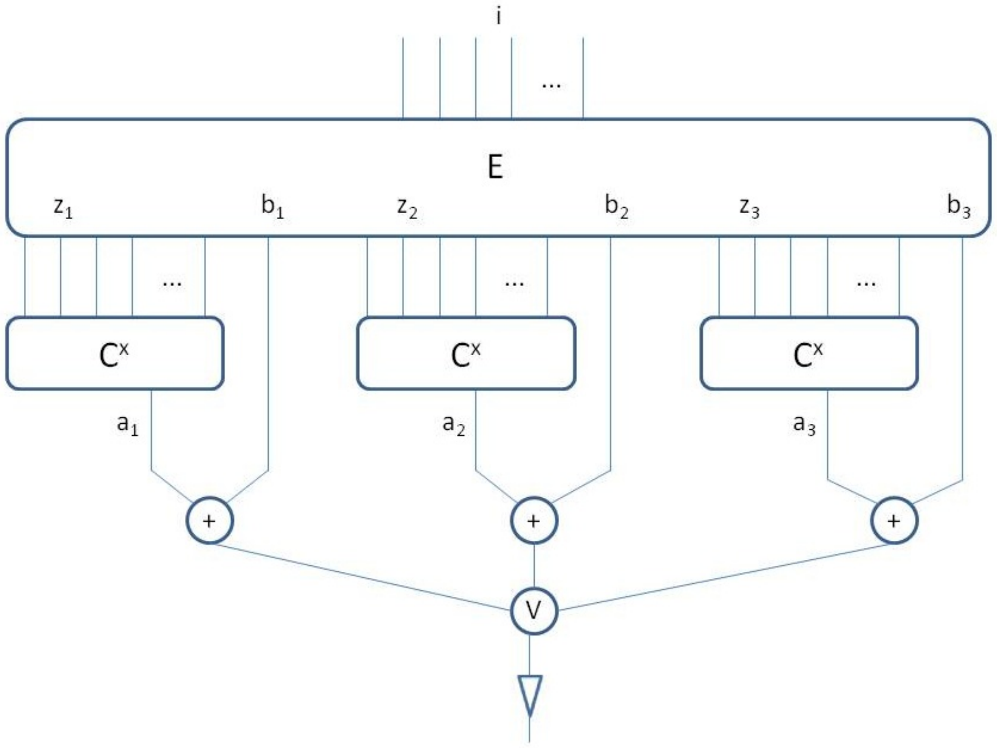

Figure 3: Circuit D constructed from E and 3 copies of C^x

In other words, for a length l binary string i representing an integer in $[b \cdot 2^n \cdot n^4]$, $D(i)$ outputs 1 if and only if the ith clause of ϕ_x is not satisfied by the values assigned to the variables in the clause by the nondeterministically guessed witness circuit C^x. In particular, if D is satisfiable then there exists a clause in ϕ_x that is not satisfied. The circuit D has size $O(n^8 + n^k)$ and l inputs, so the nondeterministic algorithm N uses the fast CircuitSAT algorithm to determine whether D is satisfiable in $O(2^l \cdot \frac{(n^8+n^k)^c}{s(n)})$ time. The total running time is then

$$\begin{aligned} O\left(\frac{2^n}{a(n)} \cdot 2^l \cdot \frac{(n^8+n^k)^c}{s(n)}\right) &= O\left(\frac{2^n}{a(n)} \cdot 2^n \cdot n^4 \cdot \frac{(n^8+n^k)^c}{s(n)}\right) \\ &\leq O\left(\frac{2^n}{a(n)} \cdot 2^n \cdot \frac{1}{a(n)}\right) \\ &\leq O(\frac{2^n}{a(n)}). \end{aligned}$$

Therefore, $L \in \mathtt{NTIME[2^n/a(n)]}$ implying that $\mathtt{NTIME[2^n/a(n)]} \subseteq \mathtt{NTIME[2^n]}$, but this contradicts the nondeterministic-time hierarchy theorem. □

Under the assumptions of the theorem $\mathtt{NTIME[2^n]}$ does not have n^k-sized witness circuits and this holds for every k. Therefore, $\mathtt{NTIME[2^n]}$ does not have polynomial size witness circuits. Then by the contrapositive of lemma 61 we have that $\mathtt{NEXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$. Therefore, we have proved our goal.

Theorem 65. *Suppose that there exists a superpolynomial function $s(n)$ such that* CircuitSAT *on circuits with n variables and n^k gates can be solved in $2^n \cdot poly(n^k)/s(n)$ time by a deterministic algorithm for all k. Then $\mathtt{NEXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$.*

Therefore, if there exists a slightly better CircuitSAT algorithm, then this method would get around relativization, natural proofs, and algebrization. It gets around the naturalization barrier by using diagonalization in the form of the nondeterministic-time hierarchy theorem. The hierarchy theorems use diagonalization to single out a specific function rather than reasoning about a whole set of functions with a property. Furthermore, Williams claims that faster algorithms would avoid relativization and algebrization because all known nontrivial SAT algorithms do not relativize or algebrize i.e. a faster SAT algorithm cannot be trivially adapted to solve SAT^A (CNF formulas with A predicates) when given the oracle A.

Consider the following. Let B be the oracle such that $\mathtt{P}^B \neq \mathtt{NP}^B$. Suppose we are trying to prove that $\mathtt{P} = \mathtt{NP}$ by coming up with a polynomial time algorithm for SAT and succeed by writing down a polynomial time algorithm G. Now suppose that for any oracle A that this same algorithm G with access to A can be blindly applied to solve SAT^A. Then in particular, G with access to B solves SAT^B proving that $\mathtt{P}^B = \mathtt{NP}^B$, a contradiction. Therefore, the algorithm G *must* break down when trying to apply G to solve SAT^B, and for any other oracle C for which $\mathtt{P}^C \neq \mathtt{NP}^C$ holds for that matter. The claim is that this is true for all known slight, yet interesting, improvements on SAT algorithms.

However, it is believed that $\mathtt{P} \neq \mathtt{NP}$, so one would hope that there actually is no genuinely polynomial time algorithm G for SAT that breaks down when applied to SAT^B for appropriate B. Yet, at the same time, we do want a slightly weaker algorithm G' that does break down for appropriate B's, so that we can prove results like $\mathtt{NEXP} \not\subseteq \mathtt{P}_{/\mathtt{poly}}$ that we should be able to prove. The algebrization barrier says that all of the above reasoning holds in the context of extension polynomials for oracles i.e. the potential polynomial time algorithm G when given access even to $\widetilde{B}$ would have to break down when applied to solving SAT^B.

Thus, we see that this result draws together many ideas of complexity theory, working on a fine duality between upper bounds and lower bounds. In particular, Williams [WIL11] used this technique to prove that $\mathtt{NEXP} \not\subseteq \mathtt{ACC}^0$ by designing a faster algorithm to solve $\mathsf{CircuitSAT}$ restricted to $\mathtt{ACC}^0$ circuits. The algorithm draws on multiple ideas in algorithm design, namely that of matrix multiplication and dynamic programming. This result backs up the previous claims that this technique avoids relativization and algebrization because there is a known oracle A such that $\mathtt{NEXP}^A \subseteq \mathtt{ACC}^{0A}$ and an oracle B and extension $\widetilde{B}$ such that $\mathtt{NEXP}^{\widetilde{B}} \subseteq \mathtt{ACC}^{0B}$. This is indeed a non-algebrizing result. Considering the natural proofs barrier however, there is not much evidence $\mathtt{ACC}^0$ has pseudorandom generators, so it may be that this result did not have to confront the natural proofs barrier at all. Thus, it cannot be used as concrete evidence that this technique does not naturalize, although the use of diagonalization is good evidence that it does not.

References

[AB] S. Arora and B. Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[AW] S. Aaronson, A. Wigderson. Algebrization: A New Barrier in Complexity Theory. *ACM Transactions on Theory of Computing* 1(1), 2009.

[Ajt] M. Ajtai. $\sum_1^1$ Formulas on finite structures. *Annals of Pure and Applied Logic*, 1983, vol. 24, pp 1-48.

[Ba1] L. Babai. E-mail and the unexpected power of interaction, in: *Proc. 5th Ann. IEEE Structures in Complexity Theory Conf.*, 1990, 30-44.

[Ba2] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing* (Providence, R.I., May 6-8). ACM, New York, 1985, pp. 421-429.

[BF] L. Babai, L. Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:1 (1991), 41-66.

[BFL] L. Babai, L. Fornow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3-40, 1991.

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

[BaM] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254-276, 1988.

[BGS] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=NP question. *SIAM Journal of Computing*, 4(4):431-442, 1975.

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

[BFT] H. Buhrman, L. Fortnow, T. Thierauf. Nonrelativizing separations. *IEEE Conference on Computational Complexity*, 8-12, 1998.

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

[DH] E. Danstin 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.

[For] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the EATCS*, 52:229-244, February 1994.

[FS] L. Fortnow, M. Sipser. Are there interactive protocols for coNP languages?, *Inf. Proc. Letters* 28 (1988), pp. 249-251.

[FSS] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *Math. Systems Thoery*, 17:13-27, 1984.

[GGM] O. Goldreich, S. Goldwasser, S. Micali. How to construct random functions. *Jour Assoc. for Computing Machinery* 33(4) (1986), 792-807.

[GMR] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs, *SIAM Journal of Computing* 18 (1989), 186-208.

[GS] S. Goldwasser, M. Sipser. Private coins versus public coins in interactive proof systems. In *Proc. 18th ACM STOC,* Berkeley CA 1986, pp. 59-68.

[Has1] J. Hastad. Almost optimal lower bounds for small depth circuits. *Proc. 18th ACM STOC*, 1986, 6-20.

[Has2] J. Hastad. *Computational Limitations of Small-Depth Circuits.* MIT Press, 1987. ACM Doctoral Dissertation Award Series (1986).

[IKW] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Sys. Sci.*, 65(4):672-694, 2002.

[LFKN] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systsms. *J. ACM*, 39:859-868, 1992.

[Lup] O. Lupanov. Implementing the Algebra of Logic Functions in Terms of Constant Depth Formulas in the Basis $\&, \wedge, \neg$. *Dokl. Ak. Nauk. SSSR* 136 (1961), pp 1041-1042 (in Russian). English translation in *Sov. Phys. Dokl*, 6 (1961), pp 107-108.

[Raz] A.A. Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\&, \oplus\}$. *Mathematicheskie Zametiki*, 41(4):598-607, 1987. English translation in *Math. Notes. Acad. Sci. USSR* 41(4):333-338, 1987.

[RR] A.A. Razborov and S. Rudich. Natural proofs. *J. Comput. Sys. Sci.*, 55(1):24-35, 1997.

[RW] S. Rudich, A. Wigderson (editors). Computational Complexity Theory. *IAS/Park City Mathematics Series No. 10.* AMS 2004.

[San] R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. In *Proc. ACM STOC*, pages 275-283, 2007.

[Sham] A. Shamir. IP=PSPACE. *J. ACM*, 39(4):869-877, 1992.

[Smo] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proc. ACM STOC*, pages 77-82, 1987.

[Tod] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865-877, 1991.

[Val] L. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.* 8 (1979), 189-201.

[Vin] N. V. Vinodchandran. A note on the circuit complexity of PP. ECCC TR04-056, 2004.

[WIL10] R. Williams. Improving exhaustive search implies super polynomial lower bounds. In *STOC*, 231-240, 2010.

[WIL11] R. Williams. Non-uniform ACC circuit lower bounds. In *IEEE Conference on Computational Complexity*, 2011.

[Yao1] A. Yao. Separating the polynomial-time hierarchy by oracles. In *FOCS*, pages 1-10. IEEE, 1985.

[Yao2] A. Yao. Theory and applications of trapdoor functions. In *Proceeding of the 23rd IEEE Symposium on Foundations of Computer Science,* IEEE, New York, 1982, 80-91.