A Study of the NEXP **vs.** P/poly **Problem and Its Variants**

by

Barış Aydınlıoğlu

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2017

Date of final oral examination: August 15, 2017

This dissertation is approved by the following members of the Final Oral Committee:

Eric Bach, Professor, Computer Sciences
Jin-Yi Cai, Professor, Computer Sciences
Shuchi Chawla, Associate Professor, Computer Sciences
Loris D'Antoni, Asssistant Professor, Computer Sciences
Joseph S. Miller, Professor, Mathematics

To Azadeh

ACKNOWLEDGMENTS

I am grateful to my advisor Eric Bach, for taking me on as his student, for being a constant source of inspiration and guidance, for his patience, time, and for our collaboration in [9].

I have a story to tell about that last one, the paper [9]. It was a late Monday night, 9:46 PM to be exact, when I e-mailed Eric this:

> `Subject: question`
> `Eric, I am attaching two lemmas. They seem simple enough. Do they seem plausible to you? Do you see a proof/counterexample?`

Five minutes past midnight, Eric responded,

> `Subject: one down, one to go.`
> `I think the first result is just linear algebra.`

and proceeded to give a proof from The Book. I was ecstatic, though only for fifteen minutes because then he sent a counterexample refuting the other lemma. But a third lemma, inspired by his counterexample, tied everything together. All within three hours. On a Monday midnight.

I only wish that I had asked to work with him sooner. (And maybe that he had taken a little longer to prove the lemma.)

My sincerest thanks to Jin-Yi Cai, for his support during a critical period in my studies. Although I have made some rather unfortunate choices early on, it was ultimately my good fortune to have (besides Eric) Jin-Yi there. I also thank him for his expertise and his guidance.

Thanks to Dan Gutfreund and Akinori Kawachi, for our collaboration in [10], and to my committee members Jin-Yi Cai, Shuchi Chawla, Loris D'Antoni, and last but not least, Joseph Miller from the math department, for their time.

CONTENTS

LIST OF FIGURES

ABSTRACT

A central question in computational complexity is NEXP $\stackrel{?}{\subset}$ P/poly, which roughly asks whether one can efficiently decide the satisfiability of a given Boolean formula of low time-bounded Kolmogorov complexity.

This thesis describes the author's work on and around the NEXP $\stackrel{?}{\subset}$ P/poly question. The contributions are two-fold: (i) three conditional results that derive conclusions akin to NEXP $\not\subset$ P/poly under certain plausible assumptions, and (ii) a metamathematical result explaining why current techniques have not so far been able to settle NEXP $\stackrel{?}{\subset}$ P/poly.

For the sake of a streamlined presentation, only a select subset of author's results are included in this thesis. For an extensive development the reader is referred to the original papers [9], [10], and [11].

1 INTRODUCTION

Given a Boolean circuit $C(x)$ of size s, how hard is it to find an input x at which C outputs 1, or at least to tell whether such an x exists?

The conjecture, widely held within the theoretical computer science community, is that no efficient program exists for either task. Here (and throughout) "efficient" means having a runtime bounded by a polynomial in s, for C of size s. In notation, CircuitSAT $\notin$ P, or equivalently, NP $\not\subset$ P since CircuitSAT is complete for the class NP.

In fact a stronger conjecture, also widely believed, is NP $\not\subset$ P/poly, meaning not only efficient programs fail at solving CircuitSAT, but also small circuits. Here "small" means, similar to "efficient" above, of size polynomial in the input size. This is a stronger conjecture because every efficient program can be realized by a family of small circuits — one circuit for each input size — by the fundamental Cook-Levin theorem [25, 44]. (The notation 'P/poly' is not to be interpreted as taking a quotient in any algebraic sense, or in any other sense for that matter.)

Of course, circuits solving problems about circuits might seem like the sort of thing only theoreticians would postulate a conjecture about, until one considers what might happen if the conjecture is *false*: if CircuitSAT can be solved by circuits of size s^{10}, say, then it is conceivable that a massive effort akin to the Manhattan project would find such a circuit, e.g., one that solves the problem for all C of some large enough size s, thereby attaining the ability to factor any number of size about $s/\log s$ bits, and bringing the demise of RSA cryptography.[1]

Compared to what is conjectured, however, little is currently known. Con-

[1]This is because factoring reduces to CircuitSAT with a logarithmic blowup in input size. More generally, *every* public key cryptosystem would perish under a device solving CircuitSAT.

sider the following variant of CircuitSAT. Instead of being given a circuit C directly, we are given a size-s circuit D that *describes* C, such as

$$D(i) := \text{ type of the } i\text{th gate in } C\text{, and the indices of the gates connected to it}$$

where i is in binary. The task now is at least as hard as the original because D can describe not only all circuits of size roughly s, but also some much larger — even exponentially large in s.[2]

Yet even this problem eludes classification. In notation, it is not even known whether NEXP $\not\subset$ P/poly, where NEXP is the class for which this new problem is complete.

* * *

Although it is the NP $\not\subset$ P conjecture that gets most of the publicity outside theoretical computer science, perhaps a more tantalizing one to the insiders is NEXP $\not\subset$ P/poly, because there are results that come "close" to proving it. For example:

Fact 1.1. NEXP $\not\subset$ P/poly *or* Σ_2EXP $\not\subset$ NP/poly.

To put the "or" part of Fact 1.1 in context, and because it will be relevant in the sequel, let us begin by recalling that NP, in the modern view, is the set of those decision problems with efficiently verifiable solutions. I.e., it is the set of functions f of the form $f(x) := \exists y \in \{0,1\}^{\ell(|x|)}\ V(x,y)$, for some polynomial ℓ and "verifier" predicate $V \in P$ checking that y is a valid solution to the problem

[2]For example, if ϕ is a circuit of size s, and if C is the circuit that computes, via brute-force, the XOR of the truth table of ϕ, i.e. if $C = \oplus_x \phi(x)$, then C can be exponentially larger than ϕ (depending on the number of inputs to ϕ), but it is easy to see that a circuit for describing C is essentially the same size as one for describing ϕ (which can be worked out as $O(s \log s)$).

instance x — e.g., if V interprets x as a circuit and evaluates it on y, then we get CircuitSAT.

So the joke "P = NP is easy to solve — N = 1" has some intelligence in it after all, since the 'N' in NP can be viewed as an operator acting on the class P: take every $V \in P$ and put a bounded $\exists$-quantifier up front; the resulting set of functions of the form $f(x) := \exists y \in \{0,1\}^{\ell}\ V(x,y)$ is NP. It is as though the 'N' operator takes every efficient predicate and confers upon it the power of an "existential prover": if a solution exists, then the prover presents it, which the predicate then checks. ('N' stands for *nondeterminism*.) The author would use '$\exists$' for this operator if he were emperor of notation, and say $\exists P$ for NP; similarly he would say $\forall P$ for coNP (recall this is the set of g of the form $g(x) := \neg f(x)$ for some $f \in$ NP). But there is a tradition to abide by — and that is to use $\Sigma_1 P$ and $\Pi_1 P$ for these classes, respectively, when they are viewed in this way.

But a couple of letters' difference is nothing compared to the main idea, which is to take all predicates comprising a given class $\mathcal{C}$ and prepend each with a quantifier to get a new class; if $\mathcal{C}$ is coNP for example, then $\Sigma_1\mathcal{C}$ becomes $\exists\forall P$, or as the tradition aptly calls it, $\Sigma_2 P$ (short for $\Sigma_1\Pi_1 P$); it is the set of all functions of the form $g(x) := \exists y \forall z V(x,y,z)$ for some $V \in P$. (A good example for g is the function CircuitMIN: given a circuit x, decide if there is a smaller circuit y that computes the same function as x over all inputs z.)

This idea naturally gives rise to a hierarchy, with P at the bottom level, NP and coNP at the first level, $\Sigma_2 P$ and $\text{co}\Sigma_2 P$ ($= \Pi_2 P$) at the second, and so on ad infinitum. This is called the *polynomial-time hierarchy*. If we do roughly the same thing, but starting from exponential-time algorithms instead of polynomial, then we get the so-called exponential-time hierarchy, with EXP at the bottom, then NEXP and coNEXP, then Σ_2EXP and Π_2EXP and so on.

So Fact 1.1 says either NEXP $\not\subset$ P/poly is true, or it is true "one-level up":

Fact 1.1 restated. $\Sigma_1\mathrm{EXP} \not\subset \Sigma_0\mathrm{P/poly}$ *or* $\Sigma_2\mathrm{EXP} \not\subset \Sigma_1\mathrm{P/poly}$.

* * *

Another way of coming "close" to NEXP $\not\subset$ P/poly is to show that plausible improvements over certain trivial algorithms imply circuit lower bounds.

A seminal example of this is a result by Impagliazzo, Kabanets, Wigderson [38] from early 00's. It involves the class BPP — the probabilistic extension of P — comprising functions of the form

$$f(x) = \text{Я}y \in \{0,1\}^{\ell(|x|)} V(x,y) \qquad (\dagger)$$

for some polynomial ℓ and some $V \in P$, where Я is the *probabilistic quantifier* [56]: $\text{Я}y\phi(y)$ is true (resp., false) if the set of y satisfying ϕ is at least twice (resp., at most half) the size of the set of y not satisfying ϕ; otherwise it is undefined.[3]

Obviously, a function of the form (†) above can be computed in exponential time, by just cycling over all y to find how many satisfy the inner predicate V and how many do not. What Impagliazzo et al. show is that even a slight improvement over this brute-force simulation yields circuit lower bounds.

Fact 1.2 ([38])**.** *If* BPP $\subset$ NP *then* NEXP $\not\subset$ P/poly*. In fact, the same conclusion holds even if* BPP $\subset$ NSUBEXP*.*

Here NSUBEXP is the subexponential analogue of NP; it is the set of functions of the form $f(x) = \exists y \in \{0,1\}^{\ell(|x|)} V(x,y)$ for some $V \in P$ and a *subexponential* $\ell \in 2^{n^{o(1)}}$. (Just like NEXP is the exponential analogue of NP.)

The first contribution of this thesis is several results in the form of Fact 1.2, showing circuit lower bounds assuming faster-than-trivial algorithms.

[3] The traditional definition of BPP requires f to be a *language*, i.e., to be defined on all x. Here and throughout, we relax that definition to include *partial* languages (cf. Section 3).

* * *

In preparation for stating our first contribution, let us consider the 'BP' in BPP as an operator acting on the class P, similar to the 'N' operator that yields NP. Again, if it were up to the author, then ЯP would be used for BPP, just as ∃P would be for NP, but tradition has made its pick.

Now, MA is the class ∃ЯP — the existential extension of BPP — comprising functions of the form

$$f(x) = \begin{cases} 1, & \text{if } \exists y \in \{0,1\}^{\ell(|x|)}\, Яz \in \{0,1\}^{\ell(|x|)} V(x,y,z) \\ 0, & \text{if } \forall y \in \{0,1\}^{\ell(|x|)}\, Яz \in \{0,1\}^{\ell(|x|)} \neg V(x,y,z) \end{cases} \tag{‡}$$

for some polynomial ℓ and some $V \in P$. It is as though we have an all-powerful prover, say *Merlin*, who presents a proof y that $f(x) = 1$, which gets verified by *Arthur*, who has access to random coin flips z. This, in fact, is where the name MA comes from: Merlin-Arthur games [14].

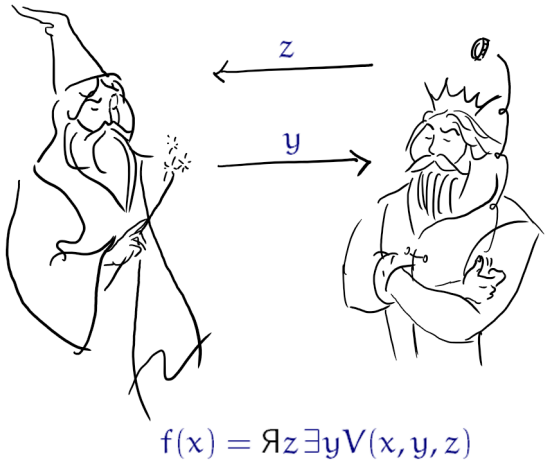

Figure 1.1: An Arthur-Merlin game

Similarly we have AM, the probabilistic extension of NP, i.e., the class ЯЭP where Arthur makes the first move and Merlin the second. A good example for a problem in AM is ApproxLB [35]: given a circuit C and a number k, output 1 (0, resp.) if there are k or more (resp., k/2 or fewer) inputs satisfying C.[4]

It is believed that the Я quantifier in general does not buy much power. In particular,

$$\text{ЯP} = \text{P} \subsetneq \exists\text{P} = \exists\text{ЯP} = \text{Я}\exists\text{P}$$

i.e.,

$$\text{BPP} = \text{P} \subsetneq \text{NP} = \text{MA} = \text{AM}$$

are all believed to be true.[5] Current knowledge is relatively little, however; we know how to express the Я quantifier in the second level of the polynomial-time hierarchy [43], and that is about it. We depict this in Figure 1.2, using dashes for BPP, dots for MA, pluses for AM. (All protrusions are believed to be empty.)

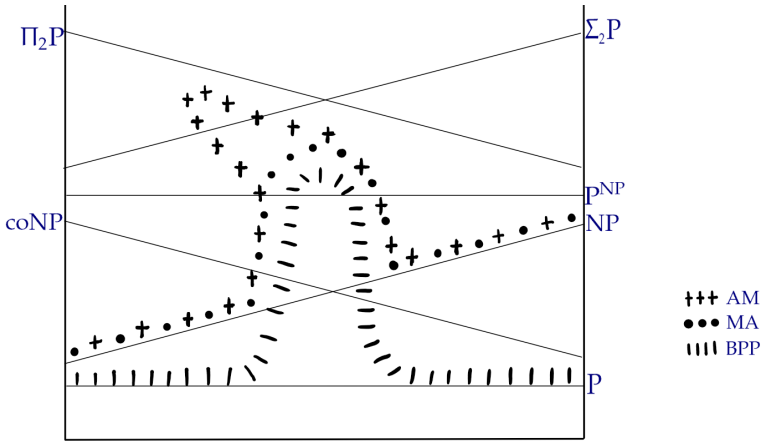

Figure 1.2: PH versus probabilistic classes.

[4] As in BPP, the traditional definition of MA, AM require f be defined on all x; we do not.

[5] By BPP $\subset$ P we mean every f $\in$ BPP can be extended to some L $\in$ P; similarly for $\mathcal{C} \subset \mathcal{D}$ in general when $\mathcal{C}$ is a class of partial languages and $\mathcal{D}$ of (total) languages.

We are ready to state the first contribution of this thesis:[6]

Theorem 1.3 (First Result)**.** *If* $\mathrm{AM} \subset \mathrm{P}^{\mathrm{NP}}$ *then* $\mathrm{EXP}^{\mathrm{NP}} \not\subset \mathrm{P}/\mathrm{subexp}$.

Theorem 1.4 (Second Result)**.** *If* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$ *then* $\mathrm{P}^{\Sigma_2\mathrm{P}} \not\subset \mathrm{NP}/n^k$ *for all constants* k.

Theorem 1.5 (Third Result)**.** *If* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$ *then* $\Sigma_2\mathrm{EXP} \not\subset \mathrm{NP}/\mathrm{poly}$.

In these theorems we think of circuits as "algorithms with advice" [40]. Such an algorithm has, for each n, an *advice* string a_n that it can consult when processing inputs of length n. For example, P/poly can be defined as the class of functions $f(x)$ for which there is a predicate $V \in \mathrm{P}$ and a family $\{a_n\}$ with $a_n \in \{0,1\}^{\ell(n)}$ such that

$$f(x) = V(x, a(|x|))$$

for every input $x \in \operatorname{dom} f$, where ℓ is some polynomial. Notice how '/poly' behaves as an operator on the class P, taking each predicate within and endowing it with polynomially bounded advice. More generally, $\mathcal{C}/\ell$ is obtained by taking every $V \in \mathcal{C}$ and giving it $\ell(n)$ bits of advice for each input length n.

So Theorems 1.3-1.5 show, assuming AM can be "derandomized" into lower levels of the polynomial-time hierarchy, that

- $\mathrm{EXP}^{\mathrm{NP}}$ contains a function of deterministic circuit complexity in $2^{n^{\Omega(1)}}$,
- $\mathrm{P}^{\Sigma_2\mathrm{P}}$ contains, for every k, a function of *non*deterministic circuit complexity in $\omega(n^k)$,
- $\Sigma_2\mathrm{EXP}$ contains a function of *non*deterministic circuit complexity in $n^{\omega(1)}$,

respectively. We compare these bounds to what is known unconditionally in Chapter 2.

[6] We can show stronger results in all three theorems, but we opt for readability here. (See Section 2.1.) Also, see footnote 5 about the notation $\mathrm{AM} \subset \mathcal{D}$ for a set $\mathcal{D}$ of languages.

* * *

Given that we can prove some variants of NEXP $\not\subset$ P/poly, just how "difficult" is it to prove the conjecture itself?

"Impossible," of course, is one possible answer since the conjecture may be false, or worse, it may be true but unprovable — thanks to Gödel who introduced this possibility to human thought. But supposing neither of these is the case, is there a scientific explanation for our failure to prove this conjecture thus far, one that sheds light on the evident limitations of our techniques?

The second contribution of this thesis concerns this question.

One way to approach this question is to find a system of axioms, powerful enough to derive the known results of complexity theory, and then to show that NEXP $\stackrel{?}{\subset}$ P/poly is independent from those axioms. This is the proper way perhaps, but also a tall order because of the balancing act required: it is one thing to find an interesting theory unable to settle some conjecture — and there is prior work of this type, in particular for the NP $\stackrel{?}{\subset}$ P question [26] — but it is another matter when "interesting" is taken to mean "able to derive known theorems". This type of work would fall in the area of *reverse mathematics* (e.g., [36]), and the author knows of no such prior work in complexity.

A second way is to take the attitude of a grossly fictional algebraist who, in his quest to settle a conjecture about fields of characteristic 7, notices that everything he has proven so far also holds for characteristic 3, and for 31 — in fact for any prime. The natural thing to do then would be, of course, to ask if the conjecture itself is insensitive to switching from 7 to any such number.

The second approach is in a sense the dual of the first: rather than looking for a "weaker" version of mathematics as in the first approach, the idea is to leave mathematics as is, and instead try to prove a "stronger" version of current theorems — parameterize them as it were — in such a way that NEXP $\stackrel{?}{\subset}$ P/poly

resolves contradictorily within the range of values for the parameter.

It is this second approach that has gained much traction in complexity theory, and that is also taken in this thesis.

* * *

In 1975, Bob Solovay, along with Theodore Baker and John Gill, published a paper [15] that turned out to influence complexity theory for two solid decades hence, and maybe even until today.

Consider broadening the definition of a program, from one that acts on bit-strings via basic logical operations, to one that also does so via an unknown Boolean function family $\mathcal{O}$ (for "oracle"). What Baker et al. found is that the resulting theory of computational complexity would be too "crude" to settle the $\mathrm{NP} \stackrel{?}{\subset} \mathrm{P}$ question. For it is not that question such a theory would attempt to resolve; it is the *relativized* version, denoted by $\mathrm{NP}^{\mathcal{O}} \stackrel{?}{\subset} \mathrm{P}^{\mathcal{O}}$. That attempt would be doomed to fail, however, because the answer depends on the choice of $\mathcal{O}$, as shown by Baker et al. Subsequent works have extended this finding to the $\mathrm{NEXP} \stackrel{?}{\subset} \mathrm{P/poly}$ question [54] (and to many other questions of complexity [31, 36, etc.]).

What makes this signicant, even today, is that when reasoning about efficient programs, the typical techniques one employs — simulation, diagonalization, and more generally techniques borrowed from computability theory — are also applicable to the above broader notion of a "program with oracle". In fact many fundamental results in complexity can be derived using solely such techniques, i.e., they *relativize*. Four decades after its introduction by Baker et al., "relativized complexity theory" remains a proxy framework for "almost all known proof techniques" in dealing with efficient programs.

Hence we go, into the shoes of the fictional algebraist of the previous section. In our quest to prove a conjecture about the class P — which can be used to define

NP and every class mentioned in this thesis — we find that almost everything we have proven so far also holds for the broader class $\mathrm{P}^{\mathcal{O}}$, relativized P. The $\mathrm{NEXP} \stackrel{?}{\subset} \mathrm{P/poly}$ question, we find however, is sensitive to switching P with $\mathrm{P}^{\mathcal{O}}$.

* * *

Suppose one day, suddenly our algebraist has a revelation. Floodgates creak, and results start trickling in — new, peculiar results. They are correct results, no doubt, and they do hold for characteristic 7, which is all he cares about; but unlike his past results, these do not hold for any prime — they do not "primize". Naturally he wonders: can this breakthrough last all the way to his longstanding conjecture?

Some years pass and our algebraist, now a bit weary, feels that progress has been slow since his big discovery. Surely there had been some great results since, including a weak version of his conjecture, but all of them seem to him as corollaries somehow, of that major development years earlier. Naturally he wonders: has he surmounted one barrier — the "primization" barrier — only to find himself at the fringes of another one?

This is pretty much what happened to complexity theorists since the early 90's when, in rapid succession, a seminal set of papers entered the scene [45, 48, 12, 28, 7]. Besides being interesting in their own right (one of them even got coverage in New York Times [41]) these results all had the added significance of *not* relativizing. Whatever it was that enabled these results, it was natural to ask whether the same ingredient can be used towards $\mathrm{NEXP} \not\subset \mathrm{P/poly}$; this question became more pertinent when something "close" actually got proved, almost as a corollary, again in the 90's:

Fact 1.6 ([19])**.** $\mathrm{MAEXP} \not\subset \mathrm{P/poly}$.

Here, MAEXP is the exponential version of MA; it sits between NEXP and

Σ_2EXP, but is conjectured to equal NEXP (just like MA versus NP and Σ_2P).[7] "We believe," the authors of Fact 1.6 wrote [19, p. 5], "our techniques give us a foot in the door that may open to many other exciting separations."

Alas, such hopes did not come to pass. It took a decade of no major progress until Scott Aaronson, together with Avi Wigderson, came up with an influential paper [2] that gave conceptual evidence for a new barrier, blocking the path to many conjectures of complexity, including NEXP $\not\subset$ P/poly.

Turning that conceptual evidence into a formal result is the second main contribution of this thesis.

Theorem 1.7 (Fourth Result)**.** *Relative to every affine oracle, Fact 1.6 holds, but not* NEXP $\not\subset$ P/poly*, nor does* NEXP $\subset$ P/poly*.*

It is as though our algebraist defines a subset of primes, say, all primes p such that $p \equiv 3 \pmod 4$. This set contains his favorite, 7, but many others as well, and for each member, all his post-breakthrough results hold. Yet his big conjecture, regarding characteristic 7, resolves contradictorily if 7 is switched with an arbitrary number in this set.

We shall see in due course what exactly an *affine oracle* is; it suffices to say in this introduction that the set of such oracles is almost as large as the set of all oracles. We shall also see that besides Fact 1.6, many other results (e.g., Theorems 1.3-1.5) hold relative to such oracles — they *relativize affinely*. If relativized complexity theory was a proxy framework for "known proof techniques" in dealing with efficient programs before the 90's, then affinely relativized complexity is the same for after. What Theorem 1.7 shows then, is that current techniques are doomed to fail at settling NEXP $\stackrel{?}{\subset}$ P/poly.

[7] More precisely, it is believed that every function in MAEXP can be extended to one in NEXP; see footnote 5 about the notation $\mathcal{C} \subset \mathcal{D}$ for $\mathcal{C}$ a set of partial languages and $\mathcal{D}$ a set of languages.

2.1 First, Second, and Third Result

In Theorems 1.3-1.4, if we want to obtain the same circuit lower bounds unconditionally, then we need to go further up in the exponential-/polynomial-time hierarchy. In particular, unconditionally,

- the lower bound of Theorem 1.3 holds "one level up", for $\mathrm{EXP}^{\Sigma_2\mathrm{P}}$ [39];
- the lower bound of Theorem 1.4 holds "half a level up", for $\Sigma_3\mathrm{P}$ [55];
- the lower bound of Theorem 1.5 holds "one level up", for $\Sigma_3\mathrm{EXP}$ [55].

If we take a closer look at the exponential-/polynomial-time hierarchy, then we can say more. The class $\mathrm{S}_2\mathrm{P}$ [47] is defined as the set of all languages L satisfying

$$\begin{aligned} L(x) = 1 &\implies \exists y\, \forall z\, V(x,y) \\ L(x) = 0 &\implies \exists z\, \forall y \neg V(x,y) \end{aligned}$$

for some $V \in \mathrm{P}$ and polynomial ℓ, where y, z are quantified over $\{0,1\}^{\ell(|x|)}$. It is easy to see that

$$\mathrm{P}^{\mathrm{NP}} \subset \mathrm{S}_2\mathrm{P} \subset \Sigma_2\mathrm{P} \cap \Pi_2\mathrm{P}$$

suggesting that $\mathrm{S}_2\mathrm{P}$ can be thought of as being a "quarter level up" from P^{NP}. Interestingly, however,

$$\mathrm{S}_2\mathrm{P} \subset \mathrm{BP}(\mathrm{P}^{\mathrm{NP}})$$

[21] so under the thesis that the BP operator (i.e., the Я quantifier) in general does not buy much power, $\mathrm{S}_2\mathrm{P}$ ought to equal P^{NP}.

Now, as we did with BPP, NP and others, we can view the 'S_2' in S_2P as an operator, which can be applied just as well to other classes. In particular, the class $S_2(P^{NP})$, located "one level up" from S_2P, satisfies

$$P^{\Sigma_2 P} \subset S_2(P^{NP}) \subset \Sigma_3 P \cap \Pi_3 P .$$

All of this is to say that, unconditionally, the lower bound of Theorem 1.4 holds for the class S_2P^{NP} [22], a class that might (but not known to) equal $P^{\Sigma_2 P}$. Similarly, the lower bound of Theorem 1.5 holds for $S_2(EXP^{NP})$ [22], the exponential analogue of $S_2(P^{NP})$.

Theorem 1.3 is published jointly with Gutfreund, Hitchcock, Kawachi [10]. Theorem 1.5 is published with Van Melkebeek [11], and Theorem 1.4 is implicit in the same paper. In fact stronger variants of all three results are derived in those papers.

2.2 Fourth Result

How difficult is it to resolve $NEXP \stackrel{?}{\subset} P/poly$? Directly or indirectly, this question has been the subject of many papers, but four among them stand out as having a close relationship to this thesis.

Recall that Baker-Gill-Solovay's framework of relativization (p. 9) already answered this question — "$NEXP \stackrel{?}{\subset} P/poly$ is beyond the reach of known techniques" — but this answer must be updated in light of the nonrelativizing results of the 90s (p. 10), which undoubtedly expanded the notion of "known techniques".

In the four papers we survey here, the overarching goal is (or so will be our view here) to propose some model for "known techniques", which involves

meeting two competing objectives: (a) derive all relevant theorems in the model, and (b) provably fail to derive in the model all relevant conjectures that are apparently beyond current reach.

We will use Figure 2.1 to roughly illustrate how each proposal fares with respect to these two objectives (a) and (b). The take-away message from this survey is that although parts of (a), (b) have been attained by prior work, ours is the first successful attempt that yields all the critical pieces under one framework.

Although the table is less precise than the discussion that follows, it does illustrate some key differences among prior work. The vertical gap in the table is a caricature of the current state of the art; to the left of the chasm are facts, and to the right are conjectures apparently out-of-reach. That gap would have been one column to the left had this been the 80s; as mentioned in Section 1, the 90s result that MAEXP $\not\subset$ P/poly (Fact 1.6) does not relativize, which bridged that chasm, or broke the relativization barrier — pick your metaphor.

We now survey each of the four proposals in turn.

1. AIV [6]: In a manuscript dating soon after the breakthrough nonrelativizing results of the early 90s mentioned in Section 1 (p. 10), Arora, Impagliazzo, and

Figure 2.1: Attempts at refining relativization

	examples for goal (a)			examples for goal (b)	
	$(\exists \mathcal{C}{:}\mathcal{C}\subset \text{NEXP} \wedge \mathcal{C}\not\subset \text{P/poly}) \Longrightarrow \text{NEXP}\not\subset\text{P/poly}$	$\Sigma_2\text{EXP}\not\subset\text{P/poly}$	$\text{MAEXP}\not\subset\text{P/poly}$	$\text{NEXP}\not\subset\text{P/poly}$	$\text{NP}\not\subset\text{P}$, $\text{EXP}\not\subset$i.o.-P/poly,..
AIV	✓	✓	✓	**?**	**?**
For	✓	✓	✓	**?**	✓
AW	**?**	✓	✓	✓	✓
IKK	✓	✓	✓	**?**	✓
this work	✓	✓	✓	✓	✓

Vazirani (AIV) [6] propose what they call "local checkability" as the key principle underlying those results.

The starting point of AIV is the classical idea, used by Cook to prove the NP-completeness of CircuitSAT, that a computation running in time t can be represented as a transcript of t rows, with each row corresponding to the state of the computation at one time step. Cook observed that given a table of t rows, we can verify that it is a valid transcript by inspecting all bits of the table in parallel, where each bit depends on only $O(\log t)$ bits elsewhere. As AIV observe, however, this property will not hold for computations with access to an arbitrary oracle $\mathcal{O}$: just consider the program that takes its input $x_1..x_n$ and outputs $\mathcal{O}(x_1..x_n)$ — the transcript of any execution of this program will have a bit that depends on n bits. This property is called *local checkability* by AIV.

We can interpret AIV's proposal as follows. Local checkability does not hold for an arbitary oracle $\mathcal{O}$, but it does if $\mathcal{O}$ itself can be computed by a locally checkable process. So the AIV framework roughly is this: take the Baker-Gill-Solovay framework of relativization, and then restrict the oracle $\mathcal{O}$, from an arbitrary function, to an arbitrary locally checkable function.

This framework derives many known nonrelativizing results, but as AIV point out, whether it can settle questions such as P versus NP or NEXP versus P/poly may be very hard to know. In fact, they observe that if P versus NP were shown beyond reach of their framework in the manner of Baker, Gill, Solovay — by giving contradictory relativizations, $NP^{\mathcal{O}} \subset P^{\mathcal{O}}$ and $NP^{\mathcal{O}} \not\subset P^{\mathcal{O}}$, using oracles satisfying local checkability — then P would actually be separated from NP. In this sense, the AIV framework is an unsatisfactory candidate for "known techniques". (Note that if all we want is a theory that can derive the current theorems, then we can just let the oracle $\mathcal{O}$ be empty.)

2. Fortnow [29]: In a response to the AIV proposal dated around the same time, Fortnow [29] argues that the nonrelativizing ingredient in the famous results of the 90s is not local checkability; rather, it is something of an algebraic nature.

We can interpret Fortnow's key insight as follows. The 90s result MAEXP $\not\subset$ P/poly (Fact 1.6) does not relativize, but it does, if every oracle $\mathcal{O}$ is constrained to have two properties:

(**i**). *Algebraic redundancy*. This means, roughly, that if we look at the truth table of $\mathcal{O}$ on inputs of length N, for any N, then we must see a table whose information content is significantly less than 2^N, in much the same way that if we look at the values of a function $f(x) = ax + b$ over an interval in $\mathbb{R}$, say, then we would see a list that can be condensed to merely two entries.

More specifically, $\mathcal{O}$ must encode a family of polynomials $G = \{G_n(x_1, .., x_n)\}_n$ that interpolate a family of Boolean functions $g = \{g_n(z_1, .., z_n)\}_n$ such that

$$G_n(x) = \sum_{z \in \{0,1\}^n} g_n(z)\Delta_z(x) \tag{2.1}$$

where $\Delta_z(x)$ denotes the monomial that is 1 if $x = z$, and 0 if $x \neq z$, for all Boolean x.

(**ii**). *Closure*. This roughly means that $\mathcal{O}$ is closed under adding redundancy. Just as $\mathcal{O}$ is an algebraically redundant version of a family g by property (i) above, there is an algebraically redundant version of $\mathcal{O}$ itself (after all $\mathcal{O}$ is a family just like g); the closure property dictates that the redundant version of $\mathcal{O}$ must essentially be $\mathcal{O}$ itself — more precisely, it must be efficiently computable given access to $\mathcal{O}$.

We will discuss the motivation behind these two properties later, in conjunction with a related paper (p. 19, IKK).

The upshot is that Fortnow takes, like AIV essentially do, the Baker-Gill-

Solovay framework of relativization, and then restricts the oracle $\mathcal{O}$ to satisfy some constraint; for lack of a better name we refer to this constraint as *closed algebraic redundancy.*

Like AIV, Fortnow does not show any formal limits of his framework. However, we can use the contributions of this thesis to show that several major conjectures of complexity can provably not be settled within it (hence the ✓ symbol in the table) — alas, we do not know how to show this for NEXP vs. P/poly. In this sense, Fortnow's framework is (in hindsight given by this thesis) a superior candidate for "known techniques" compared to AIV's, but still an unsatisfactory one in the context of NEXP vs. P/poly.[8]

3. AW [2]: A decade-and-half after the above two papers, Aaronson and Wigderson (AW) [2] come up with an influential paper that, for the first time after the breakthrough results of the 90s, sheds some light on whether "known techniques" — a notion that evidently has expanded during the 90s — can settle questions such as NEXP vs. P/poly.

We can interpret the key insight of AW as follows. In Fortnow's refinement of relativization described just above, recall that any oracle $\mathcal{O}$ must satisfy two properties that we collectively referred to as "closed algebraic redundancy". What AW found is that if we drop the closure requirement from this, then the resulting framework fails to settle many questions of complexity.[9]

This is a significant development because neither of the previous works, AIV & Fortnow, show any such limitation of the framework they propose. However, this progress by itself is not enough to yield a satisfactory framework for "known

[8]The NEXP vs. P/poly problem is representative of a host of other open problems whose provability is unknown in Fortnow's framework; see "This work" later in this survey (p. 20).

[9]Even the NEXP vs. P/poly question cannot be settled, AW found, if we go a step further and broaden the definition of algebraic redundancy, by admitting any low-degree polynomial that extends a Boolean function, and not just those of degree 1 (cf. (2.1)).

techniques", because such a framework must, as explained in the beginning of this survey, meet two objectives: (a) derive known theorems and (b) fail to settle conjectures. But all that is shown by this insight is that Fortnow's framework, which achieves goal (a), can be weakened to achieve goal (b) — albeit losing goal (a) in the process. (Notice that if all we want is a theory that cannot settle conjectures, then we can just take the empty theory.)

So what remains for AW is to figure out a way of doing what Fortnow did (attain goal (a) using two properties) by using only one of his properties, namely algebraic redundancy.

Unfortunately, AW do not succeed in this. As a compromise they come up with an ad hoc notion, called *algebraic relativization* — algebrization for short — that can only partially meet goal (a) of deriving known theorems, by taking an extremely limited view of "known theorems". For example, the statement

$$(\exists \mathcal{C} : \mathcal{C} \subset \text{NEXP} \ \wedge \ \mathcal{C} \not\subset \text{P/poly}) \implies \text{NEXP} \not\subset \text{P/poly} \tag{2.2}$$

is true no matter what NEXP or P/poly means — it is even true no matter what "is an element of" means — hence is relativizing, but it cannot be declared as algebraically relativizing in AW's framework. Consequently we have the rather questionable message: NEXP $\not\subset$ P/poly is not within reach of known techniques, but nothing stops us from coming up with a class $\mathcal{C}$, then showing $\mathcal{C} \subset$ NEXP with known techniques, and then showing $\mathcal{C} \not\subset$ P/poly with known techniques, thus concluding NEXP $\not\subset$ P/poly!

So the AW framework fails to provide a viable candidate for "known techniques". That said, its key ideas — how to meet goal (b) of showing unprovability results, using oracles with an algebraic property — influence all subsequent work, including ours.

4. IKK [37]: Motivated by the lack of basic closure properties in the AW framework — of which the above pathology (2.2) is just an example — Impagliazzo, Kabanets, and Kolokolova (IKK) [37] propose an alternative formulation soon after the AW paper.

We can view the approach of IKK as being along the same line as Fortnow's (hence also of AW's) by considering the following fact. Any Boolean formula φ can be extended to non-Boolean values, by viewing each conjunction as multiplication and each negation as subtraction from 1; the resulting expression — called the *arithmetization* of φ — is a low-degree polynomial that agrees with φ on the Boolean values, and that can be efficiently evaluated on all small values (here "low" and "small" means, as usual, polynomial in the size of φ).

Arithmetization of Boolean formulas appears to be the key technique in deriving results such as MAEXP $\not\subset$ P/poly (Fact 1.6); it is the one ingredient that clearly stands out in all proofs of nonrelativizing results from the 90s. Invariably, at some point in these proofs, some Boolean formula, used to model some efficient computation, gets arithmetized; notice this step does not seem to go through in the Baker-Gill-Solovay framework of relativization because such a formula φ would involve non-standard gates — oracle gates — yielding subformulas of the form $\mathcal{O}(\varphi_1, .., \varphi_m)$ for which there is no obvious way to proceed.

Now, in both Fortnow's framework and in IKK's, we can interpret the approach as being aimed at making this arithmetization step go through, for as large a class of oracles $\mathcal{O}$ as possible. In Fortnow's case this is achieved by constraining all oracles $\mathcal{O}$ to have "closed algebraic redundancy"; to see how this constraint helps, notice that in arithmetization, the act of replacing a conjunction $x \wedge y$ with a multiplication $x \cdot y$ is nothing other than the act of extending the Boolean function $(x, y) \mapsto x \wedge y$ to non-Boolean values via polynomial interpolation, in other words by adding algebraic redundancy (similarly for $\neg x$ versus

$1 - x$). Stated this way, arithmetization easily generalizes to Fortnow's oracles: simply replace each occurrence $\mathcal{O}$ in the formula with its algebraically redundant version, which does no harm because the class of $\mathcal{O}$'s under consideration is closed under adding algebraic redundancy. (Without closure, however, the resulting polynomial is not guaranteed to be efficiently computable, and this is where the AW framework runs into trouble.)

In the framework of IKK, on the other hand, the strategy to enable arithmetization is more direct (or more indirect, depending on the perspective): they allow $\mathcal{O}$ to be any oracle for which arithmetization, broadly construed, is possible. That is, $\mathcal{O}$ can be any family such that every Boolean formula, possibly with $\mathcal{O}$-gates besides the standard ones ($\wedge$, $\neg$, etc.), has a corresponding low-degree polynomial that extends it to non-Boolean values, and that can be efficiently evaluated given access to $\mathcal{O}$.

With this definition, IKK obtain a framework that, for the first time, meets both goal (a) of deriving known theorems, and (b) of failing to resolve conjectures — albeit not for NEXP $\stackrel{?}{\subset}$ P/poly.[10] In fact, the extent to which the IKK framework meets goal (b) is identical to what we said we can show for Fortnow's framework using the results of this thesis (the $\not\swarrow$ symbol in Figure 2.1). Thus the IKK framework is not satisfactory for our purposes either.

This work. In this thesis, we introduce *affine relativization*, the first framework that satisfactorily models "known techniques" in reasoning about efficient computation, *and* that is unable to resolve NEXP $\stackrel{?}{\subset}$ P/poly (see Figure 2.1).

Our contribution can be roughly viewed as achieving what AW aimed at but fell short of: take Fortnow's framework — relativization with oracles having

[10]The NEXP vs. P/poly problem is representative of a host of other open problems whose provability is unknown in IKK's framework; see "This work" below.

"closed algebraic redundancy" — and relax it somehow, so that it still meets goal (a) of deriving known theorems, yet it also meets goal (b) of failing to resolve conjectures.

Recall from earlier in this survey that AW did find a relaxation of Fortnow's framework that achieved goal (b), but lost goal (a) in the process — trading off one good thing with another, where both is needed. In order to fix this situation, the natural thing to try is to aim at a model between Fortnow's and AW's, in the hope of obtaining the best of both worlds.

This is what we essentially manage to do. Our model is simple to state given previous work: relativization with respect to oracles satisfying algebraic redundancy (closed or not). With this basic definition we succeed in (a) deriving nonrelativizing theorems such as MAEXP $\not\subset$ P/poly (Fact 1.6), and (b) showing that many conjectures are unresolvable, in particular NEXP $\stackrel{?}{\subset}$ P/poly.

Notice that the improvement we achieve is two-fold, regardless of whether we compare against Fortnow's work or AW's. Compared to Fortnow/AW, ours is a relaxation/tightening (respectively) that

(1) maintains the property already attained,

(2) attains the property missed,

by either work. This balancing act requires a number of ideas to pull off, and as far as we know, cannot be done by merely tweaking any of the frameworks surveyed above. Theorem 1.7 constitutes the strongest evidence yet for the difficulty of showing NEXP $\not\subset$ P/poly.

Theorem 1.7 is joint work with Eric Bach [9].

Remark: From a cursory inspection of Figure 2.1, it might seem as though NEXP $\not\subset$ P/poly is the only place where our framework has an edge over Fortnow's and IKK's — a nitpick of sorts. That is only the tip of the iceberg, however; NEXP $\not\subset$ P/poly is a representative of a host of other statements whose unprov-

ability can be shown in our framework but is not known for Fortnow's or IKK's — and in some cases even for AW's. See "Myths 3 & 4" below in Section 2.3.

2.3 Dispelling a few Myths

There are a few myths appearing occasionally — in blog posts, Q&A sites, lecture notes, etc. (e.g., [29, 23, 34, 42, 3, 46, 30]) — regarding the relativization notion and its extensions. Let us dispel the major ones here.

Myth 1. Dependence of relativization on the computer notion. By far the most widespread myth out there on relativization goes as follows. In order to define the complexity class P relative to an oracle $\mathcal{O}$, one must have defined P, in the first place, using some "computer" notion that extends naturally to a "computer with oracle". In short, as the myth goes, $\mathrm{P}^{\mathcal{O}}$ cannot be defined using P, i.e., there is no relativization operator that acts on classes.

However, it is easy to show otherwise. Consider the "oracle operator"

$$(V, \mathcal{O}, \ell) \mapsto V^{\mathcal{O}[\ell]}$$

which, given functions V and $\mathcal{O}$ on binary strings $\{0,1\}^*$, and given the function

ℓ on $\mathbb{N}$, outputs the function $V^{\mathcal{O}[\ell]}$ on binary strings, defined as

$$\begin{aligned} V^{\mathcal{O}[\ell]} : x \mapsto V(x, a), \quad & \text{where} \\ & a_1 = \mathcal{O}(V(x, \varepsilon)), \\ & a_2 = \mathcal{O}(V(x, a_1), \\ & \vdots \\ & a_i = \mathcal{O}(V(x, a_1..a_{i-1}), \\ & |a| = \ell(|x|). \end{aligned} \tag{§}$$

Now, the class FP, of efficiently computable functions (of which P is the subclass of functions with range $\{0,1\}$ instead of $\{0,1\}^*$) can be relativized simply as:

$$\mathrm{FP}^{\mathcal{O}} := \{\, V^{\mathcal{O}[n^c+c]} : V \in \mathrm{FP}, c \in \mathbb{N} \,\} \tag{¶}$$

Proposition 2.1. $\mathrm{FP}^{\mathcal{O}}$ *is exactly the set of all functions computable by a polynomial-time Turing machine with oracle access to* $\mathcal{O}$*.*

Proof. Use $\mathrm{FP}^{(\mathcal{O})}$ to denote the set for which we want to show $\mathrm{FP}^{\mathcal{O}}$ equals.

That $\mathrm{FP}^{\mathcal{O}} \subset \mathrm{FP}^{(\mathcal{O})}$ is easy. To compute a function of the form (§) with $\ell = n^c + c$ and $V \in \mathrm{FP}$, a Turing machine with access to $\mathcal{O}$ can construct $a_1..a_\ell$ bit by bit, and then output $V(x, a_1..a_\ell)$.

The converse $\mathrm{FP}^{\mathcal{O}} \supset \mathrm{FP}^{(\mathcal{O})}$ is easy as well. If M is a Turing machine with access to $\mathcal{O}$, running in time at most $|x|^c + c$ on every input x, then there is an equivalent machine M' that makes exactly $|x|^c + c$ queries to $\mathcal{O}$, by repeating if necessary the last query made by M; the running time of M' is at most $|x|^d + d$ steps for some d. Also, there is a machine Q that on input $(x, a_1..a_i)$ simulates $M'(x)$, by interpreting $a_1..a_i$ as the answers to the first i queries of M', and that

outputs the next query of M'. If M' halts during the simulation, then Q outputs whatever M' outputs. In case M' takes too long during simulation, longer than $|x|^d + d$ steps, which could happen if $a_1..a_i$ incorrectly lists the oracle answers, then Q outputs something arbitrary, say 0. Notice Q does *not* need access to $\mathcal{O}$. Therefore, if V is the function computed by Q, then $V \in \mathrm{FP}$. It follows by (§) that $V^{\mathcal{O}[n^c+c]}$ is identical to the function computed by M; by (¶), this function is in $\mathrm{FP}^{\mathcal{O}}$. □

Another way to dispel this myth, and a rather elegant one at that, is given by Arora, Impagliazzo, and Vazirani in their paper surveyed earlier (AIV, [6]). They use a variant of Cobham's definition for polynomial-time computation [24], with no reference to devices like Turing machines, to show that relativized P can be obtained by very basic constraints on what the eligible functions are, e.g., that they are closed under addition.

We caution that relativized versions of classes such as NP, P/poly, etc. must be understood as extensions of relativized P, just as in the unrelativized case. For example, $\mathrm{NP}^{\mathcal{O}}$ is the class of functions $f(x) = \exists y \in \{0,1\}^{\ell}\ V(x,y)$ for some $V \in \mathrm{P}^{\mathcal{O}}$, in other words, $(\mathrm{NP})^{\mathcal{O}}$ is *defined* as $\mathrm{N}(\mathrm{P}^{\mathcal{O}})$. This way, of defining classes in terms of P, helps us approach the subject in a disciplined way; it abstracts away from machines and avoids asking what it means to have oracle access under various computational models.

Myth 2. The naivetè of relativization. Often occurring in conjunction with the first myth, this one goes as follows. Relativization is a naive concept with a precarious foundation (oracle machines), sort of like the infinitesimals in 17th century calculus — intuitive but prone to error. It can be made rigorous however, the myth goes, via an independence result: by coming up with a set of axioms

$\mathcal{A}$ such that relativizing results are exactly those that follow from $\mathcal{A}$, hence such that contradictory-relativization results are those that are independent from $\mathcal{A}$.

Several things are wrong here. First and foremost, relativization is already a rigorous product of everyday mathematics. To illustrate, let us pick up the algebraist analogy again. Being an everyday mathematician, our fictional algebraist would likely be working under the ZFC axioms for set theory.[11] Then his universe $\mathfrak{U}$ would consist of sets, with $\in$ as the only relation on this universe.

To formalize his inquiry, our algebraist can take his big conjecture,

$$\text{for every field of characteristic 7, } \ldots \tag{ψ}$$

and restate it in more general terms as

$$\text{for every field of characteristic } p, \ldots \tag{ψ^p}$$

where p is a constant symbol that he adds to his universe $\mathfrak{U}$ and the "…" part is identical in both statements.

Now, the scenario that (ψ^p) holds for some primes p but not others — that (ψ) is beyond the "primization barrier" — can be expressed as

$$(\psi^p) \text{ is independent of ZFC} \cup \{\text{“}p \text{ is a prime”}\}$$

which would be a startling thing for our algebraist to discover, if he also finds that everything he has proven thus far on this topic — say theorems α,β,γ, of the form (ψ), with different contents for "…" — hold more generally as α^p, β^p,

[11] ZFC might not be an active part of his daily thoughts but at least there would be an awareness of it; for example, algebraists point out whenever they use the Axiom of Choice.

γ^p of the form (ψ^p), for every prime p, i.e., if

$$\alpha^p,\beta^p,\gamma^p \text{ are theorems of ZFC} \cup \{\text{“}p \text{ is a prime”}\}$$

or in short, if α,β,γ all “primize”.

Similar to our algebraist, we can introduce a constant symbol $\mathcal{O}$ and work under ZFC$\cup\{$“$\mathcal{O}$ is a language”$\}$. We can define $P^{\mathcal{O}}$ using Turing Machines with oracle access to $\mathcal{O}$ — an entirely rigorous notion — or by first defining P whatever way the reader prefers and then using the relativization operator mentioned earlier (p. 22, “Myth 1”) to get $P^{\mathcal{O}}$. Then

$$NEXP^{\mathcal{O}} \stackrel{?}{\subset} P^{\mathcal{O}}/\text{poly is independent of ZFC} \cup \{\text{“}\mathcal{O} \text{ is a language”}\} \qquad (\dagger)$$

would be a corollary of the fact that $NEXP^{\mathcal{O}} \subset P^{\mathcal{O}}/\text{poly}$ holds for some setting of $\mathcal{O}$ and $NEXP^{\mathcal{O}} \not\subset P^{\mathcal{O}}/\text{poly}$ holds for another.

To recap: relativization is a concept of everyday mathematics. A lack of confidence in relativization might well stem from a lack of confidence in the process by which everyday mathematics can be embedded in set theory, which is easily rectifiable by a good undergraduate book such as [27].

Another thing wrong about this myth is that an axiom set as stated is not known to exist. To be sure, we can set $\mathcal{A}$ to ZFC$\cup\{$“$\mathcal{O}$ is a language”$\}$, and call a theorem relativizing iff it is derivable from $\mathcal{A}$. But then it would be easy to misread a statement such as (†), as though it says something about the axiomatic complexity of the $NEXP \stackrel{?}{\subset} P/\text{poly}$ question — it does not: we certainly believe that ZFC, a sub-theory of $\mathcal{A}$, proves either $NEXP \subset P/\text{poly}$ or its negation!

The confusion arises because there are two ways of expressing the same question using the axiom set $\mathcal{A}$: $NEXP^{\mathcal{O}} \stackrel{?}{\subset} P^{\mathcal{O}}/\text{poly}$ and $NEXP \stackrel{?}{\subset} P/\text{poly}$. This is inevitable when $\mathcal{A}$ is an extension of everyday mathematics instead of a re-

striction, as one can always ignore any additional axioms and stick to everyday mathematics. For the myth to be accurate — for a genuine independence result — $\mathcal{A}$ *must be a* subset *of the axioms that govern the mathematical universe*. But such $\mathcal{A}$ is not known to exist.

If they are not independence results, then what are they, these current results on relativization? They are *crudeness* results, if we must call them something: they show that our proofs do not exploit any difference between P and an arbitrarily relativized version of P, and that such differences are key to making progress in the major questions of the field.

Myths 3 & 4. Algebraic oracles are messy; affine oracles are no different. One of the objections to investigating restricted oracles is that they are much harder to construct than unrestricted ones. It has taken a significant effort, spanning many years, to create a large library of unrestricted oracles [16, 17, 20, 18, 1, etc.], each one giving evidence that some conjecture is out of reach of "known proof techniques". On one hand, as the objection goes, these intricate oracle constructions do not seem to carry through for algebraic oracles; on the other hand, none of the conjectures addressed by these oracles have been settled since the breakthrough results of the 90s. Thus it seems a safe thesis that classical relativization is still a proxy for "known techniques".

From a cursory read of the above survey (Section 2.2), it might seem that affine oracles are no different. Indeed, as far as Figure 2.1 shows, NEXP $\not\subset$ P/poly seems to be the only place where our framework has an edge over Fortnow's and IKK's — a nitpick of sorts.

That is only the tip of the iceberg, however. NEXP $\not\subset$ P/poly is a poster child for a host of other statements whose unprovability can be shown in our framework, but is not known for Fortnow's or IKK's — and sometimes even for

AW's. The machinery we develop in Chapters 7 and 8 is powerful enough to accommodate traditional approaches to constructing oracles, with minor tweaks.

For example, our construction of an affine oracle relative to which NEXP $\subset$ P/poly (Theorem 1.7), is essentially a rephrasing of a construction by Heller [36], who gave an unrestricted oracle for the same statement. As another example, in Chapter 8 we take a fairly well-known construction, due to Beigel, Buhrman, and Fortnow [16], of an oracle relative to which $P = \oplus P \subsetneq NP = EXP$, and use it almost verbatim to give an affine oracle for the same statement. While the former construction is also done in AW's framework (in fact ours closely follow theirs), the latter does not seem doable in AW's framework, or in any other framework surveyed in Section 2.2 for that matter.

Of course, not every traditional oracle construction can yield an affine one, otherwise there would be no point in restricting oracles. But our results suggest that it is worth revisiting the classical constructions to explore which ones can be made affine. Perhaps those oracles that do not carry over correspond to conjectures whose time has come in the 90s.

Myth 5. Recent circuit lower bounds obviate relativization-based barriers. In 2011, Williams proved NEXP $\not\subset$ ACC_0 [53], a significant result both in its statement and its proof. Here ACC corresponds to "shallow circuits that can count": $AC_d[m]$ is the class of functions computable by polynomial-size $O(\log^d n)$-depth circuits that have, besides the standard gates $\wedge$, $\vee$, $\neg$, also MOD_m gates. Such gates output 1 iff ℓ of their inputs are set to 1 for some $\ell \equiv 0 \pmod m$. Finally, $ACC_d := \cup_{m \in \mathbb{N}} AC_d[m]$. (All gates have unbounded fan-in.)

Now the myth is this: as mentioned in the survey earlier (Section 2.2), a major result of ours, and of Aaronson-Wigderson for their framework, is that NEXP $\not\subset$ P/poly does not hold for all affine (or in case of AW, algebraic) oracles,

i.e., $\mathrm{NEXP}^{\mathcal{O}} \subset \mathrm{P}^{\mathcal{O}}/\mathrm{poly}$ for some eligible $\mathcal{O}$. In fact, something stronger is implied in either work: $\mathrm{NEXP}^{\mathcal{O}} \subset \mathrm{ACC}_0^{\mathcal{O}}$ for the same $\mathcal{O}$. Hence, as the myth goes, Williams' result shatters the affine/algebraic relativization barriers, just as the 90's results such as $\mathrm{MAEXP} \not\subset \mathrm{P/poly}$ did to the relativization barrier.

Several things are wrong here. First, it follows by the logic of this myth that these barriers were already shattered in the 80's, long before they were even conceived of (and long before Williams' result). This is because in both our work and AW's, something even stronger than $\mathrm{NEXP}^{\mathcal{O}} \subset \mathrm{ACC}_0^{\mathcal{O}}$ is implied, namely $\mathrm{NEXP}^{\mathcal{O}} \subset \mathrm{AC}_0^{\mathcal{O}}$, and because Furst, Saxe, Sipser [33] and Ajtai [4] already showed in the 80's that $\mathrm{P} \not\subset \mathrm{AC}_0$. So by the logic of this myth, even the original relativization barrier was shattered in the 80's and not the 90's!

Maybe it did but nobody noticed — we have to accept this if we consider Williams' result as nonrelativizing affinely/algebraically.

Even if we do, not all is lost. Be it in $\mathrm{P} \not\subset \mathrm{AC}_0$, or in $\mathrm{NEXP} \not\subset \mathrm{ACC}_0$, or in any other result on bounded-depth circuits, the techniques used clearly break down when the depth restriction is lifted. In this sense, the very nature of bounded depth constitutes a barrier.

Another defect of this myth is its ignorance of the following fact. In Williams' result, $\mathrm{NEXP} \not\subset \mathrm{ACC}_0$, the key ingredient is a better-than-brute-force algorithm for deciding $\mathrm{ACC}_0\mathrm{SAT}$, the restriction of CircuitSAT to ACC_0-circuits. For Williams' strategy to scale to $\mathrm{NEXP} \not\subset \mathrm{P/poly}$, one must find such an algorithm for the unrestricted CircuitSAT problem. But as we show in Section 8.3, such a result must not affinely relativize. So Williams' program would succeed in proving $\mathrm{NEXP} \not\subset \mathrm{P/poly}$, if it can find an ingredient that does not affinely relativize, which is to say that it will pass the barrier if it can pass the barrier.

3 PRELIMINARIES & NOTATION

We use $A \subset B$ to mean A is a subset of B; we never use '$\subseteq$'. By $\text{poly}(n)$ we mean the set of polynomials $\{n^d + d : d \in \mathbb{N}\}$. By $n^{\omega(1)}$ we mean the class of superpolynomial functions $\{(f : \mathbb{N} \to \mathbb{N}) : \forall d \in \mathbb{N}\ \forall^\infty n\ f(n) > n^d\}$. By $\operatorname{dom} f$ we mean the domain of f.

Basic complexity classes. FP is the set of all $f : \{0,1\}^* \to \{0,1\}^*$ that are efficiently computable. We do not rely on a particular implementation of efficient computability; for concreteness the reader can take the standard definition based on random access Turing machines. We rely on FP being enumerable.

A *language* is a function from $\{0,1\}^*$ to $\{0,1\}$. A *partial language* is a function that can be extended to (or already is) a language. We confuse $\{0,1\}$ with $\{\text{False}, \text{True}\}$.

In a slight abuse of notation, given sets $\mathcal{C}, \mathcal{D}$ of partial languages, we write $\mathcal{C} \subset \mathcal{D}$ to mean that every element of $\mathcal{C}$ can be extended to (or already is) an element of $\mathcal{D}$.

P is obtained by taking each function in FP and projecting its output to its first coordinate.

NP is the set of languages in $\exists \cdot P$, where $\exists \cdot \mathcal{C}$ denotes, for a set $\mathcal{C}$ of partial languages, the set of all partial languages L such that

$$\begin{aligned} L(x) = 1 &\implies \exists y \in \{0,1\}^{\ell(|x|)} : (x,y) \in \operatorname{dom} V \text{ and } V(x,y) \\ L(x) = 0 &\implies \forall y \in \{0,1\}^{\ell(|x|)} : (x,y) \in \operatorname{dom} V \text{ and } \neg V(x,y) \end{aligned}$$

for some $\ell \in \text{poly}(n)$ and $V \in \mathcal{C}$.

$\mathrm{co}\cdot\mathcal{C}$ denotes, for a set $\mathcal{C}$ of partial languages, the set of all partial languages of the form $L(x) = \neg M(x)$ for some $M \in \mathcal{C}$. It is customary to write $\mathrm{co}\mathcal{C}$ for $\mathrm{co}\cdot\mathcal{C}$.

$\forall\cdot\mathcal{C}$ denotes $\mathrm{co}\cdot\exists\cdot\mathcal{C}$. In particular, $\mathrm{coNP} = \forall\cdot P$.

Define $\Sigma_0 P = \Pi_0 P = P$, and inductively define $\Sigma_k P = \exists\cdot\Pi_{k-1}P$ and $\Pi_k P = \forall\cdot\Sigma_{k-1}P$. The set $\bigcup_{k\in\mathbb{N}}\Sigma_k P$ is called the *polynomial-time hierarchy*. Note that $\mathrm{NP} = \Sigma_1 P$ and $\mathrm{coNP} = \Pi_1 P$.

PSPACE, or $\Sigma_\infty P$, is the set of languages of the form

$$L(x) = \forall y_1 \exists z_1 \cdots \forall y_{t(|x|)} \exists z_{t(|x|)} V(x, y, z)$$

for some $V \in P$ and $t(n) \in \mathrm{poly}(n)$, where y_i, z_i are quantified over $\{0,1\}$. (We could quantify y_i, z_i over $\{0,1\}^{\ell(|x|)}$ for some $\ell \in \mathrm{poly}(n)$; the definition would be equivalent to the one given.) The justification for this definition of PSPACE comes from the well-known result of Stockmeyer and Meyer [50, Theorem 4.3] that functions computable by a polynomial-space Turing machine are contained in $\Sigma_\infty P$ (the reverse containment is clear).

We define BPP as $Я\cdot P$, where $Я\cdot\mathcal{C}$ denotes, for a set $\mathcal{C}$ of partial languages, the set of all partial languages L such that

$$\begin{aligned} L(x) = 1 &\implies \Pr_{y\in\{0,1\}^{\ell(|x|)}}[(x,y) \in \mathrm{dom}\, V \text{ and } V(x,y)] > 2/3 \\ L(x) = 0 &\implies \Pr_{y\in\{0,1\}^{\ell(|x|)}}[(x,y) \in \mathrm{dom}\, V \text{ and } \neg V(x,y)] > 2/3 \end{aligned}$$

for some $\ell \in \mathrm{poly}(n)$ and $V \in \mathcal{C}$. Traditionally BPP would be defined as the set of *languages* in $Я\cdot P$, and $Я\cdot P$ would be called prBPP, but we will not need to refer explicitly to the languages in $Я\cdot P$.

Interactive proofs. Let $Ax\varphi(x)$ denote $\mathbf{E}_x[\varphi(x)]$, the expected value of $\varphi(x)$, and let $Mx\varphi(x)$ denote $\max_x \varphi(x)$.

AM is the set of all partial languages L such that

$$\begin{aligned} L(x) = 1 &\implies \mathsf{A}y\,\mathsf{M}z\; V(x,y,z) > 2/3 \\ L(x) = 0 &\implies \mathsf{A}y\,\mathsf{M}z\; V(x,y,z) < 1/3 \end{aligned}$$

and MA is the set of all partial languages L such that

$$\begin{aligned} L(x) = 1 &\implies \mathsf{M}z\,\mathsf{A}y\; V(x,y,z) > 2/3 \\ L(x) = 0 &\implies \mathsf{M}z\,\mathsf{A}y\; V(x,y,z) < 1/3 \end{aligned}$$

for some $V \in P$ and $\ell \in \mathrm{poly}(n)$, where y, z are quantified over $\{0,1\}^{\ell(|x|)}$.

Notice that AM is the set $\mathsf{R} \cdot \exists \cdot P$ and that MA is the set $\exists \cdot \mathsf{R} \cdot P$. As in the case with BPP, traditionally AM would be defined as the set of *languages* in $\mathsf{R} \cdot \exists \cdot P$, and MA as those in $\exists \cdot \mathsf{R} \cdot P$, but we will not need to refer explicitly to the languages in either set.

IP is the set of languages L such that

$$\begin{aligned} L(x) = 1 &\implies \mathsf{A}y_1\,\mathsf{M}z_1 \cdots \mathsf{A}y_{t(|x|)}\,\mathsf{M}z_{t(|x|)} V(x,y,z) > 2/3 \\ L(x) = 0 &\implies \mathsf{A}y_1\,\mathsf{M}z_1 \cdots \mathsf{A}y_{t(|x|)}\,\mathsf{M}z_{t(|x|)} V(x,y,z) < 1/3 \end{aligned}$$

for some $V \in P$ and $t \in \mathrm{poly}(n)$, where y_i, z_i are quantified over $\{0,1\}$. (We could quantify y_i, z_i over $\{0,1\}^{\ell(|x|)}$ for some $\ell \in \mathrm{poly}(n)$; the definition would be equivalent to the one given.)

The A-quantifier in these definitions can be thought of as providing the coin tosses of a probabilistic verifier *Arthur*, who interacts with an all-powerful prover *Merlin* corresponding to the M-quantifier. Merlin's goal is to make Arthur accept, which Arthur does iff the "verdict" predicate V, given the input x and transcript (y, z) of the interaction, returns 1. The criteria by which V returns 0 or 1 is

typically described as a *protocol* between Merlin and Arthur. The quantity $t(|x|)$ is referred to as the *number of rounds* taken by — or the *round complexity* of — the protocol in computing inputs of length $|x|$.

Power of the Honest Prover. Consider the following subclass of IP. It contains languages L such that whenever $L(x) = 1$, Merlin can just compute a language $\Pi \in \mathcal{C}$ instead of using the M-quantifier. That is, there is a language $\Pi \in \mathcal{C}$ such that for all x, if $L(x) = 1$, then

$$\mathbf{Pr}_z[V(x,y,z)] = A y_1\, A y_2 \cdots A y_{t(|x|)} V(x,y,z) > 2/3, \quad \text{where} \qquad (\P)$$
$$z_1 = \Pi(x, y_1)$$
$$z_2 = \Pi(x, y_1 y_2)$$
$$\vdots$$
$$z_t = \Pi(x, y_1..y_{t(|x|)}),$$

and the case for $L(x) = 0$ remains as before. Any L in this class is said to have interactive proofs where *the power of the honest prover* is in $\mathcal{C}$.

Checkable. We call a language L *checkable* if it has an interactive protocol where the power of the honest prover reduces to L itself. I.e., in (¶), Π reduces to L via a Karp reduction (as defined below).

Perfect completeness. Replacing the condition "$> 2/3$" in the above definitions of interactive proofs, with the condition "$= 1$" yields equivalent definitions [32].

Reductions. As in the previous section (Interactive Proofs), let $Ax\varphi(x)$ denote $\mathbf{E}_x[\varphi(x)]$, and let $Mx\varphi(x)$ denote $\max_x \varphi(x)$.

Let F and G be functions into $\{0,1\}^*$ such that $\operatorname{dom} F, \operatorname{dom} G \subset \{0,1\}^*$. We

write

$$F \to G$$

and say that F *reduces to* G *via an interactive protocol*, iff there exists $R \in \mathrm{FP}$, $t \in \mathrm{poly}(n)$, and $\varepsilon \in 1/n^{\omega(1)}$, such that for every $x \in \mathrm{dom}\, F$:

$$\mathsf{A}y_1\, \mathsf{M}z_1 \cdots \mathsf{A}y_{t(n)}\, \mathsf{M}z_{t(n)} [\, F(x) = G(R(x,y,z)) \ \vee \ F(x) = R(x,y,z) \,] \geqslant 1 - \varepsilon(n)$$
$$\mathsf{A}y_1\, \mathsf{M}z_1 \cdots \mathsf{A}y_{t(n)}\, \mathsf{M}z_{t(n)} [\, F(x) \neq G(R(x,y,z)) \ \wedge \ R(x,y,z) \neq \text{`fail'} \,] \leqslant \varepsilon(n)$$

where $n = |x|$, and y_i, z_i are quantified over $\{0,1\}$. (Notice that $v = G(u)$ implies $u \in \mathrm{dom}\, G$.)

We call R an *interactive reduction* from F to G with *round complexity* $t(n)$. We caution that the word "reduction" refers to a function in FP, not to the notion that some F reduces to some G.

Intuitively, as in the previous section (Interactive Proofs), the A-quantifiers in this definition can thought of as Arthur and the M-quantifiers as Merlin. Given x, after sending random coin tosses y_i to Merlin and receiving responses z_i, Arthur uses the predicate R to obtain a string r. Arthur wants either r or $G(r)$ to equal $F(x)$. Merlin can, with high probability over Arthur's coin tosses, ensure that Arthur obtains a desired r. If Merlin is devious, then he has negligible chance in making Arthur obtain a string $r \neq$ 'fail' that is not desired.

We believe this definition to be new. There are three special cases of R being an interactive reduction that capture some classical definitions:

- in a *randomized reduction*, we have $R(x,y,z) = R(x,y)$. Intuitively, Arthur does not need to interact with Merlin to do the reduction.
- in a *Karp reduction*, we have $R(x,y,z) = R(x)$. Notice that $\varepsilon(n) = 0$ in this case. Intuitively, Arthur does not need Merlin's help to do the reduction, nor does he need to flip any coins.

- in a *Cook reduction*, we have $R(x, y, z) = R(x, z)$. Further, for every extension of G to a function G' on $\{0,1\}^*$, and for every z satisfying

$$z_i = G'(R(x, z_1..z_{i-1}))$$

  we have $F(x) = R(x, z)$.
  Notice that $\varepsilon(n) = 0$ in this case. Intuitively, Arthur does not need to flip any coins to do the reduction, and the power of the honest prover is G itself.

We call R a *strong Cook reduction* from F to G, if R is a Cook reduction from F to G, and if $R(x, z) \in \operatorname{dom} G$ for every $x \in \operatorname{dom} F$ and every z satisfying $z_i = G(R(x, z_1..z_{i-1}))$. (This would be the case, for example, when G is a language.) Intuitively, while interacting with Arthur to convince him of the value of $F(x)$, the honest prover never gets asked a question outside $\operatorname{dom} G$.

By default, all Cook reductions are strong. By default, all reductions are Karp.

The "reduces to via an interactive reduction" relation is transitive: $F \to G$ together with $G \to H$ imply $F \to H$. Further, "reduces to via a Karp reduction", "reduces to via a randomized reduction", "reduces to via a *strong* Cook reduction", are all transitive relations.

General time classes. Let $T \subset n^{\omega(1)}$ be a set of functions such that each element is computable in FP. Suppose that T is closed under taking polynomials in the following sense: for every $t \in T$ and $d \in \mathbb{N}$, there is some $t' \in T$ such that $t^d(n) < t'(n)$ for every n.

Define DTIME(T) as the set of languages L for which there exists $K \in P$ and $t \in T$ such that $L(x) = K(x, 1^{t(|x|)})$ for every x.

Define NTIME(T), Σ_2TIME(T), MATIME(T), etc., in the same way, except by picking K respectively from NP, Σ_2P, MA, etc.

Use E, NE, Σ_2E, MAE, etc., to denote respectively DTIME(linexp(n)), NTIME(linexp(n)), Σ_2TIME(linexp(n)), MATIME(linexp(n)), etc., where linexp(n) is the set $\{2^{cn} : c \in \mathbb{N}\}$.

Use EXP, NEXP, Σ_2EXP, MAEXP, etc., to denote respectively DTIME(exp(n)), NTIME(exp(n)), Σ_2TIME(exp(n)), MATIME(exp(n)), etc., where exp(n) is the set $\{2^{cn^d} : c, d \in \mathbb{N}\}$.

Circuits, Formulas, Advice. A *circuit over the basis* B is a directed acyclic graph where each internal node — nodes that can be reached from and can reach to other nodes — are labeled by a reference to some element in B. The number of incoming (and respectively, outgoing) neighbors of a node is called the *fan-in* (respectively, *fan-out*) of that node. Nodes of fan-in zero are the *input nodes* of the circuit, and of fan-out zero are the *output nodes*.

By default, circuits are over the *standard Boolean basis* $B_{\text{std}} := \{\mathbf{0}, \mathbf{1}, \wedge, \oplus\}$, where $\mathbf{0}$ is the all-zeroes language and $\mathbf{1}$ is the all-ones, $\wedge$ maps x to $\wedge_i x_i$ and $\oplus$ maps x to $\oplus_i x_i$.

A *formula* is a circuit where the fan-out of each node is at most 1. There are no output nodes in a formula, because a formula with two output nodes is really two different formulas.

The size of a circuit is the number of its edges. We say that the partial language F *has circuits of size* $s(n)$, if there is a family $\{C_n\}_{n\in\mathbb{N}}$ of circuits such that C_n has n input nodes, is of size $s(n)$, and for every $x \in \operatorname{dom} F$, $F(x) = C_{|x|}(x)$.

For $\ell : \mathbb{N} \to \mathbb{N}$ and $\mathcal{C}$ a set of languages, $\mathcal{C}/\ell$ denotes the set of languages L for which there is a language $V \in \mathcal{C}$ and a family $\{a_n\}_{n\in\mathbb{N}}$ with $a_n \in \{0,1\}^{\ell(n)}$ such

that

$$L(x) = V(x, a(|x|)).$$

P/poly denotes $\cup_{c\in\mathbb{N}}$ P/n^c. P/subexp denotes $\cap_{c\in\mathbb{N}}$ P/$2^{n^{1/c}}$.

Fact 3.1 ([40])**.** P/poly *is exactly the set of languages that have polynomial-size circuits.*

Relativized classes. For every language $\mathcal{O}$, we define the class $FP^{\mathcal{O}}$ — "FP *relative to* $\mathcal{O}$," or "FP *with oracle access to* $\mathcal{O}$" — as the set of all functions from $\{0,1\}^*$ to $\{0,1\}^*$ that Cook-reduce to $\mathcal{O}$.

All definitions built on FP above naturally generalize to their *relativized* versions: NP to $NP^{\mathcal{O}}$, IP to $IP^{\mathcal{O}}$, MAEXP to $MAEXP^{\mathcal{O}}$, etc. When we say "L is checkable with oracle access to $\mathcal{O}$", for example, we mean to replace FP with $FP^{\mathcal{O}}$ in the definition for a language to be checkable, and then declare L as checkable.

The languages $\oplus$SAT and Σ_kSAT. Define $\oplus$SAT as the language mapping

$$\phi(x) \mapsto \oplus_{\alpha\in\{0,1\}^n} \phi(\alpha)$$

where ϕ is a formula with n inputs $x_1..x_n$.

Define Σ_kSAT as the language mapping

$$\phi(X_1,..,X_k) \mapsto \exists\alpha_1 \in \{0,1\}^{n_1}\, \forall\alpha_2 \in \{0,1\}^{n_2} \ldots Q\alpha_k \in \{0,1\}^{n_k}\, \phi(\alpha_1..\alpha_k)$$

where Q is $\exists/\forall$ depending on k being odd/even, and where ϕ is a formula with k *sets* of inputs: the X_1 inputs $X_{1,1}..X_{1,n_1}$, the X_2 inputs $X_{2,1}..X_{2,n_2}$, and so on.

Approximate counting. The following fact is used in Chapters 4 and 5.

Fact 3.2 ([35]). *Let* $V(x, y) \in \mathrm{NP}$. *The function that approximates the size of the set*

$$V^{-1}(x, \cdot) := \{y : V(x, y) = 1\},$$

more precisely, the function

$$A(x, a, m) = \begin{cases} 1, \text{ if } \left|V^{-1}(x, \cdot)\right| \geqslant a \\ 0, \text{ if } \left|V^{-1}(x, \cdot)\right| < a(1 - 1/m) \end{cases}$$

where $m > 0$ *is an integer encoded in unary, is in* AM.

4 PROOF OF FIRST RESULT

In this chapter we prove Theorem 1.3 (p. 7), which we recall here in a stronger form:

Theorem 1.3, Stronger Form. *If* $\mathrm{AM} \subset \mathrm{P}^{\mathrm{NP}}$ *then* $\mathrm{EXP}^{\mathrm{NP}}$ *contains a function of circuit complexity in* $2^{\Omega(n)}$.

This is a stronger statement than the original (p. 7) because there the circuit complexity is stated as $2^{n^{\Omega(1)}}$.

The idea in proving Theorem 1.3 is what might be called "fast diagonalization via approximate counting". Consider the collection of all Boolean circuits, taking n bits to a single bit, of size $s(n)$ for a sufficiently small $s(n) \in 2^{\Omega(n)}$. Interpreting n bits as encoding an integer in the interval $[0, 2^n)$, we can quickly diagonalize against this collection with a function $L : \{0,1\}^n \to \{0,1\}$ defined as follows. Set $L(0)$ to the minority vote among all circuits in the collection regarding input 0. That is, $L(0) := b$ if the circuits that output b on input 0 comprise a subcollection smaller than those that output $\neg b$. (Break a tie arbitrarily, say as $L(0) := 0$.) Inductively, set $L(i)$ to the minority vote among the subcollection of circuits that agree with L on the interval $[0, i)$. When this process ends after 2^n steps, the function L thus obtained is guaranteed to be uncomputable by any circuit in the collection, provided the collection is of size at most 2^{2^n}, which it is since every size-$s(n)$ circuit can be encoded by a string of length $O(s(n) \log s(n))$.

This process would work just as well if, instead of exactly counting the votes in each step so as to pick the minority vote, we can approximately count so as to avoid picking the overwhelming majority vote if there is one. More precisely, in setting $L(i) := b$, if we can ensure that among the collection of circuits considered at the i^{th} step, those that output b on input i do not outnumber those that output

$\neg b$ by a factor of k or more, for any constant $k > 1$, then the resulting function L would still be hard for all size-$s(n)$ circuits.

We now fill in the details and show that this diagonalization via approximate counting can be performed, under the assumptions of Theorem 1.3, in $\mathrm{EXP}^{\mathrm{NP}}$.

Proof of Theorem 1.3. Let $s(n) = 2^{n/2}$. Under the assumption of Theorem 1.3, we construct a language $L \in \mathrm{EXP}^{\mathrm{NP}}$ such that for all but finitely many n, no circuit of size $s(n)$ computes L correctly on every $x \in \{0,1\}^n$.

Consider the following function V. On input (n, τ, i, C), where $\tau \in \{0,1\}^{2^n}$ represents the truth table of a Boolean function on n inputs, and C represents a size-$s(n)$ Boolean circuit on n inputs,

$$V(n, \tau, i, C) = 1 \text{ iff } C \text{ agrees with } \tau \text{ in the interval } [0, i),$$

where $[0, i)$ denotes the first i strings in $\{0,1\}^n$ in lex order.

Clearly, $V \in \mathrm{P}$. Thus, by Fact 3.2, the function $A(X, a)$ that approximately lower bounds the size of the set $V^{-1}(X, \cdot) := \{C : V(X, C) = 1\}$, more precisely, the function

$$A(X, a) = \begin{cases} 1, & \text{if } \left|V^{-1}(X, \cdot)\right| \geqslant a \\ 0, & \text{if } \left|V^{-1}(X, \cdot)\right| < a/2 \end{cases}$$

where X is shorthand for (n, τ, i), is in AM. By the assumption of the theorem that $\mathrm{AM} \subset \mathrm{P}^{\mathrm{NP}}$, there is a language $B \in \mathrm{P}^{\mathrm{NP}}$ that agrees with A on its domain.

We are ready to describe L. For every large enough n, construct L_n, the restriction of L to $\{0,1\}^n$, as follows. Initially set L_n to be the all zeroes function. For $i \in [0, 2^n)$, in order to decide whether to change the value of $L(i)$, let S_i denote the set of all size-$s(n)$ circuits on n inputs that agree with L on the

interval $[0, i]$. Use B to do a binary search for some a such that

$$B(n, \tau, i, a) = 1 \quad \text{and} \quad B(n, \tau, i, a+1) = 0,$$

where τ represents the truth table of L_n. The specification of A above guarantees

$$|S_i| \leqslant a \leqslant 2|S_i|, \tag{$\dagger$}$$

so we have a factor-2 approximation on $|S_i|$ from above, when $L(i) = 0$.

Now tentatively flip the value of $L(i)$ from 0 to 1, and repeat the binary search (by flipping the ith bit of τ) to obtain a new estimate a'. If $a' < a$, then keep $L(i)$ at value 1; otherwise, flip it back to 0. This completes the construction of L.

To show that L has the desired circuit complexity, it suffices to argue that the set S_i above eventually becomes empty as $i \in [0, 2^n)$ increases. So let S_{-1} be the set of all size-$s(n)$ circuits on n inputs, and let S_i be as defined earlier, i.e., as those circuits in S_{-1} that agree with L in the interval $[0, i]$, or equivalently, as those circuits in S_{i-1} that agree with L on i. Let $\overline{S}_i$ denote the complement of S_i in S_{i-1}. Finally, let a be the estimate obtained for $|S_i|$ during the construction, and let $\overline{a}$ be the one for $|\overline{S}_i|$.

By construction, $a \leqslant \overline{a}$, hence by ($\dagger$), $|S_i| \leqslant 2|\overline{S}_i|$. Since S_{i-1} is the disjoint union of S_i and $\overline{S}_i$, we have that $|S_i| + |\overline{S}_i| = |S_{i-1}|$ and putting together, that $|S_i| \leqslant 2|S_{i-1}|/3$. Therefore, $|S_i|$ is indeed empty when

$$i \geqslant 2 + \log_{3/2} |S_{-1}|,$$

and what remains is to show that the right hand side in this inequality falls in the interval $[0, 2^n)$. Since every circuit of size $s(n)$ can be represented by some string of length $O(s(n) \log s(n))$, we have $|S_{-1}| < 2^{3s(n)/2}$ for n large enough.

Hence the last inequality is implied by

$$i \geqslant 2 + \frac{3s(n)}{2\log 3 - 2}.$$

Using $\log 3 > 1.5$, we can clean up the right hand side to arrive at

$$i \geqslant 4s(n)$$

as a sufficient condition for S_i to be empty. By the setting of $s(n)$, the last quantity is less than 2^n for large enough n, as we wanted to show.

To prove that $L \in EXP^{NP}$, notice that the above construction for L starts with the all-zeroes string τ of length 2^n, and modifies it bit by bit, with each modification involving an oracle call to the language B, which takes time $O(2^n)$ to prepare. Hence $L \in E^B \subset E^{P^{NP}} \subset E^{NP}$, and the proof is done. □

5 PROOF OF SECOND RESULT

In this chapter we prove Theorem 1.4 (p. 7), which we recall here:

Theorem 1.4, Restated. *If* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$ *then for all constants* k, $\mathrm{P}^{\Sigma_2\mathrm{P}}$ *contains a function of nondeterministic circuit complexity in* $\omega(n^k)$.

Combinatorially, a nondeterministic Boolean circuit C is an ordinary Boolean circuit C' with two sets of inputs x and y. We refer to the second set of inputs as *choice* inputs — evocative of nondeterministic choice. We say that the circuit accepts input x, or $C(x) = 1$ in short, if $\exists y C'(x, y) = 1$; otherwise we say C rejects x, or $C(x) = 0$ in short.

The idea behind proving Theorem 1.4 is, at a very high level, identical to that of Theorem 1.3, namely "fast diagonalization via approximate counting". Recall this refers to mimicking the process that successively sets the next bit of the truth table of a function L_n to the minority vote of the circuits of size $s(n)$ that are consistent with the table constructed thus far. Whereas this ideal process would reduce the number of consistent circuits by at least half in each step, the mimicking process eliminates at least an α-fraction, for a value of α depending on the error rate of the approximate counting procedure. When α is a constant, we are guaranteed that after t steps, for some $t \in O(s(n) \log s(n))$ large enough, L_n is uncomputable by any size-$s(n)$ circuit, because every such circuit can be represented by a string of size $O(s(n) \log s(n))$.

There is a significant difference here, however, and it has to do with the type of circuits involved. For a deterministic circuit, whether it outputs 1 is no harder a question than whether it outputs 0, but for a nondeterministic circuit, there is an asymmetry due to the choice inputs — this is essentially the NP versus coNP problem. Consequently, it is a more delicate task to calculate the minority vote

among nondeterministic circuits, and hence to mimic that task via approximate counting. Of course, anything can be done given enough computational power; the goal here is to limit that power to $P^{\Sigma_2 P}$.

To be more specific, consider what would happen if we try to imitate the argument of the proof of Theorem 1.3 in order to prove Theorem 1.4. Right at the very first step, when we want to estimate how many circuits accept the input 0 and how many reject it, we run into a problem. While the set of accepting circuits has a membership predicate in NP — therefore, by Fact 3.2, its size can be estimated in AM, and hence in $\Sigma_2 P$ by the assumption of the theorem — the set of rejecting circuits correspond to a coNP predicate. Fact 3.2 says nothing about estimating the size of a coNP set. We need to find a way of breaking this asymmetry, and do so within $P^{\Sigma_2 P}$.

Our idea here is as follows. We may assume that every language in coNP has nondeterministic circuits of polynomial size, for otherwise there is nothing to prove. In particular, this holds for $\neg$CircuitSAT, the negation of the language CircuitSAT. So there is $k \in \mathbb{N}$ such that for all $m \in \mathbb{N}$, there is a nondeterministic circuit $C_{\neg \text{CircuitSAT}}$ of size $O(m^k)$ that receives as input an m-bit description of a deterministic circuit C, and accepts iff C is *un*satisfiable.

Now suppose we could somehow get a hold of $C_{\neg \text{CircuitSAT}}$ at any input length we want. Then to decide whether a given nondeterministic circuit C rejects its input x, i.e., whether

$$\forall y C'(x, y) = 0$$

where C' is the deterministic circuit underlying C and y represents the choice inputs of C, we could instead decide whether

$$C_{\neg \text{CircuitSAT}}(C'(x, \cdot)) = 1.$$

Since the latter is an NP predicate, we would thus get around the asymmetry problem mentioned above, and carry out the argument from Theorem 1.3 to derive Theorem 1.4 as well.

Alas, we do not have a circuit for ¬CircuitSAT, and the main contribution here is to show, using similar ideas to above — approximate counting and using circuits at a large enough input length to express coNP predicates as NP predicates — how to construct such a circuit in $\mathrm{FP}^{\Sigma_2\mathrm{P}}$. Details follow.

Proof of Theorem 1.4. Let $s(n) = n^k$ for an arbitrary $k \in \mathbb{N}$. Under the assumption that $\mathrm{AM} \subset \Sigma_2\mathrm{P}$, we construct a language $L \in \mathrm{P}^{\Sigma_2\mathrm{P}}$ such that for infinitely many n, no nondeterministic circuit of size $s(n)$ computes L correctly on every $x \in \{0,1\}^n$.

If setting $L := \neg\mathrm{CircuitSAT}$ does the job, then we are done. Hence we may assume that ¬CircuitSAT is computable at every large enough length m by some nondeterministic circuit $C_{\neg\mathrm{CircuitSAT}_m}$ of size $s(m)$.

A crucial piece of the proof is the following claim, whose proof we defer to after the proof of the theorem:

Lemma 5.1. *Suppose that* ¬CircuitSAT *is computable at every length* m *by some nondeterministic circuit of size* $s(m) \in O(m^k)$*, for some fixed* $k \in \mathbb{N}$*. Suppose further that* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$*.*

Then there is a function $C_{\neg\mathrm{CircuitSAT}} \in \mathrm{FP}^{\Sigma_2\mathrm{P}}$ *that on input* 1^m*, outputs a nondeterministic circuit (not necessarily of size* $s(m)$*) for* ¬CircuitSAT *at length* m*, for every* m*.*

With Lemma 5.1 in hand, we now follow the proof of Theorem 1.3, adjusting as necessary to the nondeterministic setting as outlined earlier in this section.

First, a bit of notation: Let $m \in O(s(n) \log s(n))$ be a function such that every nondeterministic circuit of size $s(n)$ can be represented by a string of length $m(n)$. Use m as a shorthand for $m(n)$. Also let $t \in \mathrm{poly}(m)$ be a function such

that the $C_{\neg \text{CircuitSAT}}$ function claimed to exist in Lemma 5.1 has output length at most $t(m)$ given input 1^m.

Consider the function

$$V(n, \tau, i, C) := 1 \text{ iff } C \text{ agrees with } \tau \text{ in the interval } [0, i),$$

where: (a) $\tau \in \{0,1\}^{2m}$ represents the first $2m$ entries of the truth table of a Boolean function on n inputs, (b) $i \leqslant 2m$, (c) C represents a size-$s(n)$ nondeterministic circuit on n inputs, and (d) $[0, i)$ denotes the first i strings in $\{0,1\}^n$ in lex order. (In case $2m > 2^n$, which can only happen on finitely many n since $m \in O(n^{k+1})$ by definition, let V consider the first 2^n bits of τ only.)

If we could say that $V \in \text{NP}$, then we would proceed just as in the proof Theorem 1.3 — V is almost identical to a function in that proof by the same name — but we cannot quite say that. Instead, we define a related function $W \in \text{NP}$ that serves as well as V when used with Lemma 5.1.

So let W be the following function. On input (D, n, τ, i, C), where $D \in \{0,1\}^{t(m)}$ represents a nondeterministic circuit with m inputs, and the rest of the parameters are as in V above,

$$W(D, n, \tau, i, C) := 1 \text{ iff } C \textit{ seems, with respect to } D, \text{ to agree with } \tau \text{ in the interval } [0, i),$$

where the condition on the right means the following. For every $j \in [0, i)$,

$$\begin{aligned} \tau(j) = 1 &\implies C(j) = 1 \\ \tau(j) = 0 &\implies D(C'(j, \cdot)) = 1 \end{aligned}$$

where $\tau(j)$ denotes the jth bit of τ, and C' is the deterministic circuit underlying C and $C(j) = 1$ is shorthand for $\exists z C'(j, z) = 1$, i.e., for $C'(j, \cdot)$ being a satisfiable

circuit. The following observation needs no proof.

Lemma 5.2. *Whenever* D *is a circuit for* $\neg$CircuitSAT, C *agrees with* τ *in the interval* $[0, i)$ *iff* C *seems, with respect to* D*, to agree with* τ *in the interval* $[0, i)$.

Clearly, $W \in \mathrm{NP}$. By Fact 3.2, the function $A(Y, a)$ that approximately lower bounds the size of the set $W^{-1}(Y, \cdot) := \{C : W(Y, C) = 1\}$, more precisely, the function

$$A(Y, a) := \begin{cases} 1, \text{ if } \left|W^{-1}(Y, \cdot)\right| \geqslant a \\ 0, \text{ if } \left|W^{-1}(Y, \cdot)\right| < a/2 \end{cases}$$

where Y is shorthand for (D, n, τ, i), is in AM. Further, by Lemma 5.1 and Lemma 5.2, A satisfies

$$A(C_{\neg\text{CircuitSAT}}(1^m), X, a) = \begin{cases} 1, \text{ if } \left|V^{-1}(X, \cdot)\right| \geqslant a \\ 0, \text{ if } \left|V^{-1}(X, \cdot)\right| < a/2 \end{cases} \tag{§}$$

where X is shorthand for (n, τ, i).

We are ready to describe L. For every large enough n, construct L_n, the restriction of L to $\{0,1\}^n$, as follows. Initially set L_n to be the all zeroes function. For $i \in [0, 2m)$, in order to decide whether to change the value of $L(i)$, let S_i denote the set of all size-$s(n)$ circuits on n inputs that agree with L on the interval $[0, i]$.

By the assumption of the theorem that AM $\subset \Sigma_2\mathrm{P}$, there is a language $B \in \Sigma_2\mathrm{P}$ that agrees with A on its domain. Use B, and the function $C_{\neg\text{CircuitSAT}}$ from Lemma 5.1, to do a binary search for some $a \in [0, 2^m]$ such that

$$B(C_{\neg\text{CircuitSAT}}(1^m), n, \tau, i, a) = 1 \quad \text{and} \quad B(C_{\neg\text{CircuitSAT}}(1^m), n, \tau, i, a+1) = 0,$$

where τ represents the first $2m$ entries of the truth table of L_n. ($2m \leqslant 2^n$ for n

large enough.) The observation (§) above regarding A guarantees that

$$|S_i| \leqslant a \leqslant 2|S_i|, \tag{†}$$

so we have a factor-2 approximation on $|S_i|$ from above, when $L(i) = 0$.

Now tentatively flip the value of $L(i)$ from 0 to 1, and repeat the binary search (by flipping the ith bit of τ) to obtain a new estimate a'. If $a' < a$, then keep $L(i)$ at value 1; otherwise, flip it back to 0. This completes the construction of L.

To show that L has the desired circuit complexity, it suffices to argue that the set S_i above eventually becomes empty as $i \in [0, 2m)$ increases. So let S_{-1} be the set of all size-$s(n)$ circuits on n inputs, and let S_i be as defined earlier, i.e., as those circuits in S_{-1} that agree with L in the interval $[0, i]$, or equivalently, as those circuits in S_{i-1} that agree with L on i. Let $\overline{S}_i$ denote the complement of S_i in S_{i-1}. Finally, let a be the estimate obtained for $|S_i|$ during the construction, and let $\overline{a}$ be the one for $|\overline{S}_i|$.

By construction, $a \leqslant \overline{a}$, hence by (†), $|S_i| \leqslant 2|\overline{S}_i|$. Since S_{i-1} is the disjoint union of S_i and $\overline{S}_i$, we have that $|S_i| + |\overline{S}_i| = |S_{i-1}|$ and putting together, that $|S_i| \leqslant 2|S_{i-1}|/3$. Therefore, $|S_i|$ is indeed empty when

$$i \geqslant 2 + \log_{3/2} |S_{-1}|,$$

and what remains is to show that the right hand side in this inequality falls in the interval $[0, 2m)$. Since every circuit of size $s(n)$ can be represented by some string of length m, we have $|S_{-1}| \leqslant 2^m$. Hence the last inequality is implied by

$$i \geqslant 2 + \frac{m}{(\log 3) - 1},$$

and since $\log 3 > 1.5$, by

$$i \geqslant 2m - 1$$

for all large enough n, as we wanted to show. (Recall m is a function of n.)

To prove that $L \in P^{\Sigma_2 P}$, notice that the above construction for L starts with the all-zeroes string τ of length $2m$, and modifies it bit by bit, with each modification involving $\leqslant m$ oracle calls to the language B while doing the binary search. Each call to B takes time $t(m) + \text{poly}(m) \subset \text{poly}(n)$ to prepare, provided the function $C_{\neg\text{CircuitSAT}}(1^m)$ of Lemma 5.1 is already computed, once, ahead of time. By Lemma 5.1, the latter can be done in time $\text{poly}(m)$ with oracle access to a $\Sigma_2 P$ function. Since the language B itself is in $\Sigma_2 P$, it follows that $L \in P^{\Sigma_2 P}$, and the proof is complete modulo the proof of Lemma 5.1.

(Theorem 1.4, mod Lemma 5.1) □

We now turn to the proof of Lemma 5.1. When viewed the right way, our approach here will not be much different from the high level idea of Theorem 1.4 above, which recall is the same as in Theorem 1.3. To bring out the similarity, let us abstract the idea of Theorem 1.3 a bit.

In the proof of Theorem 1.3 (and of Theorem 1.4) the process of constructing a truth table of high circuit complexity can be viewed as a prefix search for a high quality string, where quality is measured as the fraction of circuits that disagree with the string.

More precisely, let $\gamma \in (0, 1)$, and let

$$q : \{0,1\}^* \to [0, 1 - \gamma] \cup \{1\}$$

be a function satisfying, for every $x \in \{0,1\}^*$

$$\max(q(x0), q(x1)) \geqslant q(x) + \frac{1}{2}(1 - q(x)).$$

Starting with the empty string and appending one bit at each step, we can construct a string x with $q(x) = 1$ in roughly $\log(1/\gamma)$ steps, provided we have a selector function that helps us avoid a bad choice:

$$\mathrm{SEL}_q(x) = b \quad \text{whenever} \quad q(xb) - q(x\overline{b}) > \frac{1}{3}(1 - q(x)) \tag{♯}$$

where $b \in \{0,1\}$ and $\overline{b}$ denotes $\neg b$.

To make the connection to Theorem 1.3, given $\tau \in \{0,1\}^*$, let

$$q_n^{\text{Thm 1.3}}(\tau) := \text{fraction of size-}s(n)\text{ circuits on } n \text{ inputs that disagree with } \tau$$

and write

$$q(xb) - q(x\overline{b}) = (q(xb) - q(x)) - (q(x\overline{b}) - q(x))$$

to see that when $q = q_n^{\text{Thm 1.3}}$, the inequality in (♯) is equivalent to

$$\beta - \overline{\beta} > \frac{1}{3}(\beta + \overline{\beta})$$

where

$$\beta := \text{number of size-}s(n)\text{ circuits that agree with } x \text{ but disagree with } xb$$

and $\overline{\beta}$ is the same definition except for $x\overline{b}$ instead of xb.

Thus, to compute the selector function $\mathrm{SEL}_{q_n^{\text{Thm 1.3}}}$, it suffices to detect the case $\beta > 2\overline{\beta}$. One way to do this is to compute the approximations $\alpha \in [\beta, 2\beta]$ and

$\overline{a} \in [\overline{\beta}, 2\overline{\beta}]$, and then to return b or $\overline{b}$ depending on which of a or $\overline{a}$ is greater — this works because $\beta > 2\overline{\beta}$ implies $a > \overline{a}$, and because $a > \overline{a}$ implies $\overline{\beta} \leqslant 2\beta$. The tasks to compute a and $\overline{a}$ both Cook reduce to some function in AM. This is what the proof of Theorem 1.3 shows.

Now we turn to Lemma 5.1 with the same approach. We want to show an algorithm that constructs a small circuit for ¬CircuitSAT under certain assumptions. Following the approach above, the algorithm we show performs a prefix search for a high quality string, namely the string that describes a circuit for ¬CircuitSAT.

Before we proceed with the algorithm, we need to generalize the above approach a bit. As before, let $\gamma \in (0, 1)$. Call

$$q : \{0,1\}^* \to [0, 1-\gamma] \cup \{1\}$$

a *quality measure* if it satisfies two properties:

i. the *maintain* property: for every $x \in \{0,1\}^*$,

$$\max(q(x0), q(x1)) \geqslant q(x)$$

ii. the *gain* property: for some integer $p > 0$, for every $x \in \{0,1\}^{0 \bmod p}$,

$$\max(q(x0), q(x1)) \geqslant q(x) + \frac{1-q(x)}{1+1/p}.$$

By setting $p = 1$, we obtain the previous treatment where a significant gain in quality was possible at every step. In contrast, here such a gain is possible once every p steps only, and the maintain property makes it possible to preserve the quality in other steps. Call p the *plateau length* of q.

Call a function a *selector for* q, if for every $x \in \{0,1\}^{0 \bmod p}$ and $y \in \{0,1\}^{<p}$,

$$\mathrm{SEL}_q(xy, \varepsilon) = b \quad \text{whenever} \quad q(xyb) - q(xy\overline{b}) > \varepsilon(1 - q(x))$$

where $b \in \{0,1\}$, $\overline{b}$ denotes $\neg b$, and $\varepsilon = 1/d$ for some integer $d > 0$ encoded as 1^d. Then we can use SEL_q as an oracle to come up with a string of maximum quality 1 in about $\log(1/\gamma)$ steps. More precisely:

Lemma 5.3. *Let $\{q_n\}_{n\in\mathbb{N}}$ be a family of quality measures, and $\{\mathrm{SEL}_{q_n}\}_{n\in\mathbb{N}}$ of corresponding selector functions. Let p_n be the plateau length of q_n. The task to produce, given 1^n, a string $x \in \{0,1\}^{0 \bmod p_n}$ with $q_n(x) = 1$, lies in*

$$\mathrm{FTIME}^S(\mathrm{poly}(n, p_n, \log \frac{1}{\gamma_n}))$$

for every extension S of the function $(1^n, x, \varepsilon) \mapsto \mathrm{SEL}_{q_n}(x, \varepsilon)$ to a language.

Proof. Given 1^n, consider the process that initializes x to the empty string, and then for m steps, appends $S(1^n, x, \varepsilon)$ to x, and finally returns x. With foresight, set $m = p_n \lceil \log \frac{1}{\gamma_n} \rceil$ and $\varepsilon = \frac{1}{6p_n}$.

At each step of this process, the current value of x gets appended a bit $b \in \{0,1\}$ satisfying $S(1^n, x, \varepsilon) = b$, hence satisfying $\mathrm{SEL}_{q_n}(x, \varepsilon) \neq \overline{b}$. Therefore, since SEL_{q_n} is a selector for q_n,

$$q_n(xy\overline{b}) - q_n(xyb) \leqslant \varepsilon(1 - q_n(x)). \qquad (\dagger)$$

At steps $1, p_n + 1, 2p_n + 1, \ldots$ of the process, the string x in (†) is of length 0 mod p_n and y is empty, so

$$q_n(xb) \geqslant q_n(x\overline{b}) - \varepsilon(1 - q_n(x))$$

at those steps. Putting together with the gain property, we get that even if the process does not make the best choice at such a step, i.e., even if $q_n(x\overline{b}) > q_n(xb)$,

$$\begin{aligned} q_n(xb) &\geqslant q_n(x) + (\frac{1}{1+1/p_n} - \varepsilon)(1 - q_n(x)) \\ &\geqslant q_n(x) + (\frac{1}{2} - \varepsilon)(1 - q_n(x)). \end{aligned}$$

Since $\varepsilon \leqslant 1/6$, it follows that at steps $1, p_n + 1, 2p_n + 1, \dots$, the process closes at least a third of the remaining gap between the quality value $q_n(x)$ at the beginning of the step and the maximum possible value 1.

During other steps, even if the process does not make the better choice, i.e., even if $q_n(xy\overline{b}) > q_n(xyb)$, the maintain property applied to (†) gives

$$q_n(xyb) \geqslant q_n(xy) - \varepsilon(1 - q_n(x))$$

which implies, since $\varepsilon \leqslant \frac{1}{6p_n}$, that during steps $2 \dots p_n$, the quality lost is bounded by $1/6$ of the gap between the initial quality (of the empty string) and the maximum possible quality 1. More generally, during steps $jp_n + 2 \dots jp_n$, the quality lost is bounded by $1/6$ of the gap between the quality at the beginning of step $jp_n + 1$ and the maximum possible quality 1.

Putting together, at every block of p_n steps, the process closes at least a $\frac{1}{6}$-fraction of the remaining gap between the quality at the beginning of the block and the maximum possible value 1. It follows, because q_n takes no value in the interval $(\gamma_n, 1)$, that after $\lceil \log_{1/6} \gamma_n \rceil$ blocks, hence certainly after $p_n \lceil \log 1/\gamma_n \rceil$ steps, the quality reached is guaranteed to be 1. □

We are ready to prove Lemma 5.1 and thus finish the proof of Theorem 1.4.

Proof of Lemma 5.1. We want to show that given 1^n, a nondeterministic circuit

for ¬CircuitSAT at length n can be constructed in $\mathrm{FP}^{\Sigma_2 \mathrm{P}}$, assuming: a) such a circuit of size $s(n) \in O(n^k)$ exists for all n, for some fixed k, and, b) $\mathrm{AM} \subset \Sigma_2 \mathrm{P}$.

First, some notation: Let $\tilde{n} \in O(n \log n)$ be such that every nondeterministic circuit of size n can be represented by a string of size $\tilde{n}$. Use $\tilde{s}(n)$ for $\widetilde{s(n)}$. Given a language L and a circuit, call that circuit *sound with respect to* L iff $L(x) = 1$ for every input x accepted by the circuit.

For $n \in \mathbb{N}$, consider the quality measure $q_n^{\text{Lem5.1}}$ that interprets its input as $C_1 C_2 \dots C_t D$, where each $C_i \in \{0,1\}^{\tilde{s}(n)}$ represents a nondeterministic circuit of size $s(n)$ on n inputs, and $D \in \{0,1\}^{<\tilde{s}(n)}$ is the prefix of such a representation. Define $q_n^{\text{Lem5.1}}$ as follows.

$$q_n^{\text{Lem5.1}}(C_1 C_2 \dots C_t D) := \begin{cases} 0, & \text{if } C_i \text{ is not sound w.r.t. } \neg\text{CircuitSAT for some } i \\ \dfrac{\left|\left(\bigvee_i C_i\right)^{-1}(1)\right|}{\left|\neg\text{CircuitSAT}_n{}^{-1}(1)\right|}, & \text{if } D \text{ is the empty string} \\ \max\limits_{D^* \in \mathrm{Suffix}_D} \dfrac{\left|\left(\bigvee_i C_i \vee D^*\right)^{-1}(1)\right|}{\left|\neg\text{CircuitSAT}_n{}^{-1}(1)\right|}, & \text{else} \end{cases}$$

where $\neg\text{CircuitSAT}_n$ is the restriction of CircuitSAT to length-n inputs, and

$$\mathrm{Suffix}_D := \{D^* \in \{0,1\}^{\tilde{s}(n)} : D^* \text{ is sound w.r.t. } \neg\text{CircuitSAT} \\ \text{and } D \text{ is a prefix of } D^*\}.$$

We claim that $q_n^{\text{Lem5.1}}$ is a quality measure, with $\gamma_n = 1/2^n$ and $p_n = \tilde{s}(n)$. The gain property follows from the assumption that there does exist a size-$s(n)$ circuit for $\neg\text{CircuitSAT}_n$, hence a length-$\tilde{s}(n)$ representation of that circuit, so that on inputs of the form $C_1 C_2 \dots C_t \in \{0,1\}^{0 \bmod p_n}$, one always has the option of appending the first bit of that representation, thereby getting a string of quality

1 (albeit not of length 0 mod p_n). The maintain property is clear.

By Lemma 5.3, the task to produce, given 1^n, a circuit for $\neg\text{CircuitSAT}_n$ is in

$$\text{FTIME}^S(\text{poly}(n, p_n, \log \frac{1}{\gamma_n})) = \text{FP}^S$$

for every extension S of the function

$$\text{SEL}_{q^{\text{Lem5.1}}} : (1^n, x, \varepsilon) \mapsto \text{SEL}_{q_n^{\text{Lem5.1}}}(x, \varepsilon)$$

to a total function. Therefore, the proof is complete once we show:

Lemma 5.4. *Under the assumptions of Lemma 5.1,* $\text{SEL}_{q^{\text{Lem5.1}}}$ *can be extended to some* $S \in \text{P}^{\Sigma_2\text{P}}$.

which we will do in the rest of this section.

(Lemma 5.1, mod Lemma 5.4) □

Proof of Lemma 5.4. First, let us unwind definitions to see what $\text{SEL}_{q^{\text{Lem5.1}}}$ does. Since it is a selector function for $q^{\text{Lem5.1}}$,

$$\text{SEL}_{q^{\text{Lem5.1}}}(1^n, xy, \varepsilon) = b \quad \text{if} \quad q_n^{\text{Lem5.1}}(xyb) - q_n^{\text{Lem5.1}}(xy\overline{b}) > \varepsilon(1 - q_n^{\text{Lem5.1}}(x)).$$

Letting q denote $q^{\text{Lem5.1}}$, we can rewrite the last inequality as

$$(q_n(xyb) - q_n(x)) - (q_n(xy\overline{b}) - q_n(x)) > \varepsilon(1 - q_n(x)).$$

Recalling how q_n behaves — it interprets its input xy, where $|x| \equiv 0 \bmod \tilde{s}(n)$ and $|y| < \tilde{s}(n)$, as $C_1C_2 \ldots C_tD$ where $C_i \in \{0,1\}^{\tilde{s}(n)}$ represents a nondeterministic circuit of size $s(n)$ on n inputs, and $D \in \{0,1\}^{<\tilde{s}(n)}$ is the prefix of such a

representation — we can write the first term in the last inequality as

$$(q_n(C_{1..t}Db) - q_n(C_{1..t})) = \max_{D_b^* \in Suffix_{Db}} \frac{|(D_b^*)^{-1}(1) \setminus (\bigvee_i C_i)^{-1}(1)|}{|\neg \text{CircuitSAT}_n{}^{-1}(1)|}.$$

provided that each C_i is sound with respect to CircuitSAT.

Repeating this for the other two terms in that inequality, we get

$$\begin{aligned} \text{SEL}_{q^{\text{Lem5.1}}}(1^n, C_{1..t}D, \varepsilon) = b \quad \text{if} \quad & \max_{D_b^* \in Suffix_{Db}} \left|(D_b^*)^{-1}(1) \setminus (\vee_i C_i)^{-1}(1)\right| \\ & - \max_{D_{\overline{b}}^* \in Suffix_{D\overline{b}}} \left|(D_{\overline{b}}^*)^{-1}(1) \setminus (\vee_i C_i)^{-1}(1)\right| \\ & > \varepsilon \left|\neg \text{CircuitSAT}_n{}^{-1}(1) \setminus (\vee_i C_i)^{-1}(1)\right| \end{aligned}$$

provided that each C_i is sound with respect to $\neg$CircuitSAT.

What if some C_i is not sound? Then the quality measure $q_n^{\text{Lem5.1}}$ gives 0 no matter what, and $\text{SEL}_{q^{\text{Lem5.1}}}$ would be correct no matter what it returns. Thus from here onwards we do assume that each C_i is sound with respect to $\neg$CircuitSAT.

At this point, it is becoming clearer what we are to do. Letting

$$\begin{aligned} \beta &:= \max_{D_b^* \in Suffix_{Db}} \left|(D_b^*)^{-1}(1) \setminus (\vee_i C_i)^{-1}(1)\right|, \\ \overline{\beta} &:= \max_{D_{\overline{b}}^* \in Suffix_{D\overline{b}}} \left|(D_{\overline{b}}^*)^{-1}(1) \setminus (\vee_i C_i)^{-1}(1)\right| \end{aligned}$$

and observing that

$$\beta, \overline{\beta} \leqslant \left|\neg \text{CircuitSAT}_n{}^{-1}(1) \setminus (\vee_i C_i)^{-1}(1)\right|$$

we see that the function R defined as

$$R(1^n, C_{1..t}D, \varepsilon) = b \quad \text{if} \quad \beta - \overline{\beta} > \frac{\varepsilon}{2}(\beta + \overline{\beta})$$

extends $\mathrm{SEL}_{q^{\mathrm{Lem5.1}}}$. Therefore it suffices to show:

Lemma 5.5. *Under the assumptions of Lemma 5.1,* R *can be extended to some* $S \in P^{\Sigma_2 P}$.

which we will do in the rest of this section.

(Lemma 5.4, mod Lemma 5.5) □

Proof of Lemma 5.5. We want to show how to detect the condition

$$\beta > \frac{(1+\varepsilon/2)}{(1-\varepsilon/2)}\,\overline{\beta} \tag{‡}$$

in $P^{\Sigma_2 P}$. One way to detect this is to compute the approximations α and $\overline{\alpha}$ such that

$$\beta \leqslant \alpha \leqslant (1-\varepsilon/2)^{-1}\beta, \qquad \overline{\beta} \leqslant \overline{\alpha} \leqslant (1-\varepsilon/2)^{-1}\overline{\beta} \tag{𝔄}$$

and then to return b or $\overline{b}$ depending on which of α or $\overline{\alpha}$ is greater — this works because (‡) implies $\alpha > \overline{\alpha}$ (think of the contrapositive) and because $\alpha > \overline{\alpha}$ implies the negation of ($\overline{‡}$), where ($\overline{‡}$) is the same as (‡) except β and $\overline{\beta}$ are swapped.

So it suffices to show that under the assumptions of the lemma, the following task is in $FP^{\Sigma_2 P}$: given $(1^n, C_{1..t}D, \varepsilon)$, output the approximations α and $\overline{\alpha}$ satisfying (𝔄). Actually it suffices to do this for α only; the case for $\overline{\alpha}$ follows by symmetry.

Let the length of the input $(1^n, C_{1..t}D, \varepsilon)$ be N. Recall that each $C_i \in \{0,1\}^{s(n)}$ represents a nondeterministic circuit of size $s(n)$ on n inputs, and $D \in \{0,1\}^{<s(n)}$ is the prefix of such a representation. Let $r(n)$ be the size of the circuit $\bigvee_i C_i$,

and let $\tilde{r}(n)$ be the length of the string representing that circuit. Notice that since each C_i is nondeterministic, so is $\bigvee_i C_i$.

By the assumption that ¬CircuitSAT has a nondeterministic circuit of size $s(n)$ for every n, we know that for every input $x \in \{0,1\}^n$,

$$\bigvee_i C_i(x) = 0 \quad \text{iff} \quad M(\bigvee_i C_i(x)) = 1$$

for some nondeterministic circuit M computing ¬CircuitSAT at input length $\tilde{r}(n)$. M is of size $s(\tilde{r}(n))$. Recalling that $s \in \text{poly}(n) \subset \text{poly}(N)$, the size of M is in $\text{poly}(N)$.

Consider the function

$$Q(1^n, C_1..C_t D, a, \varepsilon) := \begin{cases} 1, & \text{if } \exists M, D^*\text{: } M, D^* \text{ are sound w.r.t. ¬CircuitSAT and } \left|(M(\vee_i C_i) \wedge D^*)^{-1}(1)\right| \geqslant a \\ 0, & \text{if } \forall M, D^*\text{: } M, D^* \text{ are sound w.r.t. ¬CircuitSAT implies } \left|(M(\vee_i C_i) \wedge D^*)^{-1}(1)\right| < a(1-\varepsilon) \end{cases}$$

where M is quantified over $\{0,1\}^{\tilde{s}(\tilde{r}(n))}$, i.e., over circuits of size $s(\tilde{r}(n))$ on $\tilde{r}(n)$ inputs, and D^* is quantified over $\{D\} \times \{0,1\}^{\tilde{s}(n)-|D|}$, i.e., over circuits of size $s(n)$ whose descriptions start as D.

We claim that Q can be extended to some $L \in \Sigma_2 P$. Once we show this, the proof will be done because it is guaranteed, by the specification of Q and by the existence of a circuit M for ¬CircuitSAT at an appropriate length, that a binary

search for $\alpha \in [0, 2^n]$ satisfying

$$L(1^n, C_1..C_t Db, \alpha, \varepsilon/2) = 1 \quad \text{and} \quad L(1^n, C_1..C_t Db, \alpha + 1, \varepsilon/2) = 0$$

yields an α satisfying ($\mathfrak{A}$), as desired.

By the assumption that AM $\subset \Sigma_2 P$ and by Fact 3.2, there is a language $A \in \Sigma_2 P$ such that

$$A(1^n, C, a, \varepsilon) = \begin{cases} 1, & \text{if } \left|C^{-1}(1)\right| \geqslant a \\ 0, & \text{if } \left|C^{-1}(1)\right| < a(1-\varepsilon) \end{cases}$$

where C denotes a nondeterministic circuit on n inputs. Therefore, the language L defined as

$$\begin{aligned} L(1^n, Cb_1..C_t D, a, \varepsilon) := \exists M, D^* : M, D^* \text{ are sound w.r.t. } \neg\text{CircuitSAT} \\ \text{and} \quad A(1^n, M(\vee_i C_i) \wedge D^*, a, \varepsilon) \end{aligned}$$

extends Q. Here M and D^* are quantified in the same way as in Q.

Since the predicate for checking soundness of a circuit with respect to $\neg$CircuitSAT is in coNP — for every satisfiable nondeterministic circuit x, does M reject x? — L is in $\Sigma_2 P$ as claimed. This finishes the proof. $\square$

6 PROOF OF THIRD RESULT

In this chapter we prove Theorem 1.5 (p. 7), which we recall here:

Theorem 1.5, Restated. *If* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$ *then* $\Sigma_2\mathrm{EXP}$ *contains a function of nondeterministic circuit complexity in* $n^{\omega(1)}$.

The proof uses two main ingredients. The first is a variant of Theorem 1.4.

Lemma 6.1. *If* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$ *then* $\mathrm{EXP}^{\Sigma_2\mathrm{P}}$ *contains a function of nondeterministic circuit complexity in* $n^{\omega(1)}$.

Lemma 6.1 is proved the same way as Theorem 1.4, with straightforward adjustments to some parameters. We defer its proof to after the proof of Theorem 1.5.

The second ingredient is quite involved. It says that assuming $\mathrm{AM} \subset \Sigma_2\mathrm{P}$, if the language $\oplus\mathrm{SAT}$ (Section 3) has small nondeterministic circuits, then the exponential-time hierarchy collapses to the second level.

Lemma 6.2. *Suppose* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$. *If* $\oplus\mathrm{SAT}$ *has nondeterministic circuits of polynomial size, then* $\mathrm{EXP}^{\Sigma_2\mathrm{P}} \subset \Sigma_2\mathrm{EXP}$.

We defer the proof of Lemma 6.2 to later in this chapter, barring one result from Chapter 7 needed in the proof.

We now proceed with the proof of Theorem 1.5.

Proof of Theorem 1.5. We may assume that the language $\oplus\mathrm{SAT}$ has nondeterministic circuits of polynomial size, for otherwise there is nothing to prove since $\oplus\mathrm{SAT} \in \mathrm{EXP} \subset \Sigma_2\mathrm{EXP}$. The theorem now follows from Lemmas 6.1-6.2.

(Theorem 1.5, mod Lemmas 6.1-6.2) □

Proof of Lemma 6.1. The proof is the same as that of Theorem 1.4, with minor adjustments to the parameters.

For notational convenience, let $\widetilde{\mathrm{P}}$ denote $\mathrm{DTIME}(2^{\mathrm{poly}\log(n)})$. Similarly, let $\widetilde{\mathrm{FP}}$ denote $\mathrm{FTIME}(2^{\mathrm{poly}\log(n)})$, $\Sigma_2\widetilde{\mathrm{P}}$ denote $\Sigma_2\mathrm{TIME}(2^{\mathrm{poly}\log n})$, etc.

Let $s(n) = 2^{\log^k n}$ for an arbitrarily large $k \in \mathbb{N}$. Under the assumption that $\mathrm{AM} \subset \Sigma_2\mathrm{P}$, we construct a language $L \in \mathrm{EXP}^{\Sigma_2\mathrm{P}}$ such that for infinitely many n, no nondeterministic circuit of size $s(n)$ computes L correctly on every $x \in \{0,1\}^n$.

If setting $L := \neg\mathrm{CircuitSAT}$ does the job, then we are done. Hence we may assume that $\neg\mathrm{CircuitSAT}$ is computable at every large enough length m by some nondeterministic circuit $C_{\neg\mathrm{CircuitSAT}_m}$ of size $s(m)$.

A crucial piece of the proof is the following lemma, which is a variant of Lemma 5.1 used crucially in Theorem 1.4.

Lemma 6.3. *Suppose that* $\neg\mathrm{CircuitSAT}$ *is computable at every length* m *by a nondeterministic circuit of size* $s(m) \in O(2^{\log^k m})$*, for some fixed* $k \in \mathbb{N}$*. Suppose further that* $\mathrm{AM} \subset \Sigma_2\mathrm{P}$*.*

Then there is a function $C_{\neg\mathrm{CircuitSAT}} \in \widetilde{\mathrm{FP}}^{\Sigma_2\mathrm{P}}$ *that on input* 1^m*, outputs a nondeterministic circuit (not necessarily of size* $s(m)$*) for* $\neg\mathrm{CircuitSAT}$ *at length* m*, for every* m*.*

Given Lemma 6.3, the rest of the proof proceeds identical to that of Theorem 1.4, provided we put $2^{\mathrm{poly}\log(n)}$ in place of $\mathrm{poly}(n)$ whenever it occurs (and it occurs only once, in the analysis at the end).

(Lemma 6.1, mod Lemma 6.3) □

Proof Sketch for Lemma 6.3. The proof is almost identical to that of Lemma 5.1; the only adjustments needed are: (a) replacing $s \in O(n^k)$ with $s \in O(2^{\log^k n})$, and (b) replacing occurrences of $\Sigma_2\mathrm{P}$ with $\Sigma_2\widetilde{\mathrm{P}}$ whenever it has to do with the complexity of the selector function.

With these adjustments, the complexity of the function $C_{\neg\text{CircuitSAT}}$ becomes $\widetilde{FP}^S$ for some language S in $P^{\Sigma_2\widetilde{P}}$, where we use, as above, $\widetilde{FP}$ (and respectively, $\widetilde{P}$) to denote $\text{FTIME}(2^{\text{poly}\log n})$ (respectively, $\text{DTIME}(2^{\text{poly}\log n})$). Simplifying, we get $\widetilde{FP}^{\Sigma_2\widetilde{P}}$. Finally, because the time complexity of oracle queries can be absorbed into the queries themselves by padding, and because functions of the form $2^{\text{poly}\log n}$ are closed under composition, we get $C_{\neg\text{CircuitSAT}} \in \widetilde{FP}^{\Sigma_2 P}$ as claimed.

□

We now turn to Lemma 6.2. Actually we prove something stronger:

Lemma 6.4. *Suppose* $\text{AM} \subset \Sigma_2\text{P}$. *If* $\oplus\text{SAT}$ *has nondeterministic circuits of polynomial size, then* $P^{\Sigma_2 P} \subset \Sigma_2\text{P}$.

Lemma 6.2 follows from Lemma 6.4 by a so-called padding argument:

Proof of Lemma 6.2. For $L \in \text{EXP}^{\Sigma_2 P}$, let L' be the language mapping input pairs of the form (x, y) to $L(x)$ whenever $|y| \geqslant 2^{|x|^c}$, where c is the smallest integer such that $L \in \text{DTIME}^{\Sigma_2 P}(2^{O(n^c)})$. On the rest of input pairs let L' behave trivially, say by mapping to zero. Then $L' \in P^{\Sigma_2 P}$.

Suppose that $\text{AM} \subset \Sigma_2\text{P}$. Suppose further that $\oplus\text{SAT}$ has nondeterministic circuits of polynomial size. Then by Lemma 6.4, $L' \in \Sigma_2\text{P}$. Hence the language K mapping x to $L'(x, 1^{2^{|x|^c}})$ is in $\Sigma_2\text{EXP}$. But K is identical to L.

(Lemma 6.2, mod Lemma 6.4) □

To prove Lemma 6.4, the first ingredient we will use is a technical augmentation of Arthur-Merlin protocols.

Definition 6.5 (Augmented Arthur-Merlin protocol)**.** *The class* MAH *consists of all functions* $\Pi(x)$ *for which there exists* $\ell \in \text{poly}(n)$, *a function* $\Gamma(x, y) \in \text{AM}$, *and a*

function $V(x, y) \in \text{coNP}$ *such that*

$$\Pi(x) = \begin{cases} 1, & \text{if } \exists y : ((x, y) \in \operatorname{dom} \Gamma \wedge \ \Gamma(x, y)) \wedge V(x, y) \\ 0, & \text{if } \forall y : ((x, y) \in \operatorname{dom} \Gamma \wedge \neg\Gamma(x, y)) \vee \neg V(x, y) \end{cases}$$

where y *ranges over* $\{0,1\}^{\ell(|x|)}$.

Similar to the class AM, underlying each function in MAH there is a protocol between an all-powerful prover, Merlin, and — in this case — *two* verifiers, Arthur and *Henry*, who cannot communicate with each other. Arthur is the usual randomized polynomial-time verifier for the AM-function Γ; Henry is the coNP-verifier deciding V. Merlin goes first and sends a common message to both verifiers. At this point, Henry has to make a decision to accept/reject, whereas Arthur can interact with Merlin as in the Arthur-Merlin protocol for Γ before making a decision. The input is accepted by the protocol iff both verifiers accept. (Since the word "verifier" connotes restricted computational power, it may be helpful to think of Henry as having private access to a *second* all-powerful prover who competes with Merlin by providing a certificate that is to serve as a counter-certificate to Merlin's initial message.)

The class MAH is useful because the assumption of Lemma 6.4, that $\text{AM} \subset \Sigma_2\text{P}$, can be equivalently stated using MAH instead:

Lemma 6.6. $\text{AM} \subset \Sigma_2\text{P}$ *iff* $\text{MAH} \subset \Sigma_2\text{P}$.

Proof. One direction is trivial. As for the other, replacing Γ in Definition 6.5 by a Σ_2P-predicate turns Π into a Σ_2P-predicate. □

The second ingredient in proving Lemma 6.4 is a substantial result from Chapter 7, showing that the language $\oplus$SAT is checkable (as defined in Section 3).

Theorem 7.1, Pre-stated. $\oplus$SAT *is checkable.*

The third and final ingredient is a famous result of Toda [51].

Fact 6.7 ([51])**.** Σ_3SAT $\to \oplus$SAT *via a randomized reduction.*

We are ready to derive Lemma 6.4, thus finish the proof of Theorem 1.5.

Proof of Lemma 6.4. Suppose AM $\subset \Sigma_2$P. Suppose that $\oplus$SAT has nondeterministic circuits of polynomial size.

For warmup, suppose further that $\oplus$SAT has *deterministic* circuits of polynomial size. Under this additional assumption, there is an MA-protocol for Σ_3SAT that proceeds roughly as follows. Given input $\varphi \in \{0,1\}^n$ to Σ_3SAT, Arthur performs the reduction in Fact 6.7, thereby obtaining an input $\psi \in \{0,1\}^m$ to $\oplus$SAT such that

$$\Pr[\Sigma_3\text{SAT}(\varphi) = \oplus\text{SAT}(\psi)] \geqslant 1 - \varepsilon$$

for some $m \in \text{poly}(n)$ and some negligible $\varepsilon \in (1/n)^{\omega(1)}$. Merlin sends Arthur a circuit for $\oplus$SAT on ℓ-bit inputs, for a sufficiently large $\ell \in \text{poly}(m) \subset \text{poly}(n)$. Arthur then uses that circuit as the prover in an interactive protocol for the $\oplus$SAT instance ψ.

Since MA $\subset \Sigma_2$P and $\text{P}^{\Sigma_2\text{P}} \subset \Sigma_3$P, it follows that under the additional assumption, $\text{P}^{\Sigma_2\text{P}} \subset \Sigma_2$P as desired.

Now we remove the additional assumption and try to mimic the same argument, this time using an MAH protocol instead of MA. By assumption, there is a nondeterministic circuit C such that for every $\psi \in \{0,1\}^\ell$, C accepts ψ iff $\oplus\text{SAT}(\psi) = 1$. In fact, because $\oplus$SAT is closed under complement, there is another nondeterministic circuit that accepts ψ iff $\oplus\text{SAT}(\psi) = 0$. Putting together, there is a nondeterministic circuit D such that for every $\psi \in \{0,1\}^\ell$ and $b \in \{0,1\}$, D accepts (b, ψ) iff $\oplus\text{SAT}(\psi) = b$. Say that D *two-sidedly* decides $\oplus$SAT.

Now, the MAH protocol for $\oplus$SAT starts the same way as in the above MA-protocol, with Arthur doing the reduction $\varphi \mapsto \psi$ and Merlin sending a circuit D that (in this case is nondeterministic and two-sidedly) decides $\oplus$SAT. In contrast to the above protocol, here Henry, the second verifier, also receives the circuit D; his role is to check that D is *consistent* in the following sense: D should never accept both $(0, x)$ and $(1, x)$ for any x. If this check fails, the protocol rejects.

If the consistency check passes, then Arthur and Merlin continue with the protocol as follows. Arthur initiates a simulation of the interactive protocol for $\oplus$SAT on ψ, by picking a random string $y_1 \dots y_t$ and sending it to Merlin. Merlin responds with the string $z_1 \dots z_t$, purported to be the responses of the honest prover in the interactive protocol for $\oplus$SAT on ψ where the messages of the verifier are $y_1 \dots y_t$. Merlin also sends a *sequence* of strings $s_1 \dots s_t$; these are for Arthur to use as the choice inputs when evaluating D during the simulation.

Arthur then performs the simulation as follows. Since $\oplus$SAT is checkable, there is a reduction $R \in \mathrm{FP}$ such that each message z_i of the honest prover — if it is indeed from the honest prover — must satisfy

$$z_i = \oplus\mathrm{SAT}(R(\psi, y_1, \dots, y_i)),$$

or, in terms of the circuit D that two-sidedly decides $\oplus$SAT — if it indeed two-sidedly decides $\oplus$SAT — each z_i must satisfy

$$D(z_i, R(\psi, y_1, \dots, y_i)) = 1.$$

Arthur checks this last condition for each z_i, by using the string s_i as the choice inputs in evaluating D. If this check passes, the protocol accepts. Otherwise it rejects.

Notice that in a regular interactive protocol, the soundness error probability

(i.e., the probability that the verifier rejects the input, if the input is to be rejected) would be compromised if the verifier sends his random choices y all at once to the prover. However, here we have a circuit D that acts as a prior commitment by the prover, who cannot adapt his responses z to the values sent by the verifier.

More specifically, suppose $\Sigma_3\text{SAT}(\varphi) = 1$. Then Merlin can just send the circuit for $\oplus$SAT at the appropriate input length. That circuit passes the coNP-verifier Henry, and also passes Arthur's verification with high probability.

As for the case $\Sigma_3\text{SAT}(\varphi) = 0$, suppose that Merlin sends a circuit D' that passes Henry's test. This means that D' is consistent, and corresponds to a fixed prover strategy. Then Arthur rejects with high probability by the soundness of the original interactive proof system for $\oplus$SAT. This finishes the proof. $\square$

7 PROOF OF FOURTH RESULT

In this chapter we prove Theorem 1.7 (p. 11), which we recall here:

Thm 1.7, Restated.

i. MAEXP $\not\subset$ P/poly *holds relative to every affine oracle.*

ii. NEXP $\not\subset$ P/poly *does* not *hold relative to every affine oracle.*

iii. NEXP $\subset$ P/poly *does* not *hold relative to every affine oracle.*

The first order of business is to state what affine oracles are, and what it means for something to hold relative to an oracle. Hence we start with definitions. After that we prove a useful property of $\oplus$SAT, namely that it is checkable (p. 33). After that we turn to Theorem 1.7 and prove each part in a separate section.

7.1 Definitions

Oracle access capability. Given $r \in \text{poly}(n)$ and $f \in \text{FP}$, consider the function

$$f^* : (\mathcal{O}, x) \mapsto f^{\mathcal{O}}(x)$$

that takes as input any language $\mathcal{O}$ and string x, and outputs $g(x)$, where f is a Cook reduction of round complexity r from the language g to $\mathcal{O}$. We call the set of all such f^*, over all $f \in \text{FP}$ and all $r \in \text{poly}(n)$, the class FP^* — "FP *with oracle access capability*".

For readability, we always use the notation in the previous paragraph: a starred symbol such as f^* denotes a member of FP^*, and its un-starred version f denotes the member of FP on which f^* is based.

For each $f^* \in \mathrm{FP}^*$, we say f^* *has oracle access capability*. We use $f^{\mathcal{O}}$ to denote $f^*(\mathcal{O}, \cdot)$, the restriction of f^* obtained by setting its first argument to $\mathcal{O}$, and refer to $f^{\mathcal{O}}$ as f^* *when given access to* $\mathcal{O}$.

All definitions built on FP above naturally generalize to their *oracle-access-capable* versions. For example,

$$\begin{aligned}\mathrm{NP} := \{L : \exists V \in \mathrm{P},\ \exists \ell \in \mathrm{poly}(n),\ \forall x \text{ string} \\ L(x) = 1 \iff \exists y \in \{0,1\}^{\ell(|x|)}\ V(x,y) \quad \}\end{aligned}$$

generalizes to

$$\begin{aligned}\mathrm{NP}^* := \{L^* : \exists V^* \in \mathrm{P}^*,\ \exists \ell \in \mathrm{poly}(n),\ \forall x \text{ string},\ \forall \mathcal{O} \text{ language}, \\ L^*(\mathcal{O},x) = 1 \iff \exists y \in \{0,1\}^{\ell(|x|)}\ V^*(\mathcal{O},x,y) \quad \}.\end{aligned} \tag{7.1}$$

Similarly IP generalizes to IP^*, etc. When we say "there is an interactive protocol where the verifier has oracle access capability", for example, we are merely referring to a function in IP^*.

Enumeration. We take it as a fact that FP is enumerable. It follows that every class defined above is enumerable. For example, to find an enumeration of NP^*, by (7.1) it suffices to find an enumeration of P^* and cross with $\mathbb{N}$. To find an enumeration of P^*, it suffices to find an enumeration of FP^* since P^* is obtained by taking every function in FP^* and projecting its output to the first coordinate. Finally, to find an enumeration of FP^*, it suffices to take an enumeration for FP and cross it with $\mathbb{N}$, because by definitions above, underlying every $f^* \in \mathrm{FP}$ are some $f \in \mathrm{FP}$ and some $r \in \mathrm{poly}(n)$.

Query complexity. Let $f^* \in \mathrm{FP}^*$. By definition, underlying f^* are some $r \in \mathrm{poly}(n)$ and $f \in \mathrm{FP}$, where f is a Cook reduction of round complexity r. We refer to r as the *query complexity* of f^*.

Well-behaved resource bound. Call a function $s : \mathbb{N} \to \mathbb{N}$ a *well-behaved resource bound* if it is increasing, satisfies $O(s(n)) \subset s(O(n)) \subset s(n)^{O(1)} \subset s(n^{O(1)})$ and $n \leqslant s(n)$, and if the function that maps the binary encoding of n to the binary encoding of $s(n)$ is in FP. Functions of the form $n^d, (n^d \log n)^{d'}, 2^{(\log n)^d}, 2^{dn}$ are well-behaved resource bounds.

This generalizes to $s : \mathbb{N}^2 \to \mathbb{N}$ if fixing either of the inputs yields a well-behaved resource bound.

Languages as families. In this chapter we specify languages as families of Boolean functions $\{L_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$. Sometimes we also specify them as $\left\{f_m : \{0,1\}^{s(m)} \to \{0,1\}\right\}_{n\in\mathbb{N}}$ or as $\left\{f_{m,k} : \{0,1\}^{s(m,k)} \to \{0,1\}\right\}_{m,k\in\mathbb{N}}$ for some well-behaved resource bound s that is bounded by a polynomial (respectively, in m or in mk).

It is an elementary fact that a family of the form $\{f_m\}$ or $\{f_{m,k}\}$ as above can be efficiently viewed as a language of the form $\{L_n\}$ as above, and vice versa. For concreteness, let $m \diamond k$ denote the Cantor pairing of m and k. Then given $\{f_{m,k}\}$, define $\{L_n\}$ as $L_n(x) := f_{m,k}(x_{1..s(m,k)})$ for the largest $m \diamond k$ such that $s(m \diamond k, m \diamond k) \leqslant n$. Conversely, given $\{L_n\}$, define $\{f_{m,k}\}$ as $f_{m,k}(x) := L_n(x0^p)$, where p is set so that the input to L is of length exactly $n = s(m \diamond k, m \diamond k)$.

Representing $\mathbb{F}_{2^k}$. We represent each element of $\mathbb{F}_{2^k}$ by a k-bit Boolean string, forming the coefficients of a polynomial in the ring $\mathbb{F}_2[x]$ mod some irreducible $p_k(x)$ of degree k. We fix a uniform collection $\{p_k\}_{k\in\mathbb{N}}$ of such irreducibles, i.e., there is a function in FP that outputs p_k given k in unary [49].

The **Boolean version** of a function $q : \mathbb{F}_{2^k}^m \to \mathbb{F}_{2^k}$ is, for concreteness, the function $\mathrm{bool}(q)$ mapping (x, y) to the y^{th} bit of $q(x)$. (Our results do not depend on this definition; any other equivalent function under Cook reductions would work.)

Affine extensions / oracles. (This definition uses all the definitions above.)

Given $f_m : \{0,1\}^m \to \{0,1\}$, we define its *affine extension polynomial* as the unique m-variate polynomial over $\mathbb{F}_2$, with individual degree $\leqslant 1$, that agrees with f_m over $\mathbb{F}_{2^k}$ for all k, i.e., as

$$\widehat{f}_m(x) := \textstyle\sum_{b\in\{0,1\}^m} f_m(b) \cdot \prod_{i=1}^m (1 + x_i + b_i)$$

By the *affine extension* of $f_m : \{0,1\}^m \to \{0,1\}$, we mean the family

$$\widetilde{f}_m := \left\{\widetilde{f}_m^k\right\}_{k\in\mathbb{N}}$$

where $\widehat{f}_m^k$ denotes the function that evaluates $\widehat{f}_m$ over $\mathbb{F}_{2^k}$, and $\widetilde{f}_m^k$ denotes the Boolean version of $\widehat{f}_m^k$.

Given a family $f := \{f_m\}$ we define its affine extension $\widetilde{f}$ (or its affine extension polynomial $\widehat{f}$) as the family obtained by applying the above definitions to each member. In particular, for the language

$$\mathcal{O} = \{\mathcal{O}_m : \{0,1\}^m \to \{0,1\}\}_{m\in\mathbb{N}}$$

its affine extension $\widetilde{O}$, which we denote here by $\mathcal{A}$, is

$$\mathcal{A} := \left\{\mathcal{A}_{m,k} : \{0,1\}^{mk+\lceil \log k\rceil} \to \{0,1\}\right\}_{k,m\in\mathbb{N}}$$
$$\mathcal{A}_{m,k} : (y_1..y_m z) \mapsto z^{\text{th}} \text{ bit of } \widehat{\mathcal{O}}_m(y_1,..,y_m)$$

where each y_i is interpreted as a member of $\mathbb{F}_{2^k}$. (By the previous definitions, $\mathcal{A}$ can be efficiently viewed as a family of the form $\{\mathcal{A}_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$, and vice versa.)

By an *affine oracle*, we mean the affine extension of a language.

Boolean bases. Recall from Chapter 3 that the *standard Boolean basis* is $B_{\text{std}} = \{\mathbf{0}, \mathbf{1}, \oplus, \wedge\}$. More generally, we define a Boolean basis inductively, as either the standard Boolean basis, or the set $B \cup \{f\}$ where B is a Boolean basis and f is a language. We refer to $B \cup \{f\}$ as the basis B *extended with* f, and when B is the standard basis, as the f-*extended basis*.

For representing circuits (hence formulas) over extended bases $B \supsetneq B_{\text{std}}$, we assume a generic labeling of gates — using labels such as 'the i^{th} nonstandard element' — so that a given circuit can be interpreted over different bases.

The language $\oplus\text{SAT}^f$. We extend the definition of $\oplus$SAT given in Chapter 3.

For every Boolean basis B and language f, define $\oplus\text{SAT}^f$ as the language mapping $\phi(\vec{x})$ to the evaluation of the mod-2 sum $\oplus_{\vec{\alpha}}\phi(\vec{\alpha})$, where ϕ denotes a formula over the basis B extended with f. By default B is the standard basis and f is an element of the standard basis, say the all-zeroes language $\mathbf{0}$. (So the default case is $\oplus$SAT from Chapter 3.)

We index $\oplus$SAT by n, any upper bound on the number of nodes of the formula ϕ. That is, we view $\oplus$SAT as $\{\oplus\text{SAT}_n\}_{n\in\mathbb{N}}$, where $\oplus\text{SAT}_n$ is defined on length-$s(n)$ strings for some fixed $s(n) \in \text{poly}(n)$, with each such string representing a formula ϕ of at most n nodes.

Since $(\oplus\text{SAT}^f)^g$ is equivalent to $(\oplus\text{SAT}^g)^f$ under Karp reductions, we write $\oplus\text{SAT}^{f,g}$ to mean either.

Relativization. We say that a statement ψ *holds relative to* the language $\mathcal{O}$, iff ψ can be proven when two things are done: (a) the standard Boolean basis is extended with $\mathcal{O}$, and (b) FP is redefined as $\mathrm{FP}^{\mathcal{O}}$. (See Section 3 for the definition of $\mathrm{FP}^{\mathcal{O}}$.)

Cook-Levin Theorem. For every function $F \in \mathrm{FP}$, there is a function $\mathrm{Desc}_F \in \mathrm{FP}$ satisfying the following. For every $n \in \mathbb{N}$, $\mathrm{Desc}_F(1^n)$ outputs a circuit C_{F_n} such that for every $x \in \{0,1\}^n$, $F(x) = C_{F_n}(x)$. This holds relative to every oracle.

We refer to this fact as the *Cook-Levin theorem*.

7.2 Checkability of $\oplus$SAT

The main result of this section is the following.

Theorem 7.1. $\oplus$SAT *is checkable. This holds relative to every affine oracle.*

Proving Theorem 7.1 requires a fair amount of machinery to be developed. We start by defining an arithmetic analogue of $\oplus$SAT.

Definition 7.1 (Arithmetic basis)**.** *For every Boolean basis* B*, define the* arithmetic basis $\widehat{B}$ *as the set comprising all constants in* $\mathbb{F}_{2^k}$ *for each* k*, and* $\widehat{f}$ *for each* $f \in B$*. By the standard arithmetic basis we mean* $\widehat{B}_{\mathrm{std}}$ *where* B_{std} *is the standard Boolean basis.*

Definition 7.2 ($+\mathrm{ASAT}^f$)**.** *For every Boolean basis* B *and language* f*, define* $+\mathrm{ASAT}^f$ *as the Boolean version (as defined in Section 7.1) of the map* $\Psi(\vec{x}) \mapsto \Sigma_{\vec{\alpha}}\Psi(\vec{\alpha})$ *where* Ψ *denotes a formula over the arithmetic basis corresponding to* $B \cup \{f\}$*. By default* B *is the standard basis and* f *is an element of the standard basis, say the all-zeroes language* $\mathbf{0}$*.*

We index $+\mathrm{ASAT}$ *by* n *and* k*, and write the corresponding member as* $+_k\mathrm{ASAT}_n$*; here* n *upper bounds the number of nodes in formula* Φ*, and* k *denotes the field* $\mathbb{F}_{2^k}$ *where the constants of* Φ *reside.*

For our purposes (to become clear in the proof of Lemma 7.6) we require $k \geqslant \log^2 n$.

Five lemmas regarding $\oplus$SAT and $+$ASAT are used to prove Theorem 7.1:

Lemma 7.2. $\oplus\mathrm{SAT}^f \to \oplus\mathrm{SAT}^g$ *for every language* f *and* g *such that* f *Cook-reduces to* g. *The reduction works over any Boolean basis for formulas.*

Lemma 7.3. $\oplus\mathrm{SAT}^{\tilde{f}} \to \oplus\mathrm{SAT}^f$ *for every language* f. *The reduction works over any Boolean basis for formulas.*

Lemma 7.4. $\oplus\mathrm{SAT} \to +\mathrm{ASAT}$. *The reduction works over any Boolean basis for formulas and its corresponding arithmetic basis.*

Lemma 7.5. $+\mathrm{ASAT} \to \oplus\mathrm{SAT}$. *The reduction works over any Boolean basis for formulas and its corresponding arithmetic basis.*

Lemma 7.6. $+\mathrm{ASAT}$ *is checkable. In fact, for every language* $\mathcal{O}$, $+\mathrm{ASAT}^{\mathcal{O}}$ *is checkable with oracle access (as defined in Section 3) to* $\widetilde{\mathcal{O}}$.

We defer the proof of these lemmas to after the proof of Theorem 7.1.

Proof of Theorem 7.1. Let $\mathcal{O}$ be an arbitrary language and $\mathcal{A}$ its affine extension. We want to show that $\oplus\mathrm{SAT}^{\mathcal{A}}$ is checkable with oracle access to $\mathcal{A}$. (To recall what it means for a language to be checkable with oracle access, see Section 3.)

To begin with, we claim that

$$\oplus\mathrm{SAT}^{\mathcal{A}} \to +\mathrm{ASAT}^{\mathcal{O}} \tag{$\dagger$}$$

and that

$$+\mathrm{ASAT}^{\mathcal{O}} \to \oplus\mathrm{SAT}^{\mathcal{A}}. \tag{$\ddagger$}$$

To see the first claim, use Lemma 7.3 to get $\oplus\mathrm{SAT}^{\mathcal{A}} \to \oplus\mathrm{SAT}^{\mathcal{O}}$, and then use Lemma 7.4 with the $\mathcal{O}$-extended Boolean basis to get $\oplus\mathrm{SAT}^{\mathcal{O}} \to +\mathrm{ASAT}^{\mathcal{O}}$.

For the second claim, use Lemma 7.5 with the $\mathcal{O}$-extended Boolean basis to get $+\mathrm{ASAT}^{\mathcal{O}} \to \oplus\mathrm{SAT}^{\mathcal{O}}$, and then use Lemma 7.2, together with the fact that $\mathcal{O}$ Karp-reduces, hence Cook-reduces, to $\mathcal{A}$, to get $\oplus\mathrm{SAT}^{\mathcal{O}} \to \oplus\mathrm{SAT}^{\mathcal{A}}$.

Now, by Lemma 7.6, we know that $+\mathrm{ASAT}^{\mathcal{O}}$ is checkable with oracle access to $\mathcal{A}$. But then $\oplus\mathrm{SAT}^{\mathcal{A}}$ is also checkable with access to $\mathcal{A}$: On input x, use the reduction (†) to get an input x' for $+\mathrm{ASAT}^{\mathcal{O}}$, and then simulate the checking protocol for $+\mathrm{ASAT}^{\mathcal{O}}$, by using the reduction (‡) to translate each query for $+\mathrm{ASAT}^{\mathcal{O}}$ to one for $\oplus\mathrm{SAT}^{\mathcal{A}}$.

(Theorem 7.1, mod Lemmas 7.2-7.6)

□

In the rest of this section we prove Lemmas 7.2-7.6. To do so, first we extend $\oplus$SAT and $+$ASAT to expressions involving summations *within* the formula (not just in front). We call these extensions $\oplus^*$SAT and $+^*$ASAT, respectively. After that, we derive certain facts relating $\oplus^*$SAT, $\oplus$SAT, $+^*$ASAT, and $+$ASAT. Finally, we put these together to prove Lemmas 7.2-7.6.

7.2.1 Extending $\oplus$SAT to $\oplus^*$SAT and $+$ASAT to $+^*$ASAT

We give four definitions, two for extending $\oplus$SAT and two for $+$ASAT.

Definition 7.3 (bbs)**.** *For every Boolean basis* B*, consider the set of expressions obtained inductively, by letting in: (i) every variable; (ii)* $f(\psi_1..\psi_m)$ *for every* $\psi_1,..,\psi_m$ *already let in, for every element* f *in the basis, for every* $m \in \mathbb{N}$*; (iii)* $\oplus_y \psi$ *for every* ψ *already let in, for every free variable* y *of* ψ*. Call this the set of* Boolean expressions involving binary sums *(bbs) over the basis* B*.*

Definition 7.4 ($\oplus^*\mathrm{SAT}^{\mathsf{f}}$)**.** *Define* $\oplus^*\mathrm{SAT}^{\mathsf{f}}$ *over the basis* B *as the map* $\psi(\vec{x}) \mapsto \oplus_{\vec{\alpha}}\psi(\vec{\alpha})$ *where* ψ *is a bbs over* $\mathsf{B} \cup \{\mathsf{f}\}$*, with input variables* $\vec{x}$*. By default* B *is the standard basis and* f *is the all-zeroes language* $\mathbf{0}$*.*

Definition 7.5 (abs)**.** *For every arithmetic basis* A*, consider the set of expressions obtained inductively, by letting in: (i) every variable; (ii)* $\mathsf{f}(\Psi_1..\Psi_m)$ *for every* $\Psi_1,..,\Psi_m$ *already let in, for every element in* A*, for every* $m \in \mathbb{N}$*; (iii)* $\sum_{y\in\{0,1\}}\Psi$ *for every* Ψ *already let in, for every free variable* y *of* Ψ*. Call this the set of* arithmetic expressions involving binary sums *(abs) over the basis* A*.*

Definition 7.6 ($+^*\mathrm{ASAT}^{\mathsf{f}}$)**.** *Define* $+^*\mathrm{ASAT}^{\mathsf{f}}$ *over the basis* A *as the Boolean version (as defined in Section 3) of the map* $\Psi(\vec{x}) \mapsto \Sigma_{\vec{\alpha}}\Psi(\vec{\alpha})$*, where* Ψ *is an abs over* $\mathsf{A} \cup \{\widehat{\mathsf{f}}\}$ *with input variables* $\vec{x}$*, and each* α_i *ranges over* $\{0,1\}$*. By default,* A *is the standard arithmetic basis and* f *is the all-zeroes language* $\mathbf{0}$*.*

7.2.2 Relating $\oplus$SAT, $\oplus^*$SAT, $+$ASAT and $+^*$ASAT

We show six relationships in this subsection:

Lemma 7.7. $\oplus\mathrm{SAT}^{\mathsf{f}} \to \oplus^*\mathrm{SAT}^{\mathsf{g}}$ *for every language* f *and* g *such that* f *Cook-reduces to* g*. The reduction works over any Boolean basis for formulas.*

Lemma 7.8. $\oplus\mathrm{SAT}^{\oplus\mathrm{SAT}} \to \oplus^*\mathrm{SAT}$*. The reduction works over any Boolean basis for formulas.*

Lemma 7.9. $\oplus^*\mathrm{SAT} \to +^*\mathrm{ASAT}$*. The reduction works over any Boolean basis for formulas and its corresponding arithmetic basis.*

Lemma 7.10. $+^*\mathrm{ASAT} \to +\mathrm{ASAT}$*. The reduction works over any arithmetic basis for formulas.*

Lemma 7.11. $+\text{ASAT} \to \oplus\text{SAT}$. *The reduction works over any Boolean basis for formulas and its corresponding arithmetic basis.*

Corollary 7.12. $\oplus^*\text{SAT} \to \oplus\text{SAT}$. *The reduction works over any Boolean basis for formulas.*

We prove each of these in turn. Before we begin, we derive an auxiliary fact that will be useful in proving Lemma 7.7 and Lemma 7.8.

Lemma 7.13. *For every function* $R \in \text{FP}$ *there is a function* $R' \in \text{FP}$ *such that the following holds. If* R *is a Cook reduction from some language* f *to some language* g*, then for every* n*,* $R'(1^n)$ *gives a formula* $\xi(x, y)$ *over the* g*-extended Boolean basis such that*

$$f(x) = \oplus_y \xi(x, y)$$

for every $x \in \{0,1\}^n$*.*

This holds relative to every language.

Proof. Let $\mathcal{O}$ be an arbitrary language, and let $R \in \text{FP}^{\mathcal{O}}$ be a Cook reduction from some language f to some language g. That is, there is some $\ell \in \text{poly}(n)$ such that for every x,

$$f(x) = R(x, z), \quad \text{where } z_i = g(R(x, z_1..z_{i-1})) \text{ and } |z| = \ell(|x|).$$

By the Cook-Levin theorem (Section 7.1) applied to R, it follows that there is $\ell \in \text{poly}(n)$, and there are circuits $C_0, .., C_{\ell(n)}$, each over the $\mathcal{O}$-extended Boolean basis, such that for every x,

$$f(x) = C_{\ell(n)}(x, z), \quad \text{where } z_i = g(C'_{i-1}(x, z_1..z_{i-1})) \text{ and } |z| = \ell(|x|),$$

and where

$$C'_i(y) := (C_i(y))_{1..|R(y)|},$$

the idea being to calculate the output length of $R(y)$ and trim the excess from the output of $C_i(y)$ before making any use of it. Notice that the function that calculates the output length of R is in $FP^{\mathcal{O}}$, because R is. Hence by the Cook-Levin theorem again, each C'_i can be implemented by a circuit; moreover, each such circuit can be produced by a function in $FP^{\mathcal{O}}$ given 1^n.

It follows that the restriction of f to $\{0,1\}^n$ satisfies

$$f(x) = \bigoplus_{z \in \{0,1\}^\ell} C_\ell(x,z) \wedge \bigwedge_{i=1..\ell} (z_i \equiv g(C'_{i-1}(x, z_{1..i-1}))$$

where ℓ denotes $\ell(|x|)$; further, the righthand side is produced by some function in $FP^{\mathcal{O}}$ given input 1^n.

The expression inside the sum is a circuit $E(x,z)$ over the g- and $\mathcal{O}$-extended basis, and can be equivalently written as the sum $\oplus_\nu \xi(x,z,\nu)$ where $\xi(a,\nu)$ checks that ν describes the computation of $E(a)$, and ν ranges over $\{0,1\}^s$ for some $s \in \text{poly}(\text{size of } E) \subset \text{poly}(n)$.

It follows that there is a function in $R' \in FP^{\mathcal{O}}$ that, given input 1^n, outputs a formula ξ over the g- and $\mathcal{O}$-extended basis satisfying

$$f(x) = \bigoplus_{z,\nu} \xi(x,z,\nu)$$

for every $x \in \{0,1\}^n$. This was to be shown. □

Proof of Lemma 7.7. Suppose f Cook-reduces to g. By Lemma 7.13, there is a function in FP that, given a formula ϕ over the basis $B \cup \{f\}$, where B is any Boolean basis, takes each subformula of the form $f(\phi_1..\phi_n)$ and performs the

replacement

$$f(\phi_1..\phi_n) \mapsto \bigoplus_y \xi(\phi_1..\phi_n, y)$$

where ξ is a formula over the basis $B \cup \{g\}$. This proves $\oplus SAT^f \to \oplus^* SAT^g$ over B, for every basis B. □

Proof of Lemma 7.8. Let B be a Boolean basis for formulas. Given a formula $\phi(x)$ over the basis $B \cup \{\oplus SAT^B\}$, we want a reduction from the task of computing $\oplus_x \phi(x)$ to that of computing $\oplus_z \psi(z)$, for some bbs $\psi(z)$ over B. We want the reduction to work for every choice of B.

Intuitively, replacing each occurrence of $\oplus SAT^B$ in $\phi(x)$ with the actual sum to be computed, would constitute a reduction as desired. More precisely, let FormulaEval^B be the language that, on input (t, u), interprets t as a formula τ over the basis B, and outputs $\tau(u)$, the evaluation of τ on u. (In case τ has fewer inputs than $|u|$, let FormulaEval^B output $\tau(u)$ only if the extra bits in u are set to zero, else let it output zero.)

Each subformula in $\phi(x)$ of the form

$$\oplus SAT^B(\phi_1..\phi_m) \tag{7.2}$$

can be viewed as the sum

$$\oplus_{u \in \{0,1\}^m} \text{FormulaEval}^B(\phi_1..\phi_m, u) \tag{7.3}$$

for each setting of x, since the subformulas $\phi_1(x), .., \phi_m(x)$ collectively describe a Boolean formula τ_x with $\leqslant m$ input variables.

Now, FormulaEval^B Cook-reduces to the basis B, more precisely, to the function

$$\text{IIB} : (i, x) \mapsto B_i(x)$$

where B_i is the ith element of the basis B. Notice that this reduction does not depend on what the basis B is, provided we have a reasonable representation of formulas that uses generic labels for gates — 'the i^{th} nonstandard element' etc. — which is the case by the way we set things up in Section 7.1 (subsection "Boolean bases").

It follows by Lemma 7.13 that there is a function in FP that, given input 1^m, outputs a formula $\xi^{\amalg B}$ over the basis $\amalg B$ satisfying

$$\text{FormulaEval}^B(a) = \bigoplus_y \xi^{\amalg B}(a, y) \tag{7.4}$$

for every input $a \in \{0,1\}^m$. If the basis B contains d elements, then $\amalg B$ can be written as

$$\amalg B(i, x) = ((i \equiv 1) \wedge B_1(x)) \oplus \cdots \oplus ((i \equiv d) \wedge B_d(x)) \tag{7.5}$$

where '$i \equiv j$' is shorthand for the formula checks that i is the binary encoding of the number j. The righthand side of (7.5) is a formula over the basis B. Combining with (7.4), we get a function in FP that, given input 1^m, outputs a formula ξ^B over the basis B satisfying

$$\text{FormulaEval}^B(a) = \bigoplus_y \xi^B(a, y) \tag{7.6}$$

for every input $a \in \{0,1\}^m$.

It follows, from (7.3) and (7.6), that there is a function in FP that takes each subformula of the form (7.2), and performs the replacement

$$\oplus\text{SAT}^B(\phi_1..\phi_m) \mapsto \bigoplus_{u,y} \xi^B(\phi_1..\phi_m, u, y)$$

proving $\oplus\text{SAT}^{\oplus\text{SAT}} \to \oplus^*\text{SAT}$. The reduction works over any choice of a Boolean

basis B. □

Proof of Lemma 7.9. Given a bbs ϕ over any Boolean basis B, let Φ be its "arithmetization", obtained by replacing each non-input gate f in ϕ with its affine extension polynomial $\widehat{f}$, and by replacing each mod-2 sum with a generic sum so that a subexpression of ϕ of the form $\oplus_{y\in\{0,1\}}\phi'$ becomes $\Sigma_{y\in\{0,1\}}\Phi'$.

Because $\widehat{f}$ agrees with f on Boolean settings of its inputs by definition (Section 7.1), it follows that ϕ agrees with Φ on every Boolean input. And because we represent $\mathbb{F}_{2^k}$ as k-bit vectors (Section 7.1), computing $\oplus_{\vec{\alpha}}\phi(\vec{\alpha})$ reduces to computing the least significant bit of $\Sigma_{\vec{\alpha}}\Phi(\vec{\alpha})$ over $\mathbb{F}_{2^k}$ for any k, where each α_i ranges over $\{0,1\}$ in both sums. The reduction works over any choice of the basis B and its corresponding arithmetic basis. □

Proof of Lemma 7.10. Given an abs Ψ over any arithmetic basis A, we give a reduction that produces a (summation-free) formula Φ over A satisfying

$$\Psi(x) = \Sigma_y \Phi(x, y)$$

for every setting of inputs x of Ψ over $\mathbb{F}_{2^k}$, for every k. Here y ranges over $\{0,1\}^m$ for some m depending on Ψ.

There is nothing to do if Ψ is just a variable or constant, so suppose not.

If $\Psi(x)$ is of the form $\Psi_1 \cdot \Psi_2$, and if by recursion Ψ_1 is already brought to the desired form $\Sigma_y \Phi_1(x, y)$, and Ψ_2 to $\Sigma_z \Phi_2(x, z)$, then the rest is easy: just make sure y and z refer to disjoint sets of variables by renaming as needed, and write $\Psi(x) = \Sigma_{y,z} \Phi_1(x, y) \cdot \Phi_2(x, z)$.

In case $\Psi = \Psi_1 + \Psi_2$, after recursing and renaming as before, write

$$\Psi(x) = \sum_{b,y,z} \big(\Phi_1(x, y) \cdot b \cdot \Pi_i z_i \;+\; \Phi_2(x, z) \cdot (1 - b) \cdot \Pi_i y_i\big), \tag{7.7}$$

where b is a single variable.

In case Ψ is of the form $\widehat{f}(\Psi_1,..,\Psi_m)$, where f is a nonstandard basis element, use the definition of $\widehat{f}_m$ (Section 7.1) to rewrite Ψ as

$$\Psi(x) = \sum_{b_1..b_m} \widehat{f}(b_1,..,b_m) \cdot \prod_{i=1..m}(1 + \Psi_i(x) + b_i), \tag{7.8}$$

then recurse into the product on the right side, and then finish by going to the first case, $\Psi = \Psi_1 \cdot \Psi_2$.

The reduction works over any choice of an arithmetic basis A for formulas. If Ψ is of size s, then the resulting formula Φ is of size $O(s^2)$. To see this, first pretend there are no $\widehat{f}$-gates in Ψ; then the reduction introduces at most s additional variables, so that if $\Psi = \Psi_1 + \Psi_2$, with Ψ_1 of size s_1 and Ψ_2 of size s_2, and if $t(s)$ is the size of the resulting formula Φ, then by (7.7),

$$t(s) \leqslant t(s_1) + t(s_2) + 2(s_1 + s_2) + k$$

for some constant k. Now allow $\widehat{f}$-gates back in Ψ, and use (7.8) to transform every $\widehat{f}$-gate so that it becomes "isolated", in the sense that its inputs become variables as opposed to subformulas. This transformation increases the size of Ψ from s to some function in $O(s)$, and the claim follows. □

Proof of Lemma 7.11. Given an arithmetic formula $\Phi(x)$ and given ℓ, we give a reduction from finding the ℓ^{th} bit of $\Sigma_x \Phi(x)$, where each x_i ranges over $\{0,1\}$, to evaluating the mod-2 sum $\oplus_z \phi(z)$ for some Boolean formula ϕ.

To begin with, let us assume that there are no nonstandard $\widehat{f}$-gates in Φ, in other words, that Φ is a $\mathbb{F}_{2^k}$-polynomial for some k. By the way we represent $\mathbb{F}_{2^k}$ (Section 7.1), there is a Boolean circuit $C(X)$ that takes as input a k-bit vector X_j corresponding to each input x_j of $\Phi(x)$, and outputs k bits representing the

value $\Phi(x)$. C can be produced by an FP-function given Φ.

Because the original task is to find the ℓ^{th} bit of the sum $\Sigma_x \Phi(x)$, and because addition in $\mathbb{F}_{2^k}$ corresponds to componentwise addition in $\mathbb{F}_2^k$, we can ignore all output bits of C except the ℓ^{th} one. Further, because the summation variables x_i range over binary values, we can fix in each X_i all the bits to 0 except the least significant bit, which we can call x_i. So we now have a circuit $C(x)$ returning the ℓ^{th} bit of $\Phi(x)$ for every x from the Boolean domain.

It follows that the ℓ^{th} bit $\Sigma_x \Phi(x)$ equals $\oplus_{x,y} \phi(x, y)$, where ϕ is the formula verifying that y describes the computation of the circuit C on input x. This proves the lemma when $\Phi(x)$ is a polynomial.

Now suppose that Φ contains $\widehat{f}$-gates for an arbitrary f. Mimicking the above reasoning for the standard basis, we want to express the evaluation of Φ as a Boolean circuit C over the f-extended Boolean basis. Once this is done, the rest follows as in the earlier case with no $\widehat{f}$-gates.

Perform the process, explained in the proof of Lemma 7.10 just above, of bringing Φ to prenex form — a seemingly useless thing to do as Φ does not involve sums. But notice from (7.8) that as a side effect, the process transforms the summation-free $\Phi(x)$ into the sum $\Sigma_B \Phi'(x, B)$, where each $\widehat{f}$-gate in Φ', say the i^{th} one, is "isolated" in the sense that its inputs now come from some $B_{i1}, .., B_{im_i}$ among the variables B, which all range over Boolean values. Since $\widehat{f}$ agrees with f on Boolean inputs, now the $\widehat{f}$-gates can be replaced with f-gates.

It thus follows, with the same reasoning as earlier, that the ℓ^{th} bit of $\Sigma_x \Phi(x)$ — which is the same as the ℓ^{th} bit of $\Sigma_{x,B} \Phi'(x, B)$ — equals $\oplus_{x,B,y} \phi'(x, B, y)$, where ϕ' is a formula over the Boolean basis corresponding to the basis of Φ. ϕ' can be produced by an FP-function given Φ; this function works over any choice of a basis for Φ. □

Proof of Corollary 7.12. Immediate from Lemmas 7.9, 7.10, and 7.11. □

7.2.3 Proof of Lemmas 7.2-7.5

We are ready to derive the first four of the five lemmas used in proving Theorem 7.1.

Proof of Lemma 7.2. Immediate by combining Lemma 7.7 and Corollary 7.12. □

Proof of Lemma 7.3. Being the affine extension of f, by the definitions in Section 7.1, on input x, $\widetilde{f}$ gives the z^{th} bit of the value $\widehat{f}$ takes at y, where y and z are computable in polynomial-time out of x. In other words, $\widetilde{f}$ gives the $+\text{ASAT}^f$ instance (Φ, z) where Φ is the formula '$\widehat{f}(y)$'. Thus $\widetilde{f} \to +\text{ASAT}^f$. Combining with Lemma 7.5 gives $\widetilde{f} \to \oplus\text{SAT}^f$. Therefore,

$$\oplus\text{SAT}^{\widetilde{f}} \to \oplus\text{SAT}^{\oplus\text{SAT}^f} \to \oplus^*\text{SAT}^f \to \oplus\text{SAT}^f$$

by Lemma 7.2, Lemma 7.8, and Corollary 7.12, respectively. The composite reduction $\oplus\text{SAT}^{\widetilde{f}} \to \oplus\text{SAT}^f$ works over any choice of a Boolean basis since each constituent reduction does. □

Proof of Lemma 7.4. Immediate by combining Lemma 7.9 and Lemma 7.10. □

Proof of Lemma 7.5. This is just Lemma 7.11. □

7.2.4 Proof of Lemma 7.6

We finish the proof of Theorem 7.1 by proving Lemma 7.6.

Proof of Lemma 7.6. Let $\mathcal{O}$ be an arbitrary language. We want to show an interactive protocol for $+\text{ASAT}^{\mathcal{O}}$ in which the verifier has access to $\widetilde{\mathcal{O}}$ (Section 3, "Relativized classes"), and the power of the honest prover reduces to $+\text{ASAT}^{\mathcal{O}}$ itself (Section 3, "Interactive proofs"). The verifier in this protocol, V, is given

(Φ, ℓ, b), and must check that the ℓ^{th} bit of $\Sigma_x \Phi(x)$ equals b. Here $x_i \in \{0,1\}$ for each i, and Φ is over the $\widehat{\mathcal{O}}$-extended arithmetic basis and has all its constants in $\mathbb{F}_{2^k}$ for some k; hence the sum is over $\mathbb{F}_{2^k}$. V works as follows:

First, it obtains the claimed values for the rest of the k bits of $\Sigma_x \Phi(x)$, so that the claim becomes '$\Sigma_x \Phi(x) = u$' for some $u \in \mathbb{F}_{2^k}$.

Second, it performs the sumcheck protocol [12, Section 3.2] over $\mathbb{F}_{2^k}$ to get rid of the sum and update the claim to '$\Phi(y) = v$' for some y, v over the *same* field as that of x, u.

At this point, V obtains the value of each gate in the evaluation of $\Phi(y)$ — i.e., the value of each subformula of Φ, when evaluating Φ on y — and checks all of them.

The analysis of the protocol is standard: if the original claim, that the ℓ^{th} bit of $\Sigma_x \Phi(x)$ equals b, is false, where Φ has $\leqslant n$ nodes, then the sumcheck erroneously yields a true claim with probability at most

$$\#\text{ of rounds} \cdot \deg \Phi \;/\; \text{size of the field}$$

which grows slower than $1/n^d$ for any d, due to the requirement $k \geqslant \log^2 n$ in the definition of +ASAT (Definition 7.2). This finishes the proof. □

7.3 Proof of Theorem 1.7-(i)

In this section we prove the first part of Theorem 1.7, which we recall here in a slightly stronger form:

Theorem 1.7-(i), Stronger form. MAEXP *contains a language of circuit complexity in* $n^{\omega(1)}$. *This holds relative to every affine oracle.*

This is a stronger statement than earlier (p. 67) because MAEXP, by our definition (Section 3), contains partial languages as well as (total) languages.

The proof uses two main ingredients. First is a result of Kannan [39] akin to Theorem 1.3.

Fact 7.14 ([39])**.** $\mathrm{EXP}^{\Sigma_2\mathrm{P}}$ *contains a language of circuit complexity in* $2^{\Omega(n)}$*. This holds relative to every oracle.*

Although Kannan gave a different proof of it, Fact 7.14 can be derived similarly to the proof of Theorem 1.3 (p. 39). Recall the idea there was to do "fast diagonalization by approximate counting", by mimicking the process that successively sets the next bit of a truth table to the minority vote of the circuits that are consistent with the table constructed thus far. Whereas this ideal process would eliminate at least half of the circuits at each step, the mimicking process eliminates at least an $(1/2 - \varepsilon)$-fraction for an arbitrarily small constant $\varepsilon > 0$, since the minority count can be estimated to within any constant factor, via a function in AM. Now, since every function in AM can be extended to a language in $\Pi_2\mathrm{P}$ [14], and since $\mathrm{EXP}^{\Sigma_2\mathrm{P}} = \mathrm{EXP}^{\Pi_2\mathrm{P}}$, the first part of Fact 7.14 follows.

To see why this argument relativizes, notice that it makes no particular assumption regarding what basis the circuits are on, as long as they can be evaluated. So if we extend the standard Boolean basis by an arbitrary language $\mathcal{O}$, then the same argument carries through, provided we replace FP by $\mathrm{FP}^{\mathcal{O}}$, and hence AM by $\mathrm{AM}^{\mathcal{O}}$, etc. So the second part of Fact 7.14 also follows.

The second ingredient in the proof of Theorem 1.7-(i) says that if the language $\oplus$SAT has small circuits, then the exponential-time hierarchy collapses to MAEXP.

Lemma 7.15. *If* $\oplus$SAT *has circuits of polynomial size, then* $\mathrm{EXP}^{\Sigma_2\mathrm{P}} \subset \mathrm{MAEXP}$*. This holds relative to every affine oracle.*

We prove Lemma 7.15 after proving Theorem 1.7-(i):

Proof of Theorem 1.7-(i). Let $\mathcal{A}$ be an affine oracle. If $\oplus\mathrm{SAT}^{\mathcal{A}}$ does not have polynomial size circuits, then neither does $\mathrm{MAEXP}^{\mathcal{A}}$ since $\oplus\mathrm{SAT}^{\mathcal{A}} \in \mathrm{EXP}^{\mathcal{A}} \subset \mathrm{MAEXP}^{\mathcal{A}}$. The theorem now follows from Fact 7.14 and Lemma 7.15.

(Theorem 1.7-(i), mod Lemma 7.15) □

We now turn to Lemma 7.15. The proof will be very similar to that of Lemma 6.2 which, recall, says that under a derandomization assumption, if $\oplus$SAT has small *nondeterministic* circuits, then the exponential-time hierarchy collapses to the *second* level. Here we want to deepen that collapse to MAEXP, under the stronger assumption that $\oplus$SAT has deterministic circuits. (While we don't have a derandomization assumption here, the task is still doable because the class to which we want the collapse to occur involves randomness by itself.)

Just as was the case for Lemma 6.2, we prove a stronger version of Lemma 7.15:

Lemma 7.16. *If* $\oplus$SAT *has circuits of polynomial size, then* $\mathrm{P}^{\Sigma_2\mathrm{P}} \subset \mathrm{MA}$. *This holds relative to every affine oracle.*

Lemma 7.15 follows from Lemma 7.16 by a padding argument identical to that in the proof of Lemma 6.2. So all that remains is the proof of Lemma 7.16.

Proof of Lemma 7.16. The proof is already implicit in the proof of Lemma 6.4. Let $\mathcal{A}$ be an affine oracle. Extend the standard Boolean basis with $\mathcal{A}$, and use FP for $\mathrm{FP}^{\mathcal{A}}$, MA for $\mathrm{MA}^{\mathcal{A}}$, etc.

Suppose that $\oplus$SAT has circuits of polynomial size. Then there is an MA-protocol for Σ_3SAT that proceeds as follows.

Given input $\varphi \in \{0,1\}^n$ to Σ_3SAT, Arthur performs the randomized reduction from Σ_3SAT to $\oplus$SAT (Fact 6.7), thereby obtaining an input $\psi \in \{0,1\}^m$ to $\oplus$SAT

such that

$$\Pr[\Sigma_3\mathrm{SAT}(\varphi) = \oplus\mathrm{SAT}(\psi)] \geqslant 1 - \varepsilon \qquad (\dagger)$$

for some $m \in \mathrm{poly}(n)$ and some negligible $\varepsilon \in (1/n)^{\omega(1)}$. After this, Merlin sends Arthur a circuit for $\oplus$SAT on ℓ-bit inputs, for a sufficiently large $\ell \in \mathrm{poly}(m) \subset \mathrm{poly}(n)$. Arthur then uses that circuit as the prover in an interactive protocol for the $\oplus$SAT instance ψ, which he can do because $\oplus$SAT is checkable (Theorem 7.1).

If $\Sigma_3\mathrm{SAT}(\varphi) = 1$, then by the definition of an interactive protocol, and by (†), Merlin can send the actual circuit for $\oplus$SAT and make Arthur accept with probability greater than $(2/3 - \varepsilon)$, where $\varepsilon \in (1/n)^{\omega(1)}$.

If $\Sigma_3\mathrm{SAT}(\varphi) = 0$, then again by the definition of an interactive protocol, and by (†), no matter what Merlin sends, the probability that Arthur accepts is less than $(1/3 + \varepsilon)$, where $\varepsilon \in (1/n)^{\omega(1)}$.

By running the checker twice, these probabilities can be polarized to $(8/9 - \varepsilon) \geqslant 2/3$ and $(1/9 + \varepsilon) \leqslant 1/3$ respectively. Therefore $\Sigma_3\mathrm{SAT} \in \mathrm{MA}$, and the proof is complete. □

7.4 Proof of Theorem 1.7-(ii)

In this section we prove the second part of Theorem 1.7, which we recall here:

Theorem 1.7-(ii), Restated. *Relative to some affine oracle,* $\mathrm{NEXP} \subset \mathrm{P/poly}$.

If we wanted to show $\mathrm{NEXP} \subset \mathrm{P/poly}$ relative to *some* oracle $\mathcal{O}$, affine or not, or more generally, to show $\mathcal{C}^{\mathcal{O}} \subset \mathcal{D}^{\mathcal{O}}$ for classes $\mathcal{C}$ and $\mathcal{D}$, then there is a simple approach to this, due to Heller [36]: at iteration $n \in \mathbb{N}$, take the first n algorithms underlying $\mathcal{C}^{\mathcal{O}}$, and partially fix $\mathcal{O}$ so as to *force* the behavior of these algorithms on $\{0,1\}^n$. Assuming $\mathcal{C}$ is not too powerful, this forcing can be done

without having to fix $\mathcal{O}$ on all of $\{0,1\}^{kn}$, for some constant k, even considering prior iterations. The free inputs of $\{0,1\}^{kn}$ on which $\mathcal{O}$ is yet undefined can then be used to store information on how the forced algorithms behave, in such a way that some algorithm in $\mathcal{D}^{\mathcal{O}}$ can retrieve that information.

When it comes the *affine* oracles, however, we face a difficulty in making this strategy work. An affine oracle $\widetilde{\mathcal{O}}$, being the algebraically redundant version (see (2.1)) of the oracle $\mathcal{O}$, is less "dense" in its information content then $\mathcal{O}$. So how do we guarantee that partially fixing $\widetilde{\mathcal{O}}$, as done in the above paragraph, still leaves sufficiently many free inputs on which we can do encoding?

The following two results provide that guarantee. The first one states that knowing t bits of a binary codeword exposes at most t bits of its information word, and the second scales this result to affine extensions.

Lemma 7.17 (Interpolation)**.** *Let $\mathcal{E} : \mathbb{F}_2^K \to \mathbb{F}_2^N$ be linear and injective. Given a "dataword" $u \in \mathbb{F}_2^K$ and a set of indices $A \subseteq [N]$, consider the collection U of all datawords $u' \in \mathbb{F}_2^K$ such that $\mathcal{E}(u)$ and $\mathcal{E}(u')$ agree on A.*

There is a set of indices $B \subseteq [K]$, no larger than A, such that projecting U onto $G := [K] \setminus B$ gives all of $\mathbb{F}_2^G$.

Proof. The claim of the lemma on U is true iff it is true on $U^+ := U + u$. So it suffices to show that U^+ is a subspace of $\mathbb{F}_2^K$ with dimension at least $K - |A|$.

Now, $y \in U^+$ iff $y + u \in U$, which is iff $\mathcal{E}(y + u)$ and $\mathcal{E}(u)$ agree on A, which is iff $\mathcal{E}(y)$ vanishes on A. Therefore U^+ is identical to the space of all datawords whose encodings vanish on A.

All that is left is to bound $\dim U^+$, or equivalently, to bound $\dim \mathcal{E}(U^+)$ since $\mathcal{E}$ is injective. The latter quantity is the dimension of the space $\mathcal{C} \cap \mathcal{Z}$, where $\mathcal{C}$ is the image of $\mathcal{E}$, and $\mathcal{Z}$ is the space of all N-bit vectors that vanish on A. But

then by the theorem on the dimension of a sum of subspaces (e.g. [8, Thm 1.4])

$$\begin{aligned}\dim(U^+) &= \dim(Z) \quad + \dim(C) - \dim(Z + C) \\ &= (N - |A|) + \quad K \quad - \dim(Z + C)\end{aligned}$$

which is at least $K - |A|$ because $Z + C \subseteq \mathbb{F}_2^N$. This finishes the proof. □

Theorem 7.18 (Interpolation)**.** *Given a language* f *and a finite set* A *of inputs, consider the collection* $\mathcal{F}$ *of all languages* g *such that* $\widetilde{f}$ *and* $\widetilde{g}$ *agree on* A*.*

There is a set B *of inputs, no larger than* A*, such that every partial Boolean function* g' *defined outside* B *can be extended to some* $g \in \mathcal{F}$*.*

Further, in extending g' *to* g*, the values of* g *at length-*n *inputs depend only on those of* g' *at length* n*.*

Proof. To begin with, consider the special case where $A \subseteq \mathrm{dom}(\widetilde{f}^k_m)$ for some fixed k and m. For the purpose of invoking Lemma 7.17, let $\mathcal{E}$ be the map that takes as input the truth table of a Boolean function g_m on m bits, and outputs the truth table of $\widetilde{g}^k_m$. So $\mathcal{E} : \mathbb{F}_2^K \to \mathbb{F}_2^N$, where $K = 2^m$ and $N = k2^{km}$ (to see the value of N, recall that $\widetilde{g}^k_m(y, z)$ gives the z^{th} bit of $\widehat{g}^k_m(y)$, where $\widehat{g}^k_m$ is the extension of g_m to $\mathbb{F}_{2^k}^m$).

Clearly $\mathcal{E}$ is injective; it is also linear because $\widehat{g}^k_m$ is additive and because we represent $\mathbb{F}_{2^k}$ with $\mathbb{F}_2^k$ where addition is componentwise (Section 7.1). So $\mathcal{E}$ fulfils the conditions of Lemma 7.17, which yields a set $B \subseteq \{0,1\}^m$ that is no larger than A, such that every partial Boolean function on $\{0,1\}^m \setminus B$ can be extended to a language in $\mathcal{F}$. This proves the theorem in the special case.

To handle the general case, partition A into $A_{m,k} := A \cap \mathrm{dom}(\widetilde{f}^k_m)$, and use the above special case as a building block to create a bigger code. In detail, for every m involved in the partition, define $\mathcal{E}_m$ as the map sending the truth table of g_m to the list comprising the truth tables of $\widetilde{g}^{k_1}_m, \widetilde{g}^{k_2}_m, \ldots$ for every A_{m,k_j} in

the partition. Now, take each $\mathcal{E}_m$ thus obtained, and let $\mathcal{E}$ be their product. In other words, let $\mathcal{E}$ take as input a list $T_{m_1}, T_{m_2}, ..$ where T_{m_i} is the truth table of some Boolean function g_{m_i} on m_i bits, and outputs $\mathcal{E}_{m_1}(T_{m_1}), \mathcal{E}_{m_2}(T_{m_2}), ...$ The theorem now follows from Lemma 7.17. □

With Lemma 7.17 and Theorem 7.18 in hand, we are ready to implement Heller's approach described in the beginning of this section, and prove Theorem 1.7-(ii).

Proof of Theorem 1.7-(ii). It is a basic fact that NEXP has polynomial-size circuits iff NE (Section 3), the linear-exponential version of NEXP, has circuits of size a *fixed* polynomial, and that this relativizes. In notation, for every language $\mathcal{O}$,

$$\mathrm{NEXP}^{\mathcal{O}} \subset \mathrm{P}^{\mathcal{O}}/\mathrm{poly} \iff \exists d \in \mathbb{N} : \mathrm{NE}^{\mathcal{O}} \subset \mathrm{P}^{\mathcal{O}}/n^d$$

Therefore, to prove the theorem it suffices to show a language $\mathcal{O}$ satisfying

$$\mathrm{NE}^{\widetilde{\mathcal{O}}} \subset \mathrm{P}^{\mathcal{O}}/n^d, \tag{7.9}$$

for some constant d because $\mathcal{O}$ reduces to $\widetilde{\mathcal{O}}$.

Take an enumeration $N_0^*, N_1^*, ..$ of the class NE^*. Such an enumeration can be obtained from one for NP^* (Section 7.1), since by definition (Section 3),

$$\mathrm{NE}^* = \{L^* : (\exists c \in \mathbb{N}, K^* \in \mathrm{NP}^*)(\forall x, \mathcal{O})\ L^*(\mathcal{O}, x) = K^*(\mathcal{O}, x, 1^{2^{c|x|}})\}.$$

We will want to talk about the query complexity of each N_i^* in the enumeration. Underlying each N_i^* is a constant $c \in \mathbb{N}$ and a function $K^* \in \mathrm{NP}^*$. Underlying K^* is some $\ell \in \mathrm{poly}(n)$, and some $g^* \in \mathrm{P}^*$. Underlying g^* is some $f^* \in \mathrm{FP}^*$, of query complexity (Section 7.1) q_f, say. Define the query complexity

q_g of g^* be that of f^*. Define the query complexity q_K of K^* as $q_g(n + \ell(n))$. Define the query complexity q_i of N_i as $q_K(n + 2^{cn})$.

The point of query complexity here is this: if $N_i^*(\mathcal{O}, x) = 1$, then this equality can be maintained by fixing only $q_i(|x|)$ bits of $\mathcal{O}$ and changing the rest arbitrarily.

Now, modify the list N_0, N_1,.. into a list M_0, M_1,.. (repetitions allowed) such that if M_i has query complexity q_i, then $q_i(n) \leqslant 2^{n \log n}$ for all $n > i$.

Initialize $\mathcal{O}$ to the all-zeroes language. The plan is to modify $\mathcal{O}$ in such a way that for every $n > 1$, a size-n^d circuit with access to $\mathcal{O}$, say $C_n^{\mathcal{O}}$, can compute the function

$$\begin{aligned} L_n &: \{0,1\}^{\lfloor \log n \rfloor} \times \{0,1\}^n \to \{0,1\} \\ L_n &: (i, x) \mapsto M_i^{\widetilde{\mathcal{O}}}(x). \end{aligned} \tag{7.10}$$

This yields (7.9), hence the theorem, because each language $K \in \mathrm{NE}^{\widetilde{\mathcal{O}}}$ corresponds to some $M_i^{\widetilde{\mathcal{O}}}$, and in order to compute $K(x)$ on all but finitely many inputs x (in particular for $x \in \{0,1\}^{>2i}$) we can just provide (i, x) to the circuit $C_{|x|}^{\mathcal{O}}$, implying $K \in \mathrm{P}^{\mathcal{O}}/n^d$.

We modify $\mathcal{O}$ iteratively; in iteration $n > 1$ we finalize $\mathcal{O}$ on all inputs in $\{0,1\}^{\leqslant n^d}$, plus some additional $2^{4n \log n}$ inputs at most. Let f_n denote the finalized portion of $\mathcal{O}$ at the end of iteration n, i.e., f_n is the restriction of $\mathcal{O}$ to those inputs on which it is finalized by the end of iteration n.

In iteration 1 we do nothing, so $f_1 : \{\lambda, 0, 1\} \to \{0\}$ where λ is the empty string. At iteration $n > 1$, consider all possible ways of extending f_{n-1} to a language f. Out of all such f, pick one such that when $\mathcal{O} = f$, the collection

$$S_f := \{(i, x) : L_n(i, x) = 1\} \tag{7.11}$$

is maximal. Set $\mathcal{O} = f$.

Now we want to "open up space" in f by un-defining it at some inputs, the idea being then to encode the function L_n in the freed space so that a small circuit can look it up. In doing so, of course, we do not want to disturb (7.11), which, by the way we picked f, is equivalent to wanting that $\mathcal{S}_f$ does not shrink — i.e., as we restrict f to some f', no matter how we extend f' back to some language g, we want $\mathcal{S}_g = \mathcal{S}_f$.

Consider a pair (i, x) in $\mathcal{S}_f$. Because M_i has query complexity less than $2^{n \log n}$ on input $x \in \{0,1\}^n$, the membership of (i, x) in $\mathcal{S}_f$ can be preserved by fixing $\widetilde{f}$ on at most $2^{n \log n}$ inputs only. There are at most $n2^n$ pairs in $\mathcal{S}_f$. Thus if we want $\mathcal{S}_f$ not to shrink, it suffices to fix $\widetilde{f}$ at $2^{3n \log n}$ inputs. By the Interpolation theorem, this means we only need to reserve a small set of "bad" inputs B, of size $\leqslant 2^{3n \log n}$, beyond those already reserved in previous iterations, i.e., beyond $\operatorname{dom} f_{n-1}$, such that on B we have no control as to how f behaves, but on the "good" inputs $\{0,1\}^* \setminus (B \cup \operatorname{dom} f_{n-1})$, we can change f arbitrarily. So let f_n be the restriction of f to $B \cup \operatorname{dom} f_{n-1}$.

Now that we opened up space in f, we are ready to store the information in (7.10) so that a small circuit can look it up. That information is the truth table of a function on $n + \log n$ bits, so it suffices to have $2^{2n \log n}$ bits available in $\operatorname{dom} f_n$ for this purpose. Since there are at most $2^{3n \log n}$ bad inputs in f_n by the previous paragraph, and since there are at most $2^{4(n-1)\log(n-1)}$ inputs in $\operatorname{dom} f_{n-1}$ that are outside $\{0,1\}^{\leqslant (n-1)^d}$ by induction, we know there are at most $2^{4n \log n}$ inputs currently in $\operatorname{dom} f_n$ that are outside $\{0,1\}^{\leqslant (n-1)^d}$. So there is sufficient space in $\{0,1\}^{n^d}$ for storage when d is large enough.

As for how to actually store the information, initially consider each input (i, x) to L_n as prepended with zeroes until it becomes a string $Y_{(i,x)}$ of length n^d, and then set $f_n(Y_{(i,x)}) := L_n(i, x)$. Of course this may not work as some bad inputs may coincide with some $Y_{(i,x)}$, but this can be handled simply by

changing the encoding of (i, x) to $Y_{(i,x)} \oplus Z$ for a suitably picked $Z \in \{0,1\}^{n^d}$; such Z exists because it can be picked at random with non-zero probability (by a union bound on the event that some bad input coincides with $Y_{(i,x)} \oplus Z$ for some (i, x)). This Z can then be hardwired to a circuit of size n^d, as we wanted to do.

To finish, let f_n behave arbitrarily on the rest of the good inputs in $\{0,1\}^{\leqslant n^d}$, and then accordingly adjust f_n on the bad inputs in $\{0,1\}^{\leqslant n^d}$ — recall from the Interpolation theorem that on a bad input, f_n is a function of how it behaves on non-bad inputs of same length. We have thus constructed f_n as desired. □

7.5 Proof of Theorem 1.7-(iii)

In this section we prove the last part of Theorem 1.7, which we recall here:

Theorem 1.7-(iii), Restated. *Relative to some affine oracle,* $\mathrm{NEXP} \not\subset \mathrm{P/poly}$.

Proof. We construct $\mathcal{O}$ such that some $L \in \mathrm{NEXP}^{\mathcal{O}}$ does not have polynomial size circuits over the $\widetilde{\mathcal{O}}$-extended basis; in notation,

$$L \notin \mathrm{P}^{\widetilde{\mathcal{O}}}/\mathrm{poly}. \tag{†}$$

Since $\mathcal{O}$ reduces to $\widetilde{\mathcal{O}}$, this will prove $\mathrm{NEXP}^{\widetilde{\mathcal{O}}} \not\subset \mathrm{P}^{\widetilde{\mathcal{O}}}/\mathrm{poly}$.

Initialize $\mathcal{O}$ arbitrarily. Extend the standard Boolean basis with $\widetilde{\mathcal{O}}$. Let $s(n) = n^{\log n}$ and let $\tilde{s}(n) \in O(s(n) \log s(n))$ be such that every size-$s(n)$ circuit can be represented by a string of size $\tilde{s}(n)$.

By a counting argument (or by the "fast diagonalization via minority vote" trick from the proof of Theorem 1.3) it follows that for all but finitely many $n \in \mathbb{N}$, say for all $n \geqslant n_0$, there is a string τ_n of length $\lceil \tilde{s}(n) \rceil$ such that, when

extended arbitrarily to length 2^n, τ_n becomes the truth table of a function that is uncomputable by any size-$s(n)$ circuit on n inputs.

So modify $\mathcal{O}$ as follows. For $n := n_0, n_0+1, \ldots$, pick a string τ_n as above, and modify $\mathcal{O}$ at length 2^n so that its truth table starts as τ_n.

Finally, define L simply as

$$L(x) := \mathcal{O}(0^{2^{|x|}-|x|}x).$$

Clearly, $L \in \mathrm{NEXP}^{\mathcal{O}}$. To show that (†) holds for L, let $d \in \mathbb{N}$ arbitrary, and consider a circuit family $C^{\widetilde{\mathcal{O}}} := \{C_n^{\widetilde{\mathcal{O}}}\}$ of size $t \in O(n^d)$ (over the $\widetilde{\mathcal{O}}$-extended basis). Eventually, $t(n) < n^{\log n}$, say for all $n \geqslant n_1 \geqslant n_0$. In the above construction, at the end of iteration $n \geqslant n_1$, we have that $C_n^{\widetilde{\mathcal{O}}}$ cannot compute L. This situation does not change in a later iteration because the behavior of $C_n^{\widetilde{\mathcal{O}}}$, a size-$t(n)$ circuit, does not depend on the values of $\widetilde{\mathcal{O}}$, hence of $\mathcal{O}$, at length $> 2^n$. So $L \notin \mathrm{P}^{\mathcal{O}}/\mathrm{poly}$. $\square$

8 EXTENSIONS TO FOURTH RESULT

Chapter 1 proclaimed: in dealing with efficient programs, i.e., in studying FP and classes built from FP (Chapter 3), if relativization is a proxy for "known proof techniques" for pre-90s, then affine relativization is one for post-90s. To support this claim, in Chapter 7 we gave two kinds of results. First, we showed that a famously nonrelativizing result from the 90's, namely MAEXP $\not\subset$ P/poly (Fact 1.6), relativizes affinely. Second, we showed that NEXP $\not\subset$ P/poly does not relativize affinely, nor does NEXP $\subset$ P/poly. This second result gives support to our claimed status of affine relativization because any model for "known proof techniques" must also explain our inability to settle conjectures.

In this chapter we give more results such as these. Our focus will be on the latter kind, showing certain conjectures to not relativize affinely. For results of the first type, we merely point out that the first three contributions of this thesis — Theorems 1.3, 1.4, and 1.5 — all relativize affinely. (In fact Theorems 1.3 and 1.4 both relativize, period.)

We split the remainder of the chapter into three sections. In the first section we derive a small technical ingredient that will be useful in the rest of the chapter.

In the second section, we demonstrate the strength of the machinery developed in Chapter 7 and here, by taking an existing oracle construction in the sense of Baker-Gill-Solovay, and importing it to our framework to make it an affine oracle. This result ties into the discussion in Section 2.3 – Myths 3 & 4.

In the last section we show, among other things, that a better-than-brute-force algorithm for CircuitSAT must be affinely nonrelativizing. This ties into the discussion in Section 2.3 – Myth 5.

8.1 Affine Extensions and Disjoint Unions

The disjoint union of languages $\mathcal{O}_0$ and $\mathcal{O}_1$ is the language mapping (b, x) to $\mathcal{O}_b(x)$. This is a frequently occurring element of unrestricted oracle constructions. In order to import such constructions to our setting (as in Section 8.2), as well as to construct new affine oracles from ground up (as in Section 8.3), we will need that disjoint unions and affine extensions are compatible in the following sense.

Proposition 8.1. *Let $\mathcal{A}_0, \mathcal{A}_1$ be the affine extension of the languages $\mathcal{O}_0, \mathcal{O}_1$ respectively. Then the disjoint union $\mathcal{A}_0 \coprod \mathcal{A}_1 : bx \mapsto \mathcal{A}_b(x)$ is equivalent, under Cook reductions, to the affine extension of the disjoint union $\mathcal{O}_0 \coprod \mathcal{O}_1 : bx \mapsto \mathcal{O}_b(x)$.*

Proof. Let $\mathcal{O} := \mathcal{O}_0 \coprod \mathcal{O}_1$. By definition, the affine extension of $\mathcal{O}$ is the Boolean version of the function that evaluates, given $B, X_1, .., X_n \in \mathbb{F}_{2^k}$ for any k, the polynomial

$$\begin{aligned}
\widehat{O}(BX) &= \sum_{b,x_1,..,x_n \in \{0,1\}} \mathcal{O}(bx) \cdot \textstyle\prod_i (1 + (BX)_i + (bx)_i) \\
&= \sum_{x_1,..,x_n \in \{0,1\}} \mathcal{O}_0(x) \cdot \textstyle\prod_i (1 + (BX)_i + (0x)_i) \\
&\quad + \sum_{x_1,..,x_n \in \{0,1\}} \mathcal{O}_1(x) \cdot \textstyle\prod_i (1 + (BX)_i + (1x)_i) \\
&= (1 + B) \cdot \widehat{\mathcal{O}_0}(X) + B \cdot \widehat{\mathcal{O}_1}(X) \, .
\end{aligned}$$

It follows that $\widetilde{\mathcal{O}} \in P^{\widetilde{\mathcal{O}_0} \amalg \widetilde{\mathcal{O}_1}}$ and $\widetilde{\mathcal{O}_0} \in P^{\widetilde{\mathcal{O}}}$ and $\widetilde{\mathcal{O}_1} \in P^{\widetilde{\mathcal{O}}}$, implying the claim. □

So Proposition 8.1 says, essentially, that we can use the disjoint union operator as though it maintains the property of being affine.

8.2 Renovating Classical Oracles

Recall Myths 3 & 4 (p. 27): (i) affine oracles are much harder to construct than traditional oracles, and (ii) the NEXP $\overset{?}{\subset}$ P/poly question is the only place where our framework seems to have an edge over the competition. We now debunk both (i) and (ii). We do this by showcasing how the machinery we developed, in Chapter 7 and here, can be used to import traditional oracle constructions and turn them into affine oracles.

The particular construction we showcase is due to Beigel, Buhrman, and Fortnow [16], of an oracle relative to which $\mathrm{P} = \oplus\mathrm{P} \subsetneq \mathrm{NP} = \mathrm{EXP}$. Here $\oplus$P is basically the class for which $\oplus$SAT is complete; in other words, given a set $\mathcal{C}$ of languages, if $\oplus \cdot \mathcal{C}$ denotes the set of all languages of the form

$$L(x) = \bigoplus_{y \in \{0,1\}^{\ell(|x|)}} V(x, y) \tag{8.1}$$

for some $\ell \in \mathrm{poly}(n)$ and $V \in \mathcal{C}$, then $\oplus$P is defined as $\oplus \cdot \mathrm{P}$.

The Beigel et al. oracle is an interesting one because it shows, among other things, that $\oplus\mathrm{P} = \mathrm{PSPACE}$ cannot be derived via relativizing techniques. From the definitions (Section 3) it is clear that $\mathrm{PSPACE}(= \Sigma_\infty \mathrm{P})$ is sandwiched between NP and EXP. It is also not hard to see that $\oplus\mathrm{P} \subset \mathrm{PSPACE}$: think of an interactive proof for $\oplus$SAT where the error probability is allowed to be < 1 instead of $< 1/3$. The opposite containment, however, is wide open.

We now show that affinely relativizing techniques cannot derive this containment, hence $\oplus\mathrm{P} = \mathrm{PSPACE}$, either.

Theorem 8.2. *Relative to some affine oracle,* $\mathrm{P} = \oplus\mathrm{P} \subsetneq \mathrm{NP} = \mathrm{EXP}$.

In order to prove Theorem 8.2 we use two facts regarding EXP.

Definition 8.1 (SCE)**.** *Let* SCE *(short for* SuccinctCircuitEval*) be the language that, given* $(D, x, 1^t)$*, decides if* $C(x) = 1$*, where* C *is a circuit of size* $\leqslant 2^t$ *described by the circuit* D *via a function such as*

$$D(i, j) := \textit{ type of the } i\textit{th gate in } C\textit{, and the index of the } j\textit{th neighbor of it.}$$

Let $\text{SCE}^{\mathcal{O}}$ *denote, given the language* $\mathcal{O}$*, the extension of* SCE *to circuits over the* $\mathcal{O}$*-extended Boolean basis.*

Fact 8.3. SCE *is complete for* EXP*. This holds relative to every oracle.*

Fact 8.4. $\text{EXP} \supsetneq \text{P}$*. This holds relative to every oracle.*

Both facts are well-known in the classical study of complexity classes based on Turing machines. If we do not want to rely on Turing machines, as we said we wouldn't (Chapter 3), then it suffices for FP to be efficiently enumerable in the following sense: there is an enumeration of FP, say enum, and there is a function $\mathsf{exec} \in \text{FP}$, such that for every $i \in \mathbb{N}$, there is $t_i \in \text{poly}(n)$ such that

$$\mathsf{enum}_i(x) = \mathsf{exec}(i, x, 1^{t(|x|)})$$

for every $x \in \{0,1\}^*$ and $t(|x|) \geqslant t_i(|x|)$.

We are ready to prove Theorem 8.2 and finish this section.

Proof of Theorem 8.2. By Fact 8.4, and because $\text{P} \subset \oplus\text{P}$ and $\text{NP} \subset \text{EXP}$ relative to every oracle by definition, it suffices to show an affine oracle $\mathcal{A}$ such that

$$\oplus\text{P}^{\mathcal{A}} \subset \text{P}^{\mathcal{A}} \quad \text{and} \quad \text{EXP}^{\mathcal{A}} \subset \text{NP}^{\mathcal{A}}. \tag{8.2}$$

In fact, by Proposition 8.1, instead of an affine oracle, it suffices for $\mathcal{A}$ to be the disjoint union of two affine oracles $\mathcal{A}_0 := \widetilde{\mathcal{O}_0}$ and $\mathcal{A}_1 := \widetilde{\mathcal{O}_1}$.

We claim:

Lemma 8.5. *There exists a language $\mathcal{A}$ such that $\mathcal{A} = \widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$ for some $\mathcal{O}_0$, $\mathcal{O}_1$ and*

$$\oplus\mathrm{SAT}^{\mathcal{A}}(u) = \mathcal{O}_0(u, 1^{|u|^2}) \tag{8.3}$$

$$\mathrm{SCE}^{\mathcal{A}}(u) = \bigvee_{z \in \{0,1\}^{|u|^2}} \mathcal{O}_1(u, z) \tag{8.4}$$

for every $u \in \{0,1\}^*$.

Such an $\mathcal{A}$ satisfies (8.2), hence the theorem, because:

- $\oplus\mathrm{SAT}^{\mathcal{A}}$ is complete for $\oplus\mathrm{P}^{\mathcal{A}}$ (see (8.1)), so that the condition (8.3) guarantees that $\oplus\mathrm{P}^{\mathcal{A}} \subset \mathrm{P}^{\mathcal{O}_0}$, hence that $\oplus\mathrm{P}^{\mathcal{A}} \subset \mathrm{P}^{\mathcal{A}}$ because $\mathcal{O}_0$ Karp-reduces to $\widetilde{\mathcal{O}_0}$ $(= \mathcal{A}_0)$ which in turn Karp-reduces to $\mathcal{A}_0 \coprod \mathcal{A}_1$ $(= \mathcal{A})$,
- $\mathrm{SCE}^{\mathcal{A}}$ is complete for $\mathrm{EXP}^{\mathcal{A}}$ (Fact 8.3), so that the condition (8.4) guarantees that $\mathrm{EXP}^{\mathcal{A}} \subset \mathrm{NP}^{\mathcal{O}_1}$, hence that $\mathrm{EXP}^{\mathcal{A}} \subset \mathrm{NP}^{\mathcal{A}}$ similarly to the previous item.

So all that remains is to prove Lemma 8.5.

(Theorem 8.2, mod Lemma 8.5) □

Proof of Lemma 8.5. Initialize $\mathcal{O}_0$ and $\mathcal{O}_1$ to the all-zeroes language. Update $\mathcal{O}_0$ to the output of the following procedure.

Ensure-Condition-(8.3)
input: $\mathcal{O}_0, \mathcal{O}_1$; *output:* $\mathcal{O}_0'$ such that $\widetilde{\mathcal{O}_0'} \coprod \widetilde{\mathcal{O}_1}$ satisfies condition (8.3)
initialize $\mathcal{O}_0' := \mathcal{O}_0$
for every $v \in \{0,1\}^*$ in lex order,
  $\mathcal{A} := \widetilde{\mathcal{O}_0'} \coprod \widetilde{\mathcal{O}_1}$
  $\mathcal{O}_0'(v, 1^{|v|^2}) := \oplus\mathrm{SAT}^{\mathcal{A}}(v)$
return $\mathcal{O}_0'$

Let $\mathcal{A} := \widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$. Note that at this point $\mathcal{A}$ satisfies condition (8.3). Now perform the following procedure:

Ensure-Condition-(8.4)
for every $v \in \{0,1\}^*$ in lex order,
  if $\mathrm{SCE}^{\mathcal{A}}(v) = 1$ then
    for every $S \subset \{0,1\}^{|v|^2}$
      let $\mathcal{O}_1^S$ be the same as $\mathcal{O}_1$ except $\mathcal{O}_1^S(v,z) := 1$ for all $z \in S$
      let $\mathcal{O}_0^S :=$ Ensure-Condition-(8.3)$(\mathcal{O}_0, \mathcal{O}_1^S)$
      let $\mathcal{A}^S := \widetilde{\mathcal{O}_0^S} \coprod \widetilde{\mathcal{O}_1^S}$
    pick a nonempty $S \subset \{0,1\}^{|v|^2}$ such that
      $\mathrm{SCE}^{\mathcal{A}^S}(u) = \mathrm{SCE}^{\mathcal{A}}(u)$ for all $u \leqslant v$ (in lex order)
    $\mathcal{A} := \mathcal{A}^S$

We claim that after this second procedure, $\mathcal{A}$ satisfies both conditions (8.3) and (8.4), proving the lemma.

That $\mathcal{A}$ satisfies condition (8.3) is clear: it initially satisfies the condition, and whenever it is updated, it is updated to some $\mathcal{A}^S$ satisfying the condition.

As for condition (8.4), we proceed by induction on $\{0,1\}^*$ to argue the following claim: for every $v \in \{0,1\}^*$, at the beginning of iteration v, condition (8.4) is satisfied for every $u < v$, and $\mathcal{A}$ is of the form $\widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$ where $\mathcal{O}_1(u,\cdot) = 0$ for every $u \geqslant v$.

The claim is true for the smallest string v, because at the beginning of the procedure, $\mathcal{A}$ is of the form $\widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$ where $\mathcal{O}_1$ is the all-zeroes language.

Suppose the claim is true for some fixed $v \in \{0,1\}^n$, and consider iteration v.

If $\mathrm{SCE}^{\mathcal{A}}(v) = 0$, then without anything done, condition (8.4) is already satisfied for $u = v$. So the right thing to do in this case is to do nothing, which the procedure does. The inductive step follows in this case.

In case $\mathrm{SCE}^{\mathcal{A}}(v) = 1$, suppose for a moment that a nonempty set $S \subset \{0,1\}^{|v|^2}$ as stated in the procedure does exist. Then the procedure updates $\mathcal{A} = \widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$ to some $\mathcal{A}^S = \widetilde{\mathcal{O}_0^S} \coprod \widetilde{\mathcal{O}_1^S}$ where $\mathcal{O}_1$ and $\mathcal{O}_1^S$ are identical on inputs of the form $(u, \cdot)$ for $u < v$ and for $u > v$. Further, $\mathrm{SCE}^{\mathcal{A}^S}(u)$ is identical to $\mathrm{SCE}^{\mathcal{A}}(u)$ for $u \leqslant v$. The inductive step follows.

All that remains is to prove that a set $S \subset \{0,1\}^{|v|^2}$ as stated does exist:

Lemma 8.6. *Let $v \in \{0,1\}^*$. Let $\mathcal{A}$ be a language of the form $\widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$ for some $\mathcal{O}_0$, $\mathcal{O}_1$ such that $\mathcal{O}_1(v, z) = 0$ for every $z \in \{0,1\}^{|v|^2}$.*

Suppose that $\mathcal{A}$ satisfies condition (8.3) for all u, and condition (8.4) for all $u < v$. Suppose further that $\mathrm{SCE}^{\mathcal{A}}(v) = 1$.

There is a nonempty $S \subset \{0,1\}^{|v|^2}$ such that $\mathrm{SCE}^{\mathcal{A}^S}(u) = \mathrm{SCE}^{\mathcal{A}}(u)$ for all $u \leqslant v$, where $\mathcal{A}^S$ is defined in procedure Ensure-Condition-(8.4) *above.*

(Theorem 8.2, mod Lemma 8.6) □

Proof of Lemma 8.6. Given $v \in \{0,1\}^n$, for each $z \in \{0,1\}^{n^2}$, consider replacing the value of $\mathcal{O}_1(v, z)$ with a variable α_z. Call the resulting function $\mathcal{O}_1^\alpha$.

Using $\mathcal{O}_1^\alpha$, define $\mathcal{O}_0^\alpha$ in the same way that the procedure Ensure-Condition-(8.4) defines $\mathcal{O}_0^S$ using $\mathcal{O}_1^S$. That is, let

$$\mathcal{O}_0^\alpha := \text{Ensure-Condition-(8.3)}(\mathcal{O}_0, \mathcal{O}_1^\alpha)$$

Use $\mathcal{O}_0^\alpha$ and $\mathcal{O}_1^\alpha$ to define $\mathcal{A}^\alpha$ in the same way that using $\mathcal{O}_0^S$ and $\mathcal{O}_1^S$, the procedure Ensure-Condition-(8.4) defines $\mathcal{A}^S$. That is, let

$$\mathcal{A}^\alpha := \widetilde{\mathcal{O}_0^\alpha} \coprod \widetilde{\mathcal{O}_1^\alpha}$$

Notice that $\mathcal{A}^\alpha = \mathcal{A}$ when we put $\vec{\alpha} := \vec{0}$. (In particular, notice that since $\mathcal{A} = \widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1}$ satisfies condition (8.3) for every $u \in \{0,1\}^*$, Ensure-Condition-

(8.3)$(\mathcal{O}_0, \cdot)$ returns just $\mathcal{O}_0$.) Therefore, the expression

$$\bigvee_{u \leqslant v} \mathrm{SCE}^{\mathcal{A}}(u) \oplus \mathrm{SCE}^{\mathcal{A}^\alpha}(u), \tag{Δ}$$

which is a function $\Delta(\vec{\alpha})$ on the variables α_z, gives 0 when $\vec{\alpha} = \vec{0}$.

Suppose towards a contradiction that the claim of the lemma is false. Then

$$\Delta(\vec{0}) = 0, \text{ and } \Delta(\vec{a}) = 1 \text{ for every } \vec{a} \neq \vec{0}$$

implying that $\Delta = \mathrm{OR}_{2^{n^2}}$, the OR function on 2^{n^2} variables. But this is impossible because the affine extension polynomial (Section 7.1) of OR_m, for any $m \in \mathbb{N}$, satisfies

$$\widehat{\mathrm{OR}}_m(x) = 1 + \prod_{i=1}^{m}(1 + x_i)$$

hence has total degree m, whereas:

Lemma 8.7. *$\widehat{\Delta}$ has total degree $\leqslant 2^{4n+1}$.*

All that remains is to prove Lemma 8.7.

(Lemma 8.6, mod Lemma 8.7) □

Proof of Lemma 8.7. Since $x^2 = x$ for binary x, it suffices to show that there is *some* $\mathbb{F}_2$-polynomial of total degree $\leqslant 2^{4n+1}$ that agrees with Δ on the Boolean values.

In fact since Δ is, by definition, the OR function applied to $\leqslant 2^{n+1}$ terms of the form

$$\mathrm{SCE}^{\mathcal{A}}(v) \oplus \mathrm{SCE}^{\mathcal{A}^\alpha}(v),$$

for various $v \in \{0,1\}^{\leqslant n}$, and because $\widehat{\mathrm{OR}}_m$ has total degree m for all m, it suffices to show:

Lemma 8.8. $\mathrm{SCE}^{\mathcal{A}^\alpha}(v)$ *can be expressed as a* $\mathbb{F}_2$*-polynomial (in* α_z *variables) of total degree* $\leqslant 2^{3|v|}$*, for every* $v \in \{0,1\}^*$.

(Lemma 8.7, mod Lemma 8.8) □

Proof of Lemma 8.8. Given v, by definition $\mathrm{SCE}^{\mathcal{A}^\alpha}(v)$ views v as a triple $(D, x, 1^t)$, where D is a circuit that describes a circuit C of size $\leqslant 2^t < 2^{|v|}$ over the $\mathcal{A}^\alpha$-extended basis, and

$$\mathrm{SCE}^{\mathcal{A}^\alpha}(v) = C(x).$$

Take each gate of C, say the ith gate, and assign a variable b_i so that it can be expressed as

$$b_i = f(b_{i_1} \dots b_{i_{\ell_i}})$$

meaning that the i^{th} gate is of type f (an element of the basis), and has its inputs coming from gates $i_1, \dots, i_{\ell_1}$ in that order. Then

$$C(x) = \bigoplus_{b \in \{0,1\}^m} \bigwedge_{i=1}^{m} (1 \oplus b_i \oplus f_i(b_{i_1} \dots b_{i_{\ell_i}})) \qquad (\S)$$

where m denotes the number of gates in C. So $m \leqslant (\text{size of } C) \leqslant 2^t < 2^{|v|}$.

If the i^{th} gate of C is an input node, say x_j, then the $f_i(\cdots)$ term in the sum (§) is to be replaced with x_j. Notice that the only place where α_z variables arise in (§) is the term $f_i(\cdots)$, and that no such variable arises when the i^{th} gate is an input gate x_j, or a standard gate $\wedge$, $\oplus$, etc.

On the other hand, if the i^{th} gate is of type $\mathcal{A}^\alpha$, then the $f_i(\cdots)$ term is

$$\mathcal{A}^\alpha(b_{i_1} \dots b_{i_{\ell_i}})$$

which equals, since $\mathcal{A}^\alpha = \widetilde{\mathcal{O}_0} \coprod \widetilde{\mathcal{O}_1^\alpha}$, either

$$\widetilde{\mathcal{O}_0}(b_{i_2} \dots b_{i_{\ell_i}}) \tag{$\dagger$}$$

or

$$\widetilde{\mathcal{O}_1^\alpha}(b_{i_2} \dots b_{i_{\ell_i}}) \tag{$\ddagger$}$$

depending on whether b_{i_1} equals 0 or 1 respectively.

Since $\ell_i \leqslant$ (number of gates) $= m < 2^{|v|}$, the proof is done once we show:

Lemma 8.9. *For every $w \in \{0,1\}^*$,*

- $\widetilde{\mathcal{O}_1^\alpha}(w)$ *can be expressed as a $\mathbb{F}_2$-polynomial (in α_z variables) of total degree* 1,
- $\widetilde{\mathcal{O}_0^\alpha}(w)$ *can be expressed as a $\mathbb{F}_2$-polynomial (in α_z variables) of total degree $\leqslant |w|^2$.*

(Lemma 8.8, mod Lemma 8.9) □

Proof of Lemma 8.9. By definition (Section 7.1), $\widetilde{f}(w)$ interprets its input w as a pair (X, y), where $X \in \mathbb{F}_{2^k}^m$ (for some m and k encoded in X), and outputs the yth bit of the sum

$$\sum_{b \in \{0,1\}^m} f(b) \cdot \Pi_{i=1}^m (1 + X_i + b_i)$$

The only place where α_z variables can arise in this sum is the $f(b)$ term. Putting $f := \mathcal{O}_0^\alpha$ and $f := \mathcal{O}_1^\alpha$, we see that it suffices to show:

Lemma 8.10. *For every $w \in \{0,1\}^*$,*

- $\mathcal{O}_1^\alpha(w)$ *can be expressed as a $\mathbb{F}_2$-polynomial (in α_z variables) of total degree* 1,
- $\mathcal{O}_0^\alpha(w)$ *can be expressed as a $\mathbb{F}_2$-polynomial (in α_z variables) of total degree $\leqslant |w|^2$.*

(Lemma 8.9, mod Lemma 8.10) □

Proof of Lemma 8.10. The claim regarding $\mathcal{O}_1^\alpha$ is trivial: by definition, $\mathcal{O}_1^\alpha(w)$ is either some constant $0/1$, or is some variable α_z.

As for $\mathcal{O}_0^\alpha$, by definition, $\mathcal{O}_0^\alpha(w)$ is some constant $0/1$, or is $\oplus\mathrm{SAT}^{\mathcal{A}^\alpha}(v)$ for some $v \in \{0,1\}^{<|w|}$. So it suffices to show:

Lemma 8.11. *$\oplus\mathrm{SAT}^{\mathcal{A}^\alpha}(v)$ can be expressed as a $\mathbb{F}_2$-polynomial (in α_z variables) of total degree $\leqslant |v|^2$, for every $v \in \{0,1\}^*$.*

(Lemma 8.10, mod Lemma 8.11) □

Proof of Lemma 8.11. By induction on $|v|$. The base case, $|v| = 0$, is trivial.

Suppose the claim is true for all $v \in \{0,1\}^{<s}$ for some $s \in \mathbb{N}^+$. Given $v \in \{0,1\}^s$, $\oplus\mathrm{SAT}^{\mathcal{A}^\alpha}$ interprets v as a formula φ over the $\mathcal{A}^\alpha$-extended basis with $m \leqslant s$ variables, and satisfies

$$\oplus\mathrm{SAT}^{\mathcal{A}^\alpha}(v) = \bigoplus_{x \in \{0,1\}^m} \varphi(x).$$

Replacing each subformula in φ of the form

$$\mathcal{A}^\alpha(\varphi_1 .. \varphi_h)$$

where each φ_i is a subformula, with

$$\bigoplus_{b \in \{0,1\}^h} \mathcal{A}^\alpha(b) \wedge (\varphi_1 \oplus b_1 \oplus 1) \wedge \cdots \wedge (\varphi_h \oplus b_h \oplus 1)$$

we get, assuming there are ℓ such subformulas to be replaced,

$$\varphi(x) = \bigoplus_{\substack{B_1 \in \{0,1\}^{h_1} \\ \vdots \\ B_\ell \in \{0,1\}^{h_\ell}}} \mathcal{A}^\alpha(B_1) \wedge \cdots \wedge \mathcal{A}^\alpha(B_\ell) \wedge \psi(x, B)$$

where ψ is a formula over the *standard* basis (hence can contribute no α_z-variables). Viewing the i^{th} occurrence of $\mathcal{A}^{\alpha}(\cdot)$ inside the sum as a $\oplus\text{SAT}^{\mathcal{A}^{\alpha}}$ instance of size s_i, where $s_1 + \cdots + s_\ell \leqslant s$, we can apply the induction hypothesis that it can be expressed as a polynomial of degree $\leqslant s_i^2$. The claim then follows by induction. $\square$

8.3 Oracles That Render Brute Force Optimal

Recall Myth 5 (p. 28): the connection recently discovered by Williams [52], from better-than-brute-force algorithms for CircuitSAT to circuit lower bounds NEXP, more precisely, the result:

Fact 8.12 ([52])**.** *For* $\ell \in \text{poly}(n)$*, let* CircuitSAT$_\ell$ *denote the restriction of* CircuitSAT *to circuits of size* $\leqslant \ell(n)$*, if* n *is the number of inputs to the circuit.*

If CircuitSAT$_{n^d} \in$ DTIME$(O(2^n/n^{\omega(1)}))$ *for every constant* d*, then* NEXP $\not\subset$ P/poly.

obviates relativization-based barriers.

We now counter this myth by showing that the very existence of such an algorithm constitutes a theorem that is affinely nonrelativizing. So Williams' program would succeed in proving NEXP $\not\subset$ P/poly, if it can find an ingredient that does not affinely relativize, which is to say that it will pass the barrier if it can pass the barrier.

To proceed, we need to make some definitions. Recall from Section 3 that we defined DTIME(T), the extension of P to time complexity T, where T is any class of functions closed under polynomials in the following sense: if $t \in T$ then for all constants d, $t^d < t'$ for some $t' \subset T$. In this section we will need a finer definition that allows T to be *closed under quasi-linear functions* only: if $t \in T$ then then for all constants d, $t \log^d t < t'$ for some $t' \subset T$. One way to do this is to change our notion of efficient computability, from FP to FQLIN.

Definition 8.13 (FQLIN)**.** *Define* FQLIN *as the set of all* $f : \{0,1\}^* \to \{0,1\}^*$ *computable in quasi-linear time.*

As was the case for FP, we do not rely on a particular implementation of quasi-linear time computability; for concreteness the reader can take the stan-

dard definition based on Turing machines running in time $O(n \log^d n)$ for some constant d.

Hence, FP is the set of all F such that $F(x) = f(x, 1^{t(|x|)})$ for some polynomial t and $f \in \mathrm{FQLIN}$. More generally, the same idea gives $\mathrm{FTIME}(T)$, $\mathrm{DTIME}(T)$, etc., for any class T closed under quasi-linear functions.

We will also need a finer definition of nondeterminism in this section:

Definition 8.14 ($\mathrm{N}_\ell\mathrm{P}$)**.** *For every function $\ell : \mathbb{N} \to \mathbb{N}$, define $\mathrm{N}_\ell\mathrm{P}$ as the set of all languages of the form*

$$L(x) = \exists y \in \{0,1\}^{\ell(|x|)} : V(x, y)$$

for some $V \in \mathrm{P}$.

Hence, $\mathrm{NP} = \cup_{\ell \in \mathrm{poly}(n)} \mathrm{N}_\ell\mathrm{P}$. The motivation for Definition 8.14 is the following relationship to CircuitSAT:

Proposition 8.15. *For $\ell \in \mathrm{poly}(n)$, let $\mathrm{CircuitSAT}_\ell$ denote the restriction of $\mathrm{CircuitSAT}$ to circuits of size $\leqslant \ell(n)$, if n is the number of inputs to the circuit. Let $T(n) \subset n^{\omega(1)}$ be a class of functions eligible for defining $\mathrm{DTIME}(T(n))$ (c.f. Definition 8.13 and remarks).*

Suppose that for every $\ell \in \mathrm{poly}(n)$, $\mathrm{CircuitSAT}_\ell \in \mathrm{DTIME}(T(n))$. Then $\mathrm{N}_n\mathrm{P} \subset \mathrm{DTIME}(T(n))$. This holds relative to every oracle.

Proof. Let $\mathcal{O}$ be an arbitrary oracle. Extend the standard Boolean basis with $\mathcal{O}$. Let $L \in \mathrm{N}_n\mathrm{P}^{\mathcal{O}}$ so that

$$L(x) = \exists y \in \{0,1\}^{|x|} : V(x, y)$$

for some $V \in \mathrm{P}^{\mathcal{O}}$. By the Cook-Levin theorem (Section 7.1), there exists $\mathrm{Desc}_V \in \mathrm{FP}^{\mathcal{O}}$ that outputs, given input 1^n, a circuit $C_{V_{2n}}$ of size $\ell(n)$ for some $\ell \in \mathrm{poly}(n)$ such that

$$V(x, y) = C_{V_{2n}}(x, y)$$

for every $x, y \in \{0,1\}^n$. Therefore, if CircuitSAT$_\ell \in \mathrm{DTIME}^{\mathcal{O}}(t(n))$, then $\mathrm{N_nP}^{\mathcal{O}} \subset \mathrm{DTIME}^{\mathcal{O}}(O(t(n)+p(n)))$ for some $p \in \mathrm{poly}(n)$. The result follows because $t \in T$ for some class $T \supset n^{\omega(1)}$ closed under quasi-linear functions. □

Thus, to show that the existence of a deterministic time-$T(n)$ algorithm for CircuitSAT would be affinely nonrelativizing, it suffices to show an oracle relative to which $\mathrm{N_nP}$ is not contained in deterministic time $T(n)$. This is what we do in the proof of the next theorem.

Theorem 8.16. *Relative to some affine oracle,* CircuitSAT$_{n^d} \notin \mathrm{DTIME}(O(2^n/n^{\omega(1)}))$ *for some constant* d.

To prove Theorem 8.16, we follow a strategy invented by Aaronson and Wigderson [2]. The idea is to import existing lower bounds from communication complexity to the setting of oracles. To get more specific, we need to make some definitions.

In order to avoid lengthy technicalities, in the rest of this section we will embrace the Turing machine based jargon — running time, algorithm, etc.

Definition 8.2 (cc, ticc)**.** *Define* $\mathrm{DTIME}(t(n))_{\mathrm{ticc}}$ *as the class of families* $f := \{f_n\}$ *satisfying the following. (i) Each* f_n *is a Boolean function on pairs of* 2^n*-bit strings, (ii) There is a protocol involving two algorithms* M_0, M_1 *such that for all* n *and all* $(X,Y) \in \mathrm{dom}(f_n)$*, the two parties* $M_0^X(1^n), M_1^Y(1^n)$ *compute* $f_n(X,Y)$ *in time* $O(t(n))$.

Let $\mathrm{DTIME}(t(n))_{\mathrm{cc}}$ *denote the relaxation of* $\mathrm{DTIME}(t(n))_{\mathrm{ticc}}$ *where* M_0, M_1 *are allowed to be non-uniform, and where only the communication between* M_0, M_1 *is counted towards time elapsed.*

Define $\mathrm{P}_{\mathrm{ticc}} := \mathrm{DTIME}_{\mathrm{ticc}}(n^{O(1)})$. *Use* $\mathrm{P}_{\mathrm{ticc}}$ *to define* $\mathrm{NP}_{\mathrm{ticc}}, \mathrm{BPP}_{\mathrm{ticc}}$, *etc., similar to how we define* NP, BPP, *etc., from* P.[12] *Similarly define* P_{cc} *from* $\mathrm{DTIME}_{\mathrm{cc}}$, *and* $\mathrm{NP}_{\mathrm{cc}}, \mathrm{BPP}_{\mathrm{cc}}$, *etc. from* P_{cc}.

The notation $\mathcal{C}_{\mathrm{ticc}}$ is meant to indicate that time is measured on equal grounds with communication. A function in $\mathcal{D}_{\mathrm{cc}}$ according to the classical definition [13] is defined on strings of every even length, whereas Definition 8.2 requires length a power of two; our convention causes nothing but convenience in this section.

We formalize the high-level idea of Aaronson and Wigderson with the following generic theorem in our framework:

Theorem 8.3. *Let* $\mathcal{C}$ *and* $\mathcal{D}$ *be any two complexity classes defined in Chapter 3 or in this section (Section 8.3), such that* $\mathrm{P} \subset \mathcal{C}, \mathcal{D} \subset \mathrm{EXP}$.

If $\mathcal{C}_{\mathrm{ticc}} \not\subset \mathcal{D}_{\mathrm{cc}}$, *then relative to some affine oracle,* $\mathcal{C} \not\subset \mathcal{D}$.

Proof. Supposing there is some $f := \{f_n\}$ in $\mathcal{C}_{\mathrm{ticc}} \setminus \mathcal{D}_{\mathrm{cc}}$, we want to show an affine oracle $\mathcal{A}$ relative to which $\mathcal{C} \not\subset \mathcal{D}$. For concreteness, the reader may take $\mathcal{C}$ to be NP, say, and $\mathcal{D}$ to be BPP.

By Proposition 8.1, instead of an affine oracle, it suffices for $\mathcal{A}$ to be the disjoint union of two affine oracles $\mathcal{A}_0 := \widetilde{\mathcal{O}_0}$ and $\mathcal{A}_1 := \widetilde{\mathcal{O}_1}$. In fact, since every language reduces to its affine extension, it suffices to show $\mathcal{O}_0, \mathcal{O}_1$ such that

$$\mathcal{C}^{\mathcal{O}_0 \amalg \mathcal{O}_1} \not\subset \mathcal{D}^{\widetilde{\mathcal{O}_0} \amalg \widetilde{\mathcal{O}_1}}.$$

For every $n \in \mathbb{N}$, pick an arbitrary pair $(X_n, Y_n) \in \mathrm{dom}\, f_n \subset \{0,1\}^{2^n} \times \{0,1\}^{2^n}$. Initialize $\mathcal{O}_0$ to have the same truth table as X_n for every n, and similarly for $\mathcal{O}_1$

[12] Recall that definitions of BPP, NP, etc. involve some counting of the witnesses w of a P-predicate $L(x, w)$. Here, that predicate would be of the form $f((X, w), (Y, w))$ where $|w|$ is polynomially bounded in n for f_n, i.e., polylogarithmic in $|X|$.

versus Y_n. Because $f \in \mathcal{C}_{\text{ticc}}$, the language $L := \{L_n\}$ defined as

$$L(1^n) := f(\mathcal{O}_{0,n}, \mathcal{O}_{1,n}), \quad L(\neq 1^n) := 0$$

is in $\mathcal{C}^{\mathcal{O}_0 \amalg \mathcal{O}_1}$; to see this just consider using $\mathcal{O}_0 \coprod \mathcal{O}_1$ to simulate a $\mathcal{C}_{\text{ticc}}$-protocol for f. Our objective is to modify $\mathcal{O}_0$,$\mathcal{O}_1$ so that L remains in $\mathcal{C}^{\mathcal{O}_0 \amalg \mathcal{O}_1}$ and becomes out of $\mathcal{D}^{\widetilde{\mathcal{O}_0} \amalg \widetilde{\mathcal{O}_1}}$.

To that end, for any pair of strings $(X, Y) \in \{0,1\}^{2^n} \times \{0,1\}^{2^n}$, let $\mathcal{O}_0 \leftarrow X$ denote the result of updating $\mathcal{O}_0$ so that at length-n inputs, it has the same truth table as X; similarly use $\mathcal{O}_1 \leftarrow Y$ to denote the result of updating $\mathcal{O}_1$ with Y.

Now let $N_1, N_2, \ldots$ be an enumeration of $\mathcal{D}$-algorithms endowed with an oracle access mechanism. (To be precise, we need to consider the class $\mathcal{D}^*$, defined using the class FP^* from Section 7.1 in the same way that $\mathcal{D}$ would be defined from FP. As stated earlier, however, for convenience in this section we embrace the Turing Machine based jargon.) For each algorithm in the enumeration, say for N_i, define $g^i := \{g^i_n\}$ as

$$g^i_n(X, Y) := N_i^{(\widetilde{\mathcal{O}_0 \leftarrow X}) \amalg (\widetilde{\mathcal{O}_1 \leftarrow Y})}(1^n) \tag{8.5}$$

where (X, Y) ranges over $\operatorname{dom} f_n$. In case N_i's output is not well-defined on 1^n — due to N_i computing a partial language which 1^n is outside the domain of — just let g^i_n take the value '$\perp$'.

We claim that g^i differs from f on infinitely many inputs. Indeed, the right-hand-side of (8.5) can be computed by a protocol where one party is given access to X and knows $\mathcal{O}_0$ (up to a finite length, beyond which N_i is guaranteed not to access when run on 1^n), the other party is given Y and knows $\mathcal{O}_1$ (again finitely bounded), and the two parties simulate N_i by using each other as an oracle for the missing side of the disjoint union. So $g^i \in \mathcal{D}_{\text{cc}}$. Since $f \notin \mathcal{D}_{\text{cc}}$, the claim

follows.

Now, for $i = 1..\infty$, find a pair (X_{n_i}, Y_{n_i}) in $\operatorname{dom} f_{n_i} = \operatorname{dom} g^i_{n_i}$ on which f and g^i differ, for some n_i arbitrarily large. Update $\mathcal{O}_0$ to $\mathcal{O}_0 \leftarrow X_{n_i}$ and $\mathcal{O}_1$ to $\mathcal{O}_1 \leftarrow Y_{n_i}$, so that $L(1^{n_i})$ differs from $N_i^{(\widetilde{\mathcal{O}_0 \leftarrow X})} \coprod {}^{(\widetilde{\mathcal{O}_1 \leftarrow Y})}(1^{n_i})$. Since n_i is arbitrarily large, this update does not disturb the previous iterations — e.g., $n_i > 2^{2^{n_{i-1}}}$ suffices since $\mathcal{D} \subset \mathrm{EXP}$. □

We are finally ready to prove Theorem 8.16.

Proof of Theorem 8.16. By Proposition 8.15, it suffices to show an affine oracle relative to which $\mathrm{N}_n\mathrm{P} \not\subset \mathrm{DTIME}(O(2^n/n^{\omega(1)}))$.

Let

$$\neg\mathrm{Disj}(X, Y) := \exists i \; (X(i) \wedge Y(i))$$

be the non-disjointness predicate. It is clear that $\neg\mathrm{Disj} \in \mathrm{NP}_{\mathrm{ticc}}$. On the other hand, it is well-known (e.g., [5, Example 13.6]) that 2^n bits of communication is needed to compute $\neg\mathrm{Disj}$, implying $\mathrm{Disj} \notin \mathrm{DTIME}_{\mathrm{cc}}(o(2^n))$. The claim now follows from Theorem 8.3. □

REFERENCES

[1] Aaronson, Scott. 2006. Oracles are subtle but not malicious. In *Proceedings of the IEEE Conference on Computational Complexity*, 340–354.

[2] Aaronson, Scott, and Avi Wigderson. 2009. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory* 1(1).

[3] Aaronson, Scott (https://cstheory.stackexchange.com/users/1575/scott-aaronson). Can relativization results be used to prove sentences formally independent? Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/3207 (version: 2010-11-20).

[4] Ajtai, Miklós. 1983. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic* 24(1):1–48.

[5] Arora, Sanjeev, and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

[6] Arora, Sanjeev, Russell Impagliazzo, and Umesh Vazirani. 1992. Relativizing versus nonrelativizing techniques: the role of local checkability. Manuscript retrieved from http://cseweb.ucsd.edu/~russell/ias.ps.

[7] Arora, Sanjeev, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof verification and the hardness of approximation problems. *Journal of the ACM* 45(3):501–555.

[8] Artin, Emil. 1957. *Geometric Algebra*. John Wiley & Sons.

[9] Aydınlıoğlu, Barış, and Eric Bach. Affine relativization: Unifying the relativization and algebrization barriers. *ACM Transactions on Computation The-*

ory. Under revision. Available at https://eccc.weizmann.ac.il/report/2016/040/.

[10] Aydınlıoğlu, Barış, Dan Gutfreund, John M. Hitchcock, and Akinori Kawachi. 2011. Derandomizing Arthur-Merlin games and approximate counting implies exponential-size lower bounds. *Computational Complexity* 20(2):329–366.

[11] Aydınlıoğlu, Barış, and Dieter van Melkebeek. 2017. Nondeterministic circuit lower bounds from mildly derandomizing Arthur-Merlin games. *Computational Complexity* 26(1):79–118.

[12] Babai, László, Lance Fortnow, and Carsten Lund. 1991. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity* 1:3–40.

[13] Babai, László, Peter Frankl, and Janos Simon. 1986. Complexity classes in communication complexity theory (preliminary version). In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 337–347.

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

[15] Baker, Theodore P., John Gill, and Robert Solovay. 1975. Relativizations of the P =? NP question. *SIAM Journal on Computing* 4(4):431–442.

[16] Beigel, Richard, Harry Buhrman, and Lance Fortnow. 1998. NP might not be as easy as detecting unique solutions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 203–208.

[17] Beigel, Richard, and Alexis Maciel. 1999. Circuit lower bounds collapse relativized complexity classes. In *Proceedings of the IEEE Conference on Computational Complexity*, 222–226.

[18] Buhrman, Harry, Stephen A. Fenner, Lance Fortnow, and Leen Torenvliet. 2001. Two oracles that force a big crunch. *Computational Complexity* 10(2): 93–116.

[19] Buhrman, Harry, Lance Fortnow, and Thomas Thierauf. 1998. Nonrelativizing separations. In *Proceedings of the IEEE Conference on Computational Complexity*, 8–12.

[20] Buhrman, Harry, and Leen Torenvliet. 2000. Randomness is hard. *SIAM Journal on Computing* 30(5):1485–1501.

[21] Cai, Jin-Yi. 2007. $S_2^P \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences* 73(1): 25–35.

[22] Cai, Jin-Yi, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. 2005. Competing provers yield improved Karp-Lipton collapse results. *Information and Computation* 198(1):1–23.

[23] Chow, Timothy (http://mathoverflow.net/users/3106/timothy-chow). Definition of relativization of complexity class. MathOverflow. URL: http://mathoverflow.net/q/76021 (version: 2011-09-21).

[24] Cobham, Alan. 1964. The intrinsic computational difficulty of functions. In *Proceedings of the International Congress for Logic, Methodology, and Philosophy of Science II*, 24–30.

[25] Cook, Stephen A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 151–158.

[26] DeMillo, Richard A., and Richard J. Lipton. 1980. The consistency of "P = NP" and related problems with fragments of number theory. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 45–57.

[27] Enderton, Herbert B. 1977. *Elements of Set Theory*. Academic Press.

[28] Feige, Uriel, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. 1996. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM* 43(2):268–292.

[29] Fortnow, Lance. 1994. The role of relativization in complexity theory. *Bulletin of the EATCS* 52:229–243.

[30] ———. 2016. [Blog post: The great oracle debate of 1993]. Retrieved from http://blog.computationalcomplexity.org/2009/06/great-oracle-debate-of-1993.html.

[31] Fortnow, Lance, and Michael Sipser. 1988. Are there interactive protocols for coNP languages? *Information Processing Letters* 28(5):249–251.

[32] Furer, Martin, Oded Goldreich, Yishay Mansour, Michael Sipser, and Statis Zachos. 1989. On completeness and soundness in interactive proof systems. *Advances in Computing Research: A Research Annual, vol. 5 (Randomness and Computation, S. Micali, ed.)*.

[33] Furst, Merrick L., James B. Saxe, and Michael Sipser. 1984. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory* 17(1):13–27.

[34] Goldreich, Oded. 1999. Lecture 26: Relativization. Introduction to Complexity Theory – Lecture Notes for a Two-Semester Course. Retrieved from http://www.wisdom.weizmann.ac.il/~oded/cc99.html.

[35] Goldwasser, Shafi, and Michael Sipser. 1986. Private coins versus public coins in interactive proof systems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 59–68.

[36] Heller, Hans. 1986. On relativized exponential and probabilistic complexity classes. *Information and Control* 71(3):231–243.

[37] Impagliazzo, Russell, Valentine Kabanets, and Antonina Kolokolova. 2009. An axiomatic approach to algebrization. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 695–704.

[38] Impagliazzo, Russell, Valentine Kabanets, and Avi Wigderson. 2002. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences* 65(4):672–694.

[39] Kannan, Ravi. 1982. Circuit-size lower bounds and nonreducibility to sparse sets. *Information and Control* 55(1):40–56.

[40] Karp, Richard M., and Richard J. Lipton. 1982. Turing machines that take advice. *L'Enseignement Mathématique* 28(2):191–209.

[41] Kolata, Gina. "New shortcut found for long math proofs". *New York Times* 4 April 1992.

[42] Kothari, Robin (cstheory.stackexchange.com/users/206/robin-kothari). Is relativization well-defined? Theoretical Computer Science Stack Exchange. URL: http://cstheory.stackexchange.com/q/21606 (version: 2014-03-18).

[43] Lautemann, Clemens. 1983. BPP and the polynomial hierarchy. *Information Processing Letters* 17(4):215–217.

[44] Levin, Leonid. 1973. Universal sequential search problems. *Problemy Peredachi Informatsii* 9(3):265–266.

[45] Lund, Carsten, Lance Fortnow, Howard J. Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *Journal of the ACM* 39(4): 859–868.

[46] Nikolov, Sasho (https://cstheory.stackexchange.com/users/4896/sasho-nikolov). What are natural examples of non-relativizable proofs? Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/20515 (version: 2014-01-27).

[47] Russell, Alexander, and Ravi Sundaram. 1998. Symmetric alternation captures bpp. *Computational Complexity* 7(2):152–162.

[48] Shamir, Adi. 1992. IP = PSPACE. *Journal of the ACM* 39(4):869–877.

[49] Shoup, Victor. 1990. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation* 54(189):435–447.

[50] Stockmeyer, Larry J., and Albert R. Meyer. 1973. Word problems requiring exponential time: Preliminary report. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1–9.

[51] Toda, Seinosuke. 1989. On the computational power of PP and +P. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 514–519.

[52] Williams, Ryan. 2010. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 231–240.

[53] ———. 2014. Nonuniform ACC circuit lower bounds. *Journal of the ACM* 61(1):2:1–2:32.

[54] Wilson, Christopher B. 1985. Relativized circuit complexity. *Journal of Computer and System Sciences* 31(2):169–181.

[55] Yap, Chee-Keng. 1983. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science* 26:287–300.

[56] Zachos, Stathis. 1988. Probabilistic quantifiers and games. *Journal of Computer and System Sciences* 36(3):433–451.