Interactive Proofs in Bounded Arithmetics

by
Martín Soto

supervised by
Albert Atserias

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Sciences
(Mathematical Logic)

at the
University of Barcelona
2024

Interactive Proofs in Bounded Arithmetics

Martín Soto

Abstract

Previous work [3] has shown that V^0_2, the theory of bounded arithmetic in Buss' language equipped with comprehension for boundedly definable sets, is consistent with the conjecture $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. That work entertains two different formalizations of the inclusion $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ inside V^0_2, termed α and β. Both formalizations are provably equivalent in the standard model of arithmetic, by invoking the Easy Witness Lemma (EWL), a technically deep modern result in complexity theory. While the implication $\beta \to \alpha$ is provable in V^0_2, it is open whether V^0_2 proves the converse implication $\alpha \to \beta$. Since this converse implication can be interpreted as a formalization of the EWL, whether V^0_2 proves the equivalence of the two formalizations amounts to whether V^0_2 proves (this formalization of) the EWL.

In the present work, we make progress towards resolving this question in the positive. More concretely, we show that $\mathsf{V}^0_2 + \alpha$ does prove a suitable formalization of $\mathsf{IP} = \mathsf{PSPACE}$, which is a central ingredient in the proof of the EWL. In the process of doing so, we lay the foundations necessary to discuss exact counting of large sets and formalization of interactive proofs in V^0_2 and other second-order bounded arithmetics.

Agraïments

Estic enormement agraït al meu tutor Albert Atserias, per tota la seva ajuda, per la claredat del seu pensament i explicacions, pel seu esforç en apropar-me a mi i altres alumnes al meravollós món de la complexitat computacional, i pel seu bon humor i amabilitat. Vull agraïr-li en especial la seva constant paciència i altruisme, malgrat els canvis de plans que en ocasions m'he vist obligat a demanar.

Voldria també aprofitar aquesta oportunitat per agraïr als professors i les meves companyes de classe una experiència educativa excepcional. Heu estat fonts d'ajuda i felicitat. Hauria desitjat poder compartir encara més experiències amb vosaltres, però sé que ens tornarem a veure aviat.

I finalment, a Douglas Hofstadter, per fer-me enamorar de la lògica fa 7 anys.

Contents

Introduction

Meta-complexity

It is infamously hard to prove inclusions, equalities or inequalities between complexity classes. Since the seminal works on computational complexity in the 1970s, and barring a few surprising flagship results that can probably be counted without resorting to one's toes, many deceptively simple questions, querying the most fundamental properties of the computational universe, remain not only unsolved by the existing incremental progress, but in fact evidently far from any solution. The most famous of these is undoubtedly the P vs NP problem. But there exist a truckload of other conjectures that are, in a sense, even more egregiously unproven. Many inequalities between radically efficient and radically expensive classes of computations that, while strongly believed, we still don't have the mathematical tools to assert. For a famous example of this, at the time of writing of this thesis, it is still open whether $\mathsf{NEXP} \subseteq \mathsf{P/poly}$: can any problem solved in exponential time by a non-deterministic Turing machine also be solved by polynomial-size circuits?

It is thus not surprising that in recent decades a sub-field has crystallized that jumps one level up, and asks the question we're all wondering about:

Why are complexity results so hard to prove?

Research in the *meta-mathematics of complexity theory* brings forth perspectives from mathematical logic and proof theory to study why, and to what extent, such results are infeasible, or even impossible, to prove from our working axioms. As an exemplification,

an ultimate and ambitious goal from this field could be to show that $\mathsf{P} = \mathsf{NP}$ is independent of the axioms we use to formalize most of mathematics. But of course, it is entirely plausible that human mathematicians' failure is explained by way less fundamental and bombastic reasons. And even if such ambitious consistency results did hold, while they remain unattainable we must content ourselves with smaller wins.

Albeit here we'll be focusing on this meta-theoretical direction, the recently popular field termed *meta-complexity* also encompasses an alternative complimentary perspective. Instead of jumping a meta-level up by studying the complexity of the proofs we use to study complexity, we could also introduce complexity one level down, and study the complexity of problems *that are themselves about complexity*. One such example is the Minimum Circuit Size Problem (MCSP): given the input-output table of a Boolean function, what's the smallest Boolean circuit implementing it? The topic of this decision problem is the complexity of a Boolean function. And of course, we can ask about the complexity of MCSP itself. It is not surprising to find out that this complementary branch in fact presents many interesting connections to the proof-theoretic side of meta-complexity.

Bounded arithmetics

One common tool in meta-complexity (and more generally proof complexity) is the use of bounded arithmetics, an array of weak sub-theories of Peano Arithmetic through which to study different notions of feasible reasoning. These theories, initiated by Parikh in 1971 and developed further by Buss in his 1986 dissertation, are characterized by restricting the induction axiom of PA, so that we can only induct on formulas up to a certain complexity (usually meaning amount of quantifiers).

The key useful feature of bounded arithmetics is their ability to characterize complexity classes. Many results are of the following form: a function is provably total in a given theory if and only if it belongs to a corresponding complexity class. This connection allows for

a systematic study of complexity classes through proof-theoretic methods. For instance, Buss's first-order theories S_2^i form a hierarchy that corresponds to the polynomial-time hierarchy PH in complexity theory.

Another significant result in this field is Buss's witnessing theorem, which establishes that $\forall\Sigma_1^b$ theorems of S_2^1 are witnessed by polynomial-time functions. That is, the existential witness making the Σ_1^b formula true can be computed by a polynomial-time function.

Bridges have also been built between (first-order) bounded arithmetics and propositional proof systems. This correspondence allows for the translation of proofs in bounded arithmetic into short proofs in propositional systems, particularly useful for constructing efficient proofs in propositional systems like *Extended Frege*.

This framework offers a structured approach to understanding relationships between complexity classes and the limits of provability. By studying what can and cannot be proved in various theories of bounded arithmetic, we gain insights into why certain complexity results are challenging to establish, as is the agenda of meta-complexity.

While some of these results pertain to the exact length or complexity of proofs, in the present work we put the focus on provability and consistency results, without regard for the low-level details of the proofs involved. This is because we will be dealing with significantly complex complexity-theoretic statements (like $\mathsf{IP} = \mathsf{PSPACE}$ or the Easy Witness Lemma), and correspondingly using a rather strong bounded arithmetic (the theory V_2^0), such that even results on provability are already of high interest, and advance the frontier of our understanding.

This work

The contents of this thesis are best understood as an extension of previous work in [3]. There, the authors showed that the second-order bounded arithmetic V^0_2 is consistent with the conjecture $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. We will explain in more technical detail in Chapter 2, but in short, the fact that this moderately strong theory doesn't disprove the conjecture could be understood as heuristic evidence in favor of the conjecture holding "in the real world" (that is, the standard model of arithmetic).

In fact, it is not immediately obvious how to formalize a conjecture like $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ in the language of bounded arithmetic, and this previous work [3] presents two different formalizations $\neg\alpha$ and $\neg\beta$. In the standard model of arithmetic, we know they are both equivalent. But we don't know yet $\alpha \leftrightarrow \beta$ can be proved inside V^0_2. In fact, the statement $\alpha \leftrightarrow \beta$ is equivalent to the Easy Witness Lemma from [7] (in the form of Theorem 3.1 from [15]), another deeply technical and very celebrated result of modern complexity theory.

The main motivation for our work is showing that indeed V^0_2 proves $\alpha \leftrightarrow \beta$ (or equivalently, the Easy Witness Lemma). This would be desirable for several different technical reasons (see Section 2.2), but mainly, it would provide further evidence for the strength of V^0_2, and enable the transfer of certain previous results and existing tools to new settings.

In this work we make progress towards this longer term goal. A main ingredient of the proof of the Easy Witness Lemma is $\mathsf{IP} = \mathsf{PSPACE}$ [13], another famous and celebrated result in modern complexity theory. The focus of this thesis is to show that $\mathsf{V}^0_2 + \alpha$ proves (a suitable formalization of) this result.

In the process of doing so, we also lay the foundations for exact probabilistic reasoning in V^0_2 on exponentially-large event spaces (which really amounts to reasoning about the sizes of exponential-size sets), and use these tools to formalize inside V^0_2 the interactive

proof protocols involved in IP. Most of these techniques transfer to other second-order bounded arithmetics. This is presented in Chapter 3, which is an independent technical contribution not relying on α as an assumption.

The rest of the thesis is structured as follows. In Chapter 1, we present necessary preliminaries on complexity classes, the proof of $\mathsf{IP} = \mathsf{PSPACE}$ (that we will be reconstructing inside V_2^0), and bounded arithmetics. In Chapter 2, we contextualize in all necessary technical detail the previous results of [3], and discuss precisely the new results we are working towards. In Chapter 3, we lay out the foundations for probabilistic reasoning inside V_2^0 that we will need to work with $\mathsf{IP} = \mathsf{PSPACE}$. In Chapter 4, we present our formalization of $\mathsf{IP} = \mathsf{PSPACE}$, and prove it inside V_2^0, which is our main result.

A previous publication that was brought to our attention ([10]) already dealt with $\mathsf{IP} = \mathsf{PSPACE}$ in the context of bounded arithmetics. More concretely, one of its results is that the theory $\mathsf{S}_2^1 + 1\text{-Exp}$ (which is equivalent to V_2^1, see [12][Theorem 5.5.16]) correctly proves the basic properties of the Sumcheck Protocol, which is the central piece of the proof of $\mathsf{IP} = \mathsf{PSPACE}$. Our results are indeed different: we are using $\mathsf{V}_2^0 + \alpha$, and also correspondingly need to use different techniques. Indeed, the main ingredient that makes our result possible (already present in [3], and as highlighted in our proofs below) is using α to turn a high-complexity induction (that would usually require V_2^1 to perform) into a low-complexity one (that V_2^0 can perform).
We should also note that the theory $\mathsf{V}_2^0 + \alpha$ is not necessarily interesting *by itself* (since indeed it is strongly believed that the conjecture $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ is true, and thus its negation α is believed to be false), but rather as a useful jumping point from which to prove certain collapses or unconditional results in V_2^0, like $\alpha \to \beta$.

Chapter 1

Preliminaries

1.1 Complexity classes

We assume basic familiarity with the Turing machine paradigm, and complexity classes of computational problems. Let us begin by recapitulating the definitions of some complexity classes that play a major role in this work. The first four are relatively straightforward.

Definition 1.1.1 (Polynomial Time)**.** P *is the class of decision problems that can be solved by a deterministic Turing machine in polynomial time. Formally,*

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$$

where $\mathsf{TIME}(f(n))$ *is the set of decision problems solvable by a deterministic Turing machine in* $O(f(n))$ *computational steps.*

Definition 1.1.2 (Exponential Time)**.** EXP *is the class of decision problems that can be solved by a deterministic Turing machine in exponential time. Formally,*

$$\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(2^{n^k})$$

It is known that $\mathsf{P} \subset \mathsf{EXP}$, with the containment being strict.

Definition 1.1.3 (Polynomial Space)**.** PSPACE *is the class of decision problems that can be solved by a deterministic Turing machine using a polynomial amount of memory. Formally,*

$$\mathsf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathsf{SPACE}(n^k)$$

where $\mathsf{SPACE}(f(n))$ *is the set of decision problems solvable by a deterministic Turing machine that only ever writes in the first* $O(f(n))$ *squares of its tape.*

It is known that $\mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$.

Definition 1.1.4 (Non-deterministic Exponential Time)**.** *A non-deterministic Turing machine makes its computation depend on a random seed (of length polynomial on that of the input), and accepts if any of the computational paths thus generated does.* NEXP *is the class of decision problems that can be solved by a non-deterministic Turing machine in exponential time. Formally,*

$$\mathsf{NEXP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(2^{n^k})$$

where $\mathsf{NTIME}(f(n))$ *is the set of decision problems solvable by a non-deterministic Turing machine in* $O(f(n))$ *computational steps.*

It is immediate from the definition that $\mathsf{EXP} \subseteq \mathsf{NEXP}$.

We now present two classes with slightly more complex definitions.
The first one, $\mathsf{P/poly}$, identifies problems that can be solved by small Boolean circuits.

Definition 1.1.5 (Boolean circuit)**.** *A Boolean circuit is a labeled directed acyclic graph where:*

- *Nodes with in-degree 0 are labeled as inputs.*
- *Nodes with in-degree greater than 0 are gates, labeled with one of three Boolean operations: AND (in-degree 2), OR (in-degree 2), NOT (in-degree 1).*

- *One node is labeled as the output.*

The size of a circuit is the number of gates it contains. The depth of a circuit is the length of the longest path from an input to the output.
Given a Boolean circuit C with n input nodes, and an input x of length n, the bit $C(x)$ is computed in the obvious way, by applying the Boolean operation with which each gate is labeled to its inputs, and reading the output.

Definition 1.1.6 (Non-uniform Polynomial Size)**.** $\mathsf{P/poly}$ *is the class of decision problems solvable by polynomial-sized Boolean circuits. Formally, a language L is in* $\mathsf{P/poly}$ *if there exists a sequence of Boolean circuits $\{C_n\}_{n\in\mathbb{N}}$ such that:*

- *For each n, C_n has n inputs and one output.*
- *There exists a polynomial p such that for all n, the size of C_n is at most $p(n)$.*
- *For all strings x of length n, we have $x \in L \leftrightarrow C_n(x) = 1$.*

$\mathsf{P/poly}$ and similar classes are usually referred to as *non-uniform*, meaning that problem inputs of different lengths are handled by different circuits (as opposed to using a single Turing machine for all lengths). Because of this non-uniformity, $\mathsf{P/poly}$ doesn't correspond to our intuitive notion of efficient computation, despite being labeled as polynomial. For example, it contains every *undecidable* unary language, which of course can't be solved by any one real-world machine, regardless of its computational strength.

The second class, IP, captures the problems that can be solved by an efficient trusted reasoner with the help of a more powerful, but also untrusted, helper. More concretely, the trusted reasoner will be a polynomial-time verifier V, and the untrusted one any arbitrary prover function P (without regarding computational limitations). They will engage in an interactive proof, exchanging messages back and forth, and V will be able to employ randomization of its messages to keep P truthful.

Definition 1.1.7 (Interactive Proof System)**.** *An interactive proof is an interaction between a (fixed throughout) non-deterministic machine V and a (fixed throughout) arbitrary function P. The interaction proceeds in rounds, where in each round:*

- *The verifier V sends a message (question) to the prover, possibly depending on its random seed.*
- *The prover P sends a response back to the verifier.*

After a polynomially bounded (on the length of the problem input) number of rounds, the verifier decides to accept or reject.
We say that a decision problem L is solved by the verifier V if:

1. ***Completeness:*** *For every $x \in L$, there exists a prover function P such that V accepts with probability at least 2/3.*
2. ***Soundness:*** *For every $x \notin L$, and for every prover function P, V accepts with probability at most 1/3.*

Definition 1.1.8 (IP)**.** IP *is the class of languages solved by a polynomial verifier V.*

IP is a powerful class, allowing for multiple rounds of interaction and a probabilistic verifier. And yet, the verifier can only ever perform polynomial checks, which we would expect to be a strong bottleneck.
But a surprising and important result in complexity theory is that IP = PSPACE. This means that polynomial verifiers can verify any problem solvable in polynomial space, which includes many problems believed to be much harder than those in P, or even NP (Nondeterministic Polynomial Time).

Indeed, the result IP = PSPACE is a central element of this thesis (although the exact reason for this will not be fully explained until Section 2.2). We turn now to its proof.

1.2 IP = PSPACE

We present here a proof that IP = PSPACE, reconstructed mainly from [2].
Our main result (Theorem 4.1.2), informally stated, is that the bounded arithmetic V_2^0 (defined below) can reproduce this proof, so familiarity with its details will be required.

Theorem 1.2.1 (Originally in [13])**.** IP = PSPACE

Proof.

The hard inclusion: PSPACE ⊆ IP

A Quantified Boolean Formula is a Boolean formula (the matrix) preceded by an array of Boolean quantifiers. Each quantifier is either existential $\exists_x$ or universal $\forall_x$, bearing the usual meanings, with x varying of course only between the two truth values 1 and 0. For example,

$$\exists_{x_1} \forall_{x_2} x_1 \wedge x_2 \wedge x_3$$

is a *partly* Quantified Boolean Formula, due to having x_3 as a free variable, and will be false for any value of x_3. On the contrary,

$$\forall_{x_1} \exists_{x_2} x_1 \vee x_2$$

is a *fully* Quantified Boolean Formula, due to having no free variables, and is true.

Consider the decision problem TQBF: for any Quantified Boolean Formula, determine its truth value. This problem is PSPACE-complete, which means any PSPACE problem reduces to it in polynomial time. This was shown by Stockmeyer ([14]) by, given an arbitrary PSPACE machine, constructing a (fully) Quantified Boolean Formula whose truth value is equivalent to the acceptance of the machine.
Since, as can be easily verified, the class IP is closed downwards under polynomial-time reductions, by showing TQBF ∈ IP we will have shown that PSPACE ⊆ IP.

We thus need to construct a verifier protocol that solves TQBF. For this we need the concept of **arithmetization**. By arithmetizing a formula, we mean turning any Quantified Boolean Formula φ into a polynomial $A(\varphi)$, whose values on input a Boolean assignment (that is, an assignment including only 0s and 1s), and computed in fields of high enough characteristic, correspond with the truth values of the formula on the same Boolean assignment. As we will see later, this polynomial representation will allow our verifier to more efficiently exploit the prover.

We arithmetize the Boolean connectors as follows:

$$\begin{aligned} x \wedge y &\longleftrightarrow X \cdot Y \\ \neg x &\longleftrightarrow 1 - X \\ x \vee y &\longleftrightarrow 1 - (1 - X)(1 - Y) \\ \exists_x \varphi(x) &\longleftrightarrow \sum_{X \in \{0,1\}} A(\varphi)(X) \\ \forall_x \varphi(x) &\longleftrightarrow \prod_{X \in \{0,1\}} A(\varphi)(X) \end{aligned}$$

And thus, for example

$$\exists_x \forall_y x \vee y \vee \neg z \longleftrightarrow \sum_{X \in \{0,1\}} \prod_{Y \in \{0,1\}} 1 - (1 - X)(1 - Y)Z$$

which is a polynomial on the variable Z. Indeed, just like Quantified Boolean Formulas, the quantified polynomials resulting from the arithmetization can be either partly or fully quantified.

It is easy to see that, for any $\varphi(\bar{x})$ and input $\bar{x}$, $A(\varphi)(\bar{x}) = 0 \leftrightarrow \neg\varphi(\bar{x})$. Although, when we have $\varphi(\bar{x})$, $A(\varphi)(\bar{x})$ might take any value ≥ 1, due to the summations.

Since this transformation is computable in polynomial time, our verifier V can reduce the

problem of deciding whether any given φ is true, to deciding a polynomial equality of the form[1]

$$\sum_{X_1\in\{0,1\}} \prod_{X_2\in\{0,1\}} \sum_{X_3\in\{0,1\}} \cdots \prod_{X_n\in\{0,1\}} g(X_1,\ldots,X_n) = k \qquad (\star)$$

Note, however, that what we can compute in polynomial time (from a formula φ) is the polynomial expression with quantifiers still present (like $(\star)$), and *not* the simplified expression of that same polynomial, once we have operated out all the sums and products and expressed the result as an explicit sum of monomials (like, for example, $X_1 \cdot X_3 + 4 \cdot X_2$, without any quantifiers in sight).

Indeed, turning a quantified expression like $(\star)$ into an explicit sum of monomials is PSPACE-hard, and in our protocol we will be taking advantage of the prover's power to compute the explicit expression.

We now propose a protocol to verify whether an expression like $(\star)$ holds.

It will be useful to perform all computations in a suitable prime field $\mathbb{F}_z$ (instead of the integers). Evaluated over boolean assignments, the maximal possible value of our quantified polynomial is 2^n, and if we choose z greater than that, we are sure that the evaluation over $\mathbb{F}_z$ will coincide with the evaluation over the integers. So to start, the prover sends a prime $z \in (2^n, 2^{2n}]$, and the verifier checks in polynomial time its primality. Then we can start the protocol (where all computations are modulo z):

[1] By adding a polynomial number of dummy variables if necessary, we may assume that the quantifiers alternate, the first one being a sum and last one a product, or vice versa. Here we describe the first case.

Sumcheck protocol

V: If $n = 1$, check that $g(1) + g(0) = k$ (or $g(0) \cdot g(1) = k$ if the last quantifier was a product). If so accept, otherwise reject.

If $n \geq 2$, ask P to send the coefficients (in $\mathbb{F}_z$) of the following uni-variate polynomial:

$$h(X_1) := \prod_{X_2 \in \{0,1\}} \sum_{X_3 \in \{0,1\}} \cdots \prod_{X_n \in \{0,1\}} g(X_1, X_2, \ldots, X_n)$$

P: Sends the coefficients of some polynomial $s(X_1)$

(if the prover is not "cheating", we will have $s(X_1) = h(X_1)$).

V: Reject if $s(0)+s(1) \neq k$ (or $s(0)\cdot s(1) \neq k$ if the first quantifier was a product).

Otherwise, pick a random $r \in \mathbb{F}_z$. Compute $k' := s(r)$.

Recursively use the same protocol to check the new polynomial equality:

$$\prod_{X_2 \in \{0,1\}} \sum_{X_3 \in \{0,1\}} \cdots \prod_{X_n \in \{0,1\}} g(r, X_2, \ldots, X_n) = k'$$

We now check the completeness and soundness of our protocol.

Completeness:

If the claim to prove ($\star$) is true, the prover that always returns the coefficients of the true polynomial $h(X_1)$ will always convince the verifier, since all its checks pass for the true polynomial.

Soundness:

Assume the claim to prove is false. Say d is a bound on the degree of any polynomial $h(X_1)$ during the whole procedure (we will come back later to what value d needs to take).

We will prove that

$$\Pr[V \text{ rejects}] \geq (1 - \frac{d}{z})^n \tag{1}$$

If d is polynomial on n, since z is exponential on n, the right hand side will go to 1 for large enough n, and in particular, be over $\frac{2}{3}$ as required.

We will prove (1) by induction on n.

For $n = 1$, V simply evaluates the claim directly, and so rejects with probability 1 if the claim is false.

Now assume the claim is true for polynomials of up to degree d and $n - 1$ variables.

If P returns the true polynomial $h(X_1)$, V will perform the corresponding check for $h(0) + h(1) = k$ (or $h(0) \cdot h(1) = k$ if the first quantifier was a product), which will come out false by assumption, and lead to immediate rejection. So assume the prover returns a different $s(X_1)$.

Since the degree $\leq d$ non-zero polynomial $s(X_1) - h(X_1)$ has at most d roots, there are at most d values r in $\mathbb{F}_z$ with $s(r) = h(r)$. Thus, when V picks a random $r \in \mathbb{F}_z$,

$$\Pr_r[s(r) \neq h(r)] \geq (1 - \frac{d}{z})$$

If indeed $s(r) \neq h(r)$, then the prover is left with another false claim to prove. And by induction hypothesis, it will fail with probability $\geq (1 - \frac{d}{z})^{n-1}$. Thus

$$\Pr[V \text{ rejects}] \geq (1 - \frac{d}{z})^{n-1} \cdot (1 - \frac{d}{z}) = (1 - \frac{d}{z})^n$$

finishing the induction.

The above approach is correct except for one tiny detail: the bound d on the degree of the polynomials is not guaranteed to be as small as we need it to be. Indeed, since our arithmetization includes products, the degree of the true polynomial could grow exponen-

tially up to 2^n. Such polynomials could have up to 2^n coefficients, and thus the verifier wouldn't even be able to read them in time polynomial on n.

The fix for this is to change our arithmetization procedure slightly, by introducing a new kind of quantifier: a **linearization** L_{X_i}.

We ultimately only care about the polynomial's values on Boolean assignments. Since $0^m = 0$ and $1^m = 1$ for any $m \geq 1$, we can take a polynomial $g(X_1, \ldots, X_n)$ of arbitrarily high degree, and transform it into one where a certain variable X_i is never raised to a power greater than 1, while still agreeing with g on all inputs with $X_i \in \{0, 1\}$. Indeed, this new polynomial can be defined simply as

$$L_{X_i}\, g(X_1, \ldots, X_n) := (1 - X_i) \cdot g(\overbrace{X_1, \ldots, 0}^{i}, \ldots, X_n) + X_i \cdot g(\overbrace{X_1, \ldots, 1}^{i}, \ldots, X_n)$$

Notice that this new polynomial still has X_i as a free variable. We can apply this sequentially for all variables. For example, we would turn

$$X_1^3 \cdot X_2 + X_3^2 + X_1 \cdot X_2$$

into

$$2 \cdot X_1 \cdot X_2 + X_3$$

The resulting polynomial will have degree at most n (from the maximal factor $X_1 \cdot \ldots \cdot X_n$).

The way to correspondingly alter our arithmetization procedure to ensure a low d is as follows. After each regular quantifier (which might have increased the polynomial degree if it was a product), add one linearization quantifier L_{X_i} for each variable X_i that is free in the matrix of the quantifier (at most n). For example, instead of

$$\sum_{X_1 \in \{0,1\}} \prod_{X_2 \in \{0,1\}} \sum_{X_3 \in \{0,1\}} g(X_1, X_2, X_3)$$

we would have

$$\sum_{X_1 \in \{0,1\}} L_{X_1} \prod_{X_2 \in \{0,1\}} L_{X_1} L_{X_2} \sum_{X_3 \in \{0,1\}} L_{X_1} L_{X_2} L_{X_3} g(X_1, X_2, X_3)$$

The resulting quantified polynomial expression still has length polynomial in n.

Then, in our sumcheck protocol, we treat these "linearization" quantifiers the same way we were treating the normal quantifiers. To prove $(L_{X_1}\ g(X_1))(r_1) = k$, we ask the prover to send the coefficients of the polynomial $g(X_1)$. Then, with whichever $s(X_1)$ the prover sends, we execute the check for

$$(1 - r_1) \cdot s(0) + r_1 \cdot s(1) = k$$

These incremental checks (as opposed to executing the linearization all at once) again ensure the prover cannot lie too easily. The rest of the protocol functions as before.

The easy inclusion: $\mathsf{IP} \subseteq \mathsf{PSPACE}$

For this it is enough to see that, for any problem $L \in \mathsf{IP}$ and verifier V solving it, the prover function maximizing V's acceptance probability is computable by a PSPACE machine. Indeed, if we have this, then for any $L \in \mathsf{IP}$ we can simply run the polynomial-time (and -space) interaction between V and this optimal P (on all random seeds), obtain V's probability of acceptance, and then accept if this probability is $\geq \frac{2}{3}$ or reject if it is $\leq \frac{1}{3}$.

We now describe the PSPACE machine M that computes the optimal prover.
M takes a verifier V, a claim to prove ($\star$), and a conversation history of messages (polynomials) $\bar{m}$ sent by the prover and random elements of $\mathbb{F}_z$ $\bar{r}$ chosen by the verifier.
M returns both an optimal message m, and the probability p with which V will accept (in the whole rest of the protocol) if it is sent.
To do this, M loops over all possible messages m and, for each, computes the probability p

with which the conversation history $\bar{m}^\frown m$ will be accepted. To compute p, M recursively calls on itself to decide, for every possible immediately next random choice $r \in \mathbb{F}_z$ from the verifier, the probability with which the history $\bar{m}^\frown m$ with $\bar{r}^\frown r$ is accepted, and then averages across all these r. Finally, M chooses the m with highest such p, and returns m and p.

This loop is possible in polynomial space by erasing and repurposing the same space to compute each separate recursive call, since M only needs to keep stored the best (m, p) pair seen until now (instead of all of the ones it has cycled through), and this pair (like any polynomial sent at any point during the protocol) has polynomial length. □

As it turns out, to reproduce this proof within V_2^0 we will need to implement a small change that makes z polynomial in n instead of exponential in n (see Theorem 4.1.2).

1.3 Bounded arithmetic

We assume basic familiarity with first-order logic and Peano Arithmetic (PA).

Bounded arithmetics (BAs) are weak fragments of PA, obtained by restricting PA's induction schema to formulas of a certain limited complexity.[2] They are meant to capture different classes of *feasible* computational strength. Our presentation is based on [6], [12] and [5].

To start with, bounded arithmetics employ an extension of the language of PA:

Definition 1.3.1 (BA language)**.** *In addition to the symbols* 0*,* S*,* $+$*,* $\cdot$ *and* $\leq$ *from PA, the language of bounded arithmetics includes:*

- $|x|$*, representing the length of* x*'s binary representation*
- $x\#y$*, representing* $2^{|x|\cdot|y|}$*, so that* $|x\#y| = |x| \cdot |y| + 1$
- $\mathrm{MSP}(x, i)$*, representing* $\lfloor x/2^i \rfloor$*, i.e. the "most significant part"*

[2] Strictly speaking, they are not fragments of PA due to their expanded language. But their restrictions to the language of PA are, and it's most useful to think of them through this lens.

All BAs share a core set of axioms, called BASIC, which pins down the basic properties of all these symbols, just like the non-induction axioms of PA. (The reader is advised to only skim them at this point.)

Definition 1.3.2 (BASIC)**.** *The set of axioms* BASIC *contains the following axioms:*

1. $a \leq b \to a \leq b + 1$
2. $a \neq a + 1$
3. $0 \leq a$
4. $(a \leq b \wedge a \neq b) \to a + 1 \leq b$
5. $a \neq 0 \to 2a \neq 0$
6. $a \leq b \vee b \leq a$
7. $(a \leq b \wedge b \leq a) \to a = b$
8. $(a \leq b \wedge b \leq c) \to a \leq c$
9. $|0| = 0$
10. $a \neq 0 \to (|2a| = |a|+1 \wedge |2a+1| = |a|+1)$
11. $|1| = 1$
12. $a \leq b \to |a| \leq |b|$
13. $|a\#b| = |a| \cdot |b| + 1$
14. $0\#a = 1$
15. $a \neq 0 \to (1\#(2a) = 2(1\#a) \wedge 1\#(2a+1) = 2(1\#a))$
16. $a\#b = b\#a$
17. $|a| = |b| \to a\#c = b\#c$
18. $|a| = |b| + |c| \to a\#d = (b\#d) \cdot (c\#d)$
19. $a \leq a + b$
20. $(a < b \wedge a \neq b) \to (2a + 1 \leq 2b \wedge 2a + 1 \neq 2b)$
21. $a + b = b + a$
22. $a + 0 = a$
23. $a + (b + 1) = (a + b) + 1$
24. $(a + b) + c = a + (b + c)$
25. $a + b \leq a + c \to b \leq c$
26. $a \cdot 0 = 0$
27. $a \cdot (b + 1) = a \cdot b + a$
28. $a \cdot b = b \cdot a$
29. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
30. $1 \leq a \to ((a \cdot b \leq a \cdot c) \equiv (b \leq c))$
31. $a \neq 0 \to |a| = |[a/2]| + 1$
32. $a = [(b/2)] \leftrightarrow (2a = b \vee 2a + 1 = b)$

In PA, any first-order variable is either quantified or not. But in BAs, we find intermediate notions of quantification of different "strengths", meant to capture the degree of feasibility of a computation (as represented by a formula).

Definition 1.3.3 (Bounded quantifiers)**.**

- *A* bounded *quantifier is of the form* $\forall x \leq t$ *or* $\exists x \leq t$*, where* t *is a term (possibly with free variables).*

- *A* sharply bounded *quantifier is of the form* $\forall x \leq |t|$ *or* $\exists x \leq |t|$.
- *A formula is bounded or sharply bounded when all of its quantifiers are such.*

As we will see, bounded quantifiers are meant to capture polynomial computations, while sharply bounded capture logarithmic ones (that is, polynomial on length).
Another piece of handy notation is writing $n \in Log$ to mean n is sharply bounded by some term t, which we might not want to make explicit at all times, or also $n \in Log_{>1}$ excluding 0.

Definition 1.3.4 (First-order formula classes)**.**

- $\Delta_0^b = \Sigma_0^b = \Pi_0^b$ *is the class of sharply bounded formulas*
- Σ_{i+1}^b *is the closure of* Π_i^b *under existential bounded quantification and arbitrary sharply bounded quantification, modulo prenex operations.*
- Π_{i+1}^b *is defined dually.*

To define BAs of different strengths, there exist an array of axioms parameterized by the complexity of the formulas they apply to.

Definition 1.3.5 (Induction and comprehension)**.** *For a class of formulas* Φ*, we define the following axiom schemas for* $\varphi \in \Phi$*:*

- *Induction,*

$$\Phi\text{-}IND: \qquad \varphi(0) \wedge (\forall x)(\varphi(x) \to \varphi(x+1)) \to (\forall x)\varphi(x)$$

- *Polynomial induction,*

$$\Phi\text{-}PIND: \qquad \varphi(0) \wedge (\forall x)(\varphi(\lfloor \frac{1}{2}x \rfloor) \to \varphi(x)) \to (\forall x)\varphi(x)$$

- *Length induction,*

$$\Phi\text{-}LIND: \qquad \varphi(0) \wedge (\forall x)(\varphi(x) \to \varphi(x+1)) \to (\forall x)\varphi(|x|)$$

$PIND$ requires a stronger assumption, in which an induction step adds 1 to the length of x (instead of x itself), while $LIND$ just provides a weaker consequent.

Definition 1.3.6 (First-order BAs)**.**

- S_2^i : $\mathsf{BASIC} + \Sigma_i^b$-*PIND.*
- T_2^i : $\mathsf{BASIC} + \Sigma_i^b$-*IND.*
- $\mathsf{S}_2 = \bigcup_i \mathsf{S}_2^i$ *and* $\mathsf{T}_2 = \bigcup_i \mathsf{T}_2^i$

There are several basic results on the inclusions between these theories, or also the equivalences with other axioms (including minimization and replacement axioms). But we will be working with a theory stronger than T_2, so here we will gloss over these interesting details of first-order BAs.
It is useful to think about these theories through their corresponding complexity classes.

Theorem 1.3.7 (Some corresponding complexity classes)**.**

- *A function is provably total (and definable by a* Σ_1^b *formula) in* S_2^1 *iff it is in* P*.*
- *A function is provably total in* T_2 *iff it is in* PH *(the polynomial hierarchy).*

We will be working instead with second-order BAs, which introduce an additional class of quantifiable variables: sets.

Definition 1.3.8 (Second-order BA language)**.** *To express second-order theories, we extend the BA language with a class of second-order variables (X, Y, Z, ...) and corresponding quantifiers* $\forall_2$ *and* $\exists_2$*.*

Definition 1.3.9 (Second-order formula classes)**.**

- Bounded *second-order formulas are formulas of the second-order language all of whose first order quantifiers are bounded.*
- $\Sigma_0^{1,b}$ *formulas are bounded formulas without second-order quantifiers.*

- *The classes of $\Sigma_i^{1,b}$ formulas and $\Pi_i^{1,b}$ formulas are classes of bounded formulas defined analogously to the classes Σ_i^b and Π_i^b, counting the number of alternations of second-order quantifiers and not counting the first order quantifiers.*

In the BAs we will use, second-order variables (sets) represent an exponential amount of computation. This is because each set must have its elements bounded by a number term $t(x)$ guaranteed to exist (thus polynomial on x), and so the set can encode any number up to 2^t.

A useful mental model, then, is to think of variables or parameters as three-tiered depending on their size:

- we have logarithmically small numbers $n \in Log$ (for which the number 2^n is proven to exist),
- arbitrary numbers $x \sim 2^n$ (for which the existence of 2^x is not guaranteed),
- and arbitrary sets $X \sim 2^x \sim 2^{2^n}$ (each of whose elements is bounded by a number x)

Definition 1.3.10 (Second-order BAs)**.**

- *$I\Sigma_0^{1,b}$ is the theory composed of*
  - BASIC
  - *the extensionality axiom for sets*
  - *the axiom that all sets are bounded (that is, a term bounds all their elements)*
  - *$\Sigma_0^{1,b}$-comprehension*
- *V_2^i extends $I\Sigma_0^{1,b}$ by the axiom scheme $\Sigma_i^{1,b}$-IND.*

We will be focusing on V_2^0. Despite being second-order, this theory is not stronger than T_2 (when comparing them in the first-order language), since $\Sigma_0^{1,b} - IND$ is nothing more than $\Sigma_i^b - IND$ for all i.

It is sometimes also useful to refer to the second-order analogue of a first-order theory

(obtained simply by expressing it in the second-order language). We do this by appending the symbol (α). For example, $\mathsf{T}_2(\alpha)$.

A less obvious axiom that has played a crucial role in the development of second-order BAs, and also will in this work, is the Pigeon-Hole Principle. Several variants exist, but we will focus on the following formalization.

Definition 1.3.11 (Pigeon-Hole Principle)**.** *The* (functional) pigeon-hole principle $PHP(x)$ *is the following* $\Pi^{1,b}_1$*-formula:*

$$\begin{aligned}\forall_2 X \Big(&\exists y \le x+1\, \forall z \le x\, \neg\langle y,z\rangle \in X \quad \vee \\ &\exists y \le x+1\, \exists z \le x\, \exists z' \le x \left(\neg(z=z') \wedge \langle y,z\rangle \in X \wedge \langle y,z'\rangle \in X\right) \quad \vee \\ &\exists y \le x+1\, \exists y' \le x+1\, \exists z \le x \left(\neg(y=y') \wedge \langle y,z\rangle \in X \wedge \langle y',z\rangle \in X\right)\Big)\end{aligned}$$

Intuitively, this long formula is just stating that any purported injection (coded by X) from $x+1$ into x fails in one of three ways: either it is not total, or it is not a function, or it is not injective.

This principle is the object of one of the most celebrated lower bound results, termed the *gem theorem* of proof complexity.

Theorem 1.3.12 (Gem Theorem, [1])**.** V^0_2 *does not prove* $PHP(x)$*.*

That is, V^0_2 is not able to accurately compare the sizes of large sets (i.e., exponential in $n = |x|$). This result is highly non-trivial and technical, and is the main ingredient of some of our results below.

Chapter 2

Previous results

2.1 Previous consistency results

The previous publication we're building upon ([3]) showed the following main result. For a suitable formalization of the statement that $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ in second-order bounded arithmetic:

$$\text{it is consistent with } \mathsf{V}^0_2 \text{ that } \mathsf{NEXP} \not\subseteq \mathsf{P/poly}.$$

This is interesting because V^0_2 is a moderately strong theory, and we don't actually know whether $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ "in the real world" (the standard model of arithmetic). As per the usual meta-complexity agenda, this result seems to provide evidence that $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ is hard to prove (by showcasing that strong theories aren't yet able to prove it), and thus $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ more likely.

It is also notable that the result is unconditional, meaning it doesn't assume any unproven class inequality, as is the case with most previous consistency results.

This interesting and unconditional development was only made possible by the *Gem Theorem* (1.3.12), which is technically deep and highly non-trivial.

In more detail, it is not immediately obvious how to formalize a statement like $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ inside a BA where exponentiation is not provably total. The authors presented three main complementary formalizations, which also play a main role in this present

work. To state them, we first define some useful notation.

Definition 2.1.1 (New formula classes)**.**

- $\hat{\Sigma}^{1,b}_1$ *is the class of* $\exists_2\Pi^b_1(\alpha)$*-formulas, that is, those with the existential second-order quantifier in prenex position followed by a* $\Pi^b_1(\alpha)$*. [3]*
- $\Sigma^{1,b}_{1,s}$ *is the class of* $\Sigma^{1,b}_1$*-formulas without free second-order parameters, the s standing for "sentence". Combining this with the previous definition, we also define* $\hat{\Sigma}^{1,b}_{1,s}$ *as the class of* $\exists_2\Pi^b_1(\alpha)$*-formulas without free second-order parameters. And similarly for* $\Pi^{1,b}_{1,s}$ *and* $\hat{\Pi}^{1,b}_{1,s}$*.*

The definition of $\Sigma^{1,b}_i$ is standard, that of $\hat{\Sigma}^{1,b}_1$ was introduced for the results in [3], and that of $\Sigma^{1,b}_{1,s}$ we introduce now for the purposes of this work.

Recall that, in our second-order BAs, second-order variables represent an object of exponential size, or an exponential amount of computation. Because of that, we can identify $\Sigma^{1,b}_1$ formulas with NEXP problems: each such formula states the existence of an exponential-size witness. In more technical detail, what we mean is that *in the standard model of arithmetic* such formulas define exactly such problems. Restricting ourselves to $\hat{\Sigma}^{1,b}_{1,s}$ will further constrain the NEXP problems to be presented in the prenex form $\exists_2\Pi^b_1(\alpha)$ (which we know is always possible in the standard model of arithmetic), and without access to any oracles (external second-order parameters).

This identification straightforwardly motivates the first formalization of $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

Definition 2.1.2 (Definition 1 in [3])**.** *The* direct formalization *of a* NEXP *problem* $\varphi \in \hat{\Sigma}^{1,b}_{1,s}$ *being in* P/poly *is*

$$\alpha^c_\varphi \quad :\equiv \quad \forall n \in Log_{>1} \exists C \leq 2^{n^c} \forall x < 2^n (C(x) \leftrightarrow \varphi(x))$$

That is, for any length n which can be exponentiated, there exists a circuit C (of magnitude exponential on n, thus size polynomial on n) such that for all inputs x of that length, C solves the problem. The fixed numeral c determines the polynomial bound on the growth of the circuits.

Then, the result capturing the consistency of $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ in V^0_2 is as follows:

Theorem 2.1.3 (Direct formalization, Theorem 2 in [3])**.**
There exists $\varphi \in \hat{\Sigma}^{1,b}_{1,s}$ such that $\mathsf{V}^0_2 + \{\neg\alpha^c_\varphi \mid c \in \mathbb{N}\}$ is consistent.

The infinite set of sentences $\{\neg\alpha^c_\varphi \mid c \in \mathbb{N}\}$ states that, for any possible polynomial bound n^c, it is not the case that circuits exist with size growing below this bound that solve the problem. That is, no polynomial bound at all works.

The main idea behind the proof of this result is not complicated, and we present it already.

Sketch of proof. Define $\varphi(x) :\equiv \neg PHP(x)$, which is (logically equivalent to) a $\hat{\Sigma}^{1,b}_{1,s}$ formula. Towards a contradiction, assume $\mathsf{V}^0_2 + \{\neg\alpha^c_\varphi \mid c \in \mathbb{N}\}$ inconsistent. By compactness, there exists $c \in \mathbb{N}$ such that $\mathsf{V}^0_2 \vdash \alpha^c_\varphi$.

We will show that $\mathsf{V}^0_2 + \alpha^c_\varphi \vdash PHP(x)$. Together with the above, this would imply $\mathsf{V}^0_2 \vdash PHP(x)$, which is known to be false by the *Gem Theorem* (1.3.12).

First off, it can be easily seen that V^0_2 proves $PHP(x)$ is inductive, i.e.,

$$PHP(0) \wedge \forall u < x(PHP(u) \to PHP(u+1))$$

But, in V^0_2, we can't actually perform this induction, because PHP is a $\Pi^{1,b}_1$ formula, and we only have $\Sigma^{1,b}_0$ induction available.

This is where α^c_φ comes in. Thanks to our choice of φ, α^c_φ gives us exactly a family of circuits whose negation is equivalent to PHP: given any fixed length n (in particular $n := \max\{|x|, 2\}$), we get a circuit C with

$$\forall u \leq x(\neg C(u) \leftrightarrow PHP(u))$$

By plugging $\neg C(u)$ in place of $PHP(u)$ as the inductive predicate above, the induction no longer has any second-order quantifiers, and can thus be completed as a $\Sigma_0^{1,b}$ induction, which V_2^0 has available. □

Our new results will employ the same method: showing that a certain complex predicate is inductive (although this proof is no longer straightforward), and then invoking an α_φ^c to actually perform the induction.

But before we can state what we will be proving, it is necessary to understand the two alternative formalizations of $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

The second formalization, which is very similar in spirit to the first one, employs Turing machines M (described by a fixed numeral) in place of formulas φ. Indeed, if we had a suitable formula representing *"Y is an accepting computation of the machine M on input x"*, we could replace the arbitrary $\hat{\Sigma}_{1,s}^{1,b}$ formula φ in α_φ^c by the standardized form

$$\exists_2 Y \text{ "}Y \textit{ is an accepting computation of the machine } M \textit{ on input } x\text{"}$$

where M is a fixed numeral.

The authors of [3] find such a formula, and show it satisfies certain desirable properties that capture our intuitions about these machines. Mainly, V_2^0 proves that each $\varphi \in \hat{\Sigma}_{1,s}^{1,b}$ is equivalent to such an M. And furthermore, the weak theory $\mathsf{S}_2^1(\alpha)$ can prove that this M satisfies the bounds required of a NEXP machine (we call such machines *explicit*, in the sense that they are recognized by a weak theory).

By using this well-behaved formula we can write the analogous α_M^c, and of course the consistency result now becomes that, for a certain NEXP machine M, $\mathsf{V}_2^0 + \{\neg\alpha_M^c \mid c \in \mathbb{N}\}$ is consistent.

The third and final formalization takes more explaining, and is especially central to the current work, as it underlies the connection with the Easy Witness Lemma.

We might not be too happy with the fact that the above formalizations include second-order existential quantifiers, given the fundamental limitations of V_2^0 to construct such objects. We'd be happier with a purely universal formulation, that only discusses the existence of certain large sets *conditional* on the existence of other large sets. Notice indeed that the complexity of α_φ^c (and also α_M^c) is $\Delta_2^{1,b}$, because φ is already $\hat{\Sigma}_{1,s}^{1,b}$.[1]
Crucially, the authors of [3] notice that yet another formulation is possible (β_M^c) which is completely universal, and of lower complexity $\Pi_1^{1,b}$. This re-formulation doesn't follow trivially, but rather again by a deep result of complexity theory: the Easy Witness Lemma.

Lemma 2.1.4 (Easy Witness Lemma, [7])**.** *If* $\mathsf{NEXP} \subseteq \mathsf{P/poly}$*, then every* NEXP *machine has polynomial-size* witness circuits *which, in addition, are oblivious.*

A *witness circuit* for a NEXP machine M and input x is a circuit D_x such that, if M accepts x, its truth table, $tt(D_x)$, encodes an accepting computation of M on x.
An *oblivious* witness circuit for a machine M and input length n is a circuit D with at least n inputs such that for every x of length n, if M accepts x, then the circuit D_x, obtained by fixing the n first inputs of D to be the bits of x, is a witness circuit for M and x. The qualifier oblivious refers to the fact that D depends only on the length of x, not on x itself.

This result is celebrated due to how surprising it is that the bits of arbitrary exponential-size witnesses can be generated by polynomial-size circuits.

Thus, in α_M^c, we can replace the statement about the existence of an accepting computation $(\exists_2 Y)$ by a claim that the small circuit D produces it, thus avoiding the $\exists_2$:

Definition 2.1.5 (Definition 5 in [3])**.** *The* universal formalization *of an explicit* NEXP

[1] In fact, α is a Boolean combination of $\Sigma_1^{1,b}$ formulas, and such formulas have better properties than arbitrary $\Delta_2^{1,b}$ formulas. But this realization is not important for the current work.

machine M being in $\mathsf{P/poly}$ *and having an oblivious witness circuit is*

$$\begin{aligned}\beta_M^c \;:\equiv\; & \forall n{\in}Log_{>1}\ \exists C{<}2^{n^c}\ \exists D{<}2^{n^c}\ \forall x{<}2^n\ \forall_2 Y \\ & (C(x){=}0 \;\rightarrow\; \neg\text{“}Y \textit{ is an accepting computation of } M \textit{ on } x\text{”}) \wedge \\ & (C(x){=}1 \;\rightarrow\; \text{“}tt(D_x) \textit{ is an accepting computation of } M \textit{ on } x\text{”})\end{aligned}$$

That is, when the circuit C rejects x this is also the case for any witness Y (this implication remains universal), but when C accepts x then D finds an accepting Y (this implication is no longer existential).

Finally, the authors also prove the consistency of this new formalization:

Theorem 2.1.6 (Universal formalization, Theorem 7 in [3])**.**

There exists $M \in \mathbb{N}$ such that $\mathsf{V}_2^0 + \{\neg\beta_M^c \mid c \in \mathbb{N}\}$ is consistent.

The proof of this last theorem is in fact immediate, because V_2^0 straightforwardly proves that $\{\neg\alpha_M^c \mid c \in \mathbb{N}\}$ implies $\{\neg\beta_M^c \mid c \in \mathbb{N}\}$. Strictly speaking, what we actually have is that V_2^0 (and even $\mathsf{S}_2^1(\alpha)$) proves $\beta_M^c \rightarrow \alpha_M^c$. Thus, if $\mathsf{V}_2^0 + \{\neg\beta_M^c \mid c \in \mathbb{N}\}$ were not consistent, by compactness we'd have $\mathsf{V}_2^0 \vdash \beta_M^c$ and thus $\mathsf{V}_2^0 \vdash \alpha_M^c$, which we already know is not the case (Theorem 2.1.3).

In fact, the authors define a particular universal NEXP machine M_0 for which this consistency holds. And again prove desirable properties of M_0, like V_2^0 recognizing its universality [3][Lemma 27].

The lower complexity of β is also exploited for some magnification results [3][Section 6] that we won't discuss here.

2.2 Towards new results

Despite its proof being immediate, the reason this last consistency result (2.1.6) is interesting is that, in the real world, and thanks to the Easy Witness Lemma (2.1.4), the lower

complexity statement β is equivalent to the direct formalization α.
But, while we do know $\mathsf{V}^0_2 \vdash \beta^c_{M_0} \to \alpha^c_{M_0}$, we don't know whether V^0_2 proves the other implication. In the standard model, this other implication is simply a reformulation of the Easy Witness Lemma: if the NEXP-complete problem is solved by a polynomial circuit C, then there is also a polynomial circuit D providing the witnesses. More concretely, the statement that holds in the standard model is not $\alpha^c_{M_0} \to \beta^c_{M_0}$, but rather that for every c, there is a c' such that $\alpha^c_{M_0} \to \beta^{c'}_{M_0}$, since the construction of D from C might increase the polynomial bound c.

This is where the present work comes in: our main motivation is to show that, for any c, there is a c' such that V^0_2 proves $\alpha^c_{M_0} \to \beta^{c'}_{M_0}$. This would be interesting for a variety of reasons:

- For one, it would further vindicate the strength of V^0_2, given it proves such a modern and deeply technical result. This would make the existing consistency results even more interesting, and further solidify their formalizations as the correct ones (given their equivalence).
- Additionally, it would transfer the magnification results of [3][Section 6] (only proved for the β) to the equivalent α.
- Even more importantly, it would broaden our understanding of circuits and interactive proofs in second-order BAs (which have been still barely studied). Variations on and uses of the Easy Witness could allow for the construction of many more modern tools inside these BAs.

The work here presented establishes intermediate results in the way to this final proof, but doesn't yet complete it.

The full proof (in PA) of the Easy Witness Lemma is quite involved, and we won't present

it here in full. The important aspect to know is the following: it employs $\mathsf{IP} = \mathsf{PSPACE}$ as a central piece. This thesis is mainly focused on that first step: formalizing and proving $\mathsf{IP} = \mathsf{PSPACE}$ in $\mathsf{V}_2^0 + \alpha_{M_0}^c$.

Chapter 3

Exact probabilities in V_2^0

To talk about IP inside V_2^0, we will first need to discuss how to formalize probabilities in BAs. Talking about probabilities is really talking about the sizes of sets. Indeed, by $\varphi(r)$ having a probability of p (expressed as a rational quotient) in the random seed $r \in D$, we really mean that the amount of $r \in D$ satisfying $\varphi(r)$ is $|D| \cdot p$.

If the intervening sets, like D, were small enough (polynomial instead of exponential), we'd be able to get away with representing them as numbers (first-order variables) in V_2^0. But we will see that this is not possible for our use case, and thus we need actual sets (second-order variables).

Instead of arbitrary finite sets of seeds D, we will focus on (and define probabilities for) sets of the particular shape $D_t := \{r \mid r < t\}$, for a given bound t. We will also use the notation

$$\exists_2 F : a \to b \quad (\varphi(F))$$

as an abbreviation for the formula stating that there exists a second-order F which encodes a function with domain D_a and image D_b and satisfies $\varphi(F)$. More generally, in what follows we identify t with D_t when the meaning is implicitly clear.

As it turns out, certain straightforward definitions of probabilities won't be enough for our purposes, and we instead need an additional monotonicity constraint. We now present these different approaches.

Definition 3.0.1 (Non-monotonic probabilities)**.**
We have two definitions of probability that don't enforce monotonicity:

- ***Surjective:***

$$\Pr_{r<t}[\varphi(r)] \leq p \quad :\equiv \quad \exists_2 F : t \cdot p \to t \ [\forall r < t \ (\varphi(r) \to \exists x < t \cdot p(F(x) = r))]$$

  That is, the set of $r < t$ with $\varphi(r)$ is small enough to be covered by a surjective function with domain $t \cdot p$.

- ***Injective:***

$$\Pr_{r<t}[\varphi(r)] \leq p \quad :\equiv \quad \exists_2 G : t \to t \cdot p[\forall r, r' < t(\varphi(r) \wedge \varphi(r') \wedge r \neq r' \to G(r) \neq G(r'))]$$

  That is, the set of $r < t$ with $\varphi(r)$ is small enough to be injectively embedded into $t \cdot p$.

In this and below definitions, the probability p always stands in for a rational number (expressed through its numerator and denominator) with denominator t, such that $t \cdot p$ is always well-defined (and similarly for situations in which p is multiplied by other terms).

Proposition 3.0.2. V^0_2 *proves the two definitions of non-monotonic probability equivalent.*

Proof. **Surjective** $\Rightarrow$ **Injective**:
Assume we have $F : t \cdot p \to t$ satisfying the *Surjective* definition. We define $G : t \to t \cdot p$ as follows:

$$G(r) = \begin{cases} 0 & \text{if } \neg\varphi(r) \\ \min\{x < t \cdot p : F(x) = r\} & \text{if } \varphi(r) \end{cases}$$

The existence of G is guaranteed by $\Sigma_0^{1,b}$-comprehension. The totality of G comes from the surjectivity of F, combined with the minimization principle for $\Sigma_0^{1,b}$ formulas, which we have available in V_2^0 ([5][Theorem 20]). Functionality of G is due to the uniqueness of the minimum. Injectivity of G on the set of $r < t$ with $\varphi(r)$ follows from F being a function.

Injective ⇒ Surjective:

Assume we have $G : t \to t \cdot p$ satisfying the *Injective* definition. We define $F : t \cdot p \to t$ as follows:

$$F(x) = \begin{cases} 0 & \text{if } \neg\exists r \leq t(G(r) = x) \\ \min\{r \leq t : G(r) = x\} & \text{if } \exists r \leq t(G(r) = x) \end{cases}$$

The existence of F is guaranteed by $\Sigma_0^{1,b}$-comprehension. Totality of F comes from the minimization principle for $\Sigma_0^{1,b}$ formulas. Functionality of F is due to the uniqueness of the minimum. Surjectivity of F on the set of $r < t$ with $\varphi(r)$ follows from G being total and functional. □

Definition 3.0.3 (Monotonic probabilities)**.**

We have two definitions of probability that do enforce monotonicity:

- ***Surjective monotonic:***

$$\Pr_{r<t}[\varphi(r)] \leq p \quad :\equiv \quad \exists_2 F : t \cdot p \to t \ [\forall r < t \ (\varphi(r) \to \exists x < t \cdot p(F(x) = r))] \wedge [\forall x, x' < t \cdot p(x < x' \to F(x) < F(x'))]$$

  That is, we additionally require monotonicity.

- ***Injective monotonic:***

$$\Pr_{r<t}[\varphi(r)] \leq p \quad :\equiv \quad \exists_2 G : t \to t \cdot p[\forall r, r' < t(\varphi(r) \wedge \varphi(r') \wedge r < r' \to G(r) < G(r'))]$$

  That is, we additionally require monotonicity.

Finally, we have two definitions in terms of precise probabilities, instead of inequality $\leq p$.

Definition 3.0.4 (Counting probabilities)**.**

We have two definitions of probability through direct counting:

- ***Counting:***

$$\begin{aligned}\Pr_{r<t}[\varphi(r)] = p \quad :\equiv \quad & \exists_2 C : t \to t+1 \\ & [C(0) = \mathbb{1}[\varphi(0)] \wedge \forall r < t\,(C(r+1) = C(r) + \mathbb{1}[\varphi(r+1)]) \ \wedge \\ & C(t-1) = t \cdot p]\end{aligned}$$

where $\mathbb{1}$ is the indicator function. That is, C goes over all the seeds in order, and increases its value by one whenever it encounters one satisfying $\varphi(r)$. Thus, $C(i)$ represents the exact number of seeds satisfying $\varphi(r)$ for $r \leq i$.

- ***Tree:***

Represent our seeds in base b: $r = r_1 \dots r_n$ with $r_i < b$ and $n = \log_b(t)$.[1]

Consider also the set of initial segments of such expressions:

$S := \{r_1 \dots r_k \mid k \leq n, r_i < b\}$.

With this notation at hand, we can define

$$\begin{aligned}\Pr_{r<t}[\varphi(r)] = p \quad :\equiv \quad & \exists_2 T : S \to S\ [\forall r_1 \dots r_k \in S \\ & (k = n \to T(r_1 \dots r_k) = \mathbb{1}(\varphi(r_1 \dots r_k)) \wedge \\ & (k < n \to T(r_1 \dots r_k) = \sum_{r_{k+1}<b} \frac{1}{b}\, T(r_1 \dots r_k r_{k+1})] \ \wedge \ T(\varnothing) = p\end{aligned}$$

That is, at each leaf (complete random seed r) the label is 1 or 0 depending on whether $\varphi(r)$, and each branching (incomplete random seed) recursively computes the fraction of its completions satisfying $\varphi(r)$. At the root node (corresponding to the empty prefix), we will obtain the exact probability.

[1] In fact, assume for convenience that $t = b^n$ exactly, so that our random seeds are exactly those expressible in n digits in base b.

Of course, we can recover the probabilistic inequality $\leq p$ from these precise definitions. We can do so either through existential quantification

$$\Pr_{r<t}[\varphi(r)] \leq p \quad :\equiv \quad \exists p'(\Pr_{r<t}[\varphi(r)] = p' \wedge p' \leq p)$$

or universal quantification

$$\Pr_{r<t}[\varphi(r)] \leq p \quad :\equiv \quad \forall p'(\Pr_{r<t}[\varphi(r)] = p' \rightarrow p' \leq p),$$

which we can choose between as it suits the situation.
As mentioned previously, $\exists p'$ and $\forall p'$ really stand in for the quantification of its numerator (with denominator t).

We can also recover exact probabilities from the previous inequality formulations, by defining

$$\Pr_{r<t}[\varphi(r)] = p \quad :\equiv \quad \Pr_{r<t}[\varphi(r)] \leq p \ \wedge \ \neg\Pr_{r<t}[\varphi(r)] \leq p - \frac{1}{t}$$

The *Counting* and *Tree* definitions are, in a sense, monotonic by default: *Counting* counts the seeds in order, and in *Tree* we can also recover the probability in an initial segment of the random seeds by looking through the many labels available at the intermediate levels of the tree. This is made precise by the following result.

Proposition 3.0.5.
V^0_2 *proves the two monotonic and two counting definitions of probability all equivalent.*

Proof. **Surjective monotonic $\Leftrightarrow$ Injective monotonic**:
As in the proof of Proposition 3.0.2.
Surjective monotonic $\Rightarrow$ Counting:
Assume we have $F : t \cdot p \rightarrow t$ satisfying the *Surjective monotonic* definition. We define

$C : t \to t+1$ as follows:

$$C(r) = \begin{cases} 0 & \text{if } \neg\exists y < t \cdot p(F(y) \leq r) \\ \max\{y < t \cdot p : F(y) \leq r\} + 1 & \text{if } \exists y < t \cdot p(F(y) \leq r) \end{cases}$$

Again the existence and properties of C follow from $\Sigma_0^{1,b}$-comprehension and the minimization principle.

Counting ⇒ Surjective monotonic:

Assume we have $C : t \to t$ satisfying the *Counting* definition. We define $F : t \cdot p \to t$ as follows:

$$F(r) = \min\{y < t : C(y-1) = r \wedge C(y) = r+1\}$$

(with the understanding that $C(-1) := 0$).

That is, we "spend" our rth domain element (to cover an element in the image) whenever the value of C raises for the rth time.

Counting ⇒ Tree:

Assume we have $C : t \to t$ satisfying the *Counting* definition. We define $T : S \to S$ as follows:

$$T(r_1 \dots r_k) = \frac{C(r_1 \dots r_k b-1 \dots b-1) - C((r_1 \dots r_k 0 \dots 0) \ - \ 1)}{b^{(n-k)}}$$

(with the understanding that $C(-1) := 0$).

That is, we take the difference between the highest possible completion and the lowest possible completion, which tells us how many completions satisfy φ, and then divide it by the number of such completions to obtain a fraction. This expression (which is polynomial to compute with access to oracle C) can be defined by a $\Sigma_0^{1,b}$ formula.

Tree ⇒ Counting:

Assume we have $T : S \to S$ satisfying the *Tree* definition. We define $C : t \to t$ as follows:

$$C(r) = \sum\{b^{(n-k)} \cdot T(r_1 \dots r_k) \mid r_1 \dots r_k \overbrace{b-1 \dots b-1}^{n-k} < r \wedge r_1 \dots r_{k-1} \overbrace{b-1 \dots b-1}^{n-k+1} \not< r\}$$

That is, we sum over all "maximally incomplete" random seeds that are completely below r (meaning, any completion is below r), while also multiplying them by the amount of completions they have ($b^{(n-k)}$) to turn the fraction provided by T into a counting number. We require them to be "maximally incomplete" to avoid double counting.

For example, if $b = 2$, $n = 2$ and $r = 10$ (in binary notation), then we will sum $2 \cdot T(0)$ (representing that both completions 00 and 01 are below r, and extracting from T how many of them satisfy φ) with $1 \cdot T(10)$ (simply consulting T on whether $\varphi(10)$ to add it to the count).

This way we require at most $b \cdot n$ queries to T, instead of the b^n we would need if we simply summed the value of T over all the leaves (complete random seeds). The latter would require we use a witness with length exponential on n, and thus wouldn't be $\Sigma_0^{1,b}$ definable. □

We now present the crucial property of our monotonic definition that will make our main result possible. This averaging argument intuitively states that, if the average size of a group of sets is low, then at least one of the sets must have correspondingly low size. This will be what allows us to, assuming the existence of a prover fooling our verifier on a long protocol, derive the existence of one for a slightly smaller protocol (ultimately leading us to contradiction, since our verifier cannot be fooled in a 1-step protocol).

Lemma 3.0.6 (Averaging argument)**.** V_2^0 *proves that*

$$\forall i \in Log_{>1} \forall t, p \qquad \Pr_{(r,r')<(t,i)}[\varphi(r,r'))] \leq p \quad \rightarrow \quad \exists r' < i(\Pr_{r<t}[\varphi(r,r')] \leq p)$$

Proof. As per Proposition 3.0.5, we can see it just for the *Counting* definition of proba-

bility. The antecedent thus explicitly reads

$$\begin{aligned}\exists C : t \cdot i \to t \cdot i\ [&C(0,0) = \mathbb{1}[\varphi(0,0)] \wedge \\ &\forall r < t-1, \forall r' < i\ (C(r+1,r') = C(r,r') + \mathbb{1}[\varphi(r,r')]) \wedge \\ &\forall r' < i\ (C(0,r'+1) = C(t-1,r') + \mathbb{1}[\varphi(t-1,r')]) \wedge \\ &C(t-1,i-1) \leq t \cdot i \cdot p]\end{aligned}$$

and the consequent

$$\begin{aligned}\exists r' < i\ \exists C' : t \to t\ [&C'(0) = \mathbb{1}[\varphi(0,0)] \wedge \\ &\forall r < t-1\ (C'(r+1) = C'(r) + \mathbb{1}[\varphi(r,r')]) \wedge \\ &C'(t-1) \leq t \cdot p]\end{aligned}$$

We have from the definition of C that

$$C(t-1,i-1) = \sum_{r'<i} (C(t-1,r'+1) - C(t-1,r')) + C(t-1,0)$$

We will exploit the telescoping nature of this sum.

Let us see by induction on $i \in Log_{>1}$ (and inside V^0_2) that

$$\sum_{r'<i} f(r') \leq p \cdot i \ \Rightarrow\ \exists r' < i\ f(r') \leq p$$

For $i = 1$ it's immediate. Assume it for i, and assume the antecedent $\sum_{r'<i+1} f(r') \leq p \cdot i$. Then we have

$$\sum_{r'<i} f(r') + f(i) = \sum_{r'<i+1} f(r') \leq p \cdot (i+1) = p \cdot i + p$$

We know from the BASIC set of axioms that (for numbers a, b, c, d in Log),

$$a + b \leq c + d \to a \leq c \vee b \leq d$$

Thus either $\sum_{r'<i} f(r') \leq p \cdot i$, in which case we are done by induction hypothesis, or $f(i) \leq p$, in which case we are done.
Thus we have that $\exists r' < i \ C(t-1, r'+1) - C(t-1, r') \leq t \cdot p$ (including also among those the degenerate case $C(t-1, 0) - 0$). We can thus define, with $\Sigma_0^{1,b}$-comprehension, $C'(r) := C(r, r')$, where r' is the minimum such satisfying the inequality. And it is immediate from the properties of C and the existence of such an r', that our constructed C' is the required probabilistic witness. □

We now show an additional property of the monotonic definition, which aligns with our intuitions about how probabilities should behave: negating the inside formula also negates the outside probability.

Proposition 3.0.7. V_2^0 *proves that* $\Pr_{r<t}[\varphi(r)] = p \Rightarrow \Pr_{r<t}[\neg\varphi(r)] = 1 - p$

Proof. Taking a witness C from the *Counting* definition, define

$$C'(r) := r - C(r)$$

C' exists again by $\Sigma_0^{1,b}$ comprehension, and easy to see it's a witness for our consequent. □

It intuitively seems like the set witnesses from the monotonic definitions include strictly more information than the non-monotonic ones. Indeed, we believe that this intuition can be made rigorous by showing that V_2^0 does not prove them equivalent, but we don't have a proof of this yet. Hopefully a proof exists that, assuming this equivalence, derives PHP, which V_2^0 famously does not prove (Theorem 1.3.12).
It might be possible to show in a similar way that our main result cannot be proved for the non-monotonic definition.

In the above definitions, we could have used instead a fixed Turing machine (represented by a fixed numeral) that, given φ and t, tries to construct the necessary witness of each definition (and accepts if it does). For example, a machine counting through the random

seeds one by one, as in *Counting*. But this wouldn't lower the complexity of our definition (or gain us anything else), since the computational witness (used to assert that "this fixed machine accepts on this input") would be set-sized again, and thus we would still employ second-order quantification.

As written, a definition like *Counting* has complexity $\exists_2 \forall \exists C(\varphi)$, where $C(\varphi)$ is the complexity of φ. This is undesirable, because in some places below we will need our formulas to be $\hat{\Sigma}^{1,b}_{1,s}$, thus only with universal first-order quantifiers.
For example, an internal $\exists$ could arise due to the function C being expressed as a set, and thus having to existentially quantify its output. That is, by defining $C(r) = i$ as $\exists (a_1, a_2) \in C(a_1 = r \wedge a_2 = i)$. But this quantifier in particular is easy to replace by a universal one, by defining it instead as $\forall (a_1, a_2) \in C(a_1 = r \rightarrow a_2 = i)$.

A way less obviously replaceable $\exists$ appearing in our definitions is the one stating that a certain function (like F or C) is total: $\forall a_1 \exists a_2((a_1, a_2) \in F)$. But fortunately we also have a (slightly trickier) fix for this: codifying the function in binary instead. That is, instead of $F : a \rightarrow b$, we will represent

$$F' : a \times |b| \rightarrow \{0, 1\}$$

where $F'(x, i)$ represents the ith binary bit of $F(x)$. These kinds of binary functions can be represented simply as the set of preimages of 1, that is, a subset S of $a \times |b|$. Then, $F(x) = y$ can be defined as $\forall i < |b|$ $((x, i) \in S \leftrightarrow bit(y, i) = 1)$, where *bit* is a fixed polynomial function extracting the ith bit of y. This quantification is not only bounded, but sharply bounded (due to using the length of b), and because of that doesn't increase our complexity (recall we only count first-order bounded quantifiers). And similarly, we don't need to make any statement about totality, since the subset representation immediately makes the function total (if a pair doesn't belong to the set, this simply indicates that the corresponding bit is a 0).

We have brought up this complexity-reducing "trick" in the context of our probabilistic definitions, but we also implicitly invoke it in some more places below, so that we can turn some of the formulas presented (for readability) as $\Sigma^{1,b}_{1,s}$ formulas into $\hat{\Sigma}^{1,b}_{1,s}$ formulas.

Finally, we highlight for future work that the approach and basic results of this section could present interesting connections to existing work on *Approximate counting* by Jeřábek [9]. While we have not sketched these connections in detail yet, and the approach to and results about circuits there are different from ours, it would seem like those approximate definitions are a natural weakening of our non-monotonic functional definitions (3.0.2), which allows to prove (weakened approximate versions of) interesting results like the Averaging Argument without resorting to our monotonic and exact definitions. That is, while we have sidestepped these limitations by resorting to our different definitions, approximate counting does so by weakening them slightly. This leaves the door open for proofs of (weakened approximate versions of) the results in this section, or even the main results below, formalized now with that alternate definition of probabilities.

Chapter 4

$\mathsf{IP} = \mathsf{PSPACE}$ in $\mathsf{V}^0_2 + \alpha$

4.1 Proving the hard inclusion

With these probabilistic grounds in place, we can formalize our statements of interest. We start with the hard inclusion, $\mathsf{PSPACE} \subseteq \mathsf{IP}$.

Definition 4.1.1 (Encoding $\mathsf{TQBF} \in \mathsf{IP}$)**.** *The* probabilistic formalization *of the problem* TQBF *being in* IP *(as witnessed by the verifier polynomial machine with numeral M) is*

$$\begin{aligned}\gamma^{M,t}_{\mathsf{TQBF}} :=& \forall n \in Log_{>1} \forall s < 2^n \\ & [(\mathsf{TQBF}(s) \rightarrow \exists_2 P \ \Pr_{r<t(n)}[accept(P, M, n, s, r)] \geq 2/3) \wedge \\ & (\neg\mathsf{TQBF}(s) \rightarrow \forall_2 P \ \Pr_{r<t(n)}[accept(P, M, n, s, r)] \leq 1/3)]\end{aligned}$$

where

- $\mathsf{TQBF}(s)$ *is the predicate claiming that the quantified boolean formula encoded by s is true. We can choose it to be provably $\Delta^{1,b}_1$ in V^0_2, by representing it both as "there is an (exponential on n, thus first-order) witness of the computation" and as "any (exponential on n, thus first-order) witness of the computation ends in acceptance", whose equivalence can be proven by number induction.*

- $accept(P, M, n, s, r)$ *is a formula (with second-order parameter P) stating that the*

polynomial-time interactive protocol between the prover P and the polynomial verifier M on input $s < 2^n$ and with random seed r results in acceptance. It can be chosen (provably in V^0_2) $\Delta^b_1(\alpha)$, again by considering the existential and universal phrasings of the polynomial (on n) computational witness.

- *$t(n)$ is a bounding term on the description length of the random seed.*

The above is a $\Delta^{1,b}_2$ statement, because of the presence of both $\exists_2$ and $\forall_2$. This complexity doesn't prevent our proof from going through, since we don't need to apply any comprehension or induction to this whole formula (only to its subformulas).

Notice, as we had mentioned above, that P couldn't be coded by a first-order variable. Indeed, P has a number of elements exponential on n (one for each different possibly query from the verifier), each of length polynomial on n. This means its description (coded by a number) has length exponential on n, and thus doesn't provably exist in V^0_2.

Without further ado, we state and prove our main result:

Theorem 4.1.2 (Proving $\mathsf{TQBF} \in \mathsf{IP}$)**.**
There is a numeral M and polynomial bounding term t such that, for any c,

$$\mathsf{V}^0_2 + \alpha^c_{M_0} \vdash \gamma^{M,t}_{\mathsf{TQBF}}$$

Proof.
Most of the proof of $\mathsf{IP} = \mathsf{PSPACE}$ remains structurally unchanged in our reproduction inside V^0_2, and thus we simply need to invoke various results to ensure the same reasoning can be carried out. But we do need one small change to the original proof (as presented above in Theorem 1.2) to make this possible.

As before, to each Quantified Boolean Formula will correspond an arithmetized multilinear polynomial, whose values we will calculate in an appropriate ring $\mathbb{F}_z$. In the usual

proof, this z is taken exponential in n. But this wouldn't be enough for our purposes, because we need to invoke a result about polynomial roots (4.1.4) which requires $z \in Log_{>1}$.

To make z smaller, we introduce one change into the arithmetization procedure. Say $A(\varphi)$ is the arithmetization of φ. Then, we arithmetize $\exists_{X_i}\varphi(X_1, \ldots, X_n)$ as

$$1 - \prod_{X_i \in \{0,1\}} (1 - A(\varphi)(X_1, \ldots, X_n))$$

instead of $\sum_{X_i \in \{0,1\}} A(\varphi)(X_1, \ldots, X_n)$. This ensures that the value taken by any polynomial in our procedure (on inputs 0 and 1) is always 0 or 1 (notice this is also preserved under linearization steps). This is also easily provable inside V_2^0 by a number induction on the construction of the arithmetized polynomial.

First, we note that this change still leaves the degree of our polynomials manageable. Indeed, our new operator increases degrees exactly as much as a product already did, and exactly like then the interspersed linearizations will be enough to keep this under a constant.

Now, let's see that this change allows us to pick a small z. Indeed, say d is a constant bound on the degree of any one of our arithmetized polynomials on any one of its variables during the whole procedure. For example, we can choose $d = 2n$.[1] In the soundness direction, we will end up proving that the verifier has probability of rejecting at least $(1 - \frac{d}{z})^n$, and we want this to be $\geq \frac{2}{3}$ for large enough n. We also need z to be polynomial in n, so that it is in $Log_{>1}$. Choosing, for example, $z = n^4$ satisfies both constraints.

Now that we know z, we can also fix our polynomial bound $t(n)$ on the random seed. Since each random choice from the verifier is selecting a member of $\mathbb{F}_z$, and there are at most

[1] This is because in the matrix polynomial (when no operators have yet been applied) a variable can have at most degree n. And after that, a product brings it up to $2n$, but they are immediately followed by a linearization that brings it down again to at most n.

n^2 such choices (one per quantifier, including linearizations), we can choose $t(n) := z^{n^2}$ as the bounding term.

Before going further, we first need to ensure our theory proves some basic facts about the correspondence between satisfying assignments of (partly) Quantified Boolean Formulas (that is, possibly with free variables), and the corresponding assignment of the arithmetized polynomial.

Lemma 4.1.3 (Manipulating arithmetized polynomials)**.** *There are:*

- *A numeral A, encoding a Turing machine that turns the description of a (partly) Quantified Boolean Formula into that of a (partly) quantified artihmetized polynomial, as per the arithmetization procedure explained above.*
- *A numeral A', encoding a Turing machine that turns the description of a (partly) quantified artihmetized polynomial into the explicit expression of that polynomial (without quantifiers). That is, implements all quantifiers and provides the resulting expression.*
- *Numerals E_f and E_p encoding Turing machines that, given, respectively, the description of a Boolean Formula (without quantifiers) or an arithmetized polynomial (without quantifiers) and a variable assignment, evaluate them (in the latter case, modulo z).*
- *Numerals E'_f and E'_p encoding Turing machines that, given, respectively, the description of a (partly) Quantified Boolean Formula or a (partly) quantified arithmetized polynomial and a variable assignment, evaluate them (in the latter case, modulo z).*

such that V^0_2 proves the following:

1. *A, E_f and E_p are polynomial-time (in the specific sense of [3][Definition 18])*
2. *A', E'_f and E'_p are non-deterministic exponential-time (again in the specific sense of [3][Definition 18])*

3. *Given the description of a Boolean Formula (without quantifiers) f and an assignment a, $E_f(f,a) = E'_f(f,a) = E'_p(A(f),a) = E_p(A'(f),a)$. That is, all the different methods for computing the result agree.*

4. *Given the description of a (partly) Quantified Boolean Formula f' and an assignment a, $E'_f(f,a) = E'_p(A(f),a) = E_p(A'(f),a)$. That is, all the different methods for computing the result agree.*

5. *Given the description of a formula f, the degree of $A'(f)$ is linear in $|f|$.*

Proof.

1. These functions are clearly polynomial-time. It is a standard result in bounded arithmetic that any polynomial-time function is a PV-function ([12][Lemma 5.3.3]). And it is proven in our reference paper that any PV-function has a corresponding polynomial-time machine, provingly in $\mathsf{S}^1_2(\alpha)$ ([3][Lemma 19]), and thus also in V^0_2.

2. These functions can be defined by $\Sigma^{1,b}_{1,s}$ formulas. It is proven in our reference paper that each such function has a corresponding NEXP-machine, provably in V^0_2 ([3][Lemma 26]).

3. By easy number induction on the construction of quantifier-free formulas, and noticing that $A(f) = A'(f)$.

4. The base case (without quantifiers) is the previous item. For induction steps, we use $\Sigma^{1,b}_0$ comprehension. For example, say we have our statement for $f(a_1,\ldots,a_k)$ and need to see it for $\prod_{a_k\in\{0,1\}} f(a_1,\ldots,a_k)$. Then, from the (second-order) computational witnesses of our induction hypothesis we are able to straightforwardly construct another one for this new formula with a $\Sigma^{1,b}_0$ formula, stating "put both computations in sequence and then add another short computation of the multiplication of their outputs".

5. By number induction on the number of quantifiers of the arithmetized polynomial, noticing as in our reasoning before the Lemma that the starting degree is at most

n, products can at most double it, and the linearizations between products bring it back down to n. □

In the rest of the proof, and to improve readability, we don't include all details about how our manipulations of polynomials can be straightforwardly reproduced inside V^0_2, of which the previous result was a first exemplification.

Soundness

We will show that

$$\forall n \in Log_{>1} \neg\exists s < 2^n [\neg \mathsf{TQBF}(s) \wedge \exists_2 P \ \Pr_{r \leq t(n)}[accept(P, M, n, s, r)] > 1 - (1 - \frac{d}{z})^n]$$

where $d := n^2$ and $z := n^4$. That is, there is no counterexample false Quantified Boolean Formula s for which a prover P can convince the verifier with probability higher than $1 - (1 - \frac{d}{z})^n$. As per our reasoning in the usual proof, this will be enough, since the expression $1 - (1 - \frac{d}{z})^n$ gets arbitrarily close to 0 (and in particular, $\leq \frac{1}{3}$) for large enough n.

Take a fixed $n \in Log_{>1}$ for which we want to prove this. Fix $d := n^2$ and $z := n^4$, which are thus constant from now on. Consider the induction hypothesis

$$\begin{aligned} IH(i) \quad :\equiv \quad & \neg\exists s < 2^n [quantifiers(s) \leq i \wedge \\ & \neg \mathsf{TQBF}'(s) \wedge \exists_2 P \ \Pr_{r < t(i)}[accept(P, M, n, s, r)] > 1 - (1 - \frac{d}{z})^i] \end{aligned}$$

for $i \leq n$.

Here, $\mathsf{TQBF}'(s)$ is a variant of $\mathsf{TQBF}(s)$ that can take as inputs not only fully Quantified Boolean Formulas (deciding their truth value), but also any expression of the form

$$g(\bar{a}) = k$$

that take place during the sumcheck protocol, where g is a partly quantified polynomial, $\bar{a}$ is an assignment for its free variables in $\mathbb{F}_z$, and $k \in \mathbb{F}_z$ is its alleged value (deciding the truth of the equality). This whole expression is still coded by the single number s for convenience.

This change is required for the induction to work, since at some points of the sumcheck protocol we deal with such corresponding partly quantified polynomials, instead of fully quantified ones. Of course, TQBF and TQBF' coincide when considering a fully quantified formula (and its corresponding quantified polynomial equality).

Also, $quantifiers(s)$ is of course a $\Sigma_0^{1,b}$ formula computing how many quantifiers (including existential, universal and linearizing) the quantified polynomial has in the equality coded by s. Indeed, this amount of quantifiers is exactly the length of the protocol started by the expression s, which is the quantity we need to induct on.

$IH(i)$ is a $\Sigma_{1,s}^{1,b}$ formula. Since we don't have induction for formulas this complex available on V_2^0, it would seem like, even if we prove IH inductive, we won't actually be able to perform the induction. But here enters the most important technique making this result possible (already pioneered in [3]): using α to turn this formula into an equivalent $\Sigma_0^{1,b}$ formula, by turning its second-order set witness into a first-order number (a circuit) that computes that set. And then, with $\Sigma_0^{1,b}$-induction available, we will be able to complete the induction.

Thus, for any such i and within V_2^0, let's start by proving the induction step, $\neg IH(i+1) \rightarrow \neg IH(i)$. Indeed, assume a certain s and P witness $\neg IH(i+1)$, that is, P tricks the verifier often enough on input s. For simplicity, assume we also bundle the set witnesses for $\neg\mathsf{TQBF}(s)$ and for the probabilistic statement into the single variable P.

Consider this P. The first move from the prover will be to send some polynomial. We

can construct, simply by $\Sigma_0^{1,b}$ comprehension from this P, the restriction of the prover's strategy to any of the z possible elements of $\mathbb{F}_z$ that the verifier could randomly choose immediately afterwards. Each one of them will be a prover for a protocol with one less step (trying to prove a certain, different polynomial equality, with one less quantifier). Similarly for the part of P that witnesses the probabilistic statement (the surjective function), and for the part which witnesses $\neg\mathsf{TQBF}(s)$[2].

The first step, as in the common proof of the interactive protocol, will be recognizing that only a small number of the random elements of $\mathbb{F}_z$ that the verifier could choose at this step would leave the prover trying to prove something true. That is, most restrictions of P are still proving a false claim. This is the case because there are few inputs on which the real polynomial and the sent polynomial coincide (due to them having low degree). This result has already been formalized in the bounded arithmetics literature:

Lemma 4.1.4 (Lemma 4.3.6. of [8])**.** PV_1 *proves:*
For every prime $z \in Log$, and every non-zero $f \in \mathbb{F}_z[X]$, f has at most deg(f) roots.

As usual, "at most $\deg(f)$ roots" really means that we are given a surjective function with domain $\deg(f)$ covering the roots.

Since our theory extends PV_1 ([12][Theorem 5.3.5]), a straightforward application of this result shows that indeed at most d out of the $t(i) = z^{i^2}$ random elements lead to a true statement (in which, of course, the prover can always achieve perfect credibility by reporting the true polynomials).
Thus, at least a fraction $(1 - \frac{d}{z})$ of the restrictions have a new formula s' for which we still have $\neg\mathsf{TQBF}(s')$.

We need to see that one of our restricted provers (from amongst the ones proving something

[2]Indeed, if we have hard-coded the right Turing machine to check for TQBF, the computation will route through computing each sub-formula of the original formula s, and thus from a witness for the overall computation we'll easily extract a witness for each of these sub-computations

false) is a set fooling the verifier, thus a counterexample to $IH(i)$. We will do this through the averaging argument (Lemma 3.0.6).

Say we restrict our reasoning to the immediately next random choices $r' < z$ for which the statement to prove is false, of which there are at least $(1 - \frac{d}{z}) \cdot z$. We can single these out using the surjective function given to us by Lemma 4.1.4, so assume w.l.o.g. they form an initial segment. We can thus rephrase our probabilistic hypothesis as

$$\Pr_{(r,r')<(t(i),(1-\frac{d}{z})\cdot z)}[\neg accept(P, M, i+1, s, r^\frown r')] \leq \frac{(1-\frac{d}{z})^i}{(1-\frac{d}{z})} = (1-\frac{d}{z})^{i-1}$$

And then, applying the Lemma, one of these r' will need to satisfy

$$\Pr_{r<t(i)}[\neg accept(P, M, i+1, s, r^\frown r')] \leq (1-\frac{d}{z})^{i-1}$$

A counterexample set witness for $IH(i)$ is then shown to exist by $\Sigma_0^{1,b}$-comprehension (with P as a second-order parameter), by choosing the least r such that $P \restriction r$ is a counterexample, in case there's several different restrictions satisfying the property.

We have thus shown $IH(i)$ is inductive. It is also immediate that $IH(0)$ holds, simply by checking the case $s = 0$, showing through a short computation that M rejects[3]).

A priori we can't induct on IH, due to its complexity. But we will simulate this induction using $\alpha^c_{M_0}$, turning it into a $\Sigma_0^{1,b}$ induction by transforming second-order witnesses into circuits (first-order witnesses).

In more detail, IH is already $\Sigma_{1,s}^{1,b}$. We can perform the usual trick of moving quantifiers and encoding to make it $\hat{\Sigma}_{1,s}^{1,b}$. We are then able to use said $\hat{\Sigma}_{1,s}^{1,b}$ formula to obtain an equivalent NEXP machine M_{IH} ([3][Lemma 26]), and then invoke the universality of the NEXP machine M_0 ([3][Lemma 27]) to turn the computational witness of M_{IH} into a circuit C. Using this circuit and in summary, we are able to define a $\Sigma_0^{1,b}$ formula $\phi(i)$

[3]Recognizing s as either a not well-formed polynomial, or the null polynomial, depending on how we have defined our machines.

such that

$$\forall i \leq n \ \ (\phi(i) \leftrightarrow IH(i))$$

By plugging this equivalent ϕ as the induction predicate instead of $IH(i)$, we complete the $\Sigma_0^{1,b}$ induction.

Thus we have shown $IH(n)$ for an arbitrary $n \in Log_{>1}$, which was our goal.

Completeness

Since the proof of $\mathsf{IP} = \mathsf{PSPACE}$ actually provides perfect completeness, instead of

$$\mathsf{TQBF}(s) \rightarrow \exists_2 P \ \Pr_{r<t(n)}[\neg accept(P, M, n, s, b, r)] \leq \frac{1}{3}$$

we can prove the stronger

$$\mathsf{TQBF}(s) \rightarrow \exists_2 P \ \Pr_{r<t(n)}[\neg accept(P, M, n, s, b, r)] \leq 0$$

$\mathsf{TQBF}(s)$ provides a set computational witness W of the truth of s, and we will use it with $\Sigma_0^{1,b}$-comprehension to construct the desired P.

This will look structurally different depending on how we have defined the computational check of TQBF. But if we hard-code a machine performing it through arithmetization, simplification and evaluation of polynomials, we can directly read off the correct polynomials from W.

We are left to prove that this P has the required property. This is done by simulating $\hat{\Sigma}_{1,s}^{1,b}$-induction, as we just did for the soundness implication. Indeed, we can express our induction hypothesis (from the paragraph below) as a $\hat{\Sigma}_{1,s}^{1,b}$ formula, turn its set witness into a circuit coded by a number, and apply $\Sigma_0^{1,b}$-induction with such circuits.

Assume one check of the interactive procedure rejects. In fact, assume it is the first step at which any check rejects. If it is not the last step, we get a contradiction since the true polynomials will satisfy the checked property of $g_i(r_i) = g_{i+1}(0) \cdot g_{i+1}(1)$ (seen inside V_2^0 also by a simulated $\hat{\Sigma}_{1,s}^{1,b}$ induction). If it is the last step, in which M evaluates the polynomial quantifier-free polynomial on an input assignment, we also get a contradiction because M is proven to correctly evaluate polynomials.

With both the Soundness and the Completeness direction proven, this ends the proof of the theorem. □

What we have proven is (our formalization of) $\mathsf{TQBF} \in \mathsf{IP}$, and not yet $\mathsf{PSPACE} \subseteq \mathsf{IP}$: we still need to show that V_2^0 proves TQBF is PSPACE-complete. A rigorous proof of this assertion is left for future work, but we expect it not to present any important obstacles, by proceeding similarly to the proof in [3][Lemma 27] of a machine being NEXP-complete, albeit invoking different polynomial machines (constructed from Π_1^b-formulas) that reduce any PSPACE machine to an instance of TQBF in the usual way.

4.2 Proving the easy inclusion

Now for $\mathsf{IP} \subseteq \mathsf{PSPACE}$.

Say $witness_M(W, x, V, p_M)$ is a formula stating that the numeral M is a PSPACE machine (in the specific sense of [3][Definition 18]), and that W is a second-order computational witness to the fact that $p_M = M(x, V, \varnothing, \varnothing)$ (the reason for this specific type signature is made clear below).

And say $witness_{\mathrm{Pr}}(C, x, V, P, p_P)$ states that C is a counting set witness (as per Definition 3.0.3) to the fact that the probability of acceptance when the polynomial verifier V interacts with the prover P on input x is p_P.

We want to see the following:

Theorem 4.2.1 ($\mathsf{IP} \subseteq \mathsf{PSPACE}$)**.** *There exists a numeral M such that V_2^0 proves*

$$\forall n \in Log_{>1} \forall x < 2^n \forall V, p_M, p_P \forall_2 P, C, W$$
$$[witness_M(W, x, V, p_M) \wedge witness_{\mathrm{Pr}}(C, x, V, P, p_P) \rightarrow p_P \leq p_M]$$

That is, the prover implemented by the PSPACE machine M is optimal across all provers. This is not the usual statement of $\mathsf{IP} \subseteq \mathsf{PSPACE}$ (which would say "all problems solvable by an interactive protocol are solvable by a PSPACE machine"), but rather isolates the hard step, from which $\mathsf{IP} \subseteq \mathsf{PSPACE}$ follows immediately by a series of $\Sigma_0^{1,b}$ comprehensions.

Proof. Take a $\Sigma_0^{1,b}$ formula describing the protocol of the PSPACE machine that computes the optimal P, as described in the usual proof (1.2). Again invoking a previous result, we have that $\mathsf{T}_2^1(\alpha)$ (and thus also V_2^0) proves that a fixed numeral M is a PSPACE machine implementing this protocol ([3][Lemma 20]).

Recall that this machine returns both an optimal message m (for the conversation history as described in its input), and the probability p with which the verifier will accept after said message. By construction (due to how M performs its sequential search), V_2^0 proves that, at each single step, the chosen m and attained p are optimal relative to the previous values. That is:

$$M(x, V, (m_1, \ldots, m_k), (r_1, ..., r_k))_1 = \mathrm{argmax}_{m'} \sum_{r_{k+1} < z} \frac{1}{z} M(x, V, (m_1, \ldots, m_k, m), (r_1, ..., r_k, r))_2$$
$$M(x, V, (m_1, \ldots, m_k), (r_1, ..., r_k))_2 = \max_{m'} \sum_{r_{k+1} < z} \frac{1}{z} M(x, V, (m_1, \ldots, m_k, m), (r_1, ..., r_k, r))_2 \quad (\star)$$

But we still need to prove that the probability obtained by the whole procedure dominates that achieved by any arbitrary prover P.

As earlier for the other inclusion, we will show V_2^0 proves our target predicate is inductive (on the length n of the protocol) thanks to the averaging argument (3.0.6), and then apply

α on this $\Pi_1^{1,b}$ formula (or equivalently its $\Sigma_1^{1,b}$ dual) to actually reduce the induction to a $\Sigma_0^{1,b}$ induction.

Let's first see the predicate is inductive. Define

$$\begin{aligned} IH(i) \quad :\equiv \quad & \forall x < 2^i \forall V, p_M, p_P \forall_2 P, C, W \\ & [witness_M(W, V, x, p_M) \wedge witness_{\Pr}(C, x, V, P, p_P) \rightarrow p_P \leq p_M] \end{aligned}$$

and assume $\neg IH(i+1)$. Take its counterexample witnesses x, V, p_M, p_P, P, C, W.

Consider the first random choice the verifier needs to make in either protocol, which will be $r_1 < z$, where $\mathbb{F}_z$ is the ring where the polynomial computations will take place. As per the definition of M's procedure, and particularly ($\star$) above, we can split p_M into the probabilities of the protocol one step shorter:

$$p_M = \sum_{r_1 < z} p_{M,r_1}$$

Similarly and as in our previous proof, we can also separate p_P:

$$p_P = \sum_{r_1 < z} C(t(i) - 1, r_1) - C(t(i) - 1, r_1 - 1)$$

(again understanding that $C(t(i), -1) = 0$).
We have that

$$p_P = \sum_{r_1 < z} C(t(i) - 1, r_1) - C(t(i) - 1, r_1 - 1) > \sum_{r_1 < z} p_{M,r_1} = p_P$$

and thus

$$\sum_{r_1 < z} p_{P_j} - (C(i+1, 0) - C(i, 0)) > 0$$

We now use the averaging argument (3.0.6). More concretely (because of how our Lemma

statement is written), we need to invoke it for the equivalent

$$1 - \sum_{r_1 < z} p_{P_j} - (C(i+1,0) - C(i,0)) \leq 1 - \frac{1}{t(i+1)}$$

The Lemma implies there must exist some r_1 with $p_{P_j} - (C(i+1,0) - C(i,0)) > 0$. Taking the minimum such r_1, and restricting P, C, and W in the obvious ways, we can construct with $\Sigma_0^{1,b}$-comprehension the new witness counterexample showing $\neg IH(i)$.

It is again immediate that $IH(0)$ holds. And just as in our previous proofs (and invoking the same previous results), we use an α to perform the induction and finish the proof. $\square$

Conclusion and next steps

In this work, we have formalized and proved $\mathsf{IP} = \mathsf{PSPACE}$ inside V^0_2. In the process of doing so, we have also laid out foundations for probabilistic reasoning inside V^0_2, which really amounts to a form of exact counting. We have developed other tools transferable to different settings in bounded arithmetic, like altering the arithmetization procedure to reduce the size of the prime field we need to operate on, and have also repurposed previous techniques, like the use of α to make induction on complex formulas possible that already appeared in [3].

The work here presented is a natural middle step rather, than conclusion, to a line of work, and correspondingly there are many promising next steps available:

- The most important next step is to employ our result to build the whole proof of the Easy Witness Lemma inside V^0_2. We will need to replace the use of $\mathsf{MIP} = \mathsf{NEXP}$ present in the proof in [15] by that of $\mathsf{IP} = \mathsf{PSPACE}$. Other than that, it's yet unclear whether new ideas will be required or straightforward approaches will suffice.

- Although, of course, it is necessary before that to prove that TQBF is PSPACE-complete inside V^0_2 to fully complete the statement of $\mathsf{IP} = \mathsf{PSPACE}$. As we mentioned in Section 4.1, we expect this not to present any important obstacles, by proceeding similarly to the proof in [3][Lemma 27] of a machine being NEXP-complete.

- As mentioned in Chapter 3, we have some ideas to prove that the non-monotonic definitions of probability cannot be shown equivalent to the monotonic ones in V^0_2, nor our main results proven for them, by deriving from these states of affairs a proof

within V^0_2 of PHP, which we know cannot exist.

- As mentioned also in Chapter 3, we would like to identify rigorously the connection between our exact counting approach and that of approximate counting [9].

And more generally, second-order bounded arithmetics still lack an exhaustive treatment of the kind first-order ones have received. It seems likely that many useful technical tools remain to be discovered that would make possible future consistency results and other exciting advances in proof complexity.

Bibliography

[1] M. Ajtai. The complexity of the Pigeonhole Principle. Combinatorica, Volume 14, pages 417–433, 1994.

[2] S. Arora, B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press; 2009.

[3] A. Atserias, S. Buss, M. Müller. *On the Consistency of Circuit Lower Bounds for Non-Deterministic Time*. 55th ACM Symposium on the Theory of Computation, March 3, 2023.

[4] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, and A. Woods. *Bounded Arithmetic*. PhD Thesis, UC Berkeley, 1986.

[5] S. Buss. *Bounded Arithmetic*. PhD Thesis, UC Berkeley, 1986.

[6] S. Buss. *Tutorial on Bounded Arithmetic*. Presentation, Prague, 2009.

[7] R. Impagliazzo, V. Kabanets, A. Wigderson. *In search of an easy witness: exponential time vs. probabilistic polynomial time*. Journal of Computer and System Sciences, Volume 65, Issue 4, Pages 672-694, 2002.

[8] E. Jeřábek . *Weak pigeonhole principle and randomized computation*. PhD Thesis, Charles University in Prage, 2005.

[9] E. Jeřábek. *Approximate counting in bounded arithmetics*, AS CR, 2007.

[10] E. Khaniki. *Jump operators, Interactive Proofs and Proof Complexity Generators*. In *(Im)possibility results in Proof Complexity and Arithmetic*, PhD Thesis, Univerzita Karlova, Matematicko-fyzikální fakulta, Pages 62-107, 2023.

[11] D. C. Kozen. *Theory of Computation*. TCS Springer, 2006.

[12] J. Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995.

[13] A. Shamir. *IP=PSPACE*, in JACM, Vol 39, 4, 1992.

[14] L.J. Stockmeyer, A.R. Meyer. *Word problems requiring exponential time*, in STOC, pages 1–9. ACM, 1973.

[15] R. Williams. *Improving Exhaustive Search Implies Superpolynomial Lower Bounds*, in STOC, 2010.