On Annihilators of Explicit Polynomial Maps

Prerona Chatterjee [*]   Anamay Tengse [†]

September 14, 2023

Abstract

We study the algebraic complexity of annihilators of polynomials maps. In particular, when a polynomial map is 'encoded by' a small algebraic circuit, we show that the coefficients of an annihilator of the map can be computed in PSPACE. Even when the underlying field is that of reals or complex numbers, an analogous statement is true. We achieve this by using the class VPSPACE that coincides with computability of coefficients in PSPACE, over integers.

As a consequence, we derive the following two conditional results. First, we show that a VP-explicit hitting set generator for *all of* VP would separate either VP from VNP, or non-uniform P from PSPACE. Second, in relation to algebraic natural proofs, we show that proving an algebraic natural proofs barrier would imply either VP $\neq$ VNP or DSPACE($\log^{\log^* n} n$) $\not\subset$ P.

1 Introduction

A *polynomial map* is just a vector of polynomials, for example $\mathbf{g} = (g_1(\mathbf{z}), \ldots, g_n(\mathbf{z}))$, and an n-length vector of m-variate polynomials acts as a function (or a map) from an m-dimensional space to an n-dimensional space. Whenever $n > m$, the image of the map $\mathbf{g}$ does not cover the entire space and in such cases there is a nonzero n-variate polynomial, say $A(x_1, \ldots, x_n)$, which evaluates to zero on the entire image: $A(g_1, \ldots, g_n)$ is the zero polynomial. We call such an $A(\mathbf{x})$ an *annihilator* of the map $\mathbf{g}$. As n becomes further larger than m, the space of annihilators of $\mathbf{g}$ naturally becomes richer. Thus, given an m-variate map $\mathbf{g} = (g_1, \ldots, g_n)$ with n much larger than m, it is natural to expect a "simple" annihilator, and therefore, to try and find "the simplest" annihilator for it. In this work, we study the complexity of the annihilator of a given map in terms of the complexity of the map.

A natural model to study the complexity of multivariate polynomials is that of *algebraic circuits* (see Definition 2.1). The *algebraic circuit complexity* of a polynomial essentially captures the number of additions/subtractions and multiplications required to compute it at an arbitrary input. The landmark work of Valiant [Val79] initiated the study of algebraic circuit complexity by introducing *affine projections* as a notion of reductions between polynomials, and thus giving rise to complexity classes of polynomials that are inter-reducible. Using algebraic circuits, one then defines VP: class of efficiently computable polynomials, and VNP: the class of efficiently definable (or "explicit") polynomials (see Definition 2.5 and Definition 2.6). The central question in this area is whether all explicit polynomials are efficiently computable, or whether VP = VNP.

Similar to its boolean counterpart, the "VP vs VNP" question has proven to be notoriously hard in spite of it being a weaker[1] question (see e.g. [Bür00, Chapter 4]). Nevertheless, research on circuit lower bounds has provided significant insights in more structured variants culminating into the recent breakthrough by Limaye, Srinivasan and Tavenas [LST21] that proves a super-polynomial lower bound on an any constant-depth circuit for a polynomial in VP. We point the reader to some excellent surveys [SY10, CKW11, Sap15] for more on algebraic circuit lower bounds. Returning to unrestricted algebraic circuits, the current state is not that great: the best lower bound is a barely super-linear $O(n \log n)$ due to Baur and Strassen [BS83]. This dichotomy has led some experts to believe that there is a concrete barrier towards proving general circuit lower bounds using the techniques that have worked so far. This question about a possible barrier is one of the key motivations of our work. Before we expand on that, we will go over another central question in algebraic complexity that is intimately connected to circuit lower bounds, annihilators and even the above-mentioned question about barriers.

---

[1] Assuming the Generalised Reimann Hypothesis (GRH).

1.1 Polynomial Identity Testing

"Given an arbitrary n-variate polynomial, $f(\mathbf{x})$, of degree d that is computed by circuit of size s, how many queries to f suffice to decide whether f is identically zero?" This is the *blackbox polynomial identity testing (blackbox PIT)* question since we are treating f as a "black box" (as opposed to white-box PIT, where we are given access to the circuit of f).

When allowed randomness, this can be done in constantly many queries thanks to Polynomial Identity Lemma (or 'Schwartz-Zippel lemma') [Ore22, DL78, Zip79, Sch80]. Further, this holds regardless of f being computable by a small circuit. Asking for a *deterministic* blackbox PIT turns out to be equivalent to asking for a (fixed) set of queries that works for any valid $f(\mathbf{x})$ for the given parameters n, d and s. Such a set is called a *hitting set*. This is essentially a 'derandomization' question because a not-too-large set chosen at random works (see [HS80, For14]).

Deterministic[2] PIT, especially in the blackbox setting, is intimately connected to the question of proving strong circuit lower bounds (see, e.g., [HS80, KI04, KS19]), and is therefore a challenging problem. Almost all known blackbox PITs use the following two-step recipe.

(1) *Variable reduction* to $m \ll n$, using the bound on complexity of the input.

(2) *Trivial PIT*: use a hitting set (the "grid" S^m for a large enough set S) that works for all m-variate polynomials of the resulting degree.

The ingenuity then lies in the variable reduction, which is done using a *hitting set generator*.

A hitting set generator (HSG), for a set of polynomials P, is a vector of m-variate polynomials $\mathbf{g} = (g_1(\mathbf{z}), \ldots, g_n(\mathbf{z}))$ such that, for any nonzero $f \in P$, we have that $f(\mathbf{g}) \not\equiv 0$. A good HSG is a map $\mathbf{g}$ that has (1) a significant "stretch": $m \ll n$, and (2) low degree: $D := \max_i \deg(g_i) \in \text{poly}(d)$, both of which enable a much faster blackbox PIT. Any m-variate, degree-D HSG $\mathbf{g}$ implies the hitting set $\{\mathbf{g}(\mathbf{a}) : \mathbf{a} \in S^m\}$ for any S of size $\sim (D \cdot d)$ for the set of polynomials P.

Clearly an HSG for P is *exactly* a polynomial map whose annihilators are outside the set P and, therefore, strongly motivates studying annihilators of "efficiently computable polynomial maps" that have a significant "stretch".

1.2 Algebraic Natural Proofs

We now come to the other motivation for our work, which is about the possibility of a barrier towards proving strong circuit lower bounds. Faced with a similar situation in boolean circuit complexity — not being able to extend strong lower bounds against structured models to the general setting — Razborov and Rudich [RR97] produced their seminal work about *natural proofs*. They abstracted out a template from known boolean circuit lower bounds at the time and showed that the existence of (widely believed) strong enough cryptographic primitives implied that this

---

[2]Throughout the paper, when we say PIT, we mean deterministic PIT.

template is fundamentally incapable of proving super-polynomial boolean circuit lower bounds. Viewed constructively, this led complexity theorists to come up with road-maps to avoid this template (see, e.g., [Cho11, Wil13, CJSW21]).

The algebraic natural proofs framework introduced[3] by Forbes, Shpilka and Volk [FSV18], and Grochow, Kumar, Saks and Saraf [GKSS17], tries to capture a similar barrier in the algebraic setting, should one exist. It indeed turns out that most known algebraic circuit lower bounds follow a template that appears to be an algebraic analogue of (boolean) *natural proofs*. More formally, for most classes $\mathcal{C}$, a lower bound against $\mathcal{C}$ yields a family of polynomials in VP that vanish on the coefficient vectors of all polynomials in $\mathcal{C}$ (with the appropriate parameters). So, in order to prove lower bounds against VP in this template, there must exist such a polynomial family in VP that vanishes on the coefficient vectors of polynomial families that are in VP (see Section 2.1 for formal definitions). If we could rule out such families, even assuming some hardness assumptions like that in [RR97], this would constitute an 'algebraic natural proofs barrier'.

However, unlike the boolean world, there is no strong or widely believed assumption that proves the existence of a barrier. In fact, it can be argued that whatever is known only makes the question more intriguing. To start with, Forbes, Shpilka and Volk [FSV18] showed that any polynomial that vanishes on coefficient vectors of depth-3 multilinear formulas[4] cannot come from certain structured classes[5]; but we already know exponential lower bounds against multilinear formulas using natural proofs [Raz09, RY09]. Next, Chatterjee, Kumar, Ramya, Saptharishi and Tengse [CKR+20] showed that there are non-trivial polynomials in VP that vanish on polynomials in VP with bounded coefficients; their work can be seen as evidence against a barrier somewhat analoguous to Chow's work [Cho11]. Finally, Kumar, Ramya, Saptharishi and Tengse [KRST22] showed that if there is an exponential separation between VP and VNP then the class VNP does not have VP-natural proofs; which essentially means that any VP-natural proof for VP should be able to witness VP $\neq$ VNP! A more detailed summary of these results can be found in a survey by Volk [Vol21].

One thing that keeps the question about an algebraic natural proofs barrier from being settled is that the results in boolean cryptography do not seem to have any direct implications on it. The first (and more obvious) reason for this is that low-degree algebraic circuits are a weaker model and this prevent constructions of "pseudorandom polynomials" using boolean cryptographic primitives in a black-box fashion. The second reason is that the coefficient vector of a polynomial contains more information than the truth table of a boolean function. In particular, the truth table only captures the 'multilinear projection' of a polynomial and, even for simple polynomial families, such projections can simulate the Permanent (see Hrubeš [Hru16]). This perhaps intuitively explains the findings in [KRST22]. Quite interestingly, their result uses a seemingly

---

[3]Previously, Grochow [Gro15] and Aaronson and Drucker [AD08] had suggested similar formulations.

[4]Formulas are circuits whose underlying graphs are trees.

[5]For example, depth-3 with constant top fan-in, ROABPs (in every order) and sparse polynomials, among others.

weaker assumption than that used by Razborov and Rudich [RR97] (e.g., exponential hardness of factoring integers). We refer the readers to previous works ([AD08, FSV18]) for some more details about this.

Existence of an algebraic natural proofs barrier can naturally be viewed as a question about annihilators of explicit polynomial maps via the well-known notion of a *universal circuit* (see Lemma 2.11). In fact, explicit polynomial maps that are hard to annihilate are a fairly natural algebraic analogue of *pseudorandom function generators*.

1.3   Annihilators of polynomial maps

With these two strong connections in mind, we wish that our work generates interest towards constructing explicit polynomial maps that are hard to annihilate, under widely believed hardness assumptions. In particular, we are looking for size-s-explicit maps (see Definition 1.1), whose annihilators require sizes much larger than s. This setting is sometimes called the 'cryptographic setting', and we believe that this territory is largely unexplored. In particular, to the best of our knowledge, the only known provable example of such a generator map is due to Andrews [And22], that achieves a "super-quadratic stretch" if ω, the matrix multiplication exponent, is strictly bigger than 2. We now give a few more details about the above work and other relevant literature on the annihilators of polynomial maps.

There are multiple works that prove the HSG property for specific polynomial maps ([KS01, BT88, KI04, SV09, GKSS19]), and thus all of them implicitly prove lower bounds on annihilators of those maps. We will focus on two of these, because they prove conditional hardness against general algebraic circuits. Kabanets and Impagliazzo [KI04] adapted the construction of Nisan and Wigderson [NW94] and showed that any n-variate, $2^{n^{\varepsilon}}$-hard polynomial $h(\mathbf{z})$ can be converted into a polylog(n)-variate HSG for circuits of size roughly poly(n), which would give a quasi-polynomial blackbox PIT. They achieve this by showing that any annihilator of such a map has to be a multiple of a polylog(n)-variate $h(\mathbf{z})$. Guo, Kumar, Saptharishi and Solomon [GKSS19] devised a method of obtaining efficient blackbox PIT using an exponentially hard polynomial h. They require that h be hard because it vanishes on a hitting set for algebraic circuits, since their proof relies on the annihilator of their map vanishing on the same hitting set. Note that both these constructions give polynomial maps that require circuits of *larger* size than the circuits that they 'hit'. This is because in some sense the corresponding proofs rely on the hardness of a single polynomial in the map.

To the best of our knowledge, the earliest work that explicitly studies annihilators with the perspective of (algebraic) complexity theory is that of Kayal [Kay09], which focusses on maps that have a 'stretch' of just one: $n = m + 1$. Thus, in this case, the space of annihilators (which is an *ideal*) is generated by a single polynomial. Kayal shows an exponential lower bound on the degree of this polynomial. He further shows that even for constant degree maps of this kind, obtaining

4

any non-trivial information about their annihilator is at best NP-hard.

Next we have the works of van Melkebeek and Morgan [vMM22], Andrews and Forbes [AF22] and Andrews [And22], that study annihilators of specific polynomial maps. The work of van Melkebeek and Morgan generalizes the so-called "Shpilka-Volkovich generator" [SV09], which helps them analyse the space of annihilators. They show near-optimal bounds on the degree, sparsity and 'minimum support size' of any annihilator of their map. Andrews and Forbes [AF22] construct an HSG for constant depth circuits, using the breakthrough lower bound of Limaye, Srinivasan and Tavenas [LST21]. Their HSG generates all low-rank matrices and they show that any annihilator essentially simulates a large-enough determinant (which is hard for constant-depth circuits, due to [LST21]). Andrews [And22] extends the above work, and shows that the 'low-rank-matrices' HSG hits general circuits of slightly super-linear size, if the matrix multiplication exponent ω is strictly bigger than 2. As mentioned earlier, this is the only known instance of an HSG that works for circuits of some size s and is itself size-s'-explicit, for some $s' < s$.

To summarize the entire discussion, the difficulty in obtaining better bounds on the complexity of annihilators of explicit maps has two sides. Proving upper bounds, which is related to designing natural proofs, possibly requires us to capture a weakness of some model that computes the map. Whereas proving lower bounds on the circuit complexity of annihilators, which possibly gives blackbox PITs, seems tricky unless the ideal of annihilators is simple or structured.

Knowing these intimate connections, the question: "what is an upper bound on the annihilators of an explicit polynomial map?", is very much relevant. We will see that such annihilators definitely lie inside a class that is (probably) larger than VNP, called VPSPACE (see Section 2.2 for details). Even this seemingly simple upper bound reveals interesting connections between blackbox PITs, algebraic natural proofs, and lower bounds, including those in the boolean world. We now elaborate on our results and these connections, and then present our high level ideas, before giving some directions for future research. The technical component of our paper starts from Section 2 after that.

1.4 Our contributions

Our main result is the following upper bound on the annihilators of explicit polynomial maps.

Annihilators of explicit polynomials maps

Let us first formalize what we mean by explicit polynomial maps.

Definition 1.1 (Explicit polynomial maps)**.** *A polynomial map $\mathcal{G}(x_1, \ldots, x_m) = (g_1(\mathbf{x}), \ldots, g_n(\mathbf{x}))$ is said to have degree d if $g_i(\mathbf{x})$ has degree at most d for every $i \in [n]$. Further, we say that a circuit $C(\mathbf{z}, \mathbf{y})$ encodes $\mathcal{G}$ if there are assignments $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ to $\mathbf{y}$, such that for each $i \in [n]$, $C(\mathbf{x}, \mathbf{a}_i) = g_i(\mathbf{x})$.*

Analogously, for a family $\{\mathcal{G}_m\}$ of maps and a class $\mathcal{C}$, we say that $\{\mathcal{G}_m\}$ is $\mathcal{C}$-explicit if there is a family $\{C_m\} \in \mathcal{C}$ such that, for every m, C_m encodes $\mathcal{G}_m$. ◇

We focus on VP-explicit families of polynomial maps $\{\mathcal{G}_m\}$, where the degree and size of encoding circuits increases polynomially in the number of parameters m. As we shall soon see, showing unconditional bounds on annihilators of such maps in terms of VP or VNP would have strong consequences in complexity theory in general.

For now, the best upper bound we can show is in terms of circuits with special evaluation gates, called *projection gates* (see Definition 2.18), which characterizes the class VPSPACE in the same way as (usual) algebraic circuits characterize VP. A detailed discussion on this class can be found in Section 2.2 but over any fixed finite field, in particular, it coincides with the class of polynomials whose bits of coefficients can be computed in non-uniform PSPACE.

Theorem 1.2 (Upper bound for annihilators of explicit maps). *Let $\mathcal{G} : \mathbb{F}^m \to \mathbb{F}^n$ be a polynomial map given by $(g_1(\mathbf{z}), \ldots, g_n(\mathbf{z}))$ of individual degree d, with $n \geq 2m$.*

There exists a constant c such that, if there is a circuit $C_{\mathcal{G}}(\mathbf{z}, \mathbf{y})$ of size s that encodes $\mathcal{G}$ as per Definition 1.1, then there is a circuit with projection gates $C'(\mathbf{x})$ of size $(m \cdot d \cdot s)^c$ computing a nonzero polynomial $A(\mathbf{x})$ of individual degree at most $3 \cdot m \cdot d$ that annihilates $\mathcal{G}$; i.e. $A(g_1(\mathbf{z}), \ldots, g_n(\mathbf{z})) \equiv 0$.

In particular, we have the following.

Corollary 1.3. *For any VP-explicit family of maps $\{\mathcal{G}_m\}$, where each $\mathcal{G}_m$ has more than $2m$ outputs, there is a family $\{A_m\}$ in $\mathsf{VPSPACE}_b$ such that A_m annihilates $\mathcal{G}_m$ for all large enough m.*

In fact, it turns out that the upper bound in Theorem 1.2 holds even when the encoding circuit uses projection gates (see Theorem 3.1). That is, VPSPACE even annihilates (low-degree) VPSPACE-explicit maps. This can be phrased as 'VPSPACE is closed under taking annihilators', in the same way as 'VP is closed under taking homogenous components' and 'VNP is closed under taking arbitrary coefficients'.

Since the bound in Theorem 1.2 extends to VPSPACE-explicit families, it is believable that it can be improved; we expand on this in Section 1.6. That said, this result already reveals some interesting connections between lower bounds, identity testing and algebraic natural proofs.

Algebraic Natural Proofs

First, we get an upper bound on equations for evaluation vectors (similar to truth tables) of $\mathsf{VP}(n)$ in terms of $\mathsf{VPSPACE}(n^{\log^* n})$. It is easy to see that $\mathsf{VPSPACE}(n^{\log^* n})$ is incomparable to $\mathsf{VP}(N)$[6], and a $\mathsf{VP}(N)$ upper bound would rule out a natural proofs barrier! Using the definition of 'constant-free VPSPACE' (see Definition 2.17), we can state our theorem as follows.

---

[6]Here $\mathcal{C}(n)$ and $\mathcal{D}(N)$ mean that the complexities are polynomial in the parameters n and $N \approx 2^n$, respectively.

Theorem 1.4 (Equations for VP). *For an arbitrary $d(n) \in \text{poly}(n)$, and let $N(n) = \binom{n+d(n)}{n}$. For $t(n) := n^{\log^* n}$, there is a family of $N(n)$-variate, multilinear polynomials $\{P_N\}$ that depends only on the first $t(n)$ variables, satisfying the following.*

- *The family $\{P_N\}$ is a family of equations for the evaluation vectors of $\text{VP}_{d(n)}$.*

- *The coefficient functions of $\{P_N\}$ are computable in (uniform) space $t(n) = n^{\log^* n}$.*

Even though 'algebraic natural proofs' are defined in terms of coefficient vectors instead of evaluation vectors, these notions are more or less equivalent.

Proposition 1.5 (Informal version of Proposition 4.5). *For any $d(n)$ and $N = \binom{n+d}{n}$, equations for coefficient vectors of VP_d are computable in $\text{VP}(N)$ (or $\text{VNP}(N)$), if and only if, equations for evaluation vectors of VP_d are computable in $\text{VP}(N)$ (or $\text{VNP}(N)$).*

Thus, combining Theorem 3.1 and Proposition 1.5, we get the following corollary.

Theorem 1.6 (Hardness from absence of natural proofs). *If VP does not admit VP-natural proofs then either $\text{VP} \neq \text{VNP}$ or there is a language in $\text{DSPACE}(\log^{\log^* n} n)$ that is not in P.*

Also, if VP does not admit VNP-natural proofs then there is a language in $\text{DSPACE}(\log^{\log^* n} n)$ that is not in P.

Remark 1.7. *The function $\log^*(\cdot)$ in Theorem 1.4 and Theorem 1.6 can be replaced by any growing function in the relevant parameter.* ◇

Polynomial Identity Testing

Moving on to the problem of blackbox PIT, we first formally define hitting set generators.

Definition 1.8 (Hitting Set Generators for VP_d). *Fix $d(n) \in \text{poly}(n)$. A family of polynomial maps $\{\mathcal{H}_m\}$ from $\mathbb{C}^m$ to $\mathbb{C}^{n(m)}$, is said to be a hitting set generator for VP_d, if any family $\{A_n\}$ that annihilates $\{\mathcal{H}_m\}$ is outside the class VP_d.* ◇

We show that HSGs for algebraic circuits that have a super-polynomial 'stretch' would separate VP and VPSPACE.

Theorem 1.9 (Cryptographic HSGs separate VP from VPSPACE). *Let $\{\mathcal{H}_m\}$ be a VP-explicit family of m-variate polynomial maps with $2m$ outputs of degree m^8, and let $d(n) = n^{10}$.*

If the family $\{\mathcal{H}_m\}$ is a hitting set generator for VP_d, then $\text{VP} \neq \text{VPSPACE}_b$. As a consequence, either $\text{P/poly} \neq \text{PSPACE/poly}$ or $\text{VP} \neq \text{VNP}$.

As mentioned earlier, lower bounds shown in the work of Kayal [Kay09] do not rule out such HSGs since those bound crucially rely on the space of annihilators being generated by a single polynomial. On the other hand, our setting ensures that the space of annihilators is fairly rich and therefore guarantees low-degree annihilators. More importantly, even conditioned on standard hardness assumptions, it is no longer obvious whether any of the annihilators is computable by efficient algebraic circuits.

1.5 Proof Overview

The main ideas behind our proofs are a combination of elementary techniques and hence we will try to give a nearly complete outline of our proofs.

Upper bound on the annihilators

We begin with an overview of the proof of Theorem 1.2. Let $\mathcal{G} = (g_1(\mathbf{z}), \dots, g_n(\mathbf{z}))$ be an arbitrary tuple of degree-d polynomials, in $m < n/2$ variables, that is explicit (see Definition 1.1). The fact that **g** is explicit will be used later in the proof.

Observe that an annihilator $A(x_1, \dots, x_n) \in \mathbb{C}[\mathbf{x}]$ of *individual* degree $(D-1)$ for this tuple is precisely a (non-trivial) $\mathbb{C}$-linear dependency between products of $g_1, \dots, g_n$ of total degree $\leq n(D-1)$, where each g_i is multiplied at most $(D-1)$ times. Now, any such product is itself a polynomial in m variables of individual degree less than $d' := (nD \cdot d)$, and hence belongs to a vector space of dimension less than d'^m over $\mathbb{C}$. On the other hand, there are at least D^n of these products, so if D is large enough to ensure $D^n \geq d'^m = (ndD)^m$ then we are guaranteed some non-trivial linear dependency. For $n \geq 2m$ this happens for a $D \leq nd$.

Now we turn to finding the coefficients of this dependency, which are going to be the coefficients of our annihilator. For this, consider a matrix M with rows labelled by monomials in $\mathbf{z}$ of individual degree at most $(d'-1)$, and columns labelled by monomials in $\mathbf{x}$ of individual degree at most $(D-1)$. Order both the rows and columns lexicographically. For a monomial $\mathbf{x^e} = x_1^{e_1} \cdots x_n^{e_n}$, the $\mathbf{e}^{th}$ column of M is the coefficient vector of **g**. In this language, any column-dependency of M is the coefficient vector of an annihilator of $(g_1, \dots, g_n)$.

Since we are chasing an annihilator with as efficient a description as possible, it makes sense to pick the "lexicographically first" one. That is, let **e** be the "smallest" exponent such that $\mathbf{g^e}$ is dependent on the set of $\mathbf{g^{e'}}$s where all $\mathbf{e'}$s are "smaller than" **e**. Our annihilator will precisely be this unique (up to scaling by $\mathbb{C}$) dependency.

Given a linear system of equations, $Ax = 0$, with a unique (up to scaling) solution, *Cramer's rule* describes the solution in terms of determinants of submatrices of A. We will fix the 'last coefficient' to be $-\det(A')$ where A' is the submatrix that has all but the last column. This is done to ensure that all the coefficients are integers (and "sub-determinants of A"). In fact it turns out that, stated this way, the annihilator itself can be written as a *single* determinant by just adding a new last row with all the (symbolic) monomials $\mathbf{x^e}$! We will call this 'special Cramer's rule' for the remainder of this overview.

At this point, note that there is a minor issue. For all we know, the "first" dependency that we have chosen gives us a submatrix (all columns up to **e**) that has more rows than columns and we cannot apply Cramer's rule. Further, it is not even clear if the basis of rows has a small description (smaller than just its characteristic vector). We get around this using a rank extractor, in particular

the construction used by Forbes and Shpilka [FS12] (see Lemma 2.8), because it fits snugly with the structure of the matrix M. The arguments up to this point have been formalized in Lemma 3.2.

Now that we are in the setting of 'special Cramer's rule': we have an $(r+1) \times (r+1)$ matrix $\widetilde{M}$ whose determinant is the annihilator. Since $r \leq (md)^{O(m)}$ this structure is not enough on its own to guarantee a non-trivial upper bound. But we have even more structure in $\widetilde{M}$, owing to the explicitness of $\mathbf{g}$. Specifically we have that, for all $i \leq r$, the $(i, \mathbf{e}_j)^{th}$ entry of $\widetilde{M}$ is the evaluation of $\mathbf{g}^{\mathbf{e}_j}$ at a point, $\beta^{(i-1)} \in \mathbb{Z}^m$ and this can be calculated efficiently by a constant-free algebraic circuit when i is given in its binary representation. Since computing a circuit for $\mathbf{g}^{\mathbf{e}_j}$ given the binary encoding of $\mathbf{e}_j$ is also doable efficiently, we say that the matrix $\widetilde{M}$ is efficiently encodable using algebraic circuits. More formally, there is a size $\mathrm{poly}(\log r) = \mathrm{poly}(m, d, s)$ algebraic circuit which, when given bit-vectors corresponding to $i, j \in [r+1]$, outputs $\widetilde{M}[i, j]$, Note that this can sometimes be a monomial in $\mathbf{x}$. The formal argument can be found in the proof of Lemma 3.5.

Finally, we use the fact that determinant of an explicit matrix can itself be computed in a "memory-efficient" way. More formally, we use the elegant construction due to Mahajan and Vinay [MV97], of an *algebraic branching program* (see Definition 2.3) for the determinant polynomial, along with repeated squaring using *projection gates* (see Definition 2.18). Here, the fact that projection gates let us evaluate a polynomial at constant cost proves to be crucial. Quantitatively, given any matrix that is encoded by a circuit of size s', we can compute the determinant of this matrix using a circuit with projection gates of size $\mathrm{poly}(s')$. In fact, Malod [Mal11] uses this to give an alternative characterization of VPSPACE. This argument has been formalized in Proposition 2.27.

Putting all the pieces together (see Figure 1), we get a circuit with projection gates, of size $\mathrm{poly}(m, d, s)$ that computes the annihilator of the map $\mathcal{G} = (g_1, \ldots, g_n)$ as required.

Lower bounds from Constructive HSGs

Theorem 1.9 follows from Theorem 1.2 in a fairly straightforward manner. Suppose, for contradiction, that VP = VPSPACE$_b$. This means that any circuit with projection gates can be efficiently simulated by a circuit without projection gates. So, in particular, the circuits that compute the family of annihilators of the hitting set generator $\{\mathcal{H}_m\}$ can be converted to "usual" algebraic circuits of polynomially larger size. Finally, since the degree function $d(n)$ is large enough for the annihilators ensured by Theorem 1.2, we get that there is annihilator for $\{\mathcal{H}_m\}$ in VP$_d$, contradicting its 'hitting property' in the hypothesis.

Algebraic Natural Proofs

The observation that allows us to move from Theorem 1.2 to Theorem 1.6 is that we can work with *evaluation vectors* instead of *coefficient vectors*, especially when dealing with VP-natural or VNP-natural proofs (due to Proposition 1.5). This means that a family of equations can be constructed to vanish on the evaluation vector of the *universal circuit* of a slightly super-polynomial

size, say $n^{O(\log^* n)}$. Clearly, this evaluation vector is a $\text{size}(n^{O(\log^* n)})$-explicit map. Applying Theorem 1.2, we get a family of equations that can be computed by circuits with projection gates, of size $n^{O(\log^* n)}$. Thus, giving "VPSPACE($n^{\log^* n}$)-natural proofs for VP(n)". The theorem then follows by instantiating VPSPACE($n^{\log^* n}$) $\not\subset$ VP(N) or VNP(N) with the appropriate parameters.

1.6 Open directions

- The most important takeaway from our results, and the consequences, is the need for constructions of 'cryptographic HSGs' as in Theorem 1.9 and the more general Theorem 4.1. Specifically, we ask whether the arguments of Heintz and Schnorr [HS80] can be derandomized within space that is sub-polynomial in the parameters n, d and s. As this would satisfy the hypothesis of Theorem 4.1, achieving this even under a hardness assumption that is weaker (more believable) than $\mathsf{P} \neq \mathsf{PSPACE}$ would be interesting.

  Along the same lines, under what assumptions do 'sub-exponentially strong cryptographic HSGs' corresponding to Theorem 1.6 exist? There are some caveats[7] due to which boolean cryptographic primitives do not directly imply such HSGs.

- Another interesting thread is to improve the upper bound in this paper on annihilators of explicit polynomial maps. Since we expect VPSPACE$_b$ to be exponentially stronger than VP and VNP, any sub-exponential upper bound ($2^{o(s)}$ for size-s-explicit maps) in terms of VP or VNP would be interesting, even if it requires assuming some standard hypothesis.

  It appears that using the 'minimal dependency', as in Lemma 3.2, is unlikely to imply such a bound without observing some structure of VP. This is mainly because such an approach gives coefficients that are determinants of matrices of dimension $2^{\text{poly}(s)}$ and one has to argue that all these coefficients can be "compressed" into a smaller circuit.

  Note that Theorem 3.1 implies that the bound from Theorem 1.4 extends to equations for VNP. So, can we prove a better bound just for VP? Certainly, deriving any property of VP that does not hold for VNP should probably assume hardness of VNP (e.g. [KRST22]).

1.7 Organization of the paper

The rest of our paper is organized as follows. We first formalize some relevant concepts in Section 2, in which Section 2.2 is dedicated to the class VPSPACE. Then, we prove the upper bound on annihilators in Section 3, and formally state and prove its important consequences in Section 4. For completeness, we include a somewhat detailed proof sketch about the equivalence of two definitions of VPSPACE that we use, in Appendix A.

---

[7]This is beyond the scope of this paper. Readers may refer to the literature on algebraic natural proofs (e.g. [AD08],[FSV18]) for some details.
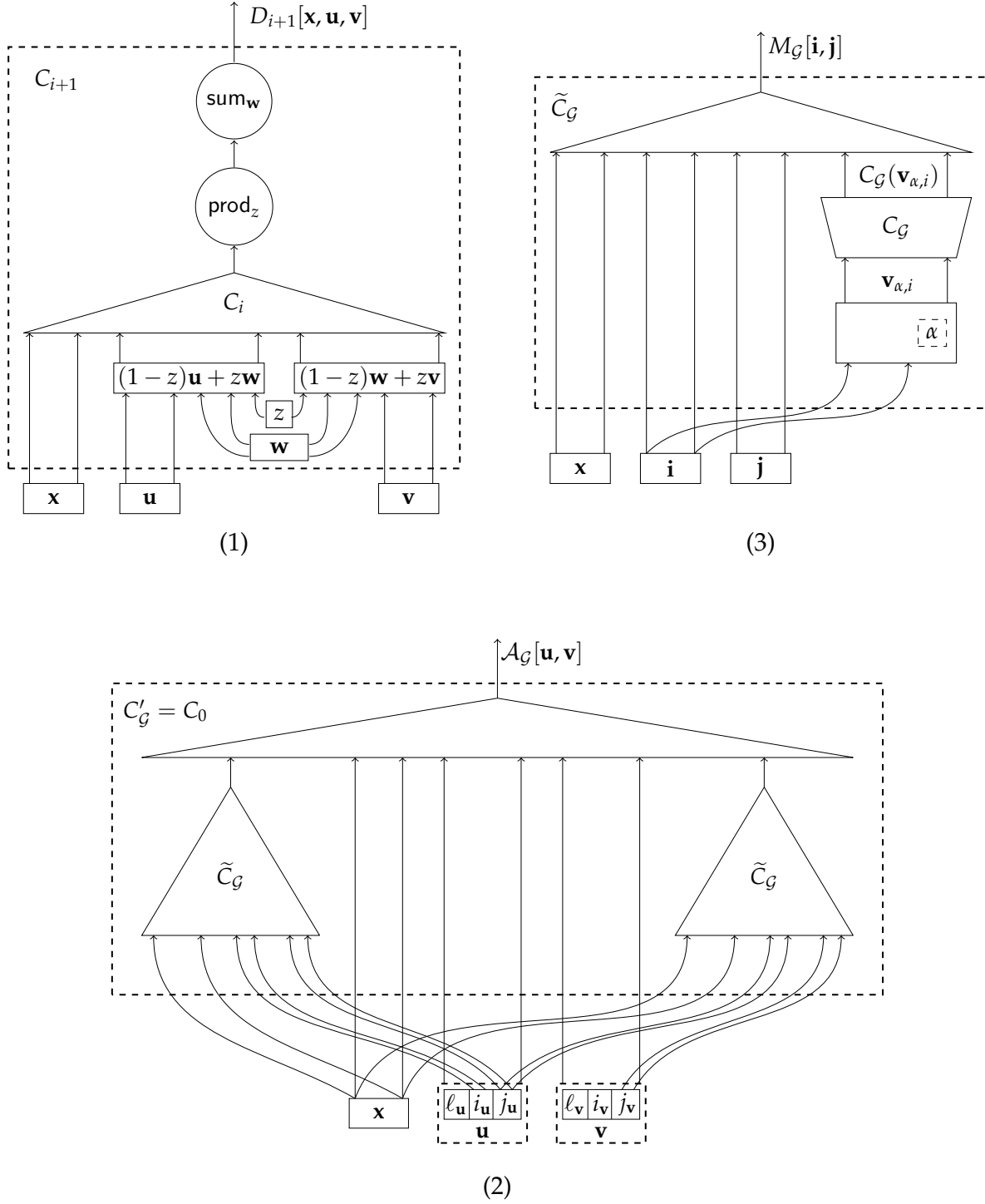
Figure 1: The annihilator is computed by using (1) repeatedly to get $C_{\log N}$, if N is the length of the ABP described by (2) (Claim 2.29); (2) describes the ABP computing $\det(\widetilde{M})$ using (3) (Claim 2.28); (3) describes the matrix $\widetilde{M}$, such that $\det(\widetilde{M}) \circ \mathcal{G} \equiv 0$, in terms of the generator $\mathcal{G}$ (Lemma 3.5).

2 Preliminaries

- We use $[n]$ to denote the set $\{1, \ldots, n\}$.

- We use boldface letters such as $\mathbf{x}, \mathbf{y}$ to denote tuples, typically of variables. When necessary, we adorn them with a subscript such as $\mathbf{y}_{[n]}$ to denote the length of the tuple. We also use $\mathbf{x}^{\mathbf{e}}$ to denote the monomial $\prod x_i^{e_i}$.

- We use $\{f_n\}_{n \in \mathbb{N}}$ to denote families of polynomials. We drop the index set whenever it is clear from context. For a given polynomial f we denote by $\deg(f)$ its degree. For a polynomial $f(\mathbf{x}, \mathbf{y}, \ldots)$ on multiple sets of variables, we use $\deg_{\mathbf{x}}(f), \deg_{\mathbf{y}}(f)$, etc., to denote the degree in the variables from the respective sets.

- For a given n-variate polynomial $f(\mathbf{x})$ of degree d, and a monomial m, we use $\operatorname{coeff}_m(f)$ to refer to the coefficient of m in f. We further use $\overline{\operatorname{coeff}}(f)$ to denote the vector[8] of coefficients of f.

- For a matrix M, we use $M[i, j]$ to refer to its $(i, j)^{th}$ entry. We commonly start our indices from 1, but sometimes start them from 0 when it helps the exposition. Whenever this is done, we make it clear.

Models of Algebraic Computation

We now formally define the models of algebraic computation that we need for this work. We first define algebraic circuits and their constant-free version.

Definition 2.1 (Algebraic Circuits)**.** *An* algebraic circuit *is specified by a directed acyclic graph, with leaves (nodes with in-degree zero, called* inputs*) labelled by field constants or variables, and internal nodes labelled by $+$ or $\times$. The nodes with out-degree zero are called the* outputs *of the circuit. Computation proceeds in the natural way, where inductively each $+$ gate computes the sum of its children and each $\times$ gate computes the product of its children.*

The size *of the circuit is defined as the number of edges (or* wires*) in the underlying graph.* ◇

Definition 2.2 (Constant-free Algebraic Circuits)**.** *An algebraic circuit is said to be "constant-free", if the only field constants that appear in it are 1 and -1. In order that the circuit is able to compute rational numbers, we allow it to have division gates, where both the inputs are field constants.* ◇

We also formally define algebraic branching programs.

Definition 2.3 (Algebraic Branching Programs (ABPs))**.** *An* algebraic branching program *is specified by a layered graph where each edge is labelled by an affine linear form and the first and the last layer have one vertex each, called the "source" and the "sink" vertex respectively. The polynomial computed by*

---

[8]We do not explicitly mention the monomial ordering used for this vector representation, since all our statements work for any monomial ordering.

an ABP is equal to the sum of the weights of all paths from the start vertex to the end vertex in the ABP, where the weight of a path is equal to the product of the labels of all the edges on it.

The width of a layer in an ABP is the number of vertices in it and the width of an ABP is the width of the layer that has the maximum number of vertices in it.

The size of an ABP is the number of edges in it. ◇

Remark 2.4. *We will use ABPs to mean ABPs that have been defined as above, but with edge labels also being allowed to be monomials of degree that is at most the size of the ABP.* ◇

Algebraic Complexity Classes

We now define the basic algebraic complexity classes.

Definition 2.5 (VP). *A family $\{f_n\}_n$ of polynomials is said to be in VP, if there exists a constant $c \in \mathbb{N}$ such that for all large n, f_n depends on at most n^c variables, has degree at most n^c, and is computable by an algebraic circuit of size at most n^c.* ◇

Definition 2.6 (VNP). *A family $\{f_n\}_n$ of polynomials is said to be in VNP, if there exists a constant $c \in \mathbb{N}$, and an m-variate family $\{g_m\} \in$ VP with $m, \text{size}(g_m) \leq n^c$, such that for all large enough n, f_n satisfies the following.*

$$f_n(\mathbf{x}) = \sum_{\mathbf{a} \in \{0,1\}^{|\mathbf{y}|}} g_m(\mathbf{x}, \mathbf{y} = \mathbf{a}) \qquad ◇$$

We will also be needing the following subclass of VP, especially in the context of algebraic natural proofs.

Definition 2.7 (VP$_d$). *For a function $d(n) \in \text{poly}(n)$, we define VP$_d$ to be the class of all degree-d families that belong to VP. Equivalently, a family $\{f_n\} \in$ VP is said to be in VP$_d$ if for all large enough n, $\deg(f_n) \leq d(n)$.*

We also define VNP$_d$ to contain all the degree-d families from VNP. ◇

Rank Extractor   We will also need the concept of rank extractors. A rank extractor is a family of maps which have rank preserving properties. The following lemma shows the existence of such a family in the setting that we will need.

Lemma 2.8 ([FS12]). *Let $1 \leq r \leq n$ and let $M \in \mathbb{C}^{n \times r}$ be of rank r. Define $A_\alpha \in \mathbb{C}^{r \times n}$ by $A_\alpha[i,j] = \alpha^{ij}$. Then there exists $\alpha \in [nr]$ such that $\text{rank}(A_\alpha \cdot M) = r$.* □

Polynomials for boolean operations   Finally we note that there are computationally simple polynomials which capture certain boolean operations. Since this is easy to check, we omit the proof.

Observation 2.9. *Given vectors $\mathbf{a}, \mathbf{b} \in \{0,1\}^\ell$, the following polynomials (or vectors of polynomials) have constant-free circuits of size $O(\ell^2)$.*

- EQ($\mathbf{a}, \mathbf{b}$): *outputs 1 if* $\mathbf{a} = \mathbf{b}$, 0 *otherwise.*

- GT($\mathbf{a}, \mathbf{b}$): *outputs 1 if* $\mathbf{a} > \mathbf{b}$ *when seen as binary encodings of positive integers,* 0 *otherwise.*

- LT($\mathbf{a}, \mathbf{b}$): *outputs 1 if* $\mathbf{a} < \mathbf{b}$ *when seen as binary encodings of positive integers,* 0 *otherwise.*

- INC($\mathbf{a}$): *outputs a vector* $\mathbf{b}$ *such that* $\mathbf{b} = \mathbf{a} + 1$, *when seen as positive integers.* □

2.1 Universal Circuits and Natural Proofs

A circuit is said to be universal for circuits of size s if every such circuit is a simple projection of it.

Definition 2.10 (Universal Circuit [SY10])**.** *A circuit $\mathcal{U}$ is called universal for n-input circuits of size s, that compute polynomials of degree d, if the following holds.*

> *For any polynomial $f(x_1, \ldots, x_n)$ of degree d that can be computed by a circuit of size s, there exists a circuit φ computing f as well, such that the computation graph of $\mathcal{U}$ is the same as the graph of φ.* ◊

The following lemma due to Raz [Raz10] shows the existence of such circuits. For a proof sketch that yields the exact statement given below, please refer to [CKR$^+$20].

Lemma 2.11 (Existence of Universal Circuits [Raz10])**.** *Let $\mathbb{F}$ be any field and $n, s \geq 1$ and $d \geq 0$. Then there exists an algebraic circuit $\mathcal{U}$ of size $\mathrm{poly}(n, d, s)$ computing a polynomial in $\mathbb{F}[x_1, \ldots, x_n, y_1, \ldots, y_r]$ with $r \leq \mathrm{poly}(n, d, s)$ such that:*

- $\deg_{\mathbf{x}}(\mathcal{U}(\mathbf{x}, \mathbf{y})), \deg_{\mathbf{y}}(\mathcal{U}(\mathbf{x}, \mathbf{y})) \leq \mathrm{poly}(d)$;

- *for any $f \in \mathbb{F}[x_1, \ldots, x_n]$ with $\deg_{\mathbf{x}}(f) \leq d$ that is computable by an algebraic circuit of size s, there exists an $\mathbf{a} \in \mathbb{F}^r$ such that $f(\mathbf{x}) = \mathcal{U}(\mathbf{x}, \mathbf{a})$.* □

Algebraic Natural Proofs

We now define the concepts around natural proofs that we will need.

Definition 2.12 (Degree-d families)**.** *For any function $d : \mathbb{N} \to \mathbb{N}$, we denote by $\mathcal{P}_d$ the class of all degree-d polynomial families. That is, a family $\{f_n\}$ of n-variate polynomials belongs to $\mathcal{P}_d$ if $\deg(f_n) \leq d(n)$ for all large enough n.* ◊

Definition 2.13 ($\mathcal{D}$-Natural Proofs for $\mathcal{C}$)**.** *Let $d(n), D(N)$ be polynomially growing functions, and let $\mathcal{C} \in \mathcal{P}_d$ and $\mathcal{D} \in \mathcal{P}_D$ be any classes of polynomial families. For $N(n) = \binom{n + d(n)}{n}$, we say that an N-variate family $\{A_N\} \in \mathcal{D}$ is a $\mathcal{D}$-natural proof for $\mathcal{C}$ if for any family $\{f_n\} \in \mathcal{C}$ we have that the polynomial $A_{N(n)}$ vanishes on the coefficient vector of f_n, for all large enough n.* ◊

Definition 2.14 (VP, VNP, and natural proofs)**.** *For a fixed $d(n) \in \mathrm{poly}(n)$, we say that VP-natural proofs exist for VP_d if for some $D(N) \in \mathrm{poly}(N)$, there exists a family $\{A_N\}$ that is a VP_D-natural proof*

for VP_d. *We say that* VP-*natural proofs exist for* VP *if there is some* VP-*natural proof for* VP_d, *for every* $d(n) \in \mathrm{poly}(n)$.

Similarly, we say that there are VNP-*natural proofs for* VP, *if there is some* VNP-*natural proof for each* $d(n) \in \mathrm{poly}(n)$. ◇

2.2 Algebraic analogues of bounded-space computation: VPSPACE

The upper bound on the annihilator that we show is in terms of space complexity.

Definition 2.15 (DSPACE)**.** *We use* $\mathsf{DSPACE}(s(n))$ *to denote the class of languages that can be decided by deterministic Turing Machines in space* $s(n)$ *for all inputs of length n.* ◇

Defining the class

One way to define the algebraic analogue of PSPACE is using the space needed to compute coefficients of the polynomial, which was the route taken by Koiran and Perifel [KP09].

Definition 2.16 (Coefficient function)**.** *Suppose* $f(x_1, \ldots, x_n)$ *is a polynomial of* individual *degree d, and each of its coefficient is an integer of absolute value* $< 2^\ell$.

Then, for $M = n \cdot \lceil \log(d+1) \rceil + (\ell + 1)$, *we define the coefficient function of f to be the M-bit boolean function* Φ_f *such that* $\Phi_f(e_1, \ldots, e_n, i)$ *outputs the* i^{th} *bit of the coefficient of the monomial* $x_1^{e_1} \cdots x_n^{e_n}$, *where the* 0th *bit encodes the sign.*

A family $\{f_n\}$ *of integer polynomials naturally defines a family* $\{\Phi_n\}$ *of coefficient functions.* ◇

Definition 2.17 ($\mathsf{VPSPACE}^0$ from coefficient functions [KP09])**.** *A family* $\{P_N\}$ *of integer polynomials is said to be in* $\mathsf{VPSPACE}^0$ *if, for all large N, the polynomial* P_N *has* $\mathrm{poly}(N)$ *variables, degree* $2^{\mathrm{poly}(N)}$ *and if the coefficient function of* P_N, Φ_{P_N} *is computable in* $\mathsf{PSPACE}/\mathrm{poly}$. ◇

Poizat [Poi08] on the other hand defined VPSPACE without going into boolean computation, using a new type of gate called projection gate, defined below. He showed[9] that this definition is equivalent to that of Koiran and Perifel.

Definition 2.18 (Projection gates)**.** *A projection gate is labelled by a variable, say w, and a constant* $b \in \{0, 1\}$. *We denote such a gate by* $\mathrm{fix}_{w=b}$. *The gate* $\mathrm{fix}_{w=b}$ *"projects" w to b in the input polynomial. That is,* $\mathrm{fix}_{w=b} f(w, \mathbf{x}) = f(b, \mathbf{x})$. ◇

Definition 2.19 (Circuits with projections [Poi08])**.** *A family* $\{P_N\}$ *of integer polynomials is said to be in* VPPROJ^0 *if for all large N, there is a constant-free algebraic circuit, say C, that additionally have access to projection gates such that C has size* $\mathrm{poly}(N)$ *and* P_N *is the polynomial computed by C.* ◇

Proposition 2.20 ([Poi08, Mal11])**.** $\mathsf{VPSPACE}^0 = \mathsf{VPPROJ}^0$.

---

[9]Poizat's paper (written in French) includes a rough sketch of this proof. Malod's paper [Mal11] (in English) quotes his result but does not provide a proof.

We believe that this statement is not entirely obvious, so we give a fairly detailed proof sketch of this in Appendix A. Finally, for polynomials with arbitrary constants, VPSPACE is defined using VPSPACE^0 as follows.

Definition 2.21 (VPSPACE). *Let $\mathbb{F}$ be the field of rationals, reals or complexes. A family $\{P_N\}$ of polynomials over $\mathbb{F}$ is in VPSPACE, if there exists an $M = \mathrm{poly}(N)$ and a family $\{Q_M\} \in$ VPSPACE^0, such that for all N, P_N is obtained from Q_M by setting some variables to constants from $\mathbb{F}$.* ◇

For convenience, we also define two new types of gates: summation and production gates.

Definition 2.22 (Summations and Productions). *The summation and production operations are defined, using the operation of projection as defined in Definition 2.18, as follows.*

- $\mathrm{sum}_z f(z, \mathbf{x}) := \mathrm{fix}_{z=0} f(z, \mathbf{x}) + \mathrm{fix}_{z=1} f(z, \mathbf{x})$

- $\mathrm{prod}_z f(z, \mathbf{x}) := \mathrm{fix}_{z=0} f(z, \mathbf{x}) \times \mathrm{fix}_{z=1} f(z, \mathbf{x})$ ◇

It is easy to see that, by definition, these gates can be simulated using projection gates and the usual sum, product gates.

Before moving on, it is important to note that there are (at least) two more equivalent characterizations of VPSPACE due to Malod [Mal11], and Mahajan and Rao [MR13]. We omit the details of these definitions as we do not directly use them.

Comparison with VP

Given that VPSPACE corresponds to a class as powerful as PSPACE, it is worth checking that the known "hard polynomial families" are outside VPSPACE as well. In this context, it makes sense to define a bounded degree analogue of VPSPACE written as VPSPACE$_b$. As Koiran and Perifel [KP09] showed, the degree bound is inconsequential when comparing VP and VPSPACE.

Proposition 2.23 (Restatement of [KP09, Lemma 4]). *Let VPSPACE$_b$ be the polynomial families of degree $\mathrm{poly}(n)$ that belong to VPSPACE, and similarly, let VP$_{nb}$ be the class of polynomial families of unbounded degree that have algebraic circuits of size $\mathrm{poly}(n)$.*

Then, VP $=$ VPSPACE$_b$ if and only if VP$_{nb} =$ VPSPACE. □

We can now summarize the relationship of VPSPACE$_b$ with known hard polynomials as follows, essentially by using the arguments in the well-known book by Bürgisser [Bür00].

- Since a *random polynomial* does not have small circuits with projection gates with high probability, random polynomial families are outside VPSPACE$_b$.

- Any construction that involves exponentially many "independent irrational numbers", e.g. $h_c(\mathbf{x}) = \sum_{\mathbf{e}} \sqrt{p_{\mathbf{e}}} \cdot \mathbf{x}^{\mathbf{e}}$ for distinct primes $\{p_{\mathbf{e}}\}$, also works against VPSPACE$_b$. This is because the coefficients of polynomials in VPSPACE$_b$ can always be written as integer polynomials that depend on at most polynomially many scalars from the underlying field.

- Finally, *Strassen's multilinear polynomial*, defined[10] as $h_s(\mathbf{x}) = \sum_{0 \le i < 2^n} 2^{2^i} \mathbf{x}^\mathbf{i}$ has coefficients that are triply exponential in the number of variables n. This means that we will need doubly exponentially many bits to even index into the bits of the coefficients, which puts the (corresponding family of) coefficient functions outside PSPACE.

Koiran and Perifel [KP09] further showed that separating VP and VPSPACE$_b$ would imply other lower bounds, as follows.

Proposition 2.24 ([KP09, Proposition 3]). *If* VP $\ne$ VPSPACE$_b$, *then either* VP $\ne$ VNP *or* P$/$poly $\ne$ PSPACE$/$poly.

Assuming the Generalized Riemann Hypothesis (GRH), the converse is also true. That is, if VP $\ne$ VNP *or* P$/$poly $\ne$ PSPACE$/$poly *then* VP $\ne$ VPSPACE$_b$. □

2.3 Explicit Objects

Next, along the same lines as *explicit* polynomial maps (Definition 1.1), we define explicit matrices and explicit ABPs. As before, we use the word *explicit* to mean encodable by efficient circuits.

Definition 2.25 (Explicit matrices). *A circuit* $C(\mathbf{x}, \mathbf{a}, \mathbf{b})$ *is said to* encode *a matrix* $M \in \mathbb{F}^{r \times c}$ *if* $|\mathbf{a}| = \lceil \log r \rceil$, $|\mathbf{b}| = \lceil \log c \rceil$ *and for* $\mathbf{i}, \mathbf{j}$ *being binary representations of* i, j *respectively,* $C(\mathbf{x}, \mathbf{i}, \mathbf{j}) = M[i, j]$.

Analogously, for a family of matrices $\{M_{r,c} : M_{r,c} \text{ has } r \text{ rows and } c \text{ columns}\}$ *and a class* $\mathcal{C}$, *we say that* $\{M_{r,c}\}$ *is* $\mathcal{C}$-explicit *if there is a family* $\{C_{r,c}\} \in \mathcal{C}$ *such that, for every* r, c, $C_{r,c}$ *encodes* $M_{r,c}$. ◇

Definition 2.26 (Explicit ABPs). *A circuit* $C_A(\mathbf{x}, (\mathbf{a}, \mathbf{b}), (\mathbf{a}', \mathbf{b}'))$ *is said to* encode *an algebraic branching program* A, *of width* w *and* d *layers, if* $|\mathbf{a}| = |\mathbf{a}'| = \lceil \log d \rceil$, $|\mathbf{b}| = |\mathbf{b}'| = \lceil \log w \rceil$ *and*

$$\forall \mathbf{i}, \mathbf{i}' \in \{0, 1\}^{\lceil \log d \rceil} \quad \forall \mathbf{j}, \mathbf{j}' \in \{0, 1\}^{\lceil \log w \rceil},$$
$$C_A(\mathbf{x}, (\mathbf{i}, \mathbf{j}), (\mathbf{i}', \mathbf{j}')) \text{ is the label on the edge } ((i, j), (i', j')) \text{ in } A.$$

Here $\mathbf{i}, \mathbf{j}, \mathbf{i}', \mathbf{j}'$ *are the binary representations of* i, j, i', j' *respectively and* $((i, j), (i', j'))$ *denotes the edge between the* j-th vertex in layer i and the j'-th vertex in layer i'.

Analogously, for a family of algebraic branching programs $\{A_{w,d} : A_{w,d} \text{ has width } w \text{ and } d \text{ layers}\}$ *and a class* $\mathcal{C}$, *we say that* $\{A_{w,d}\}$ *is* $\mathcal{C}$-explicit *if there is a family* $\{C_{w,d}\} \in \mathcal{C}$ *such that, for every* w, d, $C_{w,d}$ *encodes* $M_{w,d}$. ◇

2.4 Computing determinants of explicit matrices in VPSPACE

We now show that given a matrix that is encoded by a small circuit, its determinant can be computed using a small circuit with projection gates. A similar result already appears in a work by Malod [Mal11] and we provide a proof here for completeness.

---

[10]Here $\mathbf{i}$ is the vector corresponding to the binary representation of i.

Proposition 2.27 (Computing determinant of explicit matrices)**.** *Let $M \in \mathbb{C}^{N \times N}$ be a matrix that is encoded by a circuit $C(\mathbf{x}, \mathbf{a}, \mathbf{b})$ with $|\mathbf{a}| = |\mathbf{b}| = \lceil \log N \rceil$.*

Then $\det(M)$ can be computed by a circuit C' with projection gates, of size $O(\text{size}(C) + \log^2 N)$. Moreover, if C is constant-free, then so is C'.

Proof. The proof has two parts. First we show that $\det(M)$ can be computed by an explicit Algebraic Branching Program (ABP) and then show that the polynomial computed by an explicit ABP can also be efficiently computed by circuits with projection gates.

The first step is easy to deduce from the well-known elegant construction due to Mahajan and Vinay [MV97] and the second step is just a careful application of "repeated squaring" for matrices. We now prove these formally.

Claim 2.28. *There is an explicit ABP of size $\text{poly}(\text{size}(C_M), \log N)$, say $A(\mathbf{x})$, that computes $\det(M)$.*

Proof. We just describe the ABP computing $\det(M)$ here and also the circuit that encodes it. For a crisp proof of why this ABP computes the determinant, we direct the reader to Saptharishi's survey [Sap15, Section 3.3.3].

The ABP has $N + 1$ layers of vertices and, except for the first and the last layer, each layer has $O(N^2)$ vertices. Vertices of the ABP are labelled by $(\ell, (i, j))$ where $\ell \in [N + 1]$ denotes the layer and $(i, j) \in [N] \times [N]$ indexes a particular vertex in that layer. The edges are given as below.

- For each vertex $(\ell, (i, j))$ with $\ell \in [N - 1]$, there is an edge to each $(\ell + 1, (i, k))$ where $k > i$. The label of the edge is $M[j, k]$.

- For each vertex $(\ell, (i, j))$ with $\ell \in [N - 1]$, there is an edge to each $(\ell + 1, (k, k))$ where $k > i$. The label of the edge is $-M[j, i]$.

- All vertices $(N, (i, j))$ have an edge to the sink $(N + 1, (1, 1))$, with the label $-M[j, i]$.

We now describe the circuit $\widetilde{C}$ that encodes the above ABP, using the circuit C_M. Each vertex of the ABP is a vector of $3 \lceil \log N \rceil$ bits: $\lceil \log N \rceil$ each for ℓ, i and j as described in the above construction. In the circuit, the two input vertices are $\mathbf{u} \equiv (\ell_u, (i_u, j_u))$ and $\mathbf{v} \equiv (\ell_v, (i_v, j_v))$[11]. Using the polynomials defined in Observation 2.9, we get the following expression for $\widetilde{C}$.

$$\text{Let } G(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \text{EQ}(\ell_v, \text{INC}(\ell_u)) \cdot \text{EQ}(i_u, i_v) \cdot \text{GT}(j_v, i_u) \cdot C(\mathbf{x}, j_u, j_v)$$
$$+ \text{EQ}(\ell_v, \text{INC}(\ell_u)) \cdot \text{GT}(i_v, i_u) \cdot \text{EQ}(i_v, j_v) \cdot (-1 \cdot C(\mathbf{x}, i_u, j_u))$$
$$+ \text{EQ}(\ell_v, \text{INC}(\ell_u)) \cdot \text{EQ}(\ell_u, \mathbf{n}) \cdot \text{EQ}(0 \dots 01, i_v) \cdot \text{EQ}(i_v, j_v) \cdot (-1 \cdot C(\mathbf{x}, i_u, j_u)),$$
$$\text{and valid}(\mathbf{u}) = \text{LT}(0 \dots 00, \ell_u) \cdot \text{LT}(\ell_u, \text{INC}(\text{INC}(\mathbf{n}))) \cdot \text{int}(i_u) \cdot \text{int}(j_u),$$
$$\text{where int}(\mathbf{a}) = \text{LT}(0 \dots 00, \mathbf{a}) \cdot \text{LT}(\mathbf{a}, \text{INC}(\mathbf{n})).$$
$$\text{Then } \widetilde{C}(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \text{valid}(\mathbf{u}) \cdot \text{valid}(\mathbf{v}) \cdot g(\mathbf{x}, \mathbf{u}, \mathbf{v}).$$

---

[11]Here ℓ, i, j for both $\mathbf{u}$ and $\mathbf{v}$ are bit-vectors. We have used plain lowercase symbols since we think of them as numbers.

Here each 'product term' in the definition of G corresponds to one type of edge in the description above, and $\mathbf{n}$, $0\ldots00$ and $0\ldots01$ are the bit-vectors corresponding to the numbers N, 0 and 1 respectively. We use the polynomial "valid" to ensure that $\widetilde{C}$ outputs 0 whenever it is given a pair of labels that is of a 'non-edge'.

The correctness of this expression is easy to check using the description of the ABP. Also, $\widetilde{C}$ has size $O(\text{size}(C) + \log^2 N)$ and is a constant-free algebraic circuit (without projection gates) if C is constant free. $\qquad\square$

Claim 2.29. *There is a circuit C' with projection gates, of size $\text{poly}(\text{size}(\widetilde{C}), \log N)$ that computes the polynomial computed by the above ABP.*

Proof. Let A be the adjacency matrix of the graph that underlies the branching program. The polynomial computed by the ABP is just the (s, t)-th entry of the matrix A^N, where $s = (1, (1, 1))$ and $t = (N + 1, (1, 1))$ are the tuples corresponding to the source and the sink. Since $\widetilde{C}$ encodes A, we only need to figure out how to encode A^2, which we can then use recursively to obtain access to the entries of A^{2^k} for any k. Note that this becomes much easier when N is a power of 2.

So we modify the ABP by adding layers $N + 2, \ldots, N' + 1$, to ensure that N' is a power of 2. Each of the new layers have a single vertex with the label $(\ell, (1, 1))$ where ℓ is the layer. We also add an edge from $(\ell, 1, 1)$ to $(\ell + 1, 1, 1)$, labelled with the scalar 1, for every $N < \ell \leq N'$. This new ABP is explicit since we can add the term

$$\text{EQ}(\ell_v, \text{INC}(\ell_u)) \cdot \text{GT}(\ell_u, \mathbf{n}) \cdot \text{EQ}(0\ldots01, i_u) \cdot \text{EQ}(i_u, j_u) \cdot \text{EQ}(0\ldots01, i_v) \cdot \text{EQ}(i_v, j_v) \cdot 1$$

to the polynomial G defined above, and also modify valid$(\mathbf{u})$ appropriately, to obtain a circuit that encodes the new ABP.

Now note that the (s, t)-th entry of $A^{N'}$ is the same as the (s, t)-th entry of A^N, and therefore is the polynomial we are looking for. Further, we can now easily compute the (s, t)-th entry of $A^{N'}$ using recursion, as described earlier. We now describe a circuit using projection gates that carries out this operation.

Consider the following circuits defined using projections (Definition 2.18) and summations (Definition 2.22).

$$P_1(\mathbf{x}, \mathbf{u}, \mathbf{w}, \mathbf{v}, z) := \widetilde{C}\left(\mathbf{x}, ((1 - z)\mathbf{u} + z\mathbf{w}), ((1 - z)\mathbf{w} + z\mathbf{v})\right)$$
$$D_1(\mathbf{x}, \mathbf{u}, \mathbf{v}) := \text{sum}_{w_1} \text{sum}_{w_2} \cdots \text{sum}_{w_L} \left(\text{fix}_{z=0} P_1(\mathbf{x}, \mathbf{u}, \mathbf{w}, \mathbf{v}, z) \cdot \text{fix}_{z=1} P_1(\mathbf{x}, \mathbf{u}, \mathbf{w}, \mathbf{v}, z)\right)$$

Note that D_1 encodes the adjacency matrix A^2 when $\widetilde{C}$ encodes A and $L = O(\log N)$ is the length of the vertex labels of A.

Now for every $2 \leq i \leq k = \lceil \log(N + 1) \rceil$, we define P_{i+1} by using D_i in place of $\widetilde{C}$, and D_{i+1} by using P_{i+1} in place of P_1, in the definitions of P_1 and D_1 above.

As each of the P_{i+1}s and D_{i+1}s have circuits (with projection gates) of size poly($\log N$) with *exactly one* use of D_i and P_{i+1} respectively, all these increases are additive. So, finally, $C'(\mathbf{x}) = D_k(\mathbf{x}, s, t)$ has a circuit of size $O(\text{size}(\widetilde{C}) + \log^2 N)$ and computes the polynomial we need. Further, C' is constant-free if $\widetilde{C}$ is constant free. □

Combining both the above claims, we get that $\text{size}(C') = O(\text{size}(C) + \log^2 N)$, and also that C' is constant-free whenever C is constant-free. □

3 VPSPACE Upper Bounds for Annihilators

We now prove our main theorem, namely Theorem 1.2. In fact, we will prove the following (much) stronger statement.

Theorem 3.1 (Annihilators of explicit maps). *Let m be large enough, and let $\mathcal{G} : \mathbb{F}^m \to \mathbb{F}^n$ be a polynomial map given by $(g_1(\mathbf{z}), \dots, g_n(\mathbf{z}))$ of degree d, with $n \geq 2m$.*

There exists a constant c such that, if there is a circuit with projection gates $C_{\mathcal{G}}(\mathbf{z}, \mathbf{y})$ of size s that encodes $\mathcal{G}$ as per Definition 1.1, then there is a circuit with projection gates $C'(\mathbf{x})$ of size $(m \cdot d \cdot s)^c$ computing a nonzero polynomial $A(\mathbf{x})$ of individual degree at most $3 \cdot m \cdot d$ that annihilates $\mathcal{G}$ (that is, $A \circ \mathcal{G}_m = A(g_1(\mathbf{z}), \dots, g_n(\mathbf{z})) \equiv 0$).

To prove this, we will first show that there is an *explicit* matrix whose determinant is the annihilator, and then show that in this case the annihilator has a small circuit with projection gates.

3.1 Annihilator as determinant of a matrix

Formally, we will prove the following..

Lemma 3.2 (Annihilator as a determinant). *Let $\mathcal{G}$ be a polynomial map as given in Theorem 3.1. Further, let $C_{\mathcal{G}}(\mathbf{z}, \mathbf{y})$ be a circuit that generates $\mathcal{G}$.*

Then for $D = \left\lceil (nd)^{m/(n-m)} \right\rceil + 1$, there exists a $K \times K$ matrix $\widetilde{M}$ with $K \leq D^n$ such that $\det(\widetilde{M})$ is an annihilator of $\mathcal{G}$ that has individual degree at most $D - 1$. Moreover, $\widetilde{M}$ can be described as follows.

There is some positive integer $\alpha < D^{2n}$ such that, for $i \in \{0, \dots, K-1\}$ and $j \in [K]$,

$$
\widetilde{M}[i, \mathbf{e}^{(j)}] = \begin{cases} (\mathcal{G}(\mathbf{v}_{\alpha,i}))^{\mathbf{e}^{(j)}} & \text{if } i \leq K-2 \\ (-1)^{K-1} \cdot \mathbf{x}^{\mathbf{e}^{(j)}} & \text{if } i = K-1, \end{cases}
$$

where $\mathbf{v}_{\alpha,i} = (\alpha^i, \alpha^{i \cdot ndD}, \dots, \alpha^{i \cdot (ndD)^{m-1}})$.

Proof. First we see how linear dependencies of specific polynomials in terms of $\mathcal{G}$ give us annihilators of the map $\mathcal{G}$. Suppose $A(\mathbf{x})$ is an annihilator of $\mathcal{G}$, and let $A(\mathbf{x}) = \sum_{\mathbf{e}} f_{\mathbf{e}} \mathbf{x}^{\mathbf{e}}$ be its sparse

representation. Then $0 = A(\mathcal{G}) = \sum_{\mathbf{e}} f_{\mathbf{e}} \mathcal{G}^{\mathbf{e}}$. In other words, the coefficient vector of A is a linear dependency in the set of polynomials of the form $\mathcal{G}^{\mathbf{e}}$, as we range over all the exponent vectors $\mathbf{e}$. Clearly, the converse is also true: for any polynomial $A(\mathbf{x}) = \sum_{\mathbf{e}} f_{\mathbf{e}} \mathbf{x}^{\mathbf{e}}$, if the coefficients $\{f_{\mathbf{e}}\}$ represent a linear dependency in the polynomials $\{\mathcal{G}^{\mathbf{e}}\}$, then $F(\mathcal{G}) = 0$.

Now, for a parameter D to be fixed later, consider the following matrix M.

- Columns of M are indexed by monomials in $\mathbf{x} = \{x_1, \dots, x_n\}$ of individual degree at most $D - 1$, ordered lexicographically.

- Rows of M are indexed by monomials in $\mathbf{z} = \{z_1, \dots, z_m\}$ of individual degree at most $(ndD) - 1$, ordered lexicographically.

- For any valid $\mathbf{e} \in \mathbb{N}^n$, the $\mathbf{e}$th column of M is the coefficient vector of the product $\mathcal{G}^{\mathbf{e}} = g_1(\mathbf{z})^{e_1} \cdots g_n(\mathbf{z})^{e_n}$. Note that this product is an m-variate polynomial of individual degree at most $|\mathbf{e}| \le (n \cdot (D - 1)) \cdot d < (ndD) - 1$.

From the previous discussion, we can see that any nonzero dependency in the columns of M is the coefficient vector of some annihilator of $\mathcal{G}$. Now observe that M has fewer rows than columns whenever $D > (nd)^{m/(n-m)}$, since $(ndD)^m < D^n$ in this case. Therefore, there is a non-trivial dependency in its columns if we set $D = \left\lceil (nd)^{m/(n-m)} \right\rceil + 1$.

In order to ensure a *unique* dependency (up to scaling), we restrict M to its first K columns, where $K - 1$ is the largest number for which the first $K - 1$ columns are linearly independent. So now M has the following properties.

- M has $K \le D^n$ columns and each column is labelled by a monomial in $\{x_1, \dots, x_n\}$ of individual degree $\le D - 1$. Let $\left\{ \mathbf{e}^{(j)} \right\}_{j \in [K]}$ be the labels of the columns.

- Each row is labelled by a monomial in $\{z_1, \dots, z_m\}$ of individual degree $\le (ndD) - 1$ and so, if the number of rows in M is R, then

$$R \le (ndD)^m < D^n$$

- The $(\mathbf{z}^{\mathbf{e}}, \mathbf{e}^{(j)})$-th entry of M is the coefficient of $\mathbf{z}^{\mathbf{e}}$ in $\mathcal{G}^{\mathbf{e}^{(j)}}$.

- The first $K - 1$ columns are linearly independent and the last column is a linear combination of the previous columns.

- $A(\mathbf{x}) = \mathbf{x}^{\mathbf{e}^{(K)}} - \sum_{j=1}^{K-1} f_{\mathbf{e}^{(j)}} \cdot \mathbf{x}^{\mathbf{e}^{(j)}}$ is an annihilator of $\mathcal{G}$, if $M[\mathbf{e}^{(K)}] = \sum_{j=1}^{K-1} f_{\mathbf{e}^{(j)}} \cdot M[\mathbf{e}^{(j)}]$.

We now construct a matrix M', with $(K - 1)$ rows and K columns, such that M' has rank exactly $(K - 1)$. In fact, the columns of M' will share the same dependencies as the columns of M.

Claim 3.3. *Define for $\alpha \in \mathbb{N}$, the matrix $E_\alpha \in \mathbb{C}^{(K-1) \times R}$ such that $E_\alpha[i, j] = \alpha^{ij}$ for each $0 \le i < K - 1$ and $0 \le j < R$. For some $\alpha \le D^{2n}$, the matrix product $M' := E_\alpha \cdot M$ has rank exactly $K - 1$. Further, M' has the same dependencies in its columns as M.*

21

Proof. By [Lemma 2.8](), we immediately have that there exists $\alpha \leq R \cdot (K-1) < D^{2n}$ such that $\text{rank}(E_\alpha \cdot M) = K - 1$. Let $M' = E_\alpha \cdot M$. To show that M' has the same dependencies in its columns as M, let us first assume that $M\mathbf{u} = 0$ for some $\mathbf{u} \in \mathbb{C}^K$. Clearly this implies that $M'\mathbf{u} = E_\alpha \cdot M\mathbf{u} = 0$.

Conversely, let $\mathbf{v} \in \mathbb{C}^K$ be such that $M'\mathbf{v} = 0$ but $M\mathbf{v} \neq 0$. Then, clearly, $\mathbf{v} \neq \beta\mathbf{u}$ for any $\beta \in \mathbb{C}$. This shows that $\dim(\ker(M')) \geq 2$, contradicting the rank-nullity theorem since $\text{rank}(M') = K - 1$. Therefore no such $\mathbf{v}$ can exist, proving that for any $\mathbf{v} \in \mathbb{C}^K$, $M'\mathbf{v} = 0 \implies M\mathbf{v} = 0$. $\square$

For M' defined as in the claim above, we clearly have that $M'[\mathbf{e}^{(K)}] = \sum_{j=1}^{K-1} f_{\mathbf{e}^{(j)}} \cdot M'[\mathbf{e}^{(j)}]$. Also, note that the rows of M' are labelled by $\{0, \ldots, K-2\}$ and the columns are labelled by $\left\{ \mathbf{e}^{(j)} : j \in [K] \right\}$. We now define $\widetilde{M}$, a $K \times K$ matrix with rows labelled by $\{0, \ldots, K-1\}$ and columns labelled by $\left\{ \mathbf{e}^{(j)} : j \in [K] \right\}$, as follows.

$$\widetilde{M}[i, \mathbf{e}^{(j)}] = \begin{cases} M'[i-1, \mathbf{e}^{(j)}] & \text{if } i \leq K-2 \\ (-1)^i \cdot \mathbf{x}^{\mathbf{e}^{(j)}} & \text{if } i = K-1 \end{cases}$$

First, we show that the determinant of $\widetilde{M}$ is an annihilator of $\mathcal{G}$. In particular, we show that $\det(\widetilde{M})$ is a scalar multiple of $A(\mathbf{x})$. We use $M'[*, \mathbf{e}^{(j)}]$ to denote the $\mathbf{e}^{(j)}$th column of the matrix M', and $M' \setminus M'[*, \mathbf{e}^{(j)}]$ to denote the submatrix of M' obtained by deleting the $\mathbf{e}^{(j)}$th column.

Claim 3.4. $\det(\widetilde{M}) = \det(M' \setminus M'[*, \mathbf{e}^{(K)}]) \cdot A(\mathbf{x})$.

Proof. Note that, by expanding with respect to the last row,

$$\det(\widetilde{M}) = \sum_{j=1}^{K} (-1)^{j-1} \cdot \det(M' \setminus M'[*, \mathbf{e}^{(j)}]) \cdot (-1)^{K-1} \cdot \mathbf{x}^{\mathbf{e}^{(j)}}.$$

This can be re-written as

$$\frac{\det(\widetilde{M})}{\det(M' \setminus M'[*, \mathbf{e}^{(K)}])} = \mathbf{x}^{\mathbf{e}^{(K)}} + \sum_{j=1}^{K-1} (-1)^{K-j} \cdot \frac{\det(M' \setminus M'[*, \mathbf{e}^{(j)}])}{\det(M' \setminus M'[*, \mathbf{e}^{(K)}])} \cdot \mathbf{x}^{\mathbf{e}^{(j)}}.$$

Since $M'[\mathbf{e}^{(K)}] = \sum_{j=1}^{K-1} f_{\mathbf{e}^{(j)}} \cdot M'[\mathbf{e}^{(j)}]$, it is now easy to check the following using Cramer's Rule.

$$\frac{\det(\widetilde{M})}{\det(M' \setminus M'[*, \mathbf{e}^{(K)}])} = \mathbf{x}^{\mathbf{e}^{(K)}} - \sum_{j=1}^{K-1} f_{\mathbf{e}^{(j)}} \cdot \mathbf{x}^{\mathbf{e}^{(j)}} = A(\mathbf{x}). \qquad \square$$

Since $\det(M' \setminus M'[*, \mathbf{e}^{(K)}])$ is a fixed, nonzero scalar, $\det(\widetilde{M})$ is an annihilator of $\mathcal{G}$. Further, since every monomial in $\left\{ \mathbf{x}^{\mathbf{e}^{(j)}} : j \in [K] \right\}$ has individual degree at most $D - 1$, $\det(\widetilde{M})$ also has individual degree at most $D - 1$.

We now show that $\widetilde{M}$ can be described as claimed. To show that, we note that M' has a very special structure. For any $\ell \in \{0, \ldots, R-1\}$, let $\mathbf{v}_\ell \in [ndD-1]^m$ be the unique vector such that

$$\ell = \sum_{b=1}^m \mathbf{v}_\ell(b) \cdot (ndD)^{b-1}.$$

Then, for $i \in \{0, \ldots, K-2\}$ and $j \in [K]$,

$$M'[i, \mathbf{e}^{(j)}] = \sum_{\ell=0}^{R-1} \alpha^{i\ell} \cdot \mathrm{coeff}_{\mathbf{z}^{\mathbf{v}_\ell}}(\mathcal{G}^{\mathbf{e}^{(j)}}) = \sum_{\ell=0}^{R-1} \mathrm{coeff}_{\mathbf{z}^{\mathbf{v}_\ell}}(\mathcal{G}^{(\mathbf{e}^j)}) \cdot \alpha^{i \cdot \left(\sum_{b=1}^m \mathbf{v}_\ell(b) \cdot (ndD)^{b-1}\right)}$$

$$= \sum_{\ell=0}^{R-1} \left(\mathrm{coeff}_{\mathbf{z}^{\mathbf{v}_\ell}}(\mathcal{G}^{\mathbf{e}^{(j)}}) \prod_{b=1}^m \alpha^{(i \cdot (ndD)^{b-1}) \mathbf{v}_\ell(b)} \right) = \sum_{\ell=0}^{R-1} \left(\mathrm{coeff}_{\mathbf{z}^{\mathbf{v}_\ell}}(\mathcal{G}^{\mathbf{e}^{(j)}}) \cdot \mathbf{v}_{\alpha,i}^{\mathbf{v}_\ell} \right)$$

where $\mathbf{v}_{\alpha,i} = (\alpha^i, \alpha^{i \cdot ndD}, \ldots, \alpha^{i \cdot (ndD)^{m-1}})$. That is,

$$M'[i, \mathbf{e}^{(j)}] = (\mathcal{G}(\mathbf{v}_{\alpha,i}))^{\mathbf{e}^{(j)}}.$$

We can therefore re-write $\widetilde{M}$ as claimed. $\qquad\square$

3.2 Annihilator as determinant of an explicit matrix

We now show that the matrix described in the last section is *explicit* in the sense of Definition 2.25.

Lemma 3.5 (Annihilator as determinant of an explicit matrix). *Let $\mathcal{G}$ be a polynomial map as given in Theorem 3.1. Let $C_{\mathcal{G}}(\mathbf{z}, \mathbf{y})$ be a circuit that encodes $\mathcal{G}$. Further, for $D = \left\lceil (nd)^{m/(n-m)} \right\rceil + 1$, $\alpha < D^{2n}$ and $K \leq D^n$, let $\widetilde{M}$ be defined as follows.*
   For $i \in \{0, \ldots, K-1\}$ and $j \in [K]$,

$$\widetilde{M}[i, \mathbf{e}^{(j)}] = \begin{cases} (\mathcal{G}(\mathbf{v}_{\alpha,i}))^{\mathbf{e}^{(j)}} & \text{if } 0 \leq i \leq K-2 \\ (-1)^{K-1} \cdot \mathbf{x}^{\mathbf{e}^{(j)}} & \text{if } i = K-1. \end{cases}$$

where $\mathbf{v}_{\alpha,i} = (\alpha^i, \alpha^{i \cdot ndD}, \ldots, \alpha^{i \cdot (ndD)^{m-1}})$.
   Then, given access to a $C_{\mathcal{G}}$ gate, there is a purely algebraic circuit[12] *$\widetilde{C}_{\mathcal{G}}$ that encodes $\widetilde{M}$. Further, $\widetilde{C}_{\mathcal{G}}$ uses only a single $C_{\mathcal{G}}$ gate and has overall size $O((nd)^3 + n \cdot \mathrm{size}(C_{\mathcal{G}}))$.*

Proof. Let $\mathbf{j}$ be a bit-vector of length $n \cdot \delta$, for $\delta := \lceil \log D \rceil$, which we interpret as the tuple $(\mathbf{j}^{(1)}, \ldots, \mathbf{j}^{(n)})$ where $\mathbf{j}^{(\ell)}$ is the bit-vector encoding the ℓ-th co-ordinate of $\mathbf{e}^{(j)}$. Further, let the bit-vector $\mathbf{i}$ encode the integer i. We want to construct a circuit $\widetilde{C}_{\mathcal{G}}$ such that $\widetilde{C}_{\mathcal{G}}(\mathbf{x}, \mathbf{i}, \mathbf{j}) = \widetilde{M}[i, \mathbf{e}^{(j)}]$.

---

[12] Algebraic circuit without projection gates.

Firstly, it is easy to see that there is a constant-free multi-output circuit C', of size $O(\log^2 K)$, such that $C'(\mathbf{j}) = (\mathbf{j}^{(1)}, \dots, \mathbf{j}^{(n)})$ as described above. We will use $C'_{k,b}(\mathbf{j})$ to denote the output gate of $C'(\mathbf{j})$ which outputs the b-th bit of $\mathbf{j}^{(k)}$.

Let $\Delta = ndD$ and $\mathbf{v}_\alpha := (\alpha, \alpha^\Delta, \dots, \alpha^{\Delta^{m-1}})$. We start by computing $L = \lceil \log K \rceil$ many distinct powers of the vector $\mathbf{v}_\alpha$, namely, $\mathbf{v}_\alpha, \mathbf{v}_\alpha^2, \mathbf{v}_\alpha^4, \dots, \mathbf{v}_\alpha^{2^{L-1}}$. Here $\mathbf{v}_\alpha^r$ is used to denote the vector $(\alpha^r, \alpha^{r \cdot \Delta}, \dots, \alpha^{r \cdot \Delta^{m-1}})$. Since α can be computed by a constant-free circuit of size $O(n \log(nd))$, each of these L vectors can be computed by a constant-free circuit of size $O(L \cdot n \log(nd)) = O(\log K \cdot n \log(nd))$. Let these multi-output circuits be $C_0(\alpha), \dots, C_{L-1}(\alpha)$ with $C_{\ell,k}(\alpha)$ denoting the k-th output gate in $C_\ell(\alpha)$ for $k \in \{0, \dots, m-1\}$. That is, $C_{\ell,k}(\alpha) = \alpha^{r \cdot \Delta^k}$ for $k \in \{0, \dots, m-1\}$.

We now describe some intermediate circuits, and then finally $\widetilde{C}_\mathcal{G}$. For $\mathbf{i} = (i_0, \dots, i_{L-1})$,

$$\text{pow}(\mathbf{i}) := \left(\prod_{\ell=0}^{L-1} (i_\ell \cdot C_{\ell,0}(\alpha) + (1 - i_\ell) \cdot 1), \dots, \prod_{\ell=0}^{L-1} (i_\ell \cdot C_{\ell,m-1}(\alpha) + (1 - i_\ell) \cdot 1) \right)$$

$$\text{ROW}(\mathbf{i}) := \text{LT}(\mathbf{i}, \mathbf{k}) \cdot C_\mathcal{G}(\text{pow}(\mathbf{i})) + \text{EQ}(\mathbf{i}, \mathbf{k}) \cdot \mathbf{x}$$

$$\widetilde{C}_\mathcal{G}(\mathbf{i}, \mathbf{j}) := \prod_{a \in [n]} \left(\prod_{b=0}^{\delta-1} \left(C'_{a,b}(\mathbf{j}) \cdot (\text{ROW}(\mathbf{i})_a)^{2^b} + (1 - C'_{a,b}(\mathbf{j})) \cdot 1 \right) \right)$$

Here $\text{pow}(\mathbf{i})$ computes the vector $\mathbf{v}_{\alpha,\mathbf{i}}$, and the polynomials $\text{LT}(\cdot, \cdot)$ and $\text{EQ}(\cdot, \cdot)$ are as defined in Observation 2.9. Further, $\mathbf{k}$ is the binary encoding of the integer $K - 1$ and $\text{ROW}(\mathbf{i})_a$ denotes the a-th output gate of $\text{ROW}(\mathbf{i})$. That is, $\text{ROW}(\mathbf{i})_a = \text{LT}(\mathbf{i}, \mathbf{k}) \cdot g_a(\text{pow}(\mathbf{i})) + \text{EQ}(\mathbf{i}, \mathbf{k}) \cdot x_a$ if $\mathcal{G} = (g_1, \dots, g_n)$.

Note that $\widetilde{C}_\mathcal{G}$ uses the sub-circuit $C_\mathcal{G}$ exactly once, in ROW, as claimed. Also, all the three expressions described above are constant-free, algebraic expressions. Finally, note that the size of $\widetilde{C}_\mathcal{G}$ is $O(L \cdot n \log(nd) + L + L^2 + nd + n \cdot \text{size}(C_\mathcal{G}))$, which is $O(n^3 \cdot d^2 + n \cdot \text{size}(C_\mathcal{G}))$. $\qquad \square$

3.3 Completing the proof

We now have all the components necessary to complete the proof of Theorem 3.1.

Theorem 3.1 (Annihilators of explicit maps). *Let m be large enough, and let $\mathcal{G} : \mathbb{F}^m \to \mathbb{F}^n$ be a polynomial map given by $(g_1(\mathbf{z}), \dots, g_n(\mathbf{z}))$ of degree d, with $n \geq 2m$.*

There exists a constant c such that, if there is a circuit with projection gates $C_\mathcal{G}(\mathbf{z}, \mathbf{y})$ of size s that encodes $\mathcal{G}$ as per Definition 1.1, then there is a circuit with projection gates $C'(\mathbf{x})$ of size $(m \cdot d \cdot s)^c$ computing a nonzero polynomial $A(\mathbf{x})$ of individual degree at most $3 \cdot m \cdot d$ that annihilates $\mathcal{G}$ (that is, $A \circ \mathcal{G}_m = A(g_1(\mathbf{z}), \dots, g_n(\mathbf{z})) \equiv 0$).

Proof. Let $\mathcal{G}' = (g_1(\mathbf{z}), \dots, g_{2m}(\mathbf{z}))$, the first $2m$ co-ordinates. Clearly, $C_\mathcal{G}$ encodes $\mathcal{G}'$ as well.

Lemma 3.2 tells us that there is a $K \times K$ matrix, $\widetilde{M}$, such that $0 \not\equiv A'(\mathbf{x}) = \det(\widetilde{M})$ is an annihilator for the map $\mathcal{G}'$ (that is, $A' \circ \mathcal{G}' \equiv 0$). Here $K = (2md) + 1^{2m}$ and $A'(\mathbf{x})$ has individual

24

degree at most $\left\lceil (2md)^{m/(2m-m)} \right\rceil + 1 = 2md + 1$.

Using the fact that the entries of $\widetilde{M}$ are either evaluations of $\mathcal{G}'$ or monomials, Lemma 3.5 provides a circuit $\widetilde{C}$ that encodes the matrix $\widetilde{M}$ where $\widetilde{C}$ has size $O((2md)^3 + (2m) \cdot \text{size}(C_{\mathcal{G}}))$. Further $\widetilde{C}$ is a purely algebraic circuit[13] and also constant-free, assuming we have access to a gate computing $C_{\mathcal{G}_m}$. Finally, we use Proposition 2.27 to obtain a circuit with projections C' that computes $A'(\mathbf{x})$. The circuit C' has size $O((2md)^3 + (4m) \cdot \text{size}(C_{\mathcal{G}}) + \log^2 K) = O((2md)^3 + (4m) \cdot \text{size}(C_{\mathcal{G}})) = (m \cdot d \cdot s)^4$ for $s = \text{size}(C_{\mathcal{G}})$.

Finally, we note that for $A(x_1, \ldots, x_n) := A'(x_1, \ldots, x_{2m})$, $0 \not\equiv A(\mathbf{x})$ and $A \circ \mathcal{G}_m \equiv 0$. This completes the proof, since C' also computes A. □

Clearly Theorem 1.2 is a special case of Theorem 3.1 and therefore follows as a corollary. We restate Theorem 1.2 here for convenience.

Theorem 1.2 (Upper bound for annihilators of explicit maps). *Let $\mathcal{G} : \mathbb{F}^m \to \mathbb{F}^n$ be a polynomial map given by $(g_1(\mathbf{z}), \ldots, g_n(\mathbf{z}))$ of individual degree d, with $n \geq 2m$.*

There exists a constant c such that, if there is a circuit $C_{\mathcal{G}}(\mathbf{z}, \mathbf{y})$ of size s that encodes $\mathcal{G}$ as per Definition 1.1, then there is a circuit with projection gates $C'(\mathbf{x})$ of size $(m \cdot d \cdot s)^c$ computing a nonzero polynomial $A(\mathbf{x})$ of individual degree at most $3 \cdot m \cdot d$ that annihilates $\mathcal{G}$; i.e. $A(g_1(\mathbf{z}), \ldots, g_n(\mathbf{z})) \equiv 0$.

4 Important Consequences

We now discuss some consequences of Theorem 3.1 on two concepts: obtaining lower bounds from hitting set generators and algebraic natural proofs.

4.1 Lower Bounds from Explicit Hitting Set Generators

As stated earlier, a special case of Theorem 3.1 is the following upper bound on the complexity of annihilators of an explicit polynomial map.

Theorem 1.2 (Upper bound for annihilators of explicit maps). *Let $\mathcal{G} : \mathbb{F}^m \to \mathbb{F}^n$ be a polynomial map given by $(g_1(\mathbf{z}), \ldots, g_n(\mathbf{z}))$ of individual degree d, with $n \geq 2m$.*

There exists a constant c such that, if there is a circuit $C_{\mathcal{G}}(\mathbf{z}, \mathbf{y})$ of size s that encodes $\mathcal{G}$ as per Definition 1.1, then there is a circuit with projection gates $C'(\mathbf{x})$ of size $(m \cdot d \cdot s)^c$ computing a nonzero polynomial $A(\mathbf{x})$ of individual degree at most $3 \cdot m \cdot d$ that annihilates $\mathcal{G}$; i.e. $A(g_1(\mathbf{z}), \ldots, g_n(\mathbf{z})) \equiv 0$.

This has the below interesting consequence in the context of what are called "hardness randomness connections", via the previously mentioned result due to Koiran and Perifel [KP09] (see Proposition 2.24).

---

[13] Algebraic circuit without projection gates.

Theorem 1.9 (Cryptographic HSGs separate VP from VPSPACE). *Let $\{\mathcal{H}_m\}$ be a VP-explicit family of m-variate polynomial maps with $2m$ outputs of degree m^8, and let $d(n) = n^{10}$.*

If the family $\{\mathcal{H}_m\}$ is a hitting set generator for VP_d, then $\mathsf{VP} \neq \mathsf{VPSPACE}_b$. As a consequence, either $\mathsf{P}/\mathrm{poly} \neq \mathsf{PSPACE}/\mathrm{poly}$ or $\mathsf{VP} \neq \mathsf{VNP}$.

Informally this says that a hitting set generator with a 'super-polynomial stretch' would imply $\mathsf{VP} \neq \mathsf{VPSPACE}_b$. This, in tur, would either separate (non-uniform) P and PSPACE or separate VP from VNP. We now show how this follows from Theorem 3.1.

Proof of Theorem 1.9. We shall prove the statement via contradiction. Let us assume $\mathsf{VPSPACE}_b = \mathsf{VP}$, which means that any family $\{A_n\}$ in $\mathsf{VPSPACE}_b$ of degree $d'(n) \in \mathrm{poly}(n)$ belongs to $\mathsf{VP}_{d'}$. For the specific parameters of the generator family $\{\mathcal{H}_m\}$, Theorem 3.1 implies a family $\{A_n\} \in \mathsf{VPSPACE}_b$ of annihilators of individual degree at most $(2m \cdot m^8) = n \cdot (n/2)^8 \leq n^9$, and thus total degree at most $n^{10} = d(n)$. Now from our assumption, $\{A_n\} \in \mathsf{VP}_d$, which contradicts the HSG property of $\{\mathcal{H}_m\}$. So it must be the case that $\mathsf{VP} \neq \mathsf{VPSPACE}$ as claimed. $\qquad\square$

One can weaken the hypothesis of Theorem 1.9 to a family of VNP or even $\mathsf{VPSPACE}_b$-explicit HSGs because of Theorem 3.1, as follows.

Theorem 4.1. *Let $\{\mathcal{H}_m\}$ be a $\mathsf{VPSPACE}_b$-explicit family of m-variate polynomial maps with $n(m) > 2m$ outputs of degree $d_0(m) \in \mathrm{poly}(m)$, and let $d(n) \in \mathrm{poly}(n)$ be such that $d(n(m)) > 2m \cdot d_0(m)$, for all large m.*

If the family $\{\mathcal{H}_m\}$ is a hitting set generator for VP_d, then $\mathsf{VP} \neq \mathsf{VPSPACE}_b$. As a consequence, either $\mathsf{P}/\mathrm{poly} \neq \mathsf{PSPACE}/\mathrm{poly}$ or $\mathsf{VP} \neq \mathsf{VNP}$. $\qquad\square$

Note that the PSPACE-construction of hitting sets for algebraic circuits by Mulmuley [Mul12] (see also Forbes and Shpilka [FS18]), does not satisfy the hypothesis of Theorem 4.1. This is essentially the same reason as why the arguments of Heintz and Schnorr [HS80] do not give a "cryptographic" generator as required in Theorem 1.9. These constructions ([Mul12, FS18]) yield a (possibly) *different* family of generators constructible in space $\mathrm{poly}(n, d, s)$ for each size s. On the other hand, Theorem 4.1 requires a single family that works for all sizes $s(n) \in \mathrm{poly}(n)$.

4.2 Algebraic Natural Proofs

We also get an upper bound on equations for the *evaluation vectors* of VP. Before moving on to the formal statement, we state some definitions and observations.

Definition 4.2 (Interpolating set). *For a set of polynomials P, a set of evaluation points I is called an interpolating set for P, if there is a* linear transformation M *such that for any polynomial $f \in P$, the coefficients of f can be obtained from the evaluations of f on the set I by an application of M.* $\qquad\Diamond$

Proposition 4.3 (Restatement of [BP20, Theorem 10]). *Let $S := \{0, 1, 2, \ldots, d\}$, then the set of evaluation points $I_{n,d} := \{(a_1, a_2, \ldots, a_n) \in S^n \mid a_1 + a_2 + \cdots + a_n \leq d\}$ forms an interpolating set for the set of*

all n-variate polynomials of total degree at most d. □

Definition 4.4 (Evaluation vector). *For any n-variate polynomial $f(\mathbf{x})$ of total degree-d, its evaluation vector is simply the vector[14] formed by evaluations of f on the set $I_{n,d}$ from Proposition 4.3.* ◇

Proposition 4.5 (Equations for evaluation vectors and natural proofs). *Let $\mathcal{C}$ and $\mathcal{D}$ be either VP or VNP. For any $d(n)$, there are $\mathcal{D}$-natural proofs for $\mathcal{C}_d$ if and only if there is a family $\{A_N\} \in \mathcal{D}$ such that for each $\{f_n\} \in \mathcal{C}_d$, A_N vanishes on the evaluation vector of f_n for all large enough n.*

Sketch. These statements essentially follow from the fact that for any *n*-variate, degree-*d* polynomial, we can move between its coefficient vector and evaluation vector using linear transformations. Clearly, both these transformations have (constant-free) algebraic circuits of size poly(N). Since both VP and VNP are rich enough to implement the above circuits, we get all the required statements. □

We now state the upper bound on equations for *evaluation vectors* of VP..

Theorem 4.6. *There exists a constant c, such that for all large enough $n, d, s \in \mathbb{N}$, a multilinear equation for the* evaluation vectors *of the set of n-variate, degree-d polynomials computable by circuits of size s, is computable by constant-free circuits with projection gates of size at most $(nds)^c$.*

Proof. For the given parameters n, d, s, let $N = \binom{n+d}{n}$ and consider the universal circuit $\mathcal{U}(\mathbf{x}_{[n]}, \mathbf{y}_{[r]})$ as given in Lemma 2.11. Let $I \subseteq \mathbb{C}^n$ be an interpolating set of size N given by Proposition 4.3. Define a polynomial map $\mathcal{U}_{n,d,s} : \mathbb{C}^r \to \mathbb{C}^N$ such that each co-ordinate of $\mathcal{U}_{n,d,s}$ is exactly $\mathcal{U}(\mathbf{x} = \mathbf{a}, \mathbf{y})$ for a unique point $\mathbf{a} \in I$. We also know from Lemma 2.11 that $\mathcal{U}_{n,d,s}$ can be encoded by a *constant-free* circuit of size poly(n, d, s), and has individual degree at most poly(n, d, s).

Since any *n*-variate, degree-*d* polynomial that is computable by circuits of size *s* can be obtained by fixing the $\mathbf{y}$ variables to some scalars in $\mathcal{U}(\mathbf{x}, \mathbf{y})$, we have that any annihilator of $\mathcal{U}_{n,d,s}$ vanishes on the evaluation vector of *every* such polynomial.

Now, using Theorem 3.1, we can construct a *multilinear*, poly(n, d, s)-variate annihilator A for $\mathcal{U}_{n,d,s}$, that is computable by constant-free circuits with projections, of size at most $(nds)^c$ for some constant *c*. That the annihilator A can be multilinear is guaranteed by the bound on the individual degree from Lemma 3.2. □

Corollary 4.7. *Fix a $d(n) = \text{poly}(n)$, and let $N(n) := \binom{n+d(n)}{n}$. For $t(n) = n^{\log^* n}$, there is a family $\{P_N\}$ of $t(n)$-variate annihilators of* evaluation vectors *of VP_d that is in uniform VPSPACE^0. Further, the coefficient functions of $\{P_N\}$ are computable in space $t(n)$.*

Proof. Let *c* be a constant as chosen in the proof of Theorem 4.6, $\beta > 2$ be a constant to be fixed later and let $s = s(n) = n^{\log^* n/(\beta \cdot c)}$. Using Theorem 4.6, define a polynomial family $\{P_N\}$ such that, for each *n*, $P_N(Z_1, \ldots, Z_N)$ is an annihilator for the set of *n*-variate, degree-*d*, size-*s* polynomials

---

[14]The order can be picked arbitrarily.

and depends only on the first $(nds)^{2c} \leq n^{\log^* n} = t(n)$ variables. As each P_N is computable using constant-free algebraic circuits with projections, of size $s^{3c} \leq t(n)$, the family $\{P_N\}$ is in $\mathsf{VPSPACE}^0$. Further since the family of universal circuits $\{\mathcal{U}_{n,d,s}\}$ is $\mathsf{DTIME}(\mathrm{poly}(n,d,s))$-uniform, using the characterization of $\mathsf{VPSPACE}^0$ due to Koiran and Perifel (Definition 2.17), the coefficient functions of $\{P_N\}$ can be computed by a Turing machine that uses space $s(n)^{3c \cdot \beta'}$ for some constant β'. We now fix $\beta = 3\beta'$ so that $s(n)^{3c \cdot \beta'} \leq t(n)$.

Let $\{f_n\} \in \mathsf{VP}_d$ be arbitrary. We know that for all large enough n, $\mathrm{size}(f_n) = n^a$ for some constant a. Thus there is a finite $n_0 \in \mathbb{N}$ such that $\mathrm{size}(f_n) \leq s(n)$ for all $n > n_0$. Therefore, for all large enough n, the evaluation vectors of f_n are zeroes of the polynomial $P_N(Z_1, \ldots, Z_N)$ and so $\{P_N\}$ is a family of equations in uniform $\mathsf{VPSPACE}^0$ as claimed. $\qquad\qquad\square$

Thus, we have proven the following statement about equations for evaluation vectors of VP.

Theorem 1.4 (Equations for VP). *For an arbitrary $d(n) \in \mathrm{poly}(n)$, and let $N(n) = \binom{n+d(n)}{n}$. For $t(n) := n^{\log^* n}$, there is a family of $N(n)$-variate, multilinear polynomials $\{P_N\}$ that depends only on the first $t(n)$ variables, satisfying the following.*

- *The family $\{P_N\}$ is a family of equations for the evaluation vectors of $\mathsf{VP}_{d(n)}$.*
- *The coefficient functions of $\{P_N\}$ are computable in (uniform) space $t(n) = n^{\log^* n}$.*

Acknowledgements

References

[AD08]   Scott Aaronson and Andrew Drucker. Arithmetic natural proofs theory is sought. Shtetl Optimized: Scott Aaronson's Blog, 2008. [Cited on pages 3, 4, and 10.]

[AF22]   Robert Andrews and Michael A. Forbes. Ideals, determinants, and straightening: proving and using lower bounds for polynomial ideals. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, 2022*, pages 389–402. ACM, 2022. [Cited on page 5.]

[And22]   Robert Andrews. On Matrix Multiplication and Polynomial Identity Testing. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 356–365. IEEE, 2022. [Cited on pages 4 and 5.]

[BP20]   Markus Bläser and Anurag Pandey. Polynomial Identity Testing for Low Degree Polynomials with Optimal Randomness. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. [Cited on page 26.]

[BS83]   Walter Baur and Volker Strassen. The Complexity of Partial Derivatives. *Theoretical Computer Science*, 22:317–330, 1983. [Cited on page 1.]

[BT88]   Michael Ben-Or and Prasoon Tiwari. A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 301–309. ACM, 1988. [Cited on page 4.]

[Bür00]   Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*, volume 7 of *Algorithms and Computation in Mathematics*. Springer, 2000. [Cited on pages 1 and 16.]

[Cho11]   Timothy Y. Chow. Almost-natural proofs. *Journal of Computer and System Sciences*, 77(4):728–737, 2011. Preliminary version in the *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*. [Cited on page 3.]

[CJSW21]   Lijie Chen, Ce Jin, Rahul Santhanam, and R. Ryan Williams. Constructive Separations and Their Consequences. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 646–657. IEEE, 2021. arXiv:2203.14379. [Cited on page 3.]

[CKR+20]   Prerona Chatterjee, Mrinal Kumar, C. Ramya, Ramprasad Saptharishi, and Anamay Tengse. On the Existence of Algebraically Natural Proofs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 870–880. IEEE, 2020. Pre-print available at arXiv:2004.14147. [Cited on pages 3 and 14.]

[CKW11]   Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial Derivatives in Arithmetic Complexity. *Foundations and Trends in Theoretical Computer Science*, 2011. [Cited on page 1.]

[DL78]   Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters*, 7(4):193–195, 1978. [Cited on page 2.]

[For14]   Michael Forbes. *Polynomial Identity Testing of Read-Once Oblivious Algebraic Branching Programs*. PhD thesis, Massachusetts Institute of Technology, 2014. [Cited on page 2.]

[FS12]   Michael A. Forbes and Amir Shpilka. On identity testing of tensors, low-rank recovery and compressed sensing. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 163–172. ACM, 2012. Pre-print available at `eccc:TR11-147`. [Cited on pages 9 and 13.]

[FS18]   Michael A. Forbes and Amir Shpilka. A PSPACE construction of a hitting set for the closure of small algebraic circuits. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 1180–1192. ACM, 2018. `arXiv:1712.09967`. [Cited on page 26.]

[FSV18]   Michael A. Forbes, Amir Shpilka, and Ben Lee Volk. Succinct Hitting Sets and Barriers to Proving Lower Bounds for Algebraic Circuits. *Theory of Computing*, 14(1):1–45, 2018. Preliminary version in the *49th Annual ACM Symposium on Theory of Computing (STOC 2017)*. `arXiv:1701.05328`. [Cited on pages 3, 4, and 10.]

[GKSS17]   Joshua A. Grochow, Mrinal Kumar, Michael E. Saks, and Shubhangi Saraf. Towards an algebraic natural proofs barrier via polynomial identity testing. *CoRR*, abs/1701.01717, 2017. Pre-print available at `arXiv:1701.01717`. [Cited on page 3.]

[GKSS19]   Zeyu Guo, Mrinal Kumar, Ramprasad Saptharishi, and Noam Solomon. Derandomization from Algebraic Hardness: Treading the Borders. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 147–157. IEEE Computer Society, 2019. [Cited on page 4.]

[Gro15]   Joshua A. Grochow. Unifying Known Lower Bounds via Geometric Complexity Theory. *Computational Complexity*, 24(2):393–475, 2015. [Cited on page 3.]

[Hru16]   Pavel Hrubes. On Hardness of Multilinearization and VNP-Completeness in Characteristic 2. *ACM Transactions of Computation Theory*, 9(1):1:1–1:14, 2016. [Cited on page 3.]

[HS80]   Joos Heintz and Claus-Peter Schnorr. Testing Polynomials which Are Easy to Compute (Extended Abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 262–272, 1980. [Cited on pages 2, 10, and 26.]

[Kay09]   Neeraj Kayal. The Complexity of the Annihilating Polynomial. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 184–193, 2009. [Cited on pages 4 and 7.]

[KI04]   Valentine Kabanets and Russell Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, 13(1-2):1–46, 2004. Preliminary version in the *35th Annual ACM Symposium on Theory of Computing (STOC 2003)*. [Cited on pages 2 and 4.]

[KP09]   Pascal Koiran and Sylvain Perifel. VPSPACE and a Transfer Theorem over the Reals. *Computational Complexity*, 18(4):551–575, 2009. [Cited on pages 15, 16, 17, and 25.]

[KRST22]   Mrinal Kumar, C. Ramya, Ramprasad Saptharishi, and Anamay Tengse. If VNP Is Hard, Then so Are Equations for It. In *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 44:1–44:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. arXiv:2012.07056. [Cited on pages 3 and 10.]

[KS01]   Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001)*, pages 216–223, 2001. [Cited on page 4.]

[KS19]   Mrinal Kumar and Ramprasad Saptharishi. Hardness-Randomness Tradeoffs for Algebraic Computation. *Bulletin of EATCS*, 1(129), 2019. [Cited on page 2.]

[LST21]   Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial Lower Bounds Against Low-Depth Algebraic Circuits. Technical Report 081, 2021. [Cited on pages 1 and 5.]

[Mal11]   Guillaume Malod. Succinct Algebraic Branching Programs Characterizing Non-uniform Complexity Classes. In *Fundamentals of Computation Theory - 18th International Symposium, FCT 2011, Oslo, Norway, August 22-25, 2011. Proceedings*, pages 205–216, 2011. [Cited on pages 9, 15, 16, 17, 33, and 34.]

[MR13]   Meena Mahajan and B. V. Raghavendra Rao. Small Space Analogues of Valiant's Classes and the Limitations of Skew Formulas. *Computational Complexity*, 22(1):1–38, 2013. [Cited on page 16.]

[Mul12]   Ketan Mulmuley. Geometric Complexity Theory V: Equivalence between Blackbox Derandomization of Polynomial Identity Testing and Derandomization of Noether's

Normalization Lemma. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 629–638, 2012. [Cited on page 26.]

[MV97]   Meena Mahajan and V. Vinay. A Combinatorial Algorithm for the Determinant. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1997)*, pages 730–738, 1997. Available on `citeseer:10.1.1.31.1673`. [Cited on pages 9 and 18.]

[NW94]   Noam Nisan and Avi Wigderson. Hardness vs Randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. Available on `citeseer:10.1.1.83.8416`. [Cited on page 4.]

[Ore22]   Øystein Ore. Über höhere Kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922. [Cited on page 2.]

[Poi08]   Bruno Poizat. A la recherche de la definition de la complexite d'espace pour le calcul des polynomes a la maniere de Valiant. *J. Symb. Log.*, 73(4):1179–1201, 2008. [Cited on pages 15, 33, and 34.]

[Raz09]   Ran Raz. Multi-Linear Formulas for Permanent and Determinant are of Super-Polynomial Size. *Journal of the ACM*, 56(2), 2009. Preliminary version in the *36th Annual ACM Symposium on Theory of Computing (STOC 2004)*. Pre-print available at `eccc:TR03-067`. [Cited on page 3.]

[Raz10]   Ran Raz. Elusive Functions and Lower Bounds for Arithmetic Circuits. *Theory of Computing*, 6(1):135–177, 2010. [Cited on page 14.]

[RR97]   Alexander A. Razborov and Steven Rudich. Natural Proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997. Preliminary version in the *26th Annual ACM Symposium on Theory of Computing (STOC 1994)*. [Cited on pages 2, 3, and 4.]

[RY09]   Ran Raz and Amir Yehudayoff. Lower Bounds and Separations for Constant Depth Multilinear Circuits. *Computational Complexity*, 18(2):171–207, 2009. Preliminary version in the *23rd Annual IEEE Conference on Computational Complexity (CCC 2008)*. Pre-print available at `eccc:TR08-006`. [Cited on page 3.]

[Sap15]   Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015. [Cited on pages 1 and 18.]

[Sch80]   Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, 1980. [Cited on page 2.]

[SV09] Amir Shpilka and Ilya Volkovich. Improved Polynomial Identity Testing for Read-Once Formulas. In *APPROX-RANDOM*, pages 700–713, 2009. [Cited on pages 4 and 5.]

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. [Cited on pages 1 and 14.]

[Val79] Leslie G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC 1979)*, pages 249–261, 1979. [Cited on page 1.]

[vMM22] Dieter van Melkebeek and Andrew Morgan. Polynomial Identity Testing via Evaluation of Rational Functions. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 119:1–119:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. [Cited on page 5.]

[Vol21] Ben Lee Volk. Guest Column: Algebraic Natural Proofs Ben Lee Volk. *SIGACT News*, 52(4):56–73, 2021. Draft on author's webpage. [Cited on page 3.]

[Wil13] Ryan Williams. Natural proofs versus derandomization. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 21–30. ACM, 2013. arXiv:1212.1891. [Cited on page 3.]

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. [Cited on page 2.]

A Equivalence of the definitions of VPSPACE

This section is devoted to proving Proposition 2.20, restated below.

Proposition 2.20 ([Poi08, Mal11])**.** $\mathsf{VPSPACE}^0 = \mathsf{VPPROJ}^0$.

Throughout this section, whenever a bit-string corresponds to an integer, the 0th bit refers to the sign-bit, the first bit to the most significant bit (MSB) and the last bit to the least significant bit (LSB).

Before moving to Proposition 2.20, we sketch the proofs of a few basic facts that will be needed.

Observation A.1 (Adding in small space)**.** *Let A, B be positive integers such that $A + B$ is at most 2^{2^s}. Assuming bit-access to A and B, any bit of the sum $A + B$ can be computed in space at most $O(s)$.*

Sketch. Let $i \in [s]$ be the index of the bit of $A + B$ that we wish to compute. This will be done in i iterations, starting from the LSB to the ith bit. We maintain a counter j (using s bits) to keep track of the current iteration. We also use 4 additional bits: a, b, r and c, for A, B, the result and the carry, respectively, all initialized to 0.

In iteration j, we get the jth bits of A and B in a and b, respectively. We then set $r \leftarrow a \oplus b \oplus c$, and $c \leftarrow (a \wedge b) \vee (b \wedge c) \vee (a \wedge c)$, in that order. At the end of iteration i, we output r as the ith bit of $A + B$.

The total space used is, clearly, at most $O(s)$ for all large s. $\quad\square$

Observation A.2 (Subtracting in small space). *Let A, B be integers of absolute value at most 2^{2^s} such that the absolute value of $A - B$ is at most 2^{2^s}. Assuming bit-access to A and B, any bit of the difference $A - B$ can be computed in space at most $O(s)$.*

Sketch. As an initial step, we query the sign-bits (0th bits) of A and B, to decide whether the absolute value of $A - B$ is the sum of the difference of their absolute values.

The rest of the algorithm is the same as that for Observation A.1, but with the correct expressions for computing r and c in each of the iterations. In order to compute the sign-bit of $A - B$, we need to check the sign-bits of A and B, and check if there is a carry from iteration 1. $\quad\square$

Observation A.3 (Summing a list in small space). *Let $A_1, A_2, \ldots, A_L$ be positive integers such that their sum $T := A_1 + A_2 + \cdots + A_L$ is at most 2^{2^s}. Assuming bit-access to each of the A_is, any bit of the sum T can be computed in space at most $O(\lceil \log L \rceil \cdot s)$.*

Sketch. We prove this using Observation A.1, and induction on the size of the list, L. Assuming bit-access to the sums $T_1 = A_1 + \cdots + A_{L/2}$, and $T_2 = A_{L/2+1} + \cdots + A_L$, we can compute the sum T in space $2s$. In order to simulate bit-access to T_1 and T_2, we recursively run the algorithm of the two smaller lists. The key observation is that we can reuse the same $2 \cdot (\lceil \log L \rceil - 1) \cdot s$ bits of space for both these computations. The total space used is thus, at most $O(\lceil \log L \rceil \cdot s)$. $\quad\square$

Observation A.4 (Multiplying in small space). *Let A, B be positive integers such that $A \times B$ is at most 2^{2^s}. Assuming bit-access to A and B, any bit of the product $A \times B$ can be computed in space at most $O(s^3)$.*

Sketch. Using the "school method" the product $A \times B$ can be calculated as the sum of $L = \lceil \log B \rceil$ many numbers, each of which is either 0, or a shift of A.

Therefore, we can use Observation A.3 by simulating bit-access to each of the shifts mentioned above. Here, the jth bit in the kth shift of A, is either 0 (if the kth bit of B is 0), or the $(j + k - 1)^{th}$ bit of A. So we just need an additional space of at most $2s$ bits for this simulation.

The total space used is therefore $O(s^2) + O(s) \leq O(s^3)$, for all large enough s. $\quad\square$

We are now ready to prove Proposition 2.20, which we restate once more.

Proposition 2.20 ([Poi08, Mal11]). $\mathsf{VPSPACE}^0 = \mathsf{VPPROJ}^0$.

The statement clearly follows from the following lemmas, namely Lemma A.5 and Lemma A.6.

Lemma A.5. *Let s be large enough, and suppose $f(\mathbf{x}) \in \mathbb{Z}[\mathbf{x}]$ is computable by a constant-free, algebraic circuit with projections gates, of size s.*

Then, the coefficient function[15] Φ of f, is computable by a Turing machine that uses space at most $O(s^4)$, and takes as advice a string of length at most $O(s^2)$.

Proof. Let C be the constant-free circuit with projection gates that computes f. The advice to the Turing machine is going to be an encoding of the graph of C.

We can assume that $C(\mathbf{x}) = C_1(\mathbf{x}) - C_2(\mathbf{x})$, with C_1, C_2 being monotone (that is, they only use the positive constant 1, and addition and multiplication gates). We shall therefore assume that C itself is monotone, and show that its coefficient function can be computed in small space, and then just use Observation A.2 to finish the proof.

For each gate $g \in C$, let Φ_g be its coefficient function.

- If g is a leaf, then its coefficient function is trivial.

- If $g = u + v$, then Φ_g can be decided in space $O(s)$ using Observation A.1, by using an additional space of at most $O((s-1)^4)$ bits, to simulate Φ_u and Φ_v whenever required.

- If $g = \mathrm{fix}_{z=0}\, u(z, \mathbf{x})$, then Φ_g is essentially Φ_u with the exponent for z fixed to zero.
  When $g = \mathrm{fix}_{z=1}\, u(z, \mathbf{x})$, for any monomial $\mathbf{x}^{\mathbf{e}}$, $\mathrm{coeff}_g(\mathbf{x}^{\mathbf{e}}) = \sum_{0 \le a \le \deg_z(u)} \mathrm{coeff}_u(\mathbf{x}^{\mathbf{e}} \cdot z^a)$. Therefore, we can calculate Φ_g using Φ_u and Observation A.3.

- If $g = u \times v$, then the coefficient of a monomial $\mathbf{x}^{\mathbf{e}}$ in g is given by $\sum_{\mathbf{e}'} \mathrm{coeff}_u(\mathbf{x}^{\mathbf{e}'}) \mathrm{coeff}_v(\mathbf{x}^{\mathbf{e}-\mathbf{e}'})$. The degree of f is at most 2^s, and hence there are at most $2^{ns} \le 2^{s^2}$ terms in the above sum. Therefore, if we can simulate bit-access to each of the product terms, then we can compute any bit of the sum in additional space $O(s^3)$ using Observation A.3. Since we can inductively compute the coefficient functions of both u and v in space $O((s-1)^4)$, Observation A.4 gives us bit-access to each of the terms in the sum, using an additional space of at most $O(s^3)$ bits. Putting all the pieces together, Φ_g can be decided in space at most $O((s-1)^4 + s^3)$.

Thus, the total space used is at most $O((s-1)^4 + s^3) \le O(s^4)$, for all large s. $\qquad\square$

Lemma A.6. *Let $f(\mathbf{x}) \in \mathbb{Z}[\mathbf{x}]$ be an n-variate, degree d polynomial such that its coefficient function can be computed in space s. Then there is a constant-free circuit with projection gates of size $\mathrm{poly}(s, n, \log d)$ that can compute $f(\mathbf{x})$.*

Proof. We prove this in two steps. First we show that the coefficient function can be efficiently computed by a constant-free algebraic circuit with projection gates. We then show that if the coefficient function of a polynomial can be efficiently computed by a constant-free algebraic circuit with projection gates, then the polynomial itself can be efficiently computed by a constant-free algebraic circuit with projection gates.

---

[15] Definition 2.16.

Claim A.7. *Let $f(\mathbf{x}) \in \mathbb{Z}[\mathbf{x}]$ be an n-variate, degree d polynomial such that its coefficient function can be computed in space s. Then there is a constant-free algebraic circuit with projection gates of size $\mathrm{poly}(s, n, \log d)$ that computes the coefficient function of f.*

Proof. Using the fact that totally quantified boolean expressions capture PSPACE, since the coefficient function of f can be computed using space s, we note that the coefficient function can be written as a quantified boolean formula of size $\mathrm{poly}(s)$.

We now *arithmetize* the boolean expression in the usual way to get an algebraic circuit. Additionally, we use projection gates to simulate the quantifiers using constant sized gadgets:

- $\forall y \Psi(y) \equiv \mathrm{fix}_{y=0}\, p(y) \times \mathrm{fix}_{y=1}\, p(y),$

- $\exists y \Psi(y) \equiv 1 - (1 - \mathrm{fix}_{y=0}\, p(y)) \times (1 - \mathrm{fix}_{y=1}\, p(y)).$

We therefore get an algebraic circuit with projection gates, of size $\mathrm{poly}(s, n, \log d)$ that computes the coefficient function of f. $\qquad\square$

Claim A.8. *Let $f(\mathbf{x}) \in \mathbb{Z}[\mathbf{x}]$ be an n-variate, degree d polynomial such that its coefficient function can be computed by a constant-free algebraic circuit with projection gates of size s'. Then there is a constant-free algebraic circuit with projection gates of size $\mathrm{poly}(s', n, \log d)$ that computes f.*

Proof. It is easy to see that projection gates can efficiently simulate exponential sums. So it suffices to provide an exponential sum that computes $f(\mathbf{x})$ given access to a circuit, say $\mathrm{CF}(\mathbf{y}_e, \mathbf{i})$, computing the coefficient function, as follows. Here $m = n \cdot \lceil \log d \rceil$ is the length of the bit-vectors for exponents, and b is the base-2-logarithm of the bit complexity of the coefficients of f.

$$f(\mathbf{x}) = \sum_{\mathbf{y}_e \in \{0,1\}^m} \left(\sum_{\mathbf{i} \in \{0,1\}^b} \mathrm{pow}(\mathbf{i}) \cdot \mathrm{CF}(\mathbf{y}_e, \mathbf{i}) \right) \cdot \mathrm{check}(\mathbf{y}_e) \cdot \mathrm{mon}(\mathbf{x}, \mathbf{y}_e)$$

The polynomials pow computes the ith power of two, check outputs 1 when the exponent $\mathbf{e}$ is valid, and zero otherwise, and mon computes the $\mathbf{e}^{th}$ monomial in $\mathbf{x}$. All these polynomials have easy constructions in size $\mathrm{poly}(m, b) = \mathrm{poly}(n, \log d, s)$. $\qquad\square$

The required statement clearly follows from the above two claims. $\qquad\square$