Electronic Colloquium on Computational Complexity, Report No. 31 (2022)

Transparency Beyond VNP in the Monotone Setting

Prerona Chatterjee [*] Kshitij Gajjar [†] Anamay Tengse [‡]

February 16, 2022

Abstract

Recently Hrubeš and Yehudayoff [HY21] showed a connection between the monotone algebraic circuit complexity of *transparent* polynomials and a geometric complexity measure of their Newton polytope. They then used this connection to prove lower bounds against monotone VP (mVP). We extend their work by showing that their technique can be used to prove lower bounds against classes that are seemingly more powerful than monotone VNP (mVNP).

In the process, we define a natural monotone analogue of VPSPACE — a well-studied class in the non-monotone setting [Poi08, KP09, Mal11, MR13] — and prove an exponential separation between the computational powers of this class and mVNP.

To show this separation, we define a new polynomial family with an interesting combinatorial structure which we use heavily in our lower bound. Both the polynomial and the combinatorial nature of our proof might be of independent interest.

[*]Tata Institute of Fundamental Research, Mumbai, India. Research supported by the Department of Atomic Energy, Government of India (under project number RTI4001) and in part by the Google PhD Fellowship. Email: prerona.ch@gmail.com.

[†]School of Computing, National University of Singapore. Research supported by NUS ODPRT Grant, WBS No. R-252-000-A94-133. Email: kshitijgajjar@gmail.com.

[‡]Department of Computer Science, University of Haifa, Israel. Research supported by the Israel Science Foundation (grant No. 716/20). Email: anamay.tengse@gmail.com.

ISSN 1433-8092

1 Introduction

The aim of algebraic complexity is to classify polynomials in terms of how hard it is to compute them, and the most standard model for computing polynomials is that of an *algebraic circuit*. An algebraic circuit is rooted directed acyclic graphs where the leaves are labelled with variables or field constants and internal nodes are labelled with addition $(+)$ or multiplication $(\times)$. Every node therefore naturally computes a polynomial and the polynomial computed by the root is said to be the polynomial computed by the circuit. The central question in the area is to show super-polynomial lower bounds against algebraic circuits for *explicit* polynomials, or to show that $\mathsf{VP} \neq \mathsf{VNP}$, and is the algebraic analogue of the famed P vs. NP question.

However, proving strong lower bounds against circuits has turned out to be a difficult problem. Much of the research therefore naturally focusses on various restricted algebraic models which compute correspondingly structured polynomials. One such syntactic restriction is that of *monotonicity*, where the models are not allowed to use any negative constants. Therefore, trivially, monotone circuits always compute polynomials with only non-negative coefficients. Such polynomials are called *monotone polynomials*. We denote the class of all polynomials that are efficiently computable by monotone algebraic circuits by mVP. Also note that any monomial computed during intermediate computation in a monotone circuit can never get cancelled out, making it a very weak model. As a result, several strong lower bounds are known against monotone circuits.

Lower bounds in the monotone setting

There has been a long line of classical works that prove lower bounds against monotone algebraic circuits [Sch76, SS77, SS80, JS82, Kuz85, KZ86, Gas87]. The most well-known among these, is the result of Jerrum and Snir [JS82] where they showed exponential lower bounds against monotone circuits for many polynomial families, including the Permanent. In particular, they showed that every monotone algebraic circuit computing the n^2-variate Perm_n must have size at least $2^{\Omega(n)}$. A few of the more recent works on monotone lower bounds include [RY11, GS12, CKR20].

Additionally, many separations that are believed to be true in the general setting have actually been proved to be true in the monotone setting [SS77, HY16, Yeh19, Sri19]. Most remarkably, Yehudayoff [Yeh19] showed an exponential separation between the computational powers of the the monotone analogues of VP and VNP (denoted by mVP and mVNP respectively).

Another line of work in this setting tries to understand the power of non-monotone computational models while computing monotone polynomials. Valiant [Val80], in his seminal paper, showed that there is a family of monotone polynomials which can be computed by polynomial sized non-monotone algebraic circuits such that any monotone algebraic circuit computing them must have exponential size. More recent works [HY13, CDM21, CDGM22] have shown even stronger separations between the relative powers of monotone and non-monotone models while

computing monotone polynomials.

The Tau conjecture for Newton polygons, transparent polynomials, and monotone complexity

The Newton polytope of an n-variate polynomial f, denoted by $\mathsf{Newt}(f)$, is the convex hull in $\mathbb{R}^n$ of the *exponent vectors* of the monomials in the support of f (see Definition 2.4). Recently, Hrubeš and Yehudayoff [HY21] proposed the notion of *Shadows of Newton polytopes* (the maximum number of vertices in any linear projection of the polytope to a plane) as an approach to refute the τ-conjecture for Newton polygons made by Koiran, Portier, Tavenas and Thomassé [KPTT15].

Conjecture 1.1 (τ-conjecture for Newton polygons [KPTT15])**.** *Let* $f(x,y) = \sum_{i\in[s]} \prod_{j\in[r]} T_{i,j}(x,y)$, *with each* $T_{i,j}$ *having at most* p *monomials. If*

$$\mathsf{Newt}(f) = \mathrm{conv}\left(\left\{(a,b) \;:\; x^a y^b \text{ is a monomial in } f\right\}\right),$$

then $\mathsf{Newt}(f)$ *has at most* $\mathrm{poly}(s,r,p)$ *vertices.*

This is a very strong conjecture and it implies, among other things, that $\mathsf{VP} \neq \mathsf{VNP}$. However, observe that the Newton polygon retains no information about the coefficients of the polynomial, and since the algebraic complexity of polynomials is believed to be heavily dependent on coefficients[1], Conjecture 1.1 is believed to be false.

The approach suggested by Hrubeš and Yehudayoff used shadows of Newton polytopes as a means to move from the multivariate setting to the bivariate setting, and use polynomials like determinant (Det_n) to refute Conjecture 1.1. A crucial component of this agenda is the fact that all polynomials in VP can be written as $\sum_{i\in[s]} \prod_{j\in[r]} T_{i,j}(x,y)$ for $s \leq 2^{O(\sqrt{\deg \cdot \log^2 n})}$. Thus, if an n-variate, degree $\mathrm{poly}(n)$ polynomial in VP were to have a bivariate polynomial with $2^{\deg^{0.51}}$ vertices as its *shadow*, then Conjecture 1.1 would be proven false. The difficulty in this strategy however, is to find a polynomial in VP that exhibits high *shadow complexity*. The reason behind this is that even when a candidate polynomial is fixed, say Det_n, it is not easy to design a suitable bivariate projection.

As a means to tackle this issue, Hrubešand Yehudayoff introduced the notion of *transparent polynomials* — polynomials that can be projected to bivariates in such a way that all of their monomials become vertices of the resulting Newton polygon. Further, they also gave examples of polynomials with exponentially large sets of monomials that are provably transparent. Therefore a proof of any one of these polynomials being in VP would directly refute Conjecture 1.1.

Even though Hrubeš and Yehudayoff [HY21] were not able to actually use this approach to refute Conjecture 1.1, they used the notions of shadows and transparency to come up with yet another method for proving lower bounds against monotone algebraic circuits. They showed

[1]For example, determinant (Det_n) is efficiently computable by algebraic circuits and this is expected to not be the case for Perm_n even though they have the same set of monomials.

that the monotone circuit complexity of a polynomial is lower bounded by its shadow complexity when the polynomial is transparent.

Theorem 1.2 ([HY21, Theorem 2]). *If f is transparent then every monotone circuit computing f has size at least $\Omega(|\mathrm{supp}(f)|)$.*

As a corollary, they present an n-variate polynomial such that any monotone algebraic circuit computing it must have size $\Omega(2^{n/3})$. In this work, we examine the hardness implications of transparency against monotone models of computation in more detail.

1.1 Beyond monotone VP

The starting point for our search is an observation by Yehudayoff [Yeh19], that any lower bound against mVP that depends solely on the support of the hard polynomial, automatically "lifts" to mVNP with the same parameters (Theorem 3.3). Since transparency is a property solely of the Newton polytope, and hence the support of the polynomial, the above observation shows that any transparent polynomial that is non-sparse (has super-polynomially large support) is hard to compute even for mVNP. However, transparency seems like a very strong property and a natural question therefore is whether there are even larger classes of monotone polynomials that do not contain non-sparse transparent polynomials.

Classes larger than VNP had not been defined in the monotone world prior to our work. We thus turn to the literature in the non-monotone setting, where VPSPACE is a well studied class that is expected to be strictly larger than VNP. There are multiple definitions of VPSPACE, resulting from varied motivations, which are all essentially equivalent [Mal11, MR13].

The various definitions of VPSPACE

Koiran and Perifel [KP09, KP07] were the first to define VPSPACE in order to prove a "transfer theorem" — a collapse in one model leading to a collapse in the other — from algebraic circuit complexity to the *Blum-Shub-Smale* model. They defined (uniform) VPSPACE as the class of polynomials whose "coefficient-language" is decidable in PSPACE (Turing machines using polynomially bounded space). Their transfer theorem essentially says that if VPSPACE has efficient algebraic circuits, then $\mathsf{P} = \mathsf{NP}$ in the *Blum-Shub-Smale* model ($\mathsf{P}_{\mathbb{C}} = \mathsf{NP}_{\mathbb{C}}$).

Later, Poizat [Poi08] gave an alternate definition that does not rely on any boolean machinery, but instead uses a new type of gate called a *projection gates*. He defined VPSPACE as the class of polynomial families that are computable by poly-sized algebraic circuits that have addition, multiplication, and projection gates. Loosely, a projection gate that is labelled by a variable z and a constant b, accepts a single polynomial as input and outputs its (partial) evaluation at $z = b$.

Poizat showed[2] that this definition is equivalent to that of Koiran and Perifel.

Adding to the above mentioned work of Poizat, Malod [Mal11] characterised VPSPACE using exponentially large *algebraic branching programs (ABPs)* that are *succinct*. An algebraic branching program is a directed graph with two special vertices, source s and a sink/target t. Every edge is labelled by a linear polynomial over the underlying variables, and the polynomial computed by an s–t path is the product of the edge labels on that path. The polynomial computed by the ABP is the sum of the polynomials computed by all its s–t paths. Malod's work defines the *complexity* of an ABP as the size of the smallest algebraic circuit that encodes its graph — outputs any edge label when given the two endpoints as input. An n-variate ABP is then said to be *succinct*, if its complexity is $\text{poly}(n)$. Please refer to subsection 6.1 for a formal definition.

In the same work [Mal11], Malod alternatively characterised VPSPACE using an interesting algebraic model that resembles *(totally) quantified boolean formulas* that are known to characterise PSPACE. This model is defined using special types of projection gates called *summation* and *production* gates, and we refer to it as "quantified algebraic circuits". A summation gate labelled by a variable z returns the sum of the $(z = 0)$ and $(z = 1)$ projections of its input, and a production gate returns the product of these partial evaluations (see Definition 2.3). A quantified algebraic circuit has the form $\mathtt{Q}^1_{z_1}\mathtt{Q}^2_{z_2}\cdots\mathtt{Q}^m_{z_m}C(\mathbf{x},\mathbf{z})$, where each $\mathtt{Q}^i$ is a summation or a production, and $C(\mathbf{x},\mathbf{z})$ is a usual algebraic circuit. A formal definition can be found in subsection 6.2.

Finally, Mahajan and Rao [MR13] defined algebraic analogues of small space computation (e.g. L, NL) using the notion of *width* of an algebraic circuit. Using their definition, they import some of the relationships from the boolean world to the algebraic world (for example, they show $\mathsf{VL} \subseteq \mathsf{VP}$). They further show that their definition of uniform polynomially-bounded-space computation coincides with that of uniform VPSPACE from the work of Koiran and Perifel [KP09].

We now narrow our focus to the definitions due to Poizat [Poi08] and Malod [Mal11]. We choose these definitions because they are algebraic in nature, and have fairly natural monotone analogues. We elaborate a bit more about this decision in section 6.

Remark. *It should be noted that all the above mentioned definitions of* VPSPACE *allow for the polynomial families to have large degree — as high as* $\exp(\text{poly}(n))$. *The main focus of our work, however, is to compare the monotone analogues of these models with* mVP *and* mVNP. *Since these classes only contain low-degree polynomials, for the rest of this paper, we will only work with polynomials of degree* $\text{poly}(n)$ *unless explicitly mentioned otherwise.* ◊

1.2 Our contributions

We define monotone analogues for the various definitions of VPSPACE outlined in the previous section, and compare the powers of the resulting monotone models/classes. We then analyse the

[2] The work of Poizat is written in French, Malod [Mal11] provides an alternate exposition of some of the main results in English.

scope of support-based lower bounds (like [JS82]) and lower bounds for transparent polynomials (from [HY21]) in the context of these new computational models and complexity classes. We now describe these results in greater detail.

Beyond VNP in the monotone setting

Monotone VPSPACE. Adapting the definition due Poizat [Poi08], we define mVPSPACE — the monotone analogue of VPSPACE — as the class of polynomials computable by monotone algebraic circuits of polynomial size that have addition, multiplication *and projection gates* (formally defined in Definition 2.7). A projection gate labelled by a variable z and a constant $b \in \{0,1\}$, takes a single input $f(z, \mathbf{x})$ and outputs $f(b, \mathbf{x})$, thus allowing partial evaluations of intermediate computations. An immediate question is whether this newly defined class is indeed more powerful than mVNP. Our first main result is that this is indeed the case.

Theorem 1.3 (Separating mVNP and mVPSPACE)**.** *There is a family* $\{P_n\} \in$ mVPSPACE *such that for every* n*,*

- P_n *is a* $2n$*-variate polynomial of degree* n*;*
- *there is a monotone algebraic circuit with projection gates, of size* $O(n \log n)$*, that computes* P_n*;*
- *every monotone* VNP *circuit computing* P_n *has size* $2^{\Omega(n^{0.63})}$*.*

We construct a new polynomial family to prove the above theorem, which we call *Overlapping parities (OP)*. The motivation for choosing this family is described in greater detail in subsection 1.3.

To further highlight the power of this definition of mVPSPACE, we show that the Permanent family is also in this class.

Theorem 1.4. *There is a monotone circuit with projection gates of size* $O(n^3)$ *that computes* Perm_n*.*

Even though this already gives an $O(m^{1.5})$ vs. $2^{\Omega(\sqrt{m})}$ separation between mVNP and mVPSPACE due to the classical result of Jerrum and Snir [JS82] and the observation of Yehudayoff [Yeh19], Theorem 1.3 gives a quantitatively better separation. Further, the Overlapping Parities polynomial family has interesting combinatorial properties that allow us to prove the desired lower bound. We believe that these properties and the combinatorial nature of our arguments leading up to the lower bound are interesting in their own right. Please see subsection 1.3 for details.

Monotone succinct ABPs. Next we consider the natural monotone analogue of the definition due to Malod [Mal11] which uses succinct algebraic branching programs.

Malod showed that every family $\{f_n\}$ in VPSPACE can be computed by $2^{\mathrm{poly}(n)}$ sized ABPs that have complexity $\mathrm{poly}(n)$. Recall that the complexity of a succinct ABP is the size of the smallest

algebraic circuit that describes its graph (that is, takes the encoding of any two vertices as inputs and outputs the label of the edge between them).

We therefore define monotone succinct ABPs as ABPs that can be succinctly described by *monotone* algebraic circuits of size $\text{poly}(n)$. However, unlike in the non-monotone setting, we show that this model does not coincide with mVPSPACE. In fact we show that the computational power of monotone succinct ABPs does not go beyond mVNP.

Theorem 1.5. *If an n-variate polynomial $f(\mathbf{x})$ of degree d is computable by a monotone succinct ABP of complexity $\text{poly}(n)$, then $f(\mathbf{x}) \in$ mVNP.*

Quantified monotone circuits. As mentioned earlier, Malod [Mal11] had also characterised the class VPSPACE using the notion of quantified algebraic circuits. We now consider its natural monotone analogue — where the "inner" algebraic circuit is restricted to be monotone — which we call quantified monotone circuits. Please refer to Definition 6.5 for a formal definition.

It is easy to see that quantified monotone circuits of polynomial size can simulate monotone VNP. Yehudayoff [Yeh19] observed that any lower bound against mVP that relies on the support of the hard polynomial also extends to mVNP with the same parameters. We observe that lower bounds based on support actually extend all the way to quantified monotone circuits of polynomial size, whenever the support cannot be written as a non-trivial product of two sets (see Lemma 5.16). As our polynomials OP_k have this property, we get an identical lower bound against the quantified monotone circuits.

Theorem 1.6. *There exists a family $\{P_n\} \in$ mVPSPACE such that for all n large enough, P_n requires quantified monotone circuits of size $2^{\Omega(n^{0.63})}$.*

We also note that the Permanent family does not have poly-sized quantified monotone circuits and that an interesting open thread is to show a separation between this class and mVNP.

Extending the techniques of Hrubeš and Yehudayoff

Finally we consider a model that is perhaps even more powerful than quantified monotone circuits, where the summation and production gates are allowed to appear anywhere in the circuit. For our second main theorem, we show that even this additional power does not help much in monotone computation of transparent polynomials. Specifically, we extend the lower bound for transparent polynomials against monotone circuits ([HY21, Theorem 5.9]) to obtain an identical lower bound against circuits that have arbitrary summation and production gates along with the usual addition and multiplication gates. This shows that transparent polynomials with large support are hard even for this model, which is seemingly even more powerful than mVNP.

Theorem 1.7. *Any monotone algebraic circuit with summation and production gates that computes a transparent polynomial f, has size $|\text{supp}(f_n)|/4$.*

Although we are currently unable to prove a similar lower bound against monotone circuits with (general) projection gates, we believe that transparency is a highly restrictive property, especially for monotone computation, and therefore despite allowing projection gates, it should still be hard to obtain a circuit of size $\text{polylog}(\text{supp}(f))$ for a transparent f.

Consequences of refuting the Tau Conjecture using transparency. Recall that one way to refute Conjecture 1.1 is to show a transparent polynomial in (non-monotone) VP. Our lower bound for transparent polynomials against monotone circuits with summations and productions, also means that any transparent polynomial from VP that refutes Conjecture 1.1 would also witness a separation between VP and quantified monotone circuits of polynomial size, among other things.

Further, if transparent polynomials also happen to be hard for mVPSPACE as we believe, then a transparent polynomial in VP would essentially show that VP $\not\subset$ mVPSPACE. Even though stark separations between monotone and non-monotone models are not unheard of [HY13, CDM21], such a result would be very interesting, and would further highlight the power of subtractions.

1.3 Proof overview

Let us now go over the proof techniques for our results. We begin with Theorem 1.3.

The hard polynomial family. A famous example in the probability/information theory texts that conveys the subtlety of conditional independence/information is the following.

> Let X, Y be independent, uniformly random bits, and define $Z = X \oplus Y$. Now the random variables X, Y, Z have a peculiar property — any two of them are (pairwise) independent, but conditioned on the third variable, they become fully correlated.

In the language of mutual information, for instance $I(X : Y) = 0$, but $I(X : Y|Z) = 1$. Thus, in some sense, each of the 3 bits controls exactly half of the "total randomness". We use this property recursively to build our family *Overlapping Parities*, $\{\mathsf{OP}_k\}$.

The polynomial OP_k has monomials of degree exactly $n = 3^k$ where each monomial encodes a bit-string of length n and has coefficient 1. For $k = 1$, the monomials correspond to the bit-strings $\{001, 010, 100, 111\}$, very similar to the possible outcomes of X, Y, Z in the above example. For larger k, suppose $\mathbf{b}$ is a bit-string for some monomial in OP_k, and let $\mathbf{b} = (\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2)$ be its partition into three equal-sized, contiguous blocks. The strings $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ are such that each of them is a valid string corresponding to OP_{k-1}, and the bit-wise parity (XOR) — $\mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2$ — gives the all-ones bit-string of length 3^{k-1}.

Similar to the example of X, Y, Z above, the set of such bit-strings has the property that fixing any $1/3$ fraction of the bits, reduces the "total randomness" by at least a factor of $1/2$. This property is the key to our lower bound proof, which we describe next. A complete description of the polynomial family is given in subsection 2.2.

Lower bound against monotone VNP

The proof of the lower bound against mVNP for OP_k essentially consists of the following steps.

Step 1: If OP_k can be computed by an monotone algebraic circuit of size s then $\mathsf{OP}_k = \sum_{i=1}^{s} g_i \cdot h_i$ where for every i, g_i and h_i are monotone polynomials of degree at least $n/3$.

Step 2: We show that for every i, one of g_i and h_i has sparsity 1, and the other has sparsity at most $2^{2^{k-1}}$. As a result, the sparsity of $g_i \cdot h_i$ is also, at most $2^{2^{k-1}}$.

Step 3: Finally we show that OP_k has sparsity 2^{2^k} and so the previous two steps together imply that s is at least $2^{2^{k-1}}$.

Step 4: Using the observation by Yehudayoff [Yeh19] that support-based lower bounds against mVP extends to mVNP as well, step 3 proves that any mVNP circuit computing OP_k must have size at least $2^{2^{k-1}}$.

The first step is a standard structural lemma that has been used in various lower bound proofs against monotone algebraic circuits. Also, the sparsity count in the third step follows fairly easily from the definition of OP_k. The second step is the most non-trivial part of our proof and our main technical contribution. So we describe this step in greater detail.

Sparsity upper bound on $g_i \cdot h_i$. Note that since we are working with monotone circuits, all the monomials in the product $g_i \cdot h_i$ appear in final polynomial (none of them can ever cancel out). Therefore all of them must be of degree exactly n, and further, all of them must be "valid". Recall that monomials in OP_k correspond to bit strings. The above observation therefore means that for every i, g_i and h_i have to work with two *disjoint* sets of positions in the bit strings that together cover all the positions. For the rest of this discussion, we focus on a fixed pair g_i, h_i and therefore drop the subscripts.

There are two components of this bound, proving that either g or h has sparsity 1 and proving that the other has sparsity at most $2^{2^{k-1}}$. The first component has a simpler argument than the second and therefore we begin with the first component.

Either g or h is a monomial This property is in fact true irrespective of the degrees of g and h. That is, for any 'monotone' product $g \cdot h$ that computes only valid monomials of OP_k, either g or h is a monomial. Consider the case when $k = 1$. Clearly, either g or h has degree 2, and any monomial in the degree 2 term fixes two positions in the string. This forces the remaining bit and hence the degree one term is just a monomial.

As an intermediate case, let $k > 1$, but suppose g "controls" the first n_1 bits. If $n_1 \leq n/3$, then it follows from our construction of OP_k that h "controls" two whole "blocks" of length $n/3$ out of the three. This forces a unique choice on the bits controlled by g. When $n/3 <$

$n_1 < n/2$, we focus on the "central block" of length $n/3$, as it is not fully controlled by either g or h. Recall that the substring in the middle block is itself a valid string for OP_{k-1}. Thus we can inductively say that either g or h controls the central block in addition to its own block, and this forces a single choice for the other block.

In the general case, g and h do not control contiguous segments, and we have to abstract out a more general strategy from the above discussion. Imagine a 'complete ternary tree' of height k, so that there are $3^k = n$ leaves, identified with the n positions. We call it an *assignment fixing tree (or AFT)*, which we also use in the next component. We colour the nodes in the tree as follows.

> Leaves (positions) controlled by g are coloured blue and those controlled by h are coloured red. Internal nodes are iteratively coloured the same as the majority of their children.

We then show that if the root is coloured red then g is a monomial, and if it is coloured blue then h is a monomial. We provide an illustration of this colouring in Figure 2.

Loosely, since bit strings in OP_k are constructed iteratively all the way from $k = 1$, the colouring of a node using recursive majority in fact tells us which amongst g and h indirectly influences *all* the leaves (positions) that are under that node. This forces a unique choice for the other. In other words, if the root is red then any monomial of h has a fixed "valid" completion and hence forces g to have sparsity 1.

Note, however, that in this case h might have sparsity more than 1 since multiple monomials can have the same "fixed valid completion". For example, let $k = 2$ ($n = 9$) and suppose h controls positions $0, 1, 3, 4$, while g controls positions $2, 5, 6, 7, 8$. Note that $**1**1111$ is a valid extension for both $11*11****$ as well as $00*00****$.

We next show that even in this case, the sparsity of h can not be too much.

Upper bounding the number of completions Let us continue with the bi-coloured *assignment fixing tree (AFT)*. We now show that if the root is coloured blue then g has sparsity at most $2^{2^{k-1}}$, and if the root is red then h has sparsity at most $2^{2^{k-1}}$. Let us assume, without loss of generality, that the root is red since the other case is symmetric.

Here we will need to use the fact that at least $n/3$ leaves are blue. The simplest case is when there are exactly $n/3$ blue leaves and all of them are under the leftmost child, say, of the root. Recall that g is now a monomial and hence the bits in the left block are fixed. In order to complete this partial string, any fixing of the central block fully determines the right block. However, as it turns out from the proof of Lemma 2.10, h is free to choose any string from OP_{k-1} in the central block and therefore has exactly $2^{2^{k-1}}$ choices. So we have to show that the simplest case achieves the highest sparsity for h.

This task turns out to be bizarrely difficult and resists several attacks that use a weak induction hypothesis. We manage to get around this by first coming up with a scoring system for the nodes in the AFT that tracks their "degrees of freedom" and then coming up with an explicit formula (see Lemma 3.11) that bounds this score of a node given the number of blue leaves under it.

The score of a node exactly counts the "loss in degrees of freedom" for it. For example, the parent of one blue and two red leaves has score 1 since it has 1 less degree of freedom compared to the 'full' amount (which is 2). We then note that for any node, its "loss in degrees of freedom" (encoded by its score) is the sum of the scores of its two highest scored children. This is because fixing any two children fixes the parent and higher scored nodes have fewer degrees of freedom.

We now come to the formula that lower bounds the score of a node given the number of its blue leaves. In a nutshell, we devise a function[3] $t(m)$ such that if a node has between $t(m)$ and $t(m+1)$ blue leaves, then its score is at least m. This function has the following crucial property — if $m = a + b$ for $a \geq b$, then $t(m) \geq t(a) + 2t(b)$. This coincides with a parent node distributing the blue leaves among its children (the three children getting $t(a) \geq t(b) \geq t(c)$ blue leaves respectively) and also the scores ($a \geq b \geq c$ respectively). We then observe that $t(2^{k-1}) = 3^{k-1}$. This shows that having at least 3^{k-1} blue leaves reduces the degrees of freedom by at least 2^{k-1}, as required.

The mVPSPACE upper bound

We first note the clear advantage that projection gates provide: projecting a variable to 0 can filter out unwanted monomials. We observe that one can design a *selection gadget* by projecting multiple auxiliary variables in a "correlated" manner. For example, consider the polynomial $(zx_0 + yx_1)$. We can obtain x_0 from this by setting $y = 0, z = 1$ and obtain x_1 by setting $y = 1, z = 0$.

In order to compute OP_k, we first compute monomials corresponding to all the 2^n bit strings of length n. We then use a combination of these selections gadgets to prune out the unwanted monomials. The latter part requires us to have suitable auxiliary variables attached to the monomials computed in the first step, which we achieve using the following idea.

Here is an easy way to compute monomials for all n-bit strings:

$$(x_{1,0} + x_{1,1})(x_{2,0} + x_{2,1}) \dots (x_{n,0} + x_{n,1}).$$

Suppose we wanted to ensure that the parity of the first two bits is always 1. For this, we can add

[3] It turns out that the sequence $t(0), t(1), t(2), \dots$ has connections to the Pascal's triangle, see OEIS:A006046.

4 auxiliary variables in the first two brackets (zs go with zeroes and ys go with ones):

$$(z_1 x_{1,0} + y_1 x_{1,1})(z_2 x_{2,0} + y_2 x_{2,1}) \dots .$$

Now all that we have to ensure is that we either "select $x_{1,0}$ and $x_{2,1}$" or "select $x_{1,1}$ and $x_{2,0}$". This can be done by setting the auxiliary variables suitably, and adding the two resulting polynomials.

At this point, we observe that the valid bit strings of OP_k are solutions to a system of linear constraints. Each bit participates in exactly k constraints and each constraint involves exactly 3 bits. This gives a total of $nk/3 = O(n \log n)$ constraints and each of these 3-bit-constraints can then be satisfied, as in the above toy example, using 6 *distinct* auxiliary variables (two for each position in the constraint).

Extending the lower bound for transparent polynomials

Next, we give the proof overview of Theorem 1.7.

This result is an extension of the ideas in the work of Hrubeš and Yehudayoff [HY21]. Their argument shows that any bivariate monotone circuit of size s that computes a polynomial with *convexly independent support* outputs a polynomial with support at most $4s$. They achieve this by keeping track of the largest polygon (having most vertices) that one can build using the polynomials computed at all the gates in the circuit. They then inductively show that no gate (leaf, addition, multiplication) can increase the number of vertices by 4. We are able to show the same bound for production and summation gates, by working with a monotone bivariate circuit over y_1, y_2 that is allowed some auxiliary variables z for summations and productions.

An important component of the proof in [HY21] is that if the sum or product of two monotone polynomials in convexly independent, then so are each of the two inputs. However, the allowing for summations and productions means that some monomials that are computed internally could get "zeroed out". In fact, summation and production gates do not quite "preserve convex dependencies". For example, the convexly dependent support $\{y_1y_2, y_1y_2z, y_1y_2z^2\}$ when passed through $\mathtt{sum}_z$ produces just $\{y_1y_2\}$, which is convexly independent.

In order to prove Theorem 5.7, we get around this by working directly with the support projected down to the "true" variables, which we call $\mathbf{y}$-support in our arguments. It turns out that summations and productions indeed preserve convex dependencies that are in the $\mathbf{y}$ support of the input polynomial.

Proofs overview of our other results

The central idea in showing that $\{\mathsf{Perm}_n\} \in \mathsf{mVPSPACE}$ (Theorem 1.4), is the same as that for the upper bound for OP_k — using correlated projections to prune invalid monomials. Hence, the construction is also very similar.

The collapse of monotone succinct ABPs of polynomial complexity to monotone VNP in Theorem 1.5 essentially follows from the definition, and does not need any non-trivial ideas.

Similarly, a careful look at how a production gate modifies the support of its input polynomial tells us that summations and productions do not assist monotone circuits in computing a polynomial with an "irreducible support" (Lemma 5.16). We then observe that the polynomials OP_k have irreducible supports, proving Theorem 1.6.

1.4 Organisation of the paper

We begin in section 2 with formal definitions for all the models of computation that we will be using, as well as a complete description of the Overlapping Parities polynomial family. Then in section 3, we prove the lower bound against monotone VNP for our newly defined polynomial. In subsection 4.1, we prove the mVPSPACE upper bounds for both the Overlapping Parities polynomial as well as the Permanent. We then extend the proof techniques of Jerrum and Snir, and Hrubešand Yehudayoff, to monotone classes beyond mVNP in section 5. Finally, in section 6, we elaborate on our choice for the definition of mVPSPACE. We conclude with section 7, where we discuss some of the important open threads from our work.

2 Preliminaries

We shall use the following notation for the rest of the paper.

- We use the standard shorthand $[n] = \{1, 2, \ldots, n\}$. In some of our arguments we will be dealing with vectors that are indexed starting from 0 for notational convenience. In such cases we use $[\![n]\!]$ to refer to the set $\{0, 1, \ldots, n-1\}$.
- We use boldface letters like $\mathbf{x}, \mathbf{z}, \mathbf{e}$ to denote tuples/sets of variables or constants, individual members are expressed using indexed version of the usual symbols: $\mathbf{e} = (e_1, e_2, \ldots, e_n)$, $\mathbf{x} = \{x_1, \ldots, x_n\}$. We also use $|\mathbf{y}|$ to denote the size/length of a vector $\mathbf{y}$.

  For vectors $\mathbf{x}$ and $\mathbf{e}$ of the same length n, we use the shorthand $\mathbf{x}^{\mathbf{e}}$ to denote the monomial $x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$.
- $\mathsf{ONES} = \{(0,0,1), (0,1,0), (1,0,0), (1,1,1)\}$ is the set of all combinations of three bits whose parity is one.
- For a polynomial $f(\mathbf{x})$ and a monomial $m = \mathbf{x}^{\mathbf{e}}$, we refer to the coefficient of m in f by $\mathrm{coeff}_f(m)$.

  The support $\mathrm{supp}(f)$ of a polynomial f is given by $\{m : \mathrm{coeff}_f(m) \neq 0\}$, and the *sparsity* of a polynomial is the size of its support.

Basic models of algebraic computation. We now define algebraic circuits and branching programs, which are the basic blocks for other models that we handle in our work.

Definition 2.1 (Algebraic circuits)**.** *An algebraic circuit is a directed acyclic graph (DAG) with leaves (nodes with in-degree zero) labelled by formal variables and constants from the field, and other nodes labelled by addition $(+)$ and multiplication $(\times)$. The leaves compute their labels, and every other node computes the operation it is labelled by on the polynomials along its incoming edges. There is a unique node of out-degree zero called the root, and the circuit is said to compute the polynomial computed at the root.*

The size *of a circuit is the number of nodes in the graph, and the* depth *of the circuit is the length of the longest path from a leaf to the root.*

An algebraic circuit over $\mathbb{Q}$ or $\mathbb{R}$ is said to be monotone*, if all the constants appearing in it are non-negative.* $\Diamond$

Definition 2.2 (Algebraic Branching Programs (ABPs))**.** *An algebraic branching program (ABP in short), is an directed acyclic graph with two special nodes: source s and sink t. All edges in the ABP are labelled with linear forms in the underlying set of variables. Each edge computes its label, a path computes the product of the edges, and the ABP is said to compute the sum of all s–t paths.*

The size *of an ABP is the number of vertices in the underlying graph, and the* length *of an ABP is the length of the longest s–t path.*

An ABP over $\mathbb{Q}$ or $\mathbb{R}$ is said to be monotone*, if all the constants appearing in it are non-negative.* $\Diamond$

Projection, summation and production gates. As mentioned earlier, some of the definitions for VPSPACE use special kinds of gates. We now formally define their behaviour.

Definition 2.3 (Projection, Summation and Production gates)**.** *A* projection *gate is a* unary *gate that is labelled by a variable z and a constant $b \in \{0,1\}$, denoted by $\mathtt{fix}_{(z=b)}$. It returns the partial evaluation of its input polynomial, at $z = b$, that is, $\mathtt{fix}_{(z=b)}(f(z,\mathbf{x})) = f(b,\mathbf{x})$.*

Note that a projection gate labelled by a variable z, leaves any z-free input polynomial unchanged.

Summation *and* production *gates are also unary gates that are labelled by a variable z, and are denoted by $\mathtt{sum}_z$ and $\mathtt{prod}_z$ respectively. A summation gate returns the sum of the $(z = 0)$ and $(z = 1)$ evaluations of its input, and a production gate returns the product of those evaluations. That is, $\mathtt{sum}_z(f(z,\mathbf{x})) = f(0,\mathbf{x}) + f(1,\mathbf{x})$, and $\mathtt{prod}_z(f(z,\mathbf{x})) = f(0,\mathbf{x}) \cdot f(1,\mathbf{x})$.* $\Diamond$

Newton polytopes. We will also need the concept of the Newton polytope of a polynomial.

Definition 2.4 (Newton polytopes)**.** *For a polynomial $f(\mathbf{x})$, its Newton polytope $\mathsf{Newt}(f) \subseteq \mathbb{R}^n$, is defined as the convex hull of the* exponent vectors *of the monomials in its support.*

$$\mathsf{Newt}(f) := \mathrm{conv}\left(\{\mathbf{e} : \mathbf{x}^{\mathbf{e}} \in \mathrm{supp}(f)\}\right)$$

A point $\mathbf{e} \in \mathsf{Newt}(f)$ is said to be a vertex*, if it cannot be written as a convex combination of* other

points in $\mathsf{Newt}(f)$*. We denote the set of all vertices of a polytope* $\mathcal{P}$ *using* $\mathrm{vert}(\mathcal{P})$*.* $\Diamond$

2.1 Monotone complexity classes

Next we define the various monotone classes that we will be using.

Definition 2.5 (Monotone VP (mVP))**.** *A family* $\{f_n\}$ *of monotone polynomials is said to be in* mVP*, 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 a monotone algebraic circuit of size at most* n^c*.* $\Diamond$

Definition 2.6 (Monotone VNP (mVNP))**.** *A family* $\{f_n\}$ *of monotone polynomials is said to be in* mVNP*, if there exists a constant* $c \in \mathbb{N}$*, and a family* $\{g_m\} \in \mathsf{mVP}$*, such that for all large enough* n*, and* $m \leq n^c$*,* f_n *satisfies the following.*

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

We now formally state our proposed definition for the monotone analogue for VPSPACE.

Definition 2.7 (Monotone VPSPACE (mVPSPACE))**.** *A family* $\{f_n\}$ *is said to be in monotone* VPSPACE*, or* mVPSPACE*, if there exists a constant* $c \in \mathbb{N}$*, such that for all large enough* n*, the polynomial* f_n *depends on at most* n^c *variables, and can be computed by an algebraic circuit* with projection gates*, of size (i.e. number of all gates) at most* n^c*.* $\Diamond$

2.2 The Overlapping Parities (OP) polynomial

We define a new polynomial family, which we call the Overlapping Parities polynomial family. Given a positive integer k, the k overlapping parities polynomial (denoted by $\mathsf{OP}_k(\mathbf{x})$) is a polynomial of degree 3^k, with the underlying variables being

$$\left\{ x_{i,b} \ : \ (i,b) \in \left\{0,1,\ldots,3^k-1\right\} \times \{0,1\} \right\}.$$

To succinctly describe $\mathsf{OP}_k(\mathbf{x})$, we denote the ternary (base 3) representation of $i \in \left\{0,1,\ldots,3^k-1\right\}$ by $T(i) = T_{k-1}(i)T_{k-2}\cdots T_1(i)T_0(i)$, where each $T_j(i) \in \{0,1,2\}$ is a "trit". For example, the decimal number 38 is 1102 in base 3. Its most significant trit is 1 and its least significant trit is 2.

The Hamming difference HAM between two ternary strings is defined as the set of positions where they differ. For example, $\mathtt{HAM}(12011020, 12212021) = \{0,3,5\}$.

Now, let us describe the monomials of the polynomial $\mathsf{OP}_k(\mathbf{x})$. Each monomial m of $\mathsf{OP}_k(\mathbf{x})$ is a multilinear monomial of degree 3^k of the form $\prod_{i=0}^{3^k-1} x_{i,b_{i,m}}$. Let $\mathcal{M}_{\mathsf{OP}_k}$ be the set of monomials supporting $\mathsf{OP}_k(\mathbf{x})$. Then, a monomial $m = \prod_{i=0}^{3^k-1} x_{i,b_{i,m}}$ is in $\mathcal{M}_{\mathsf{OP}_k}$ if and only if for all ternary

strings $i_1, i_2, i_3 \in \{0, 1, \ldots, 3^k - 1\}$ and positions $j \in \{0, 1, \ldots, k-1\}$,

$$\texttt{HAM}(i_1, i_2) = \texttt{HAM}(i_2, i_3) = \texttt{HAM}(i_1, i_3) = \{j\} \quad \Longrightarrow \quad b_{i_1,m} \oplus b_{i_2,m} \oplus b_{i_3,m} = 1. \tag{2.8}$$

The polynomial $\mathsf{OP}_k(\mathbf{x})$ is defined as

$$\mathsf{OP}_k(\mathbf{x}) = \sum_{m \in \mathcal{M}_{\mathsf{OP}_k}} m.$$

For example,

$$\mathsf{OP}_1(\mathbf{x}) = x_{0,0}x_{1,0}x_{2,1} + x_{0,0}x_{1,1}x_{2,0} + x_{0,1}x_{1,0}x_{2,0} + x_{0,1}x_{1,1}x_{2,1}.$$

Let $n = 3^k$. Note that $\mathsf{OP}_k(\mathbf{x}) = \mathsf{OP}_{\log_3 n}(\mathbf{x})$ is a $2n$-variate polynomial of degree n. We denote each monomial $m = \prod_{i=0}^{n-1} x_{i,b_{i,m}}$ of $\mathsf{OP}_k(\mathbf{x})$ with its corresponding bit string $(b_{0,m}, b_{1,m}, \ldots, b_{n-1,m})$. For example, the set of 4 bit strings denoting the 4 monomials of $\mathsf{OP}_1(\mathbf{x})$ are given by the set ONES.

By definition, these bit strings satisfy Equation 2.8. The polynomial $\mathsf{OP}_k(\mathbf{x})$ also has an interesting combinatorial property. To understand this, let us look closely at $\mathsf{OP}_k(\mathbf{x})$ for $k = 2, k = 3$.

Example 2.9. *Each monomial m of $\mathsf{OP}_2(\mathbf{x})$ is denoted by a bit string $(b_{0,m}, b_{1,m}, \ldots, b_{8,m})$. However, instead of writing it as a long bit string of length $3^2 = 9$, we will write it as 3 rows of 3 bits each, or a 3×3 matrix.*

$$\begin{pmatrix} b_{00,m} & b_{01,m} & b_{02,m} \\ b_{10,m} & b_{11,m} & b_{12,m} \\ b_{20,m} & b_{21,m} & b_{22,m} \end{pmatrix}.$$

Here, the indices i in $b_{i,m}$ are written in ternary. $\mathsf{OP}_2(\mathbf{x})$ has exactly 16 monomials, denoted by the following 16 matrices.

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

The reason behind viewing these bit strings as 3×3 matrices is the following nice observation. These are precisely the 16 different 3×3 matrices with 0-1 entries, each of whose rows and columns have an odd number of ones.

For $k = 3$, each monomial of $\mathsf{OP}_3(\mathbf{x})$ is a bit string of length $3^3 = 27$. We may visualize each bit string

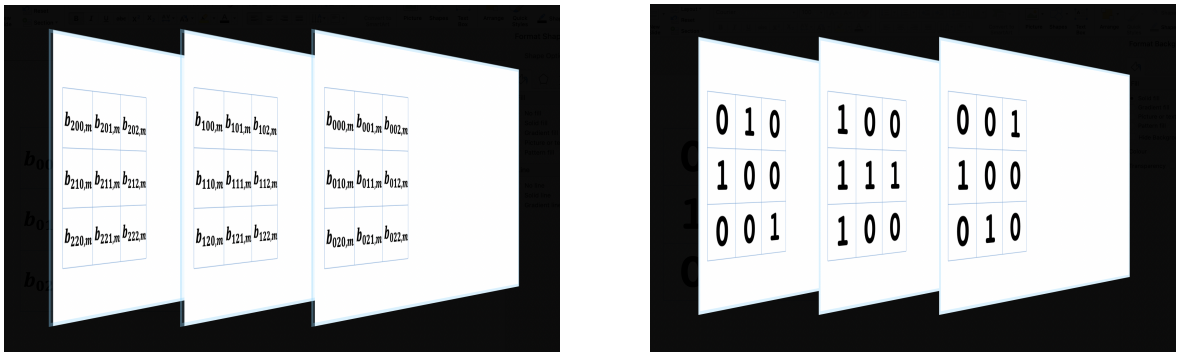

Figure 1: (Left) A 3-dimensional illustration of the bit string $(b_{0,m}, b_{1,m}, \ldots, b_{26,m})$ denoting a monomial m of $\mathsf{OP}_3(\mathbf{x})$. (Right) One such monomial, satisfying Equation 2.8, and also property † below.

as a set of three 3×3 *matrices*

$$\begin{pmatrix} b_{000,m} & b_{001,m} & b_{002,m} \\ b_{010,m} & b_{011,m} & b_{012,m} \\ b_{020,m} & b_{021,m} & b_{022,m} \end{pmatrix} \begin{pmatrix} b_{100,m} & b_{101,m} & b_{102,m} \\ b_{110,m} & b_{111,m} & b_{112,m} \\ b_{120,m} & b_{121,m} & b_{122,m} \end{pmatrix} \begin{pmatrix} b_{200,m} & b_{201,m} & b_{202,m} \\ b_{210,m} & b_{211,m} & b_{212,m} \\ b_{220,m} & b_{221,m} & b_{222,m} \end{pmatrix}$$

written out on three separate sheets of paper, stacked one behind the other (Figure 1, left). As before, the indices i in $b_{i,m}$ are written in ternary. $\mathsf{OP}_3(\mathbf{x})$ *has exactly 256 monomials. These are precisely the 256 different bit strings of length 27 that satisfy the following property (†). It is easy to check that the given example (Figure 1, right) is one such bit string.*

† *Every set of three bits that share a row on the same sheet, or share a column on the same sheet, or share the same position in the matrix on their respective sheets, has an odd number of ones.* ◊

The Overlapping Parities family. The polynomials OP_k naturally define a family where the nth polynomial is OP_k for $k = \lfloor \log_3(n) \rfloor$. We simply use $\{\mathsf{OP}_k\}$ to refer to this family to avoid cumbersome notation.

As stated earlier, OP_1 has four monomials corresponding to the strings $\{001, 010, 100, 111\}$. These strings have the property that if we focus on the first two positions, then we see all the four combinations. The following lemma generalises this observation to higher values of k, and gives an exact count for the sparsity of OP_k.

Lemma 2.10. *The sparsity of* $\mathsf{OP}_k(\mathbf{x})$ *is* 2^{2^k}.

Proof. We will prove this using induction on k. For $k = 1$, it is easy to check that

$$\mathsf{OP}_1(\mathbf{x}) = x_{0,0}x_{1,0}x_{2,1} + x_{0,0}x_{1,1}x_{2,0} + x_{0,1}x_{1,0}x_{2,0} + x_{0,1}x_{1,1}x_{2,1},$$

and so $\mathsf{OP}_1(\mathbf{x})$ has sparsity $4 = 2^{2^1}$.

For the induction hypothesis, we choose $k_0 > 1$ and assume that for every $k = k_0 - 1$, the sparsity of $\mathsf{OP}_k(\mathbf{x})$ is at most 2^{2^k}. For the induction step, let us fix $k = k_0$.

Let $\mathcal{M}_{\mathsf{OP}_k}$ denote the support of $\mathsf{OP}_k(\mathbf{x})$. We then claim that the map $\sigma : \mathcal{M}_{\mathsf{OP}_{k-1}} \times \mathcal{M}_{\mathsf{OP}_{k-1}} \to \mathcal{M}_{\mathsf{OP}_k}$, defined as follows, is a bijection.

$$\sigma\left(\prod_{i=0}^{k-2} x_{i,b_i^{(1)}}, \prod_{i=0}^{k-2} x_{i,b_i^{(2)}}\right) = \prod_{i=0}^{k-2} x_{i,b_i^{(1)}} \cdot \prod_{i=0}^{k-2} x_{3^{k-1}+i,b_i^{(2)}} \cdot \prod_{i=0}^{k-2} x_{2\cdot 3^{k-1}+i,b_i}$$

where for every $i \in \{0,1,\ldots,k-2\}$,

$$b_i = b_i^{(1)} \oplus b_i^{(2)} \oplus 1.$$

Firstly, we need to show that for arbitrary $m_1 = \prod_{i=0}^{k-2} x_{i,b_i^{(1)}}$, $m_2 = \prod_{i=0}^{k-2} x_{i,b_i^{(2)}} \in \mathcal{M}_{\mathsf{OP}_{k-1}}$, $m = \sigma(m_1, m_2)$ in indeed an element in $\mathcal{M}_{\mathsf{OP}_k}$. Note that, if we show $m_3 \in \mathcal{M}_{\mathsf{OP}_{k-1}}$, then we would be done[4]. This is what we show now.

Let $i_1, i_2, i_3 \in \{0,\ldots,3^k-1\}$ be distinct and $T_j(i_1) = T_j(i_2) = T_j(i_3)$ for all but exactly one $j \in \{0,\ldots,k-2\}$. Then,

$$\begin{aligned} b_{i_1} \oplus b_{i_2} \oplus b_{i_3} &= \left(b_{i_1}^{(1)} \oplus b_{i_1}^{(2)} \oplus 1\right) \oplus \left(b_{i_2}^{(1)} \oplus b_{i_2}^{(2)} \oplus 1\right) \oplus \left(b_{i_3}^{(1)} \oplus b_{i_3}^{(2)} \oplus 1\right) \\ &= \left(b_{i_1}^{(1)} \oplus b_{i_2}^{(1)} \oplus b_{i_3}^{(1)}\right) \oplus \left(b_{i_1}^{(2)} \oplus b_{i_2}^{(2)} \oplus b_{i_3}^{(2)}\right) \oplus 1 = 1 \oplus 1 \oplus 1 = 1. \end{aligned}$$

Therefore, $m_3 \in \mathcal{M}_{\mathsf{OP}_{k-1}}$, and hence $m \in \mathcal{M}_{\mathsf{OP}_k}$.

Next, we note that σ is clearly injective. Conversely, suppose $m \in \mathcal{M}_{\mathsf{OP}_k}$. Then

$$m = \prod_{i=0}^{k-2} x_{i,b_i^{(1)}} \cdot \prod_{i=0}^{k-2} x_{3^{k-1}+i,b_i^{(2)}} \cdot \prod_{i=0}^{k-2} x_{2\cdot 3^{k-1}+i,b_i}.$$

Then because of the conditions m satisfies (Equation 2.8), for

$$m_1 = \prod_{i=0}^{k-2} x_{i,b_i^{(1)}} \quad \text{and} \quad m_2 = \prod_{i=0}^{k-2} x_{i,b_i^{(2)}},$$

it must be the case that $m_1, m_2 \in \mathcal{M}_{\mathsf{OP}_{k-1}}$ and for every $i \in [k-2]$, b_i must satisfy the property described in the definition of σ.

σ is therefore a bijection, and so $|\mathcal{M}_{\mathsf{OP}_k}| = |\mathcal{M}_{\mathsf{OP}_{k-1}}|^2$. Now by the induction hypothesis, $|\mathcal{M}_{\mathsf{OP}_{k-1}}| = 2^{2^{k-1}}$. Hence $|\mathcal{M}_{\mathsf{OP}_k}| = 2^{2^k}$, completing the proof. □

In fact an even stronger property is true for the support of OP_1, that *any two* of its positions

[4]The last condition that m has to satisfy so that $m \in \mathcal{M}_{\mathsf{OP}_k}$ (for three distinct i_1, i_2, i_3, $T_j(i_1) = T_j(i_2) = T_j(i_3)$ for all $j \in \{0,\ldots,k-2\}$ but $T_{k-1}(i_1) \neq T_{k-1}(i_2) \neq T_{k-1}(i_3) \neq T_{k-1}(i_1)$) is true by the definition of m_3.

appear in all the four combinations. In other words, we have exactly two "degrees of freedom". Further any two bits are "independent" and they together fully determine the third bit. As we shall see in subsection 3.2, a version of this property is also exhibited by OP_k in general and this is a crucial component of our lower bound proof.

3 The Monotone VNP Lower Bound

In this section we will prove the following.

Theorem 3.1. *Any* mVNP *circuit computing* $\mathsf{OP}_{\log_3 n}(\mathbf{x})$ *has size* $2^{\Omega(n^{0.63})}$.

For this, we will use the following result by Yehudayoff [Yeh19].

Definition 3.2 ([Yeh19])**.** *Two polynomials* $p(\mathbf{x})$ *and* $q(\mathbf{x})$ *are equivalent if the monomials that appear in both are the same. That is, or every monomial* α, $\alpha \in p(\mathbf{x}) \Leftrightarrow \alpha \in q(\mathbf{x})$. ◊

Theorem 3.3 ([Yeh19])**.** *If a monotone circuit-size lower bound for* $q(\mathbf{x})$ *holds also for all polynomials that are equivalent to* $q(\mathbf{x})$ *then it also holds for every* mVNP *circuit computing* $q(\mathbf{x})$.

By Theorem 3.3, the following theorem is enough to prove Theorem 3.1

Theorem 3.4. *Every monotone algebraic circuit computing* $\mathsf{OP}_{\log_3 n}(\mathbf{x})$ *has size* $2^{\Omega(n^{\log_3 2})} \geq 2^{\Omega(n^{0.63})}$.

So we now move on to proving Theorem 3.4. For that, we need the following two statements (Lemma 3.5, Lemma 3.6), along with Lemma 2.10 that talks about the sparsity of OP_k. The first lemma is frequently used in lower bounds against monotone circuits and so we only state it here without a proof.

Lemma 3.5 (for example, [Sap15])**.** *Let* f *be a degree* d *polynomial computed by a monotone circuit of size* s. *Then,* f *can be written as*

$$f = \sum_{j=1}^{s} g_j \cdot h_j,$$

where all the g_j*s and* h_j*s are monotone polynomials satisfying the following properties.*

1. *For each* $j \in [s]$, $d/3 \leq \deg(g_j), \deg(h_j) < 2d/3$;
2. *For each* $j \in [s]$, $\mathrm{supp}(g_j) \times \mathrm{supp}(h_j) \subseteq \mathrm{supp}(f)$. □

The second statement is non-trivial to show and is our main contribution.

Lemma 3.6. *If* $\mathsf{OP}_k(\mathbf{x}) = \sum_{j=1}^{s} g_j \cdot h_j$ *where the* g_j*s and* h_j*s satisfy the properties described in Lemma 3.5. Then for every* $j \in [s]$,

$$\textit{sparsity of } g_j \times \textit{sparsity of } h_j \leq 2^{2^{k-1}}.$$

Before proving Lemma 3.6, let us first use it to complete the proof of our main theorem.

Proof of Theorem 3.4. Suppose $\mathsf{OP}_{\log_3 n}(\mathbf{x})$ can be computed by a monotone algebraic circuit of size s. Then, by Lemma 3.5,

$$\mathsf{OP}_{\log_3 n}(\mathbf{x}) = \sum_{i=1}^{s} g_i \cdot h_i,$$

where the g_is and h_is satisfy the properties mentioned in Lemma 3.5. Therefore, by Lemma 3.6, for every $i \in [s]$,

$$\text{sparsity of } g_i \times \text{sparsity of } h_i \leq 2^{2^{k-1}},$$

where $k = \log_3 n$. However, by Lemma 2.10, the sparsity of $\mathsf{OP}_k(\mathbf{x})$ is 2^{2^k}. This implies that

$$2^{2^k} = \text{sparsity of } \mathsf{OP}_k(\mathbf{x}) \leq s \cdot 2^{2^{k-1}} \implies s \geq 2^{2^{k-1}}.$$

Thus,

$$s \geq 2^{2^{(\log_3 n)-1}} = 2^{\left(2^{\log_3 n}\right)/2} = 2^{\left(n^{\log_3 2}\right)/2} = 2^{\Omega(n^{\log_3 2})}.$$

This completes the proof. □

The rest of this section is devoted to the proof of Lemma 3.6. For that, we will need the concept of the Assignment Fixing Tree (or AFT), which we define as follows.

3.1 The Assignment Fixing Tree (AFT)

The Assignment Fixing Tree of height k (denoted as AFT_k) is a complete ternary tree of height[5] k. Thus AFT_k has exactly 3^k leaves. For every $i \in \{0, 1, \ldots, 3^k - 1\}$, the i-th leaf (from the left) is labelled by a bit $b(i)$. Every other node is labelled by the concatenation of the labels of its children from left to right. That is, if a node has children 010, 110 and 101, then its label is 010110101. More precisely, if $\mathbf{b}_N$ is the label of a non-leaf node N with children A, B and C from left to right, then

$$\mathbf{b}_N = \mathbf{b}_A \mathbf{b}_B \mathbf{b}_C. \tag{3.7}$$

Note that fixing the labels of the all the leaf nodes automatically fixes the label of every node in the tree, right up to the root. Also note that the label of every non-leaf node contains (as a substring) the label of every node in its subtree.

[5] Leaves are at height 0, root is at height k.

Valid sssignments. An assignment to the labels of all the leaf nodes, say $\mathbf{b} = \{b(i)\}_{i=0}^{3^k-1}$, is said to be valid if and only if $\prod_{i=0}^{3^k-1} x_{i,b(i)} \in \mathcal{M}_{\mathsf{OP}_k}$. Furthermore, an assignment to a non-leaf node N is said to be valid if and only if the corresponding assignment to the labels of all the leaf nodes in the subtree rooted at N can be extended to a valid assignment to the rest of the leaf nodes.

Colouring the AFT. In a coloured AFT, each leaf of the tree is coloured either blue or red. Then, we iteratively colour every other node in the tree as the majority of its children's colours (each node has exactly 3 children, so there is always a *strict* majority). We exhibit an interesting example of a colouring of AFT_3 in Figure 2, wherein even though most of the leaves of the tree are coloured blue, the root of the tree gets coloured red.

Scoring the AFT. There are two ways of "scoring" the AFT, one with respect to the red leaves and one with respect to the blue leaves. For every node N, we denote its score with respect to the red leaves by $\mathrm{rscore}(N)$ and that with respect to the blue leaves by $\mathrm{bscore}(N)$. If N is a leaf, then

$$\mathrm{rscore}(N) = \begin{cases} 0 & \text{if } N \text{ is coloured blue;} \\ 1 & \text{if } N \text{ is coloured red.} \end{cases} \qquad \mathrm{bscore}(N) = \begin{cases} 1 & \text{if } N \text{ is coloured blue;} \\ 0 & \text{if } N \text{ is coloured red.} \end{cases}$$

If N is a non-leaf node and A, B and C are its three children (from left to right), then

$$\mathrm{rscore}(N) = \mathrm{rscore}(A) + \mathrm{rscore}(B) + \mathrm{rscore}(C) - \min\{\mathrm{rscore}(A), \mathrm{rscore}(B), \mathrm{rscore}(C)\},$$
$$\mathrm{bscore}(N) = \mathrm{bscore}(A) + \mathrm{bscore}(B) + \mathrm{bscore}(C) - \min\{\mathrm{bscore}(A), \mathrm{bscore}(B), \mathrm{bscore}(C)\}.$$

Alternatively, the rscore of a node is the sum of the two maximum rscores of its children, and the bscore of a node is the sum of the two maximum bscores of its children.

Thus, the maximum possible rscore of a node at level 1 is 2, even if all its three children are coloured red. Similarly, the maximum possible rscore of a node at level 2 is 4, even if all its three children have the maximum possible score of 2. Generalizing this, the following is easy to see.

Fact 3.8. *Let N be a node at level i in an AFT of height $k \geq i$. Then $\mathrm{rscore}(N) \leq 2^i$ and $\mathrm{bscore}(N) \leq 2^i$.*

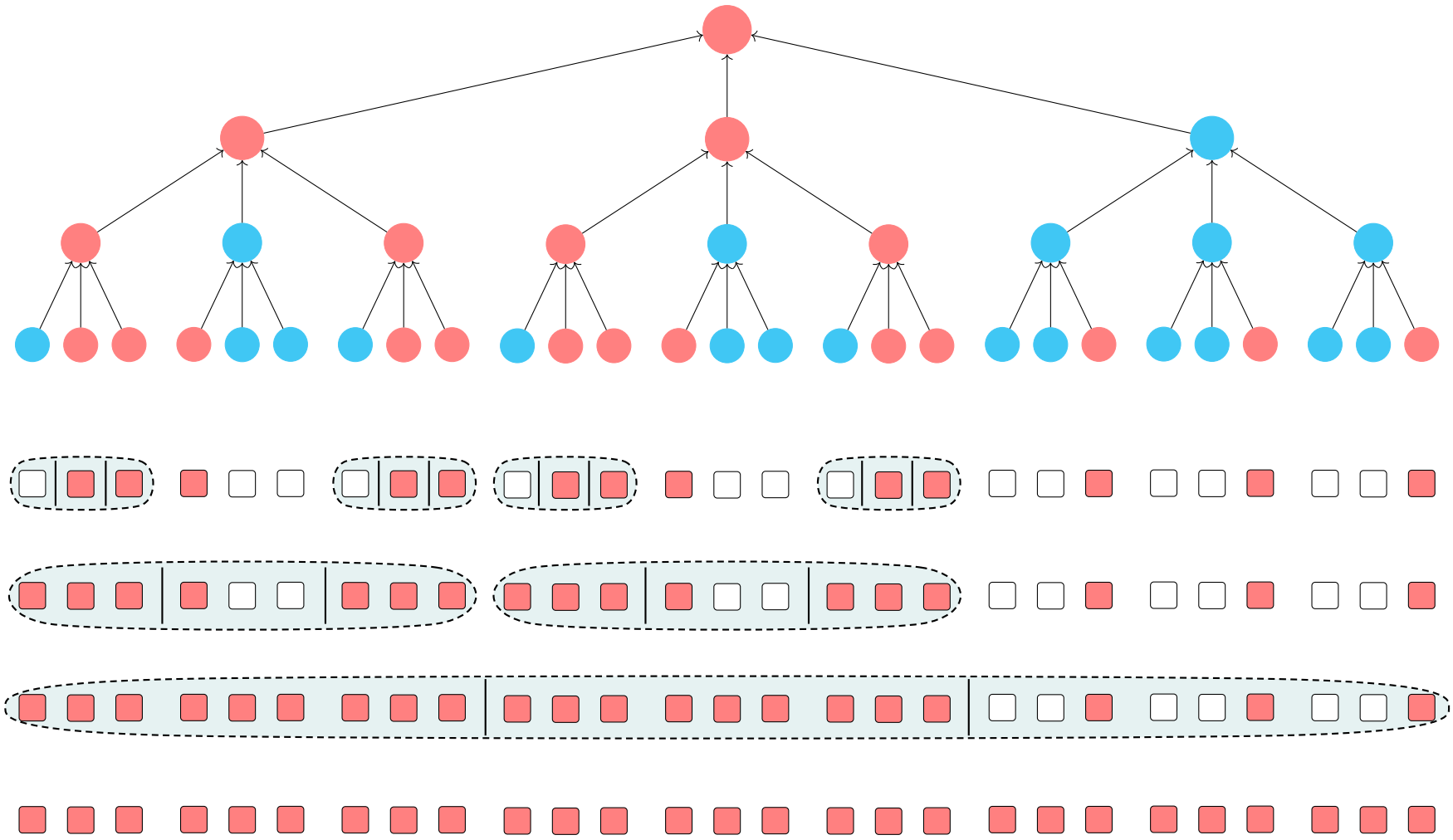

Figure 2: A colouring of AFT_3. The root of this tree is red, and so the blue entries can be completely determined by looking at the red entries. This can be seen by first considering groups of $3 = 1 + 1 + 1$, then groups of $9 = 3 + 3 + 3$, and finally $27 = 9 + 9 + 9$.

3.2 Properties of the AFT

The AFT has some very colourful and useful properties, one of which is described formally by the following lemma.

Lemma 3.9. *Suppose you are given a coloured AFT of height k.*

If the labels of all the blue leaves in AFT_k are assigned (or fixed) and none of the red leaves are assigned, then the number of possible valid assignments to the label of any node, say N, at height ht (in the subtree $\mathsf{AFT}_{\mathsf{ht}}$ rooted at N) is at most $2^{2^{\mathsf{ht}}-\mathrm{bscore}(N)}$.

Analogously, if the labels of all the red leaves in AFT_k are fixed and none of the blue leaves are assigned, then the number of possible valid assignments to the label of any node, say N, at height ht (in the subtree $\mathsf{AFT}_{\mathsf{ht}}$ rooted at N) is at most $2^{2^{\mathsf{ht}}-\mathrm{rscore}(N)}$.

Proof. We only prove the case for the blue leaves being assigned to a fixed value, since the proof of the other case is identical. So suppose the label (or bit) corresponding to each blue leaf is fixed. We prove our lemma using induction on ht.

Base Case: For $\mathsf{ht} = 0$, first suppose N is a blue node at height ht. Then N is essentially a blue leaf, and so $\mathrm{bscore}(N) = 1$. By our assumption, the label of N has already been assigned to a fixed value (either 0 or 1). Therefore, the maximum possible number of valid assignments to the label of N is

$$1 = 2^0 = 2^{2^0-1} = 2^{2^{\mathsf{ht}}-\mathrm{bscore}(N)}.$$

On the other hand, suppose N is a red node at height $\mathsf{ht} = 0$. Then $\mathrm{bscore}(N) = 0$ and by our assumption, its label has not been assigned a value. Therefore, both 0 and 1 are valid assignments. Thus, the maximum possible number of valid assignments to the label of N is

$$2 = 2^1 = 2^{2^0-0} = 2^{2^{\mathsf{ht}}-\mathrm{bscore}(N)}.$$

Induction Hypothesis: Let ht_0 be a positive integer. Assume that the maximum number of possible valid assignments to the label of every node at height $\mathsf{ht}_0 - 1$ is at most $2^{\left(2^{(\mathsf{ht}_0-1)}-\mathrm{bscore}(N)\right)}$.

Induction Step: Now fix $\mathsf{ht} = \mathsf{ht}_0$, and let N be any node at height ht. Assume that A, B, C are its three children such that $\mathrm{bscore}(A) \geq \mathrm{bscore}(B) \geq \mathrm{bscore}(C)$. Then, by definition,

$$\mathrm{bscore}(N) = \mathrm{bscore}(A) + \mathrm{bscore}(B).$$

Furthermore, we claim that the following map, σ, is injective.

$$\sigma : \begin{matrix}\text{Valid assignments}\\ \text{to the label of } N\end{matrix} \to \begin{matrix}\text{Valid assignments}\\ \text{to the label of } A\end{matrix} \times \begin{matrix}\text{Valid assignments}\\ \text{to the label of } B\end{matrix}$$

is defined as

$$\sigma(\mathbf{b}_N) = (\mathbf{b}_A, \mathbf{b}_B).$$

where $\mathbf{b}_N$ is as defined in Equation 3.7. Indeed, since $\mathbf{b}_N$ is a valid assignment,

$$\sigma(\mathbf{b}_N) = \sigma(\mathbf{b}'_N) \implies (\mathbf{b}_A, \mathbf{b}_B) = (\mathbf{b}'_A, \mathbf{b}'_B) \implies \mathbf{b}_C = \mathbf{b}'_C \implies \mathbf{b}_N = \mathbf{b}'_N.$$

Therefore,

$$\left|\begin{matrix}\text{Valid assignments}\\ \text{to the label of } N\end{matrix}\right| \leq \left|\begin{matrix}\text{Valid assignments}\\ \text{to the label of } A\end{matrix}\right| \times \left|\begin{matrix}\text{Valid assignments}\\ \text{to the label of } B\end{matrix}\right|$$

and so, by the induction hypothesis,

$$\begin{aligned}\left|\begin{matrix}\text{Valid assignments}\\ \text{to the label of } N\end{matrix}\right| &\leq 2^{2^{\mathsf{ht}-1}-\mathsf{bscore}(A)} \times 2^{2^{\mathsf{ht}-1}-\mathsf{bscore}(A)} \\ &= 2^{(2^{\mathsf{ht}-1}+2^{\mathsf{ht}-1})-(\mathsf{bscore}(A)+\mathsf{bscore}(B))} = 2^{2^{\mathsf{ht}}-\mathsf{bscore}(N)}\end{aligned}$$

This completes the proof. □

Claim 3.10. *If the root of* AFT_k, $\mathsf{root}(\mathsf{AFT}_k)$, *is coloured* red, *then* $\mathsf{rscore}(\mathsf{root}(\mathsf{AFT}_k)) = 2^k$.

Proof. To show this, we claim that given $\mathsf{root}(\mathsf{AFT}_k)$ is coloured red and N is a red node at level i, then the rscore of N is 2^i. This can be easily observed using induction on i.

Base Case: For $i = 0$, the statement is true by definition.

Induction Hypothesis: For $i_0 \geq 1$, assume that if N is a red node at level $i = i_0 - 1$, then

$$\mathsf{rscore}(N) = 2^i.$$

Induction Step: Fix $i = i_0$. Let N be any red node at level i, and A, B, C be its children. Now at least two out of A, B, C must be red. Without loss of generality, let A, B be red. Then, by the induction hypothesis (since A, B, C are at level $i - 1$),

$$\mathsf{rscore}(A), \mathsf{rscore}(B) = 2^{i-1}.$$

Further, note that $\mathsf{rscore}(C) \leq 2^{i-1}$ (Fact 3.8). Therefore, by definition,

$$\mathsf{rscore}(N) = \mathsf{rscore}(A) + \mathsf{rscore}(B) = 2^{i-1} + 2^{i-1} = 2^i.$$

This completes the proof. □

3.3 Consequences of colouring the AFT

Lemma 3.11. *Assume that the root of* AFT_k *is coloured* red. *Further, let* $t_{-1} = -1$, *and for integers* $k \geq 0$ *and* $m \in \{0, 1, \ldots, 3^k - 1\}$, *define*

$$t_m = \sum_{j=0}^{k-1} B_j(m) 2^{i_j(m)} 3^j$$

where $B(m) = B_{k-1}(m)B_{k-2}(m)B_{k-3}(m)\cdots B_1(m)B_0(m)$ *is the binary representation of* m *and* $i_j(m) = \left(\sum_{l=j}^{k-1} B_l(m)\right) - 1$.

Then for every $m \in \{0, 1, \ldots, 3^k - 1\}$ *and every* $t_{m-1} < \ell_m \leq t_m$, *if a node (say* N*) in* AFT_k *has* ℓ_m *leaves coloured* blue, *then the* bscore *of* N *is at least* m.

To prove this, we will need the following claim.

Claim 3.12. *For any set of integers* a, b, m *with* $m \geq 1$ *and* $a \geq b \geq 0$,

$$a + b \leq m \implies t_a + 2t_b \leq t_m$$

Proof. We prove the claim using induction on m.

Base Case: For $m = 1$, either $a = b = 0$ or $a = 1$ and $b = 0$. Thus $t_a \leq 1$ and $t_b = 0$. This implies that $t_a + 2t_b \leq 1 = t_m$.

Induction Hypothesis: For $m_0 > 1$, assume that for every $m \leq m_0 - 1$,

$$m > a \geq b \geq 0 \quad \text{and} \quad a + b \leq m \implies t_a + 2t_b \leq t_m.$$

Induction Step: Fix $m = m_0$. Let the position of the most significant bit (MSB) of a written in binary be k_a. That is if the binary representation of a is $B_{k-1}(a)\cdots B_0(a)$, then $B_{k_a}(a) = 1$ and $B_i(a) = 0$ for every $i > k_a$. Similarly, let the positions of the MSB of b and m be k_b and k_m respectively.

Case 1: $k_a = k_b$

Note that in this case, $k_m = k_a + 1$. Let $a' = a - 2^{k_a}$, $b' = b - 2^{k_a}$ and $m' = m - 2^{k_a+1}$.
Then $t_a = 3^{k_a} + 2t_{a'}$, $t_b = 3^{k_a} + 2t_{b'}$ and $t_m = 3^{k_a+1} + 2t_{m'}$.
Clearly,

$$a' \geq b' \geq 0 \text{ and } a' + b' \leq m'.$$

Since $m' < m$, by induction hypothesis, $t_{a'} + 2t_{b'} \leq t_{m'}$. Therefore,

$$t_a + 2t_b = (3^{k_a} + 2t_{a'}) + (2 \cdot 3^{k_a} + 4t_{b'}) = 3^{k_a+1} + 2(t_{a'} + 2t_{b'}) \leq 3^{k_a+1} + 2t_{m'} = t_m.$$

Case 2: $k_a > k_b$ and $k_m = k_a$

Let $a' = a - 2^{k_a}$, $b' = b$ and $m' = m - 2^{k_a}$.

Then $t_a = 3^{k_a} + 2t_{a'}$, $t_b = t_{b'}$ and $t_m = 3^{k_a} + 2t_{m'}$.

Clearly, $a' + b' \leq m'$. Further since m' < m, by the induction hypothesis,

$$a' \geq b' \geq 0 \implies t_{a'} + 2t_{b'} \leq t_{m'} \quad \text{and} \quad b' \geq a' \geq 0 \implies t_{b'} + 2t_{a'} \leq t_{m'}.$$

In either case, it implies that $t_{a'} + t_{b'} \leq t_{m'}$. Therefore,

$$t_a + 2t_b = (3^{k_a} + 2t_{a'}) + 2t_{b'} = 3^{k_a} + 2(t_{a'} + t_{b'}) \leq 3^{k_a} + 2t_{m'} = t_m.$$

Case 3: $k_a > k_b$ and $k_m = k_a + 1$

Note that in this case, $B_{k_a+1}(m) = 1$ and $B_{k_a}(m) = 0$.

Let $a' = a - 2^{k_a}$, $b' = b$ and $m' = m - 2^{k_a}$.

Then $t_a = 3^{k_a} + 2t_{a'}$, $t_b = t_{b'}$ and $t_m = 3^{k_a+1} - 3^{k_a} + t_{m'} = 2 \cdot 3^{k_a} + t_{m'} = 3^{k_a} + (3^{k_a} + t_{m'})$.

Clearly, $a' + b' \leq m'$. Further since m' < m, by the induction hypothesis,

$$a' \geq b' \geq 0 \implies t_{a'} + 2t_{b'} \leq t_{m'} \quad \text{and} \quad b' \geq a' \geq 0 \implies t_{b'} + 2t_{a'} \leq t_{m'}.$$

Also, since $a', b' < 2^{k_a}$, we have that $t_{a'}, t_{b'} < 3^{k_a}$. Thus, in either case, $2(t_{a'} + t_{b'}) \leq 3^{k_a} + t_{m'}$. Therefore,

$$t_a + 2t_b = (3^{k_a} + 2t_{a'}) + 2t_{b'} = 3^{k_a} + 2(t_{a'} + t_{b'}) \leq 3^{k_a} + (3^{k_a} + t_{m'}) = t_m.$$

This completes the proof of the claim, since these are the only possible cases. □

We now complete the proof of Lemma 3.11.

Proof of Lemma 3.11. We prove the lemma using induction on n.

Base Case: For $n = 0$, $t_{n-1} = -1$ and $t_n = 0$. Therefore it must be the case that $\ell_n = 0$, and hence the score of the node is at least $0 = n$. Similarly for $n = 1$, $t_{n-1} = 0$ and $t_n = 1$, implying that $\ell_n = 1$. The score of the node therefore is at least $1 = n$.

Induction Hypothesis: For $n_0 \in \{2, \ldots, 3^k - 1\}$ assume that for every $n \in \{0, \ldots, n_0 - 1\}$, if a node N in the AFT has ℓ_n leaves scored 1, for $\ell_n \in (t_{n-1}, t_n]$, then bscore$(N) \geq n$.

Induction Step: fix $n = n_0$ and assume that there is a node whose score is at most $n - 1$ even though the number of leaves it has with score 1 is in the range $(t_{n-1}, t_n]$. Let N be such a node whose level is minimal. That is, no node at a level lower than that of N has this property. We prove that this will lead to a contradiction.

Indeed, let its children be A, B, C and have ℓ_a, ℓ_b, ℓ_c leaves scored 1 respectively. Further, let $\ell_a \geq \ell_b \geq \ell_c$. Now $\ell_a + \ell_b + \ell_c = \ell_n$ and so $\ell_a + 2\ell_b \geq \ell_n > t_{n-1}$. We also assume that $t_{a-1} < \ell_a \leq t_a$ and $t_{b-1} < \ell_b \leq t_b$. Therefore,

$$t_a + 2t_b \geq \ell_a + 2\ell_b > t_{n-1}. \tag{3.13}$$

Now since $\text{bscore}(N) \leq n-1$, it must be that $\text{bscore}(A), \text{bscore}(B), \text{bscore}(C) \leq n-1$. Thus, by our choice of N, we have $\ell_a, \ell_b, \ell_c \leq t_{n-1}$. This implies that $a, b, c \leq n-1$. By the induction hypothesis,

$$\text{bscore}(A) \geq a, \text{bscore}(B) \geq b \text{ and } \text{bscore}(C) \geq c.$$

Further since $\ell_a \geq \ell_b \geq \ell_c$, we have $a \geq b \geq c \geq 0$. Therefore, $\text{bscore}(N) = \text{bscore}(A) + \text{bscore}(B)$ which implies that $\text{bscore}(N) \geq a + b$. Using Equation 3.13 and our assumption, we have integers a, b such that $a \geq b \geq 0$ with

$$a + b \leq n-1 \quad \text{but} \quad t_a + 2t_b > t_{n-1}.$$

Since $n \geq 2$, $n-1 \geq 1$ and therefore this contradicts Claim 3.12. Thus our assumption must be incorrect and $\text{bscore}(N) \geq n$.

This completes the proof. □

3.4 Wrapping up the proof

Finally, let us recall the statement of Lemma 3.6 and prove it.

Lemma 3.6. *If $\mathsf{OP}_k(\mathbf{x}) = \sum_{j=1}^{s} g_j \cdot h_j$ where the g_js and h_js satisfy the properties described in Lemma 3.5. Then for every $j \in [s]$,*

$$\textit{sparsity of } g_j \times \textit{sparsity of } h_j \leq 2^{2^{k-1}}.$$

Proof. Suppose $\mathsf{OP}_k(\mathbf{x}) = \sum_{j=1}^{s} g_j \cdot h_j$ where the g_js and h_js satisfy the two properties described in Lemma 3.5. Arbitrarily fix $j \in [s]$ and define

$$\mathcal{I}(g_j) = \{i \,:\, x_{i,b_i} \in \text{var}(g_j) \text{ for some } b_i \in \{0,1\}\}$$

where $\text{var}(g_j)$ denotes the variables g_j depends on. $\mathcal{I}(h_j)$ is defined similarly. By Lemma 3.5, note that the following must be true.

1. Every monomial in g_j must be of the form $\prod_{i \in \mathcal{I}(g_j)} x_{i,b_i}$ where, for every $i \in \mathcal{I}(g_j)$, $b_i \in \{0,1\}$.

2. $d/3 \leq \left|\mathcal{I}(g_j)\right|, \left|\mathcal{I}(h_j)\right| \leq 2d/3$.

3. $\mathcal{I}(g_j) \sqcup \mathcal{I}(h_j) = \{0, 1, \ldots, 3^k - 1\}$.

Recall that the degree of every monomial of $\mathsf{OP}_k(\mathbf{x})$ is $d = 3^k$. So, we get

$$3^k/3 \leq \left|\mathcal{I}(g_j)\right|, \left|\mathcal{I}(h_j)\right| \leq 2 \cdot 3^k/3.$$

We colour every leaf of AFT_k as follows ($\mathrm{colour}(i)$ denotes the colour of the i-th leaf from the left).

$$\text{for every } i \in \left\{0, \ldots, 3^k - 1\right\}, \quad \text{we set} \quad \mathrm{colour}(i) = \begin{cases} \text{red} & \text{if } i \in \mathcal{I}(g_j) \\ \text{blue} & \text{if } i \in \mathcal{I}(h_j) \end{cases}$$

Note that this colouring implies that AFT_k has at least 3^{k-1} red leaves, and at least 3^{k-1} blue leaves. We now colour the rest of the nodes in the tree as described earlier, and claim the following.

If $\mathrm{colour}(\mathsf{root}(\mathsf{AFT}_k)) = \text{red}$, then

sparsity of h_j is 1 and sparsity of g_j is at most $2^{2^{k-1}}$,

and $\mathrm{colour}(\mathsf{root}(\mathsf{AFT}_k)) = \text{blue}$ then

sparsity of g_j is 1 and sparsity of h_j is at most $2^{2^{k-1}}$.

We only prove the statement for the colour of the root being red, since the other case can be proved in an identical manner. By Claim 3.10,

$$\text{rscore}(\mathsf{root}(\mathsf{AFT}_k)) = 2^k. \tag{3.14}$$

Further, since AFT_k has at least $3^{k-1} = t_{2^{k-1}}$ blue leaves, by Lemma 3.11,

$$\text{bscore}(\mathsf{root}(\mathsf{AFT}_k)) \geq 2^{k-1}. \tag{3.15}$$

We first show that the sparsity of h_j is 1. Consider some monomial in g_j, say $m = \prod_{i \in \mathcal{I}(g_j)} x_{i,b_i}$. We then assign values to the label of every red leaf of AFT_k as follows.

For every $i \in \mathcal{I}(g_j)$, $b(i) = b_i$.

Then, by Lemma 3.9, the number of valid assignments to $\mathsf{root}(\mathsf{AFT}_k)$ is $2^{2^k - \text{rscore}(\mathsf{root}(\mathsf{AFT}_k))}$ which by Equation 3.14 is $2^{2^k - 2^k} = 2^0 = 1$. That is, the number of valid assignments to its blue leaves is exactly 1. Or in other words, there is a unique assignment to the labels $\{b(i)\}_{i \in \mathcal{I}(h_j)}$ such that $\mathbf{b}$ is

a valid assignment to the leaves of AFT_k. Note that this is equivalent to saying that there is exactly one m' (namely $\prod_{i\in\mathcal{I}(h_j)} x_{i,b(i)}$) such that $m \cdot m' \in \mathcal{M}_{\mathsf{OP}_k}$. This shows that the sparsity of h_j is 1.

Next we show that the sparsity of g_j is at most $2^{2^{k-1}}$. Consider some monomial in h_j, say $m = \prod_{i\in\mathcal{I}(g_j)} x_{i,b_i}$. We then assign values to the label of every blue leaf of AFT_k as follows.

For every $i \in \mathcal{I}(h_j)$, $b(i) = b_i$.

Then, by Lemma 3.9, the number of valid assignments to $\mathsf{root}(\mathsf{AFT}_k)$ is $2^{2^k - \mathsf{bscore}(\mathsf{root}(\mathsf{AFT}_k))}$ which by Equation 3.15 is at most $2^{2^k - 2^{k-1}} = 2^{2^{k-1}}$. That is, the number of valid assignments to its red leaves is at most $2^{2^{k-1}}$. Or in other words, there are at most $2^{2^{k-1}}$ assignments to the labels $\{b(i)\}_{i\in\mathcal{I}(g_j)}$ such that $\mathbf{b}$ is a valid assignment to the leaves of AFT_k. Let $\mathcal{B}$ be the set of these assignments. Then note that this is equivalent to saying that there are at most $2^{2^{k-1}}$ monomials m' $\left(\text{namely } \left\{\prod_{i\in\mathcal{I}(h_j)} x_{i,b(i)} \;:\; \mathbf{b} \in \mathcal{B}\right\}\right)$ such that $m \cdot m' \in \mathcal{M}_{\mathsf{OP}_k}$. This shows that the sparsity of g_j is at most $2^{2^{k-1}}$.

Therefore, our claim is true and

$$\text{sparsity of } g_j \times \text{sparsity of } h_j \leq 1 \times 2^{2^{k-1}} = 2^{2^{k-1}},$$

completing the proof. □

4 The Monotone VPSPACE Upper Bound

In this section, we prove mVPSPACE upper bounds for both $\{\mathsf{OP}_k\}$ as well as $\{\mathsf{Perm}_n\}$.

4.1 Upper bound for the Overlapping Parities polynomial family

We first prove the following and note that this together with Theorem 3.1 implies Theorem 1.3.

Theorem 4.1. *There is a* mVPSPACE *circuit of size* $O(n \log n)$ *that computes* $\mathsf{OP}_{\log_3 n}(\mathbf{x})$.

For that we look at the monomials of OP_k as solutions to a system of linear equations.

Monomials as solutions of a linear system

Let $n = 3^k$. Recall that the monomials m of $\mathsf{OP}_k(\mathbf{x}) = \mathsf{OP}_{\log_3 n}(\mathbf{x})$ can be represented as bit strings $(b_{0,m}, b_{1,m}, \ldots, b_{n-1,m})$ satisfying Equation 2.8. It is easy to see that the number of ternary strings $i_1, i_2, i_3 \in \{0, 1, \ldots, 3^k - 1\}$ that satisfy the LHS of Equation 2.8 is exactly $k \cdot 3^{k-1}$. These equations (or constraints) can be written as linear constraints, and thus there are $k \cdot 3^{k-1}$ such constraints[6].

We index these constraints using strings W of length k, that have exactly $(k-1)$ symbols from $\{0, 1, 2\}$ and one $*$ ("do-not-care") symbol. The constraint W applies to the three monomials (or

[6]Even though the rank of this system is much lower, as is evident from Lemma 2.10

three bit strings) whose ternary representations are obtained by replacing the $*$ by 0, 1 and 2. For example, for $k = 4$, the constraint $(1, 2, *, 0)$ applies to the following three bit strings.

$$\begin{aligned} &(1,2,0,0)\\ &(1,2,1,0)\\ &(1,2,2,0)\\ \hline W =&(1,2,*,0) \end{aligned}$$

It is easy to see that if W is a constraint such that $W[p] = *$, and $i_0 < i_1 < i_2$ are the three bit strings that W applies to, then $i_1 = i_0 + 3^p$ and $i_2 = i_0 + 2 \cdot 3^p$.

The mVPSPACE upper bound

Let $W_1, W_2, \ldots, W_{n'}$ be the constraints, where $n' = k \cdot 3^{k-1}$. The mVPSPACE-circuit that we describe starts with all possible 2^{3^k} monomials and then gradually prunes out the unwanted monomials using "correlated projections" of auxiliary variables. Let

$$A_k := \prod_{i=0}^{n-1} A_k(i), \quad \text{where } A_k(i) := (z_{i,0} z_{i,1} \cdots z_{i,k-1})\, x_{i,0} + (y_{i,0} y_{i,1} \cdots y_{i,k-1})\, x_{i,1},\ \forall i \in [\![n]\!].$$

See Figure 3 for an illustration of A_1. Fix an $i \in [\![n]\!]$. We will use the jth pair $(z_{i,j}, y_{i,j})$ for the constraint pertaining to the jth trit of i. We will use the shorthand $\texttt{sel}_{(i,j)\sim b}(g(\ldots))$ for the two "correlated projections" given by $\texttt{fix}_{(y_{i,j}=b)}(\texttt{fix}_{(z_{i,j}=\neg b)}(g(\ldots)))$ (Figure 4). This selects all monomials in g divisible by $x_{i,b}$ with the help of the jth pair of auxiliary variables for the ith position.

Let $Q_0 = A_k$. Further, for each $\ell = 1, 2, \ldots, n'$, define

$$Q_\ell := \sum_{(b_0,b_1,b_2)\in \mathsf{ONES}} \texttt{sel}_{(i_2,p)\sim b_2}\left(\texttt{sel}_{(i_1,p)\sim b_1}\left(\texttt{sel}_{(i_0,p)\sim b_0}\left(Q_{\ell-1}\right)\right)\right), \tag{4.2}$$

where $p : W_\ell[p] = *$, and i_0, $i_1 = i_0 + 3^p$ and $i_2 = i_0 + 2 \cdot 3^p$, are the positions in the constraint W_ℓ. Figure 5 illustrates how Q_ℓ is computed from $Q_{\ell-1}$. The circuit finally outputs $Q_{n'}$.

Correctness. We first show that all monomials in $Q_{n'}$ correspond to bit strings that satisfy all the n' constraints, using the following lemma.

Lemma 4.3. *For each $\ell = 1, 2, \ldots, n'$, if $\mathbf{b} \in \{0,1\}^n$ is a bit string satisfying constraints W_1, W_2, …, W_ℓ, then there is a* unique *monomial in Q_ℓ that is divisible by $\prod_{i\in[\![n]\!]} x_{i,b_i}$, and its coefficient is* 1.

Proof. We shall prove the statement by induction on ℓ.

Base case: For $\ell = 0$, it is easy to see that $Q_0 = A_k$ contains exactly 2^n monomials, each divisible

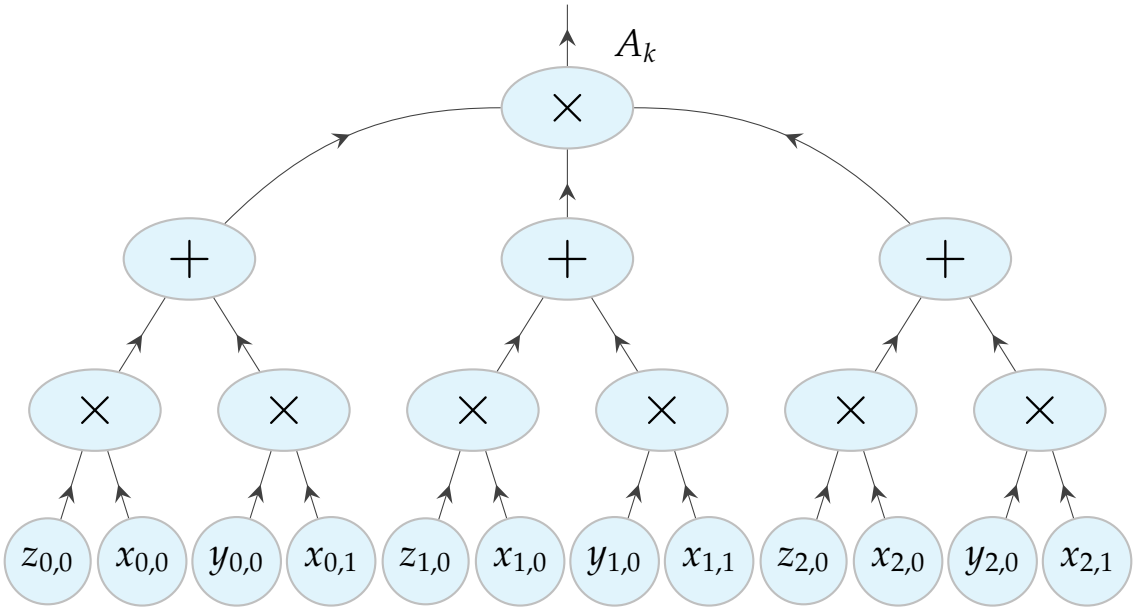

Figure 3: The circuit for A_k for $k = 1$ ($n = 3$), which is the base case (since $Q_0 = A_k$) of Lemma 4.3

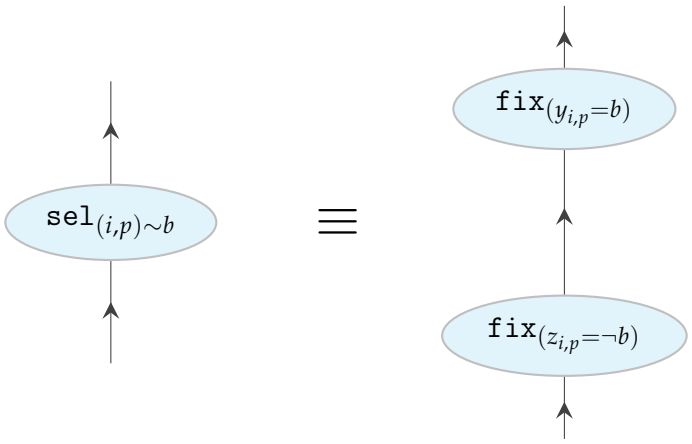

Figure 4: Each select gate $\left(\texttt{sel}_{(i,p)\sim b}\right)$ in Figure 5 first fixes $z_{i,p}$ to $\neg b$, and then fixes $y_{i,p}$ to b

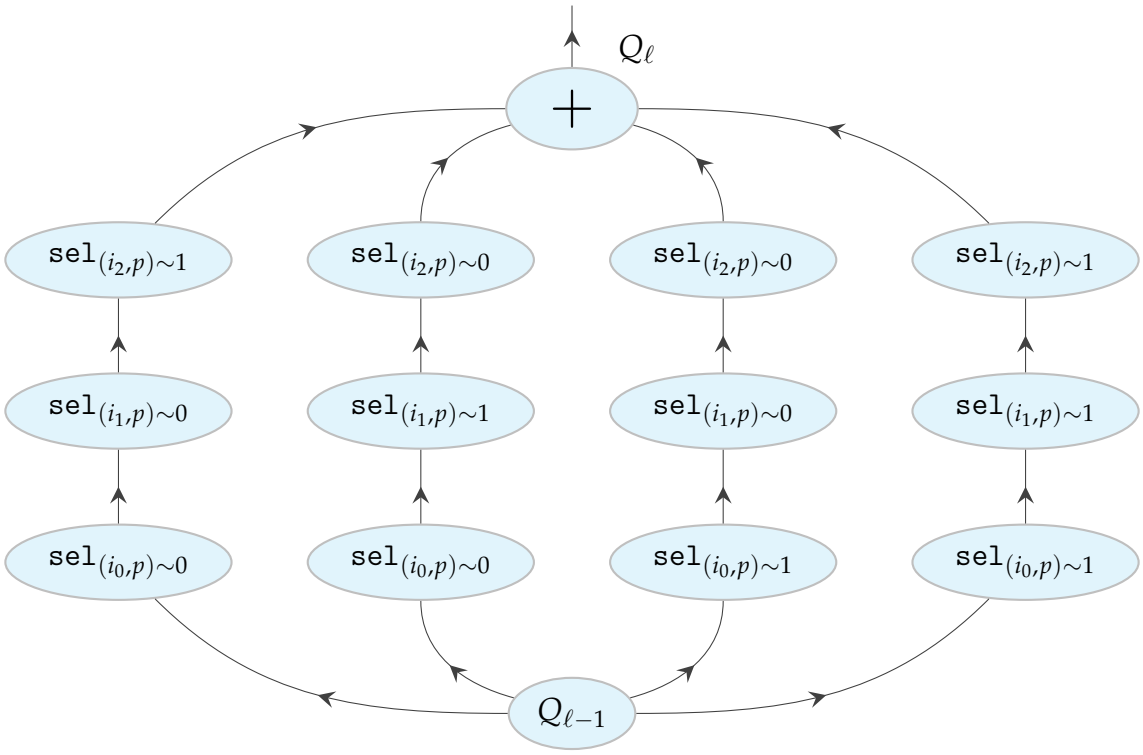

Figure 5: The circuit computing Q_ℓ from $Q_{\ell-1}$ (Equation 4.2), the inductive case of Lemma 4.3

by a unique monomial $\prod_{i\in[\![n]\!]} x_{i,b_i}$ corresponding to a distinct bit string of length n (that is vacuously valid).

Induction step: Assume $\ell > 0$, and that the lemma is true for $\ell - 1$. This means that to each $\mathbf{b}$ satisfying $W_1, W_2, \ldots, W_{\ell-1}$, we can assign a monomial $m_{\mathbf{b}} = \prod_{i\in[\![n]\!]} x_{i,b_i}$, in $Q_{\ell-1}$. Let us define $\mathcal{M}_{\ell-1}$ to be the set of all monomials $m_{\mathbf{b}}$ such that $\mathbf{b}$ satisfies the first $(\ell-1)$ constraints; so we have $Q_{\ell-1} = \sum_{m_{\mathbf{b}}\in\mathcal{M}_{\ell-1}} \mathbf{y}^{\mathbf{e}_y(m_{\mathbf{b}})}\mathbf{z}^{\mathbf{e}_z(m_{\mathbf{b}})} m_{\mathbf{b}}$, for some $\mathbf{e}_y(m_{\mathbf{b}}), \mathbf{e}_z(m_{\mathbf{b}})$. Also, if W_ℓ involves positions i_0, i_1, i_2, then $\mathcal{M}_\ell = \{m_{\mathbf{b}} \in \mathcal{M}_{\ell-1} : (x_{i_0,b_0}x_{i_1,b_1}x_{i_2,b_2} \text{ divides } m_{\mathbf{b}}), b_0 \oplus b_1 \oplus b_2 = 1\}$.

Note that for any $z_{i,j}$ or $y_{i,j}$, the indices i and j fully determine the constraint W_ℓ, and therefore also the "level" ℓ in our circuit at which the variable gets projected. Thus, if $y_{i,j}$ is present in some $Q_{\ell'}$, then every monomial in $Q_{\ell'}$ that is divisible by $x_{i,1}$, is also divisible by $y_{i,j}$, and vice versa; similarly for $z_{i,j}$ and $x_{i,0}$.

We can therefore simplify Equation 4.2 for Q_ℓ as follows.

$$
\begin{aligned}
Q_\ell &= \sum_{(b_0,b_1,b_2)\in\mathsf{ONES}} \mathtt{sel}_{(i_2,p)\sim b_2}\left(\mathtt{sel}_{(i_1,p)\sim b_1}\left(\mathtt{sel}_{(i_0,p)\sim b_0}\left(Q_{\ell-1}\right)\right)\right) \\
&= \sum_{(b_0,b_1,b_2)\in\mathsf{ONES}} \mathtt{sel}_{(i_2,p)\sim b_2}\left(\mathtt{sel}_{(i_1,p)\sim b_1}\left(\mathtt{sel}_{(i_0,p)\sim b_0}\left(\sum_{m_{\mathbf{b}}\in\mathcal{M}_{\ell-1}} \mathbf{y}^{\mathbf{e}_y(m_{\mathbf{b}})}\mathbf{z}^{\mathbf{e}_z(m_{\mathbf{b}})} m_{\mathbf{b}}\right)\right)\right) \\
&= \sum_{(b_0,b_1,b_2)\in\mathsf{ONES}} \left[\sum_{\substack{m_{\mathbf{b}}\in\mathcal{M}_{\ell-1} \\ x_{i_0,b_0}x_{i_1,b_1}x_{i_2,b_2}\mid m_{\mathbf{b}}}} \mathbf{y}^{\mathbf{e}'_y(m_{\mathbf{b}})}\mathbf{z}^{\mathbf{e}'_z(m_{\mathbf{b}})}\cdot m_{\mathbf{b}}\right], \quad \text{for suitable } \mathbf{e}'_y(m_{\mathbf{b}}), \mathbf{e}'_z(m_{\mathbf{b}})\text{s} \\
&= \sum_{m_{\mathbf{b}}\in\mathcal{M}_\ell} \mathbf{y}^{\mathbf{e}'_y(m_{\mathbf{b}})}\mathbf{z}^{\mathbf{e}'_z(m_{\mathbf{b}})}\cdot m_{\mathbf{b}}
\end{aligned}
$$

This finishes the proof. □

Further, note that for each $\ell \in [n']$, the computation of Q_ℓ fixes 6 auxiliary variables that are *unique* to W_ℓ. Since we had $2k \cdot n = 2 \cdot k \cdot 3 \cdot 3^{k-1} = 6 \cdot n'$ auxiliary variables to start with, this means that Q'_n has no auxiliary variables. Along with Lemma 4.3, we then have $Q_{n'} = \mathsf{OP}_k$.

Size. Note that $\text{size}(A_k) = \sum_i \text{size}(A_k(i)) = 3^k \cdot \text{size}(A_k(0)) = O(3^k \cdot k) = O(n \log_3 n)$. Since $n' = O(n \log n)$, the size of the circuit is at most $\text{size}(A_k) + O(n') = O(n \log n)$, as a result of the following easy claim.

Claim 4.4. *For each $\ell = 1, 2, \ldots, n'$, $\text{size}(Q_\ell) = \text{size}(Q_{(\ell-1)}) + O(1)$.*

Proof. By construction, Q_ℓ is obtained as a sum of exactly 4 projections of $Q_{\ell-1}$, each of which is obtained by projecting exactly 6 variables. Thus, $\text{size}(Q_\ell) \le \text{size}(Q_{(\ell-1)}) + 6 \times 4 + 1$. □

4.2 An upper bound for the Permanent

We now describe a monotone circuit with projection gates that computes Perm_n.

Theorem 1.4. *There is a monotone circuit with projection gates of size $O(n^3)$ that computes Perm_n.*

Similar to our circuit for OP_k, the trick is again to use "correlated projections". We use n^2-many auxiliary variables $\mathbf{y}$, one attached to each 'true' variable $x_{i,j}$.

$$\text{Let } P_0(\mathbf{x},\mathbf{y}) := \left(\sum_{j=1}^{n} y_{1,j}x_{1,j}\right)\left(\sum_{j=1}^{n} y_{2,j}x_{2,j}\right)\cdots\left(\sum_{j=1}^{n} y_{n,j}x_{n,j}\right).$$

This ensures that all monomials in P_0 contain at most one $\mathbf{x}$-variable from each row. We now want to progressively prune the monomials that pick up multiple variables from the jth column by projecting the n variables $y_{1,j},\ldots,y_{n,j}$. Let $e_1,\ldots,e_n \in \{0,1\}^n$ such that $e_i(k) = 1 \Leftrightarrow i = k$, and define for each $j \in [n]$,

$$P_j := \sum_{i\in[n]} \mathtt{fix}_{(y_{1,j}=e_i(1))}\left(\mathtt{fix}_{(y_{2,j}=e_i(2))}\left(\cdots\left(\mathtt{fix}_{(y_{n,j}=e_i(n))}\left(P_{j-1}\right)\right)\right)\right). \tag{4.5}$$

The following claim is now easy to verify.

Claim 4.6. *For all $j \in [n]$, P_j contains all the monomials from P_{j-1} that are supported on exactly one $\mathbf{x}$-variable from the jth column.* □

As a result, the monomials in P_n are exactly those of the monomials in Perm_n. Additionally for each j, the auxiliary variables in P_j are only from the columns $j+1,\ldots,n$; thus $P_n = \mathsf{Perm}_n$.

The size of our circuit is $O(n^3)$, since $\mathrm{size}(P_0) = O(n^2)$ and $\mathrm{size}(P_j) = \mathrm{size}(P_{j-1}) + O(n^2)$. This proves Theorem 1.4, and therefore shows that $\{\mathsf{Perm}_n\} \in \mathsf{mVPSPACE}$.

Remark 4.7. *As mentioned earlier, the above theorem yields a $O(m^{1.5})$ vs $2^{\Omega(\sqrt{m})}$ separation between* mVNP *and* mVPSPACE *for a family of m-variate polynomials. The $O(m\log m)$ vs $2^{\Omega(m^{0.63})}$ separation witnessed by $\{\mathsf{OP}_k\}$ in Theorem 1.3 is therefore a clear improvement over this.* ♢

Note that our upper bound above also implies that any polynomial (family) that can be expressed as the permanent of a monotone matrix of size $\mathrm{poly}(n)$ (called monotone p-projection of Perm_n), is also in mVPSPACE. Although Perm_n is complete for non-monotone VNP, it is *not* the case that all monotone polynomials in VNP are monotone p-projections of Perm_n, as shown by Grochow [Gro17].

5 Scopes of Known Techniques

In this section we look at some of the currently known techniques in the context of the concepts that are used in the literature on (non-monotone) VPSPACE.

5.1 Lower bounds for transparent polynomials

We start by recalling the concepts of *shadow complexity* and *transparent polynomials*.

Definition 5.1 (Shadow complexity [HY21])**.** *For a polynomial $f(x_1, \ldots, x_n)$, its shadow complexity $\sigma(f)$ is defined as follows.*

$$\sigma(f) := \max_{L:\mathbb{R}^n \to \mathbb{R}^2} |\mathsf{vert}(L(\mathsf{Newt}(f)))| \qquad \Diamond$$

For any n, a set of points in $\mathbb{R}^n$ is said to be *convexly independent* if no point in the set can be written as a convex combination of other points from the set. Note that if a polynomial has *convexly independent support*, then all the monomials in its support correspond to vertices of its Newton polytope. The following definition is an even stronger condition.

Definition 5.2 (Transparent polynomials [HY21])**.** *A polynomial f is said to be transparent, if $\sigma(f) = |\mathrm{supp}(f)|$.* $\Diamond$

The following lemma states that the linear map that witnesses the shadow complexity of a polynomial over the reals, can be assumed to be "integral" without loss of generality.

Lemma 5.3 (Consequence of [HY21, Lemma 4.2])**.** *Let $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$ be an n-variate polynomial. Then there is an $M \in \mathbb{Z}^{2\times n}$, such that for $L(\mathbf{e}) := M \cdot \mathbf{e}$, $|\mathsf{vert}(L(\mathsf{Newt}(f)))| = \sigma(f)$.* $\square$

We also require the following concepts from the work of Hrubešand Yehudayoff [HY21].

Definition 5.4 (Laurent polynomials and high powered circuits)**.** *A* Laurent polynomial *over the variables $\{x_1, \ldots, x_n\}$ and a field $\mathbb{F}$, is a finite $\mathbb{F}$-linear combination of terms of the form $x_1^{p_1} x_2^{p_2} \cdots x_n^{p_n}$, where $p_1, p_2, \ldots, p_n \in \mathbb{Z}$.*

A high powered circuit over the variables $\{x_1, \ldots, x_n\}$ and a field $\mathbb{F}$, is an algebraic circuit whose leaves can compute terms like $\alpha x_1^{p_1} x_2^{p_2} \cdots x_n^{p_n}$ for any $\alpha \in \mathbb{F}$ and $\mathbf{p} \in \mathbb{Z}^n$. In other words, a high powered circuit can compute an arbitrary Laurent monomial with size 1; the size of the high powered circuit is the total number of nodes as usual. $\Diamond$

Using the above definition, we can easily infer the following by replacing each leaf with the corresponding Laurent monomial.

Observation 5.5. *Let $f(\mathbf{x})$ be computable by a monotone circuit of size s, and suppose $\sigma(f) = k$. Then there exists a bivariate Laurent polynomial $P(y_1, y_2)$ that is computable by a high powered circuit of size s, whose Newton polygon has k vertices.* $\square$

We now have all the concepts required to prove the main theorem of this section, Theorem 1.7. The following results and their proofs closely follow those in [HY21]. We reproduce the overlapping parts for the sake of completeness and ease of exposition.

Lemma 5.6 ([HY21, Lemma 5.8])**.** *Let $A, B \subset R^2$ be finite sets, such that $A + B$ is convexly independent. Then if $|A| \geq |B|$, then either $|A|, |B| \leq 2$ or $|B| = 1$.* $\square$

Theorem 5.7 (Extension of [HY21, Theorem 5.9]). *Let $f(y_1, y_2)$ be a monotone Laurent polynomial with convexly independent support, and let $C(y_1, y_2, \mathbf{z})$ be a monotone high-powered circuit with summation and production gates*[7]*, that computes f. Then* $\text{size}(C) \geq |\text{supp}(f)|/4$.

Proof. For a multi-set[8] $\mathcal{A}$ that contains sets of points in $\mathbb{R}^2$, we define a measure μ that relates to the "largest" convexly independent set that can be constructed using it. For a sub-collection $\mathcal{B} \subseteq \mathcal{A}$ and a map $v : \mathcal{B} \to \mathbb{R}^2$, the resulting set $\mathcal{B}(v)$ is defined as follows.

$$\mathcal{B}(v) := \bigcup_{A \in \mathcal{B}} (\{v(A)\} + A)$$

The measure μ is then defined as follows.

$$\mu(\mathcal{A}) := \max_{\mathcal{B}, v} \{|\mathcal{B}(v)| : \mathcal{B}(v) \text{ is convexly independent}\} \tag{5.8}$$

For a Laurent polynomial $g(y_1, y_2, \mathbf{z})$, let $\text{supp}_{\mathbf{y}}(g) := \{(a, b) : \exists \mathbf{e}, y_1^a y_2^b \mathbf{z}^{\mathbf{e}} \in \text{supp}(g)\}$ be its $\mathbf{y}$-support. Corresponding to the circuit $C(y_1, y_2, \mathbf{z})$ of size s, we will consider the collection $\mathcal{A}$ of s sets, which will be the $\mathbf{y}$-supports of the polynomials computed by the s gates. The following claim will help us prove the theorem by induction.

Claim 5.9. *For $\mathcal{A}' = \mathcal{A} \cup \{B\}$, and $A_1, A_2 \in \mathcal{A}$,*

$$\mu(\mathcal{A}') \leq \mu(\mathcal{A}) + |B|\,, \tag{5.10}$$

$$\mu(\mathcal{A}') \leq \mu(\mathcal{A}) + 2 \qquad \text{if } B = u + A_1, \tag{5.11}$$

$$\mu(\mathcal{A}') \leq \mu(\mathcal{A}) + 4 \qquad \text{if } B = A_1 \cup A_2, \tag{5.12}$$

$$\mu(\mathcal{A}') \leq \mu(\mathcal{A}) + 4 \qquad \text{if } B = A_1 + A_2, \tag{5.13}$$

$$\mu(\mathcal{A}') \leq \mu(\mathcal{A}) + 4 \qquad \text{if } B = A_1 + A' \text{ for } A' \subseteq A_1\,. \tag{5.14}$$

Proof. It is trivial to see that (5.10) holds.

For (5.11), suppose $\mathcal{B}$ is the subset that achieves $\mu(\mathcal{A}') > \mu(\mathcal{A})$. Then $A_1, B \in \mathcal{B}$ as otherwise one can mimic the contribution of B using A_1; further $v(A_1) \neq v(B) + u$ because otherwise the translates of A_1 and B overlap. Now note that $(\{v(A_1)\} + A_1) \cup (\{v(B)\} + B)$ is a convexly independent set of points, and also that $(\{v(A_1)\} + A_1) \cup (\{v(B)\} + B) = \{v(A_1), v(B) + u\} + A_1$. Therefore by Lemma 5.6, we see that $|B| = |A_1| \leq 2$, which finises the proof using (5.10).

For (5.12), observe that $\mu(\mathcal{A}) \leq \mu(\mathcal{A} \cup A_1, A_2)$. The required bound then follows by two applications of (5.11).

In (5.13), if B is convexly *dependent*, then it cannot contribute to $\mu(\mathcal{A}')$, so suppose it is. Assuming $|A_1| \geq |A_2|$ without loss of generality, by Lemma 5.6, either $|B| \leq |A_1| \cdot |A_2| \leq 4$, or

[7] All auxiliary variables only appear with non-negative powers in the circuit.

[8] We assume that copies of the same set $A \in \mathcal{A}$ can be referred distinctly.

$B = u + A_1$ for some u, and (5.11) finishes the proof.

Clearly (5.13) implies (5.14), as its proof does not depend on whether $A_2 \in \mathcal{A}$, or $A_2 \not\subseteq A_1$. □

We now argue that the polynomial computed at every gate in $C(y_1, y_2, \mathbf{z})$ has convexly independent $\mathbf{y}$-support. Since the $\mathbf{y}$-supports of addition and multiplication gates are unions and Minkowski sums of their children respectively, if any of their input is convexly dependent, then so is the output. For a summation gate $g = \mathtt{sum}_z g'$, $\mathrm{supp}_{\mathbf{y}}(g) = \mathrm{supp}_{\mathbf{y}}(g')$ using Lemma 5.16. For a production gate $g = \mathtt{prod}_z g'$, $\mathrm{supp}_{\mathbf{y}}(g) = S' + \mathrm{supp}_{\mathbf{y}}(g')$ for some $S' \subseteq \mathrm{supp}_{\mathbf{y}}(g')$, so any convex dependency in $\mathrm{supp}_{\mathbf{y}}(g')$ would transfer to $\mathrm{supp}_{\mathbf{y}}(g)$. Since the output of $C(x, y, \mathbf{z})$ is convexly independent, the above observations imply that each gate $g \in C$ has convexly independent $\mathrm{supp}_{\mathbf{y}}(g)$.

Let us now prove the theorem by inductively building the collection $\mathcal{A}$ with respect to the circuit C: a gate is added only after adding all of its children. When the gate being added is a leaf, then μ increases by at most 1 due to (5.10). For an addition gate computing g, $\mathrm{supp}_{\mathbf{y}}(g)$ is the union of the (x, y)-supports of its children; so we can apply (5.12). For an multiplication gate computing g, $\mathrm{supp}_{\mathbf{y}}(g)$ is the Minkowski sum of the (x, y)-supports of its children; so we can use (5.13). For a summation gate that computes g, note that its (x, y)-support is exactly the same as that of its child ((5.15)); therefore (5.11) applies. Finally for a production gate, we can use (5.14), as $\mathrm{supp}_{\mathbf{y}}(\mathtt{prod}_z g) = \mathrm{supp}_{\mathbf{y}}(g|_{z=0}) + \mathrm{supp}_{\mathbf{y}}(g|_{z=1})$, and $\mathrm{supp}_{\mathbf{y}}(g|_{z=0}) \subseteq \mathrm{supp}_{\mathbf{y}}(g|_{z=1}) = \mathrm{supp}_{\mathbf{y}}(g)$.

Since the measure μ increases by at most 4 in each of the s steps, we have that $|\mathrm{supp}(f)| \leq \mu(\mathcal{A}) \leq 4s$, as required. □

The above result then lets us prove Theorem 1.7, which we first restate.

Theorem 1.7. *Any monotone algebraic circuit with summation and production gates that computes a transparent polynomial f, has size $|\mathrm{supp}(f_n)|/4$.*

Proof. Let C be a quantified monotone circuit computing f_n, of size s. Since $f_n(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$ is transparent, there exists a matrix $M \in \mathbb{Z}^{2\times n}$, such that the linear map $L(\mathbf{e}) = M\mathbf{e}$, satisfies $|\mathrm{vert}(L(\mathsf{Newt}(f)))| = |\mathrm{supp}(f)|$. Further using Observation 5.5, there exists a size-s high powered monotone circuit with summation and production gates, that computes a Laurent polynomial $P(y_1, y_2)$ which has $|\mathrm{supp}(f)|$ vertices in its Newton polytope. The bound then easily follows from Theorem 5.7. □

5.2 Lower bounds based on support

Next, we analyse lower bound arguments that rely solely on the support of the claimed hard polynomial. This is an extension of the following observation due to Yehudayoff [Yeh19].

Observation 5.15 ([Yeh19])**.** *Let $g(\mathbf{x}, z)$ be a monotone polynomial and let $c > 0$. Then for any monomial $m = \mathbf{x}^{\mathbf{e}} z^j$ in the support of g, $\mathbf{x}^{\mathbf{e}} \in \mathrm{supp}(g, z = c)$.*

In particular, we show the following.

Lemma 5.16. *Let $f(\mathbf{x})$ be a monotone polynomial whose support cannot be written as a non-trivial product of two sets, and for some monotone polynomial $g(\mathbf{x}, \mathbf{z})$, suppose $f(\mathbf{x}) = \mathtt{Q}_{z_1}^{(1)}\mathtt{Q}_{z_2}^{(2)} \cdots \mathtt{Q}_{z_m}^{(m)} g(\mathbf{x}, \mathbf{z})$ with $\mathtt{Q}^{(i)} \in \{\mathtt{sum}, \mathtt{prod}\}$ for each $i \in [m]$.*

Then $\mathrm{supp}(f(\mathbf{x})) = \mathrm{supp}(g(\mathbf{x}, \bar{1}))$.

Proof. Observe that it is enough to show the statement of the lemma for $m = 1$.

Therefore, suppose $f(\mathbf{x}) = \mathtt{sum}_z g(\mathbf{x}, z)$, then $f(\mathbf{x}) = g(\mathbf{x}, 0) + g(\mathbf{x}, 1)$, and hence $\mathrm{supp}(f) = \mathrm{supp}(g(\mathbf{x}, 1))$, since g is monotone.

Next, $f(\mathbf{x}) = \prod_z g(\mathbf{x}, z)$ means that $f(\mathbf{x}) = g(\mathbf{x}, 0) \cdot g(\mathbf{x}, 1)$. As $\mathrm{supp}(f)$ cannot be written as a non-trivial product of two sets, and since g is monotone, this must mean that $g(\mathbf{x}, 0)$ is a constant and $\mathrm{supp}(f(\mathbf{x})) = \mathrm{supp}(g(\mathbf{x}, 1))$ as claimed. □

6 Other Possible Definitions of Monotone VPSPACE

In this section, we discuss monotone analogues of some of the other definitions VPSPACE. We start by briefly addressing the definitions due to Koiran and Perifel [KP09], and Mahajan and Rao [MR13], that directly or indirectly rely on boolean computational models.

Definitions relying on boolean computation ([KP09, MR13]). Koiran and Perifel define uniform VPSPACE as the class of families $\{f_n\}$ of $\mathrm{poly}(n)$-variate polynomials of degree at most $2^{\mathrm{poly}(n)}$, such that there is a PSPACE machine that computes the *coefficient function* of $\{f_n\}$. Here, the coefficient function of $\{f_n\}$ can be seen to map a pair $(1^n, \mathbf{e})$ to the coefficient of $\mathbf{x}^{\mathbf{e}}$ in f_n.

Non-uniform VPSPACE is then defined by replacing PSPACE by its non-uniform analogue, PSPACE/poly. Since there are no monotone analogues of Turing machines, perhaps the only possible monotone analogue of this definition is to insist on the coefficient function being monotone, which results in an absurdly weak class (the "largest" monomial will always be present).

Mahajan and Rao [MR13] look at the notion of *width* of a circuit — all gates are assigned heights, such that the height of any gate is *exactly* one larger than the height of its highest child. The width of the circuit is the maximum number of nodes that have the same height. They then define $\mathsf{VSPACE}(S(n))$, as the class of families that are computable by circuits of width $S(n)$ and size at most $\max\left\{2^{S(n)}, \mathrm{poly}(n)\right\}$.

The class uniform $\mathsf{VSPACE}(S(n))$ further requires that the circuits be $\mathsf{DSPACE}(\mathsf{S}(\mathsf{n}))$-uniform. Although their non-uniform definition is purely algebraic, it is a bit unnatural for space $S(n) \gg \log n$ (as also pointed out in their paper), since such circuits may not even have a $\mathrm{poly}(n)$-sized description. We therefore do not analyse a monotone analogue for their definition.

6.1 Succinct Algebraic Branching Programs

We start by considering the most natural monotone version of *succinct ABPs*. Let us start by recalling the definition.

Definition 6.1 (Succinct ABPs [Mal11])**.** *A* succinct ABP *over the n variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ is a three tuple $(B, \mathbf{s}, \mathbf{t})$ with $|\mathbf{s}| = |\mathbf{t}| = r$, where*

- $\mathbf{s}$ *is the label of the source vertex, and* $\mathbf{t}$ *is the label of the sink(target) vertex.*
- $B(\mathbf{u}, \mathbf{v}, \mathbf{x})$ *is an algebraic circuit that describes the underlying graph G_B of the ABP on the vertex set $\{0,1\}^r$. For any two vertices $\mathbf{a}, \mathbf{b} \in \{0,1\}^r$, the output $B(\mathbf{u} = \mathbf{a}, \mathbf{v} = \mathbf{b}, \mathbf{x})$ is the label of the edge from $\mathbf{a}$ to $\mathbf{b}$ in the ABP.*

As usual, the polynomial computed by the ABP is the sum of polynomials computed along all $\mathbf{s}$ to $\mathbf{t}$ paths; where each path computes the product of the labels of the constituent edges.

The size of the circuit B is said to be the complexity *of the succinct ABP. The number of vertices 2^r is the* size *of the succinct ABP, and the length of the longest $\mathbf{s}$ to $\mathbf{t}$ path is called the* length *of the ABP.* ◊

The natural monotone analogue is therefore, the following.

Definition 6.2 (Monotone Succinct ABPs)**.** *A succinct ABP $(B, \mathbf{s}, \mathbf{t})$ is said to be monotone, if the circuit B is a monotone algebraic circuit.*

Naturally, this means that all the edge-labels in the ABP are monotone polynomials over $\mathbf{x}$. ◊

While Malod [Mal11] showed that every family in VPSPACE admits succinct ABPs of polynomial complexity, the power of monotone succinct ABPs of polynomial complexity collapses to mVNP!

Theorem 1.5. *If an n-variate polynomial $f(\mathbf{x})$ of degree d is computable by a monotone succinct ABP of complexity* $\mathrm{poly}(n)$*, then $f(\mathbf{x}) \in$ mVNP.*

Proof. Let $A = (B, \mathbf{s}, \mathbf{t})$ be the monotone succinct ABP computing f, with $|\mathbf{s}| = |\mathbf{t}| = r$.

Claim 6.3. *The length of A is at most* $\deg(f)$.

Proof. Let $b(\mathbf{u}, \mathbf{v}, \mathbf{x})$ be the *monotone* $(2r + n)$-variate polynomial computed by the circuit B. Due to the monotonicity of b, for any $\mathbf{e} \in \mathbb{N}^n$ we have that if the monomial $\mathbf{x}^{\mathbf{e}}$ appears in any edge-label $(\mathbf{a}, \mathbf{b})$, then it also appears in the label of $(\bar{1}, \bar{1})$; therefore $\deg_{\mathbf{x}}(B(\mathbf{a}, \mathbf{b}, \mathbf{x})) \leq \deg_{\mathbf{x}}(B(\bar{1}, \bar{1}, \mathbf{x}))$ for all $\mathbf{a}, \mathbf{b}$. Similarly, $\deg_{\mathbf{x}}(B(\mathbf{s}, \mathbf{b}, \mathbf{x})) \leq \deg_{\mathbf{x}}(B(\mathbf{s}, \bar{1}, \mathbf{x}))$ and $\deg_{\mathbf{x}}(B(\mathbf{a}, \mathbf{t}, \mathbf{x})) \leq \deg_{\mathbf{x}}(B(\bar{1}, \mathbf{t}, \mathbf{x}))$ for all $\mathbf{a}, \mathbf{b}$.

Therefore if the length of the ABP is $L > 1$, then $\deg(f) = \deg(B(\mathbf{s}, \bar{1}, \mathbf{x}) \cdot B(\bar{1}, \bar{1}, \mathbf{x})^{L-2} \cdot B(\bar{1}, \mathbf{t}, \mathbf{x})) \geq L - 2$. □

As a result of the above claim we have the following, where $d = \deg(f)$.

$$\begin{aligned}
f(\mathbf{x}) &= B(\mathbf{s},\mathbf{t},\mathbf{x}) + \sum_{j=1}^{d-1} (\text{sum of } \mathbf{s}\text{–}\mathbf{t} \text{ paths through } j \text{ intermediate vertices}) \\
&= B(\mathbf{s},\mathbf{t},\mathbf{x}) + \sum_{j=1}^{d-1} \left(\sum_{\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_j \in \{0,1\}^r} B(\mathbf{s},\mathbf{a}_1,\mathbf{x}) \cdot \left(\prod_{k=1}^{j-1} B(\mathbf{a}_k,\mathbf{a}_{k+1},\mathbf{x}) \right) \cdot B(\mathbf{a}_j,\mathbf{t},\mathbf{x}) \right) \\
&= B(\mathbf{s},\mathbf{t},\mathbf{x}) + \sum_{\mathbf{a}_1,\ldots,\mathbf{a}_{d-1} \in \{0,1\}^r} \sum_{j=1}^{d-1} 2^{-r(d-1-j)} \left(B(\mathbf{s},\mathbf{a}_1,\mathbf{x}) \cdot \left(\prod_{k=1}^{j-1} B(\mathbf{a}_k,\mathbf{a}_{k+1},\mathbf{x}) \right) \cdot B(\mathbf{a}_j,\mathbf{t},\mathbf{x}) \right) \\
&= \sum_{\mathbf{a}_1,\ldots,\mathbf{a}_{d-1}} \left(2^{-r(d-1)} B(\mathbf{s},\mathbf{t},\mathbf{x}) + \sum_{j=1}^{d-1} 2^{-r(d-1-j)} B(\mathbf{s},\mathbf{a}_1,\mathbf{x}) \left(\prod_{k=1}^{j-1} B(\mathbf{a}_k,\mathbf{a}_{k+1},\mathbf{x}) \right) B(\mathbf{a}_j,\mathbf{t},\mathbf{x}) \right)
\end{aligned}$$

The last line clearly yields a monotone VNP expression, since $d = \text{poly}(n)$ and B is a monotone circuit of size $\text{poly}(n)$. Note that the second line already writes $f(\mathbf{x})$ as a polynomially large sum of mVNP expressions with the last two lines just explicitly exhibiting the closure of mVNP under such sums. □

6.2 Quantified monotone circuits

Next, we turn to the monotone analogue of "quantified algebraic circuits" which are also known to characterise VPSPACE (see [Mal11, Corollary 1]). We begin by defining this model.

Definition 6.4 (Quantified Algebraic Circuits)**.** *A quantified algebraic circuit is an algebraic circuit with summation and production gates, that has the form*

$$\mathtt{Q}_{z_1}^{(1)} \mathtt{Q}_{z_2}^{(2)} \cdots \mathtt{Q}_{z_m}^{(m)} C(\mathbf{x},\mathbf{z}).$$

Here $\mathtt{Q}^{(i)} \in \{\mathtt{sum},\mathtt{prod}\}$ *for each* $i \in [m(n)]$*, and* C *is an algebraic circuit with only addition and multiplication gates. The size of a quantified algebraic circuit is the total number of gates in it, including the summation and production gates.*

A quantified monotone circuit *is a quantified algebraic circuit where all the constants appearing are non-negative.* ◊

One can then naturally define the class of polynomial families that are efficiently computable by such quantified circuits.

Definition 6.5 (Quantified Monotone VP)**.** *A polynomial family* $\{f_n(\mathbf{x})\}$ *is in* quantified monotone VP *if there is a constant* $c \in \mathbb{N}$*, such that for all large enough* n*,* f_n *is a polynomial in at most* n^c *variables, that is computable by quantified monotone circuits of size at most* n^c*.* ◊

We immediately have that mVNP is contained in the above class, by letting all $\mathtt{Q}^{(i)}$s be summation gates. A natural question is whether quantified monotone VP is a strictly larger class than

mVNP. While we do believe this is true, we unfortunately do not have any lower bound techniques that can separate these two classes.

In particular, it follows from Lemma 5.16 that almost all lower bounds against VP, that follow from either transparency or the support of hard polynomials[9] (for example, [JS82]), also yield almost identical lower bounds against quantified monotone VP. As a result, the separation in Theorem 1.3 also extends to an identical separation between quantified monotone VP, and mVPSPACE.

Theorem 1.6. *There exists a family* $\{P_n\} \in \mathsf{mVPSPACE}$ *such that for all* n *large enough,* P_n *requires quantified monotone circuits of size* $2^{\Omega(n^{0.63})}$.

Proof. First note that if $\mathrm{supp}(\mathsf{OP}_k) = A \times B$, then all monomials in A have the same degree. Further, it follows from Lemma 3.11 that $t_1 = 1$, which means that either A or B has degree 0 for $A \times B$ to cover all of $\mathrm{supp}(\mathsf{OP}_k)$. Therefore the polynomials in $\{\mathsf{OP}_k\}$ have irreducible supports, and Lemma 5.16 proves the statement. □

The following result says that a transparent polynomial is hard for quantified monotone circuits, as it is for 'usual' algebraic circuits; it directly follows from the results in the previous section.

Theorem 6.6 (Corollary of Theorem 1.7)**.** *Let* f *be a monotone, transparent polynomial. Then any quantified monotone circuit computing* f *must have size* $|\mathrm{supp}(f)|/4$. □

7 Conclusion

Our work starts with the aim of understanding the hardness of transparent polynomials in the context of monotone algebraic computation. We observe that the lower bound of Hrubeš and Yehudayoff [HY21] extends beyond monotone VNP, and therefore turn to exploring the class VPSPACE from the non-monotone world. This exploration reveals that the natural monotone analogues of the multiple equivalent definitions of VPSPACE have contrasting powers. As shown earlier, transparent polynomials turn out to be as hard for some of these analogues as they are for "usual" monotone circuits. There are some fairly natural open questions about transparency, and the various monotone models discussed in this paper, we now conclude by listing some of those.

- Perhaps the most natural question in light of the our results is whether mVPSPACE — poly-sized monotone circuits with projections — can compute transparent polynomials that have a large support. An immediate hurdle in extending Theorem 5.7 to mVPSPACE is that unlike summations and productions, 0-projections do not preserve convex dependencies, that is, the 0-projection of a convexly dependent polynomial could be convexly independent.

[9] All such arguments known to us, work with polynomials having an *irreducible* support.

Nevertheless, this does not rule out a weaker lower bound against monotone circuits with projections that compute transparent polynomials.

It would also be equally interesting if a transparent polynomial with large support is shown to be in mVPSPACE. Such a result would be a bit surprising, at least to us, given how restrictive the transparency condition is.

- Along similar lines, a possibly simpler goal is to show a non-monotone circuit upper bound for a transparent polynomial. Note that transparency only restricts the support of the polynomial, so one is free to choose any real coefficients that do not affect the transparency. It could therefore be possible to compute transparent polynomials that have positive and negative coefficients (Lemma 5.3 works for all real polynomials) in VP using fundamentally non-monotone tricks like interpolation. Among other things, such a result would refute the notoriously open Conjecture 1.1.

- Another open direction is to compare the power of quantified monotone VP, with mVNP. Our result in Lemma 5.16 says that any separation between these models cannot rely solely on the support of the separating polynomial. Some recent works in the monotone world [Yeh19, Sri19, CDM21, CDGM22] have devised lower bound techniques that break this "support barrier". It would be interesting to see if some of these techniques could be adapted to solve this problem.

8 Acknowledgements

We thank Ramprasad Saptharishi and Suhail Sherif for discussions about the results in this work that helped enhance the presentation.

References

[CDGM22] Arkadev Chattopadhyay, Rajit Datta, Utsab Ghosal, and Partha Mukhopadhyay. Monotone Complexity of Spanning Tree Polynomial Re-Visited. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 39:1–39:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[CDM21] Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. Lower Bounds for Monotone Arithmetic Circuits Via Communication Complexity. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC 2021)*, 2021. Pre-print available at `eccc:TR20-166`.

[CKR20] Bruno Pasqualotto Cavalar, Mrinal Kumar, and Benjamin Rossman. Monotone Circuit Lower Bounds from Robust Sunflowers. In *LATIN 2020: Theoretical Informatics - 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, volume 12118 of *Lecture Notes in Computer Science*, pages 311–322. Springer, 2020.

[Gas87] S.B. Gashkov. The complexity of monotone computations of polynomials. *Moscow University Math Bulletin*, (5):1–8, 1987.

[Gro17] Joshua A. Grochow. Monotone Projection Lower Bounds from Extended Formulation Lower Bounds. *Theory of Computing*, 13(18):1–15, 2017. eccc:TR15-171.

[GS12] S. B. Gashkov and I. S. Sergeev. A method for deriving lower bounds for the complexity of monotone arithmetic circuits computing real polynomials. *Sbornik. Mathematics*, 203(10), 2012.

[HY13] Pavel Hrubeš and Amir Yehudayoff. Formulas are Exponentially Stronger than Monotone Circuits in Non-commutative Setting. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 10–14. IEEE Computer Society, 2013.

[HY16] Pavel Hrubeš and Amir Yehudayoff. On Isoperimetric Profiles and Computational Complexity. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, pages 89:1–89:12, 2016. eccc:TR15-164.

[HY21] Pavel Hrubeš and Amir Yehudayoff. Shadows of Newton Polytopes. In *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[JS82] Mark Jerrum and Marc Snir. Some Exact Complexity Results for Straight-Line Computations over Semirings. *Journal of the ACM*, 29(3):874–897, 1982.

[KP07] Pascal Koiran and Sylvain Perifel. VPSPACE and a Transfer Theorem over the Complex Field. In *Proceedings of the 32nd International Conference on Mathematical Foundations of Computer Science*, MFCS'07, page 359–370, Berlin, Heidelberg, 2007. Springer-Verlag.

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

[KPTT15] Pascal Koiran, Natacha Portier, Sébastien Tavenas, and Stéphan Thomassé. A τ-Conjecture for Newton Polygons. *Foundations of Computational Mathematics*, 15:185–197, 2015.

[Kuz85] S. E. Kuznetsov. Monotone computations of polynomials and circuits withoutzero chains, 1985.

[KZ86] O. M. Kasim-Zade. Arithmetic complexity of monotone polynomials. *Theoretical Problems in Cybernetics. Abstracts of lectures*, page 68–69, 1986.

[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.

[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.

[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.

[RY11] Ran Raz and Amir Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *Journal of Computer and System Sciences*, 77(1):167–190, 2011. Preliminary version in the *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*.

[Sap15] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015.

[Sch76] Claus-Peter Schnorr. A Lower Bound on the Number of Additions in Monotone Computations. *Theor. Comput. Sci.*, 2(3):305–315, 1976.

[Sri19] Srikanth Srinivasan. Strongly Exponential Separation Between Monotone VP and Monotone VNP. *CoRR*, abs/1903.01630, 2019. Pre-print available at `arXiv:1903.01630`.

[SS77] Eli Shamir and Marc Snir. *Lower bounds on the number of multiplications and the number of additions in monotone computations*. IBM Thomas J. Watson Research Division, 1977.

[SS80] Eli Shamir and Marc Snir. On the Depth Complexity of Formulas. *Math. Syst. Theory*, 13:301–322, 1980.

[Val80] Leslie G. Valiant. Negation can be Exponentially Powerful. *Theor. Comput. Sci.*, 12:303–314, 1980.

[Yeh19] Amir Yehudayoff. Separating monotone VP and VNP. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 425–429. ACM, 2019.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il