Towards Optimal Depth-Reductions for Algebraic Formulas

Hervé Fournier
Université Paris Cité, IMJ-PRG, France

Nutan Limaye
IT University Copenhagen, Denmark

Guillaume Malod
Université Paris Cité, IMJ-PRG, France

Srikanth Srinivasan
Aarhus University, Denmark

Sébastien Tavenas
Université Savoie Mont Blanc, CNRS, France

Abstract

Classical results of Brent, Kuck and Maruyama (IEEE Trans. Computers 1973) and Brent (JACM 1974) show that any algebraic formula of size s can be converted to one of depth $O(\log s)$ with only a polynomial blow-up in size. In this paper, we consider a fine-grained version of this result depending on the degree of the polynomial computed by the algebraic formula.

Given a homogeneous algebraic formula of size s computing a polynomial P of degree d, we show that P can also be computed by an (unbounded fan-in) algebraic formula of depth $O(\log d)$ and size $\mathrm{poly}(s)$. Our proof shows that this result also holds in the highly restricted setting of monotone, non-commutative algebraic formulas.

This improves on previous results in the regime when d is small (i.e., $d = s^{o(1)}$). In particular, for the setting of $d = O(\log s)$, along with a result of Raz (STOC 2010, JACM 2013), our result implies the same depth reduction even for *inhomogeneous* formulas. This is particularly interesting in light of recent algebraic formula lower bounds, which work precisely in this "low-degree" and "low-depth" setting.

We also show that these results cannot be improved in the monotone setting, even for commutative formulas.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebraic formulas, depth-reduction

Digital Object Identifier 10.4230/LIPIcs.CCC.2023.28

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2023/009/`

Funding *Nutan Limaye*: A part of this work was conducted during a research visit funded by Université Paris Cité in summer 2022.
Srikanth Srinivasan: Supported by start-up package from Aarhus University, and benefited greatly from a research visit sponsored by the Guest researchers faculty program at Université Paris Cité in summer 2022.
Sébastien Tavenas: This work is supported by ANR-22-CE48-0007.

1 Introduction

In this paper, we study a basic question regarding computational tradeoffs between two resources for the model of *algebraic formulas*.

An algebraic formula F for a multivariate polynomial $P(x_1, \ldots, x_n)$ is simply an algebraic expression for P made up of nested additions and multiplications. Equivalently, it can be defined as a rooted directed tree where the leaves are labelled by variables and internal nodes (or gates) compute either linear combinations or products of their children (a formal definition can be found in Section 2 below). Unless otherwise stated, we do not bound the number of children of a gate (in other words, we consider formulas of *unbounded fan-in*).

The two basic computational resources that describe the complexity of an algebraic formula F are its *size*, which is the number of leaves in the underlying tree, and its *depth*, which naturally is the depth of the tree. Polynomials[1] that have efficient (i.e., $\mathrm{poly}(n)$-sized) algebraic formulas form the algebraic complexity class VF. Like its Boolean counterpart NC^1, this is a natural and important complexity class.

Tradeoffs between size and depth in the setting of formulas and related models of computation have been the focus of many previous works, starting from the early 1970s [28, 3, 4, 31, 25, 17, 21, 23, 1, 12, 29, 9, 11, 7, 15]. We describe a few such results here.

- In the Boolean setting, Spira [28] and independently Khrapchenko (see [32]) showed that any Boolean formula of size s can be converted to a Boolean formula of depth $O(\log s)$ while keeping the size bounded by $s^{O(1)}$. These results were replicated in the algebraic setting in results of Brent, Kuck and Marayuma [3] and Brent [4]. The constants involved in the bounds for the depth and the size were improved in many follow-up works [19, 20, 13, 5, 2].
- This question has also been studied for the more general model of *algebraic circuits,* where the underlying tree is replaced by a directed acyclic graph (DAG). A well-known result of Valiant, Skyum, Berkowitz and Rackoff [31] showed that an algebraic circuit of size s computing a polynomial of degree $\mathrm{poly}(s)$ can be converted to a circuit of depth[2] $O(\log s)$ and size $\mathrm{poly}(s)$. These results were also shown to hold for *multilinear*[3] circuits by Raz and Yehudayoff [23].
- The above results are known to be tight in various settings. In the *monotone* case[4], Shamir and Snir [25] showed the existence of an explicit polynomial P with a $\mathrm{poly}(n)$-sized circuit such that any circuit of depth $o(\log n)$ for P is of superpolynomial size. Similar results were obtained in the multilinear case by Raz [21] and Chillara, Limaye and Srinivasan [8] (for multilinear circuits and formulas, respectively).
- Beginning with the work of Agrawal and Vinay [1], a recent line of work [12, 29, 9, 11, 7, 14] has shown that algebraic circuits and formulas can be converted to formulas of *constant depth* with a *sub-exponential* blow-up in size. In contrast, our focus in this paper is primarily on reducing depth as much as possible while keeping the size bounded by $\mathrm{poly}(s)$, as in the results listed previously.

The question

In this paper, we ask the question of whether stronger *depth-reduction* results can be proved given a bound d on the degree of the polynomial $P(x_1, \ldots, x_n)$ computed by the algebraic formula. In general, an algebraic formula of size s can compute a polynomial of degree at most s. When $d = s$ (or $d = s^{\Omega(1)}$), the above results imply that an algebraic formula F for P can be converted to another formula F' of depth $O(\log d)$ without significant blow-up in size. Does such a result hold for any d (or more specifically, when $d = s^{o(1)}$)?

[1] Strictly speaking, we should refer here to infinite sequences of polynomials, but we ignore this distinction.
[2] In the bounded fan-in case, this would be depth $O(\log^2 s)$ instead.
[3] A circuit or formula is multilinear if each of its gates computes a multilinear polynomial.
[4] where the underlying field is $\mathbb{R}$ and all constants are non-negative, so cancellations do not occur.

Note that this question only makes sense for algebraic formulas of unbounded fan-in. If the fan-in of each gate is bounded by a constant, then any formula of size s must have depth at least $\Omega(\log s)$ (and so, Brent, Kuck, and Marayuma's result [3] is optimal). However, in many settings (see e.g. the third motivation below), we want a finer analysis of the formula depth that can be achieved by formulas of unbounded fan-in.

Motivation

While the question is fairly natural in our opinion, there are also many concrete reasons that lead to this line of inquiry.

- It is easy to see from the proof of Valiant, Skyum, Berkowitz, and Rackoff [31] that any algebraic *circuit* can be depth-reduced to depth $O(\log d)$ with only a $\mathrm{poly}(s, d)$ blow-up in size. So the natural generalization for small degrees is indeed true in the setting of algebraic circuits.
- It also follows from previous results that the depth bound of $O(\log s)$ can be improved when the degree is $d = o(\log s)$. This is due to a result of Raz [22]: any formula F for a homogeneous polynomial P of degree d can be converted to a *homogeneous* formula[5] F' efficiently. Further, it is easy to see that any homogeneous formula computing a polynomial of degree d has depth $O(d) = o(\log s)$. So, in this regime for the degree, standard depth-reduction results *can* be strengthened.
- Finally, very recent results in algebraic complexity [16] have suggested a way of proving lower bounds against *low-depth* algebraic formulas for computing *low-degree* polynomials, which naturally raises the question of obtaining the best possible depth-reduction results in this setting.

  More specifically, Limaye, Srinivasan, and Tavenas [16] showed how to prove lower bounds against algebraic formulas (and even circuits) of small depth. Their proof proceeds by converting an algebraic formula of size s and depth Δ to a homogeneous algebraic formula of size $\mathrm{poly}(s)$ and depth $O(\Delta)$, and then proving lower bounds against homogeneous algebraic formulas of depth $O(\Delta)$. An important point regarding the first step is that it only works in the "low-degree setting" of $d = O(\log s / \log\log s)$. The second step proves lower bounds against homogeneous formulas of depth up to $O(\log\log d)$.

  To make this proof idea work for general (unbounded-depth) algebraic formulas, we would like to be able to homogenize and depth-reduce algebraic formulas as much as possible. The aforementioned result of Raz [22] already shows that we can homogenize algebraic formulas efficiently in the low-degree setting. So it is natural to investigate the best possible depth-reduction for homogeneous algebraic formulas in the low-degree setting.

Results

Our main result is a depth-reduction result for *homogeneous* formulas that efficiently reduces the depth to $O(\log d)$, matching what was already known for algebraic circuits by the result of [31].

▶ **Theorem 1** (Main Result). *Let F be a homogeneous algebraic formula of size s computing a polynomial P of degree $d \geqslant 2$. Then P is also computed by a homogeneous formula F' of size $\mathrm{poly}(s)$ and depth $O(\log d)$. Moreover, if F is monotone and/or non-commutative, then so is F'.*

[5] A homogeneous formula is one where each gate computes a homogeneous polynomial. This means that the formula does not compute intermediate polynomials of degree larger than d.

Here, a *monotone* algebraic formula is one that does not exploit cancellations in any way, and a *non-commutative* formula describes a polynomial expression in a domain where the input variables do not commute when multiplied with each other (formal definitions are given in Section 2). These are both settings in which formula upper bounds are harder to prove, and hence the depth-reduction result in this setting implies the result in the standard setting. It can also be checked that the depth-reduction procedure above preserves other interesting properties of the formula, such as *multilinearity* and *set-multilinearity.*

Using the aforementioned result of Raz that allows us to homogenize algebraic formulas in the low-degree setting, we get the following depth-reduction even for *inhomogeneous* formulas.

▶ **Corollary 2.** *Let $d = O(\log n)$. Then a homogeneous polynomial P defined on n variables with degree $d \geqslant 2$ has an algebraic formula of size $\mathrm{poly}(n)$ if and only if it has an algebraic formula of depth $O(\log d)$ and size $\mathrm{poly}(n)$.*

In particular, this means that to prove superpolynomial lower bounds against general algebraic formulas in the low-degree setting, it suffices to prove such lower bounds against homogeneous algebraic formulas of depth $O(\log d)$. As far as we know, nothing below the trivial $O(d)$ bound was known before for such an implication. This brings us much closer to the regime of depths for which we have lower bounds [16].

The statements are even starker in the non-commutative setting, where it is a long-standing problem to prove separations between *Algebraic Branching Programs* (ABPs) and formulas. In recent work [30], it was shown how to prove such a result for depths that are $o(\sqrt{\log d})$. The results of this paper show that it suffices to prove such a result for depth $O(\log d)$.[6]

Finally, we also show that our results cannot be improved asymptotically in terms of depth, unless we use techniques that exploit cancellations in some way.

▶ **Theorem 3** (Lower Bound)**.** *Let n and $d = d(n)$ be growing parameters such that $d(n) \leqslant \sqrt{n}$. Then there is a monotone algebraic formula F of size at most n and depth $O(\log d)$ computing a polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ of degree at most d such that any monotone formula F' of depth $o(\log d)$ computing P must have size $n^{\omega(1)}$.*

It should be noted that a well-known result of Gupta, Kamath, Kayal and Saptharishi [9] shows how to exploit cancellations to obtain better depth-reduction results. However, in general this does not reduce the depth of a given formula by more than a constant factor without incurring a significant blow-up in size.[7] As a result, we believe that the above result is a strong indication that our depth reduction result is tight up to a constant factor in the depth.

2 Preliminaries

Basic notation

Throughout, unless otherwise specified, we work with polynomials over a field $\mathbb{F}$. We will work with the multivariate ring of polynomials $\mathbb{F}[x_1, \ldots, x_n]$ or its *non-commutative* analog $\mathbb{F}\langle x_1, \ldots, x_n\rangle$.

[6] We note that Raz's result, though only stated for the commutative setting, works just as well in the non-commutative case.

[7] More precisely, the result of [9] shows how to convert a low-degree *homogeneous* depth-4 formula to an *inhomogeneous* depth-3 formula efficiently. In general, this can be used to reduce the depth of a small-depth formula by a multiplicative factor of 2.

2.1 Algebraic formulas

We start with some brief definitions and results related to algebraic formulas. For much more about this model, see the standard references [27, 24].

The model

An algebraic formula over the multivariate polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$ is a rooted, directed tree with edges directed towards the root. Leaves are labelled by variables $x_1, \ldots, x_n$ or by the constant 1 and edges by non-zero field constants. Internal nodes (i.e., gates) by $+$ and $\times$ and compute linear combinations (based on the edge weights) or products of their children. We will assume, with loss of generality, that if a node α has for child a leaf labelled by 1, then α is a $+$-gate and that if a $+$-gate α has only children labelled by 1, then α is the output of the formula.[8] A *non-commutative* algebraic formula over the multivariate polynomial ring $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ is defined similarly, with the additional assumption that the children of any $\times$-gate are linearly ordered, and the corresponding product is computed in this order.

Unless explicitly stated, the algebraic formulas we consider have *unbounded* fan-in (i.e., a gate can have any number of inputs). The *size* of F will denote the number of leaves,[9] the *depth* of F the longest leaf-to-root path. The *product-depth* and the *sum-depth* of F are defined to be the maximum number of product gates and sum gates encountered on a leaf-to-root path, respectively.

A *parse tree* of a formula F is a subformula of F which corresponds to the way a monomial is built in the evaluation of F. Parse trees of F can be defined inductively as follows:

- If F has a top $+$-gate, a parse tree of F is obtained by taking a parse tree of one of its children together with the corresponding edge to the root of F;
- If F has a top $\times$-gate, a parse tree of F is obtained by a taking a parse tree of each of its children, together with the incoming edges of the root of F;
- The only parse tree of a leaf is itself.

The polynomial computed by a parse tree is a single monomial, which is equal to the product of the variables labelling its leaves, multiplied by the product of the scalars labelling its edges. The polynomial computed by a formula F is easily seen to be the sum of the monomials computed by all its parse trees.

A formula F is called *monotone* if any monomial computed by a parse tree of F has a non-zero coefficient in the polynomial computed by F.

We now recall some well-known results from the literature regarding algebraic formulas. It should be noted that these results (specifically Theorems 4, 5 and 10 later on) are usually proved in the general setting of commutative formulas. However, it is easy to see that the proofs of these results carry over to the monotone, non-commutative setting without significant change.

Depth-reduction

Classical results [3, 4] show that any algebraic formula of small size can be simulated by one of small depth and not much larger size. Formally,

[8] This ensures that a formula can compute polynomials with a constant term but forbids using many arithmetic operations just to compute constants.
[9] This is within a constant factor of the number of gates, as long as each gate has fan-in at least 2 each (which is without loss of generality).

▶ **Theorem 4.** *Let F be a (non-commutative or commutative) algebraic formula of size s computing a polynomial P. Then there is an algebraic formula F' of size at most* $\mathrm{poly}(s)$ *and depth $\Delta = O(\log s)$ computing P. We may also assume that each gate in F' has fan-in 2. Furthermore, F is homogeneous and/or monotone, then so is F'.*

Homogeneity

Each gate in an algebraic formula has a *syntactic degree* defined in a natural way. Leaves labelled by the constant 1 have syntactic degree 0, leaves labelled with a variable have syntactic degree 1, $\times$-gates have a syntactic degree that is the sum of the syntactic degrees of their children, and $+$-gates have a syntactic degree that is equal to the largest of the syntactic degrees of their children. The syntactic degree of a formula is defined as the syntactic degree of its output. Notice that in a formula the syntactic degree of any gate is bounded by the syntactic degree of the formula.

We will further assume that no gate computes the zero polynomial.

A formula is *homogeneous* if each gate in the formula computes a homogeneous polynomial. Equivalently, in terms of syntactic degrees, this means that all the children of a sum gate have the same syntactic degree.

Raz [22] showed how to convert a possibly inhomogeneous formula F to a homogeneous formula with a relatively small blow-up in size.

▶ **Theorem 5.** *Let F be a (non-commutative or commutative) algebraic formula of size s and product-depth Δ computing a polynomial P such that all gates in F have fan-in 2. Then there is a* homogeneous *algebraic formula F' of size at most $O\left(s \cdot \binom{\Delta+d+1}{d}\right)$ and product-depth Δ computing P. In particular, if $\Delta = O(\log s)$ and $d = O(\log s)$, then the formula F' has size* $\mathrm{poly}(s)$.

3 Main result

Proof Overview

While the proof of the main result is fairly short and (in our opinion) clean, we add some remarks here to clarify why previous depth-reduction proofs are not applicable in our setting.

The first attempt in proving Theorem 1 would be to try to use the proof strategy behind Theorem 4. Here, we start with a formula F of size s and find a subformula G of size roughly $s/2$ rooted at some gate α of F. It is not hard to show that the polynomial computed by F can be written as

$$F = G \times H_1 + H_2$$

where H_1 and H_2 are also computed by formulas of size $s/2$.[10] We then apply induction to these three subformulas to get the result. Unfortunately, this strategy does not use the degree of the formula at all, and therefore only yields a formula of depth $O(\log s)$.

In the homogeneous setting, it is sometimes more natural to do induction on the degree of the underlying formula, in which case G and H_1 (in the decomposition above) would be subformulas of degree roughly $d/2$. We get the following recursion on the worst-case size of the depth-reduced version of F, which we denote by $T(s,d)$

$$T(s,d) \leqslant T(s_1, d/2) + T(s - s_1, d/2) + T(s - s_1, d)$$

[10] Here, for simplicity, we are assuming that F is a commutative formula. In the non-commutative setting, we would instead get $F = H_1' G H_1'' + H_2$.

where s_1 denotes the size of G. Unfortunately, in this case, the formulas H_1 and H_2 *could* have size nearly s, resulting in considerable size blow-up. Indeed, when s_1 is much smaller than s (say $s_1 = s^{o(1)}$), the above recursion only yields $T(s,d) = s^{O(\log d)}$, which is a superpolynomial size blow-up.

It may be possible to interleave recursions with respect to size and depth, but we were unable to make this work.

Another possible strategy could be to follow the work of [31] which produces *circuits* of the required depth. Unfortunately, the proof of [31] is a memoization procedure, which seems to yield circuits even when applied to a formula F. Turning the resulting circuit of depth $O(\log d)$ into a formula seems to increase the size to $s^{\Omega(\log d)}$.

The approach we take is somewhat more global than the recursive strategies outlined above. Our first motivating example is seemingly the worst-case example for depth-reduction: a *comb* of depth greater than d with alternating sums and products. More formally, a comb computes the following polynomial (up to identifying variables).

$$C(x) = x_1 + (x_2 \times (x_3 + (x_4 \times \cdots)))$$

Note that the above yields a formula of depth greater than d where d is the degree of the underlying polynomial. However, we observe that such a comb actually computes a polynomial with only a few monomials, and hence can be written trivially as a depth-2 $\sum\prod$-formula without much of a size blow-up.

Building on this observation, the overall strategy is to decompose the formula into a top part G, which is a (generalized) comb, whose leaves are subformulas of F to which we will apply the same procedure recursively. We then write G as a $\sum\prod$-formula, and replace its leaves with the depth-reduced versions of the subformulas. This gives the depth-reduced version of F.

The correct definition of G is crucial, and somewhat subtle (at least to us), but with the proper definitions in place, the proof goes through without much trouble.

3.1 Proof of Theorem 1

In this section we prove our main result (Theorem 1). We start by showing a simple depth-reduction result for the case of *skew formulas.* A formula is said to be skew if every multiplication gate in it has at most 1 non-trivial child (i.e., a non-leaf node).

We will say that a leaf in a skew-formula is a $+$-leaf if the parent of that leaf is a $+$ gate and a $\times$-leaf otherwise. We show that any skew formula can be converted efficiently into a depth-2 formula, i.e., a $\sum\prod$-formula.

▶ **Lemma 6.** *Let G be a skew formula with sum-depth δ, wherein all the gates have fan-in* 2 *and the leaves are labelled by distinct variables. Then the polynomial computed by G is a multilinear polynomial with at most 2^δ monomials. Moreover any variable labelling in G*

- *a $+$-leaf,*
- *or a $\times$-leaf whose sibling is a leaf*

appears in exactly one monomial. We will call them the non-duplicable *variables.*

Proof. We prove this by induction on the depth of the formula. The base case is when the depth is 0 or 1. In both cases, the statement trivially holds.

For the induction case, assume that the depth is at least 2. Suppose $G = G_1 + G_2$. The sum-depths of G_1 and G_2 are at most $\delta - 1$. Let f_1, f_2 be the multilinear polynomials computed by G_1, G_2, respectively. We know that the leaves of G are labelled with distinct variables. Hence, G_1, G_2 have the same property and the variable sets labelling the leaves of G_1 and G_2 are disjoint. The depths of G_1 and G_2 are strictly smaller than the depth of G. By the induction hypothesis we have that f_1 and f_2 have at most $2^{\delta-1}$ monomials. Moreover, any non-duplicable variable in G_1 appears in at most one monomial in f_1. Similarly, any non-duplicable variable in G_2 appears in at most one monomial in f_2. Hence, the polynomial computed by G, i.e., $f_1 + f_2$, has at most 2^δ monomials and each non-duplicable variable appears in at most one monomial in it.

Suppose the top gate of G is a $\times$ gate. As G is a skew formula it is either $x \times G_1$ or $G_1 \times x$, where x is a variable. In particular since the depth of G is at least two, the variable x is duplicable. Let f_1 be the multilinear polynomial computed by G_1. By our assumption, the variable x does not appear in f_1. The depth of G_1 is strictly smaller than G. By the induction hypothesis we have that f_1 has at most 2^δ monomials and any non-duplicable variable in G_1 appears in at most one monomial. As x can distribute over the monomials of f_1, we have that the polynomial computed by G is multilinear with at most 2^δ monomials and any non-duplicable variable appears in at most one monomial.

(Note that this can also be seen using parse trees, since the polynomial computed is the sum of the monomials computed by the parse trees. The multilinearity is obvious. Note that parse trees do not really "branch" at multiplication gates because of the skewness and are therefore combs. To build a parse tree starting from the root we will have two choices for each addition gate we encounter on the path, and there are at most δ, so we get at most 2^δ parse trees. The only parse tree containing a given non-duplicable variable is defined by the path from the root to this leaf.) ◀

If a formula is homogeneous, it implies that for any gate α, the degree of the polynomial computed by α coincides with the syntactic degree d_α of α. Based on this remark, below we prove a stronger statement than Theorem 1. Specifically, Theorem 1 is stated for homogeneous formulas. But here, we show a depth-reduction for formulas for which the syntactic degree is small.

▶ **Theorem 7** (Refinement of Theorem 1). *Let F be an algebraic formula of size s and of syntactic degree $d_F \geqslant 2$. Then P is also computed by a formula F' of size $\mathrm{poly}(s)$ and depth $O(\log d_F)$. Moreover, if F is homogeneous, monotone and/or non-commutative, then so is F'.*

Proof. Let us start with a formula obtained from F after applying Theorem 4. That is, we will assume that we have a formula of size $\mathrm{poly}(s)$ such each gate in it has fan-in 2, sum-depth and product-depth are bounded by $O(\log s)$. For notational simplicity, from now on, F will refer to this new formula.

Let δ be a positive integer. For a formula G of syntactic degree $d_G \geqslant 1$ and sum-depth $\Delta(G)$ we define a potential function $\phi_\delta(G)$ as follows.

$$\begin{cases} \phi_{\delta,1}(G) = \lceil \log(d_G) \rceil \\ \phi_{\delta,2}(G) = \lceil \Delta(G)/\delta \rceil \end{cases}$$

and let

$$\phi_\delta(G) = \phi_{\delta,1}(G) + \phi_{\delta,2}(G).$$

We will show that the potential function bounds the depth of the depth-reduced formula that we will construct. We will also use it to bound the size of the resulting formula. Specifically, we prove the following lemma.

▶ **Lemma 8.** *Let δ be a positive integer. Any formula F of fan-in 2, syntactic degree $d \geqslant 1$, sum-depth Δ, and size s can be parallelized into a formula F' (of arbitrary fan-in) of product-depth at most $\phi_\delta(F)$ and size at most $s \cdot 2^{\delta \log(d)}$. Further, if F is homogeneous, monotone and/or non-commutative, so is F'.*

Since we ensured that the sum-depth is bounded by $O(\log s)$, taking $\delta = \left\lceil \frac{\log s}{\log d} \right\rceil$ and applying Lemma 8, we get that the final formula F' has size at most $\mathrm{poly}(s)$ and product-depth at most

$$\phi_\delta(F) = O\left(\lceil \log d \rceil + \left\lceil \log(s) \frac{\log d}{\log s} \right\rceil \right) = O(\log d).$$

By collapsing sum gates that feed into other sum gates, we see that the depth of F' can be assumed to be at most twice its product-depth, which is $O(\log d)$. This thus finishes the proof of the theorem. ◀

We now prove Lemma 8.

Proof of Lemma 8. For any gate α in F, let F_α denote the subformula rooted at α and d_α be its syntactic degree. We do the proof by induction on $\phi_\delta(F)$.

The base case $\phi_\delta(F) = 0$ trivially holds. Consider the following set of gates of F:

$$\mathcal{A} = \left\{ \alpha \mid \alpha \text{ is a not a leaf labelled 1 and } \phi_\delta(F_\alpha) < \phi_\delta(F) = \phi_\delta(F_{\text{parent}(\alpha)}) \right\}.$$

For any gate α in $\mathcal{A}$, the induction hypothesis tells us that we can construct a formula F'_α of product-depth at most $\phi_\delta(F) - 1$ and size at most $s_\alpha \cdot 2^{\delta \log(d_\alpha)}$ computing the same polynomial as F_α.

Let us consider the formula G obtained by replacing these gates from $\mathcal{A}$ in F by leaves (labelled with distinct variables). Notice that for a product-gate β in F, at most one of its children has syntactic degree larger than $d_\beta/2$, where d_β is the syntactic degree of β. Consequently, G is a skew formula. The other child of β is a $\times$-leaf in G. Moreover, G has sum-depth at most δ (since $\phi_{\delta,2}$ strictly decreases for gates below).

Hence, we can use Lemma 6 to simplify G: we get that the polynomial computed by G is a multilinear polynomial in its leaves and has 2^δ monomials. We can then write G as a $\sum\prod$-formula G' such that each duplicable gate appears in at most 2^δ monomials and each non-duplicable gate in at most 1.

The new formula F' for F is obtained from G' by replacing each variable leaf, which corresponds to some gate α in F, by its depth-reduced version F'_α constructed using the induction hypothesis above.

The product-depth of F' is bounded by the product-depth of the gates $\alpha \in \mathcal{A}$ plus the product-depth of G, which is equal to 1 after rewriting it as a $\sum\prod$-formula. That is, the product-depth is at most $(\phi_\delta(F) - 1) + 1 = \phi_\delta(F)$. By construction if G does not contain leaves labelled by 1, then it is also the case for G', otherwise, G' still has at most one such leaf. The size of the resulting formula is bounded by

$$\sum_{\alpha \text{ non-duplicable}} \left(s_\alpha \cdot 2^{\delta \log(d_\alpha)} \right) + \left(2^\delta \cdot \sum_{\alpha \text{ duplicable}} \left(s_\alpha \cdot 2^{\delta \log(d_\alpha)} \right) \right) + \mathbb{1}_{\text{G' has a constant leaf}}.$$

Notice that if α is duplicable, it must be a $\times$-leaf with its sibling β not a leaf. This means that $d_\alpha \leqslant d_\beta \leqslant d$ since the syntactic degree is maximal at the root. Hence $d_\alpha \leqslant d/2$ so the contribution of duplicable gates is bounded by

$$\sum_{\alpha \text{ duplicable}} 2^\delta \left(s_\alpha \cdot 2^{\delta(\log(d)-1)} \right) \leqslant \sum_{\alpha \text{ duplicable}} \left(s_\alpha \cdot 2^{\delta \log(d)} \right).$$

The contribution of non-duplicable gates is bounded by

$$\sum_{\alpha \text{ non-duplicable}} s_\alpha \cdot 2^{\delta \log d}$$

since any gate in F has syntactic degree at most d. By the choice of $\mathcal{A}$, the subformulas F_α are disjoint so $\sum_{\alpha\in\mathcal{A}} s_\alpha \leqslant s$, with strict inequality if G' has a constant leaf. Hence the size of F' is bounded by $s \cdot 2^{\delta \log(d)}$.

Finally, it is straightforward to verify that the construction preserves homogeneity, monotonicity and/or non-commutativity. ◀

▶ Remark 9. It is easy to note that our depth reduction procedure does not increase the syntactic degree of the formula.

We also observe that putting Theorem 1 together with Theorems 4 and 5 immediately implies Corollary 2.

Proof of Corollary 2. Given a (possibly inhomogeneous) formula F of size $s = \text{poly}(n)$ computing a polynomial of degree $d = O(\log n)$, we first apply standard depth-reduction (Theorem 4) to get an equivalent formula F_1 of size $s_1 = \text{poly}(n)$ and depth $\Delta_1 = O(\log n)$ where each gate has fan-in 2. Applying Raz's homogenization theorem (Theorem 5) to F_1 yields an equivalent *homogeneous* formula F_2 of size $s_2 = \text{poly}(n)$ and depth $\Delta_2 = O(\log n)$. We can now apply Theorem 1 to F_2 to get an equivalent formula F' of size $s' = \text{poly}(n)$ and depth $\Delta' = O(\log d)$. ◀

3.2 Reducing the size blow-up

We note that the above strategy can be easily adapted to yield a small depth formula of size s' that is nearly linear in the size s of the original formula, at the expense of increasing the depth by a large constant.

The proof is nearly identical to the proof of Theorem 1 above. The new ingredient is a near-linear depth-reduction in the setting where there is no bound assumed on the degree of the above formula. More precisely, Bshouty, Cleve and Eberly [5] (see also the work of Bonet and Buss [2]) showed the following (we sketch Bonet and Buss' proof in Appendix A for completeness).

▶ **Theorem 10** (Depth-reduction with near-linear size). *The following holds for any $\varepsilon > 0$. Let F be a (non-commutative or commutative) algebraic formula of size s computing a polynomial P. Then there is an algebraic formula F' of size at most $s^{1+\varepsilon}$ and depth $\Delta = 2^{O(1/\varepsilon)} \cdot \log s$ computing P. We may also assume that each gate in F' has fan-in 2. Furthermore, if F is homogeneous and/or monotone, then so is F'.*

Using the above result, we can prove the following improved version of our depth-reduction.

▶ **Theorem 11.** *Assume that F is a (commutative or non-commutative) formula of size s and syntactic degree $d \geqslant 1$ computing a polynomial P. Then P is also computed by a formula F'' of size at most $s^{1+\varepsilon}$ and depth $\Delta = 2^{O(1/\varepsilon)} \cdot \log d$. Furthermore, if F is a homogeneous and/or monotone formula, then so is F''.*

Proof. If $d^{4/\varepsilon} \geqslant s$, then the depth-reduction of Theorem 10 already gives a satisfactory solution. Indeed the size is bounded by $s^{1+\varepsilon}$ and the depth is bounded by

$$2^{O(1/\varepsilon)} \log s \leqslant \frac{4}{\varepsilon} 2^{O(1/\varepsilon)} \log d \leqslant 2^{O(1/\varepsilon)} \log d.$$

So we assume that $s > d^{4/\varepsilon}$. By first applying Theorem 10 (with $\varepsilon/2$ instead of ε), we obtain a formula F' of size at most $s^{1+\varepsilon/2}$, depth $\Delta' = 2^{O(1/\varepsilon)} \cdot \log s$, and fan-in 2 computing P.

Now, we apply Lemma 8, while setting $\delta = \lfloor(\varepsilon \log s)/(2\log d)\rfloor$. Notice that since $\varepsilon \log s > 4 \log d$, it ensures that $\delta > (\varepsilon \log s)/(4\log d) > 1$.

Then the strategy produces a formula F'' of product-depth at most

$$\phi_\delta(F') \leqslant \phi_{\delta,1}(F') + \phi_{\delta,2}(F') < \lceil \log d \rceil + 2^{O(1/\varepsilon)} \log s \frac{4 \log d}{\varepsilon \log s} = 2^{O(1/\varepsilon)} \log d$$

and size at most

$$s^{1+\varepsilon/2} \cdot 2^{\delta \log d} \leqslant s^{1+\varepsilon/2} s^{\varepsilon/2}.$$

This proves the theorem. ◀

3.3 Reducing the product fan-ins to 2

It is natural to ask if Theorem 1 can be proved while ensuring that the fan-in of each gate is bounded by 2, as in Theorem 1 and Theorem 10. This is not possible, as a formula of fan-in 2 and depth $O(\log d)$ can only compute polynomials on at most $\mathrm{poly}(d)$ variables, while formulas of size s may have up to s variables. However, this does not rule out reducing the fan-in of the product gates to 2. Indeed, the *circuit* depth-reduction of [31] does exactly this. We show now that this can also be done for algebraic formulas with bounded syntactic degree.

▶ **Theorem 12.** *Let F be a (commutative or non-commutative) algebraic formula F of size s, depth Δ, and syntactic degree $d \geqslant 1$ computing a polynomial P. Then P can also be computed by a formula F' of size s and depth $\Delta' = O(\Delta + \log d)$ where each product gate of F' has fan-in 2. Furthermore, if F is a homogeneous and/or monotone formula, then so is F'.*

Plugging this in Theorem 7 gives a depth-reduction to formulas of depth $O(\log d)$ and size $\mathrm{poly}(s)$ such that all product gates have fan-in at most 2. A similar result can be obtained with a smaller blow-up in size by combining this statement with Theorem 11.

Proof. It suffices to prove a weaker version of the above theorem where each product gate has fan-in at most 3. We can then replace each of the products of fan-in 3 by a tree of product gates of fan-in 2 and size 3. This has the effect of increasing the depth at most by a factor of 2, which does not affect the overall result.

So we will prove this slightly weaker version. In this setting, we will aim for a depth $\Delta' = \Delta + \log d$.

This is done by induction on the depth Δ of the formula. The case of $\Delta = 0$ is trivial. Let F be a formula of depth $\Delta > 0$ and syntactic degree d.

Assume that the output gate of F is a sum gate, and $F_1, \ldots, F_t$ are the subformulas of F of depth $\Delta - 1$. By definition, each F_i has syntactic degree at most d. Applying the induction hypothesis to each of the F_i yields a formula F'_i with product gates of fan-in at most 3. The formula F' can then be defined as the sum of these formulas.

Now we come to the main case, which is when the output gate of F is a product gate. Assume that $F_1, \dots, F_t$ are the subformulas of F of depth $\Delta - 1$ in the order[11] that they appear in F. Let d_i denote the syntactic degree of F_i. Define

$$m = \min\{j \mid \sum_{i=1}^{j} d_i \geqslant d/2\}.$$

Let F_ℓ be the formula obtained from F by keeping only the subformulas $F_1, \dots, F_{m-1}$, and F_r be the formula obtained by keeping only $F_{m+1}, \dots, F_t$. We use the induction hypothesis on F_ℓ, F_m, and F_r to get formulas F'_ℓ, F'_m and F'_r. Finally, we set

$$F' = F'_\ell \times F'_m \times F'_r.$$

The size[12] of F' is the sum of the sizes of F'_ℓ, F'_m and F'_r, which is at most s by the induction hypothesis. Let Δ'_ℓ, Δ'_m and Δ'_r denote the depths of F'_ℓ, F'_m and F'_r respectively. The depth of F' is

$$1 + \max\{\Delta'_\ell, \Delta'_m, \Delta'_r\} \leqslant 1 + \max\{\Delta + \log(d/2), \Delta - 1 + \log d, \Delta + \log(d/2)\} = \Delta + \log d$$

where the second inequality uses the induction hypothesis, and the fact that F_ℓ and F_r have syntactic degree at most $d/2$ and F_m has depth at most $\Delta - 1$. ◀

4 Tightness

Given integers $k \geqslant 1$ and $r \geqslant 2$, we will define a polynomial $H^{(k,r)}$. Intuitively, we want to define this polynomial as a standard universal polynomial for formulas. It is composed of k-nested inner products, each one of size r. In the following we will drop the superscript in $H^{(k,r)}$ and write simply H instead.

The polynomial H will be defined over the set of $(2r)^k$ variables

$$\{x_{\sigma,\tau} \mid \sigma \in [2]^k, \tau \in [r]^k\}.$$

Let us define recursively polynomials $H_{u,v}$ for all $(u, v) \in [2]^{\leqslant k} \times [r]^{\leqslant k}$ such that $|u| = |v|$:

$$H_{u,v} = x_{u,v} \qquad \text{when } |u| = |v| = k$$

$$H_{u,v} = \sum_{a=1}^{r} H_{u1,va} H_{u2,va} \qquad \text{otherwise.}$$

The polynomial H is defined as the polynomial $H_{\varepsilon,\varepsilon}$. Note that H is a polynomial of degree $d = 2^k$ and has r^{d-1} monomials.

From its definition, H is computed by a monotone formula M of size $(2r)^k$ and depth $2k$, with a $+$-gate at the top, alternating layers of $+$-gates and $\times$-gates, with $+$-gates of fan-in r and $\times$-gates of fan-in 2, and leaves labelled with distinct variables.

For words u and v over the same alphabet, we write $u \sqsupset v$ if u is a prefix of v. There is a natural one-to-one correspondance between prefixes of words of $([r] \times [2])^k$ and nodes of M, which is the following. Let $\sigma = \sigma_1 \dots \sigma_k \in [2]^k$, and $\tau = \tau_1 \dots \tau_k \in [r]^k$. The word $\tau_1\sigma_1 \dots \tau_k\sigma_k$ corresponds to a path from the root of M to the leaf labelled $x_{\sigma,\tau}$, while

[11] The order is important in the non-commutative setting.
[12] Recall that the size of a formula is the number of its leaves.

proper prefixes of $\tau_1\sigma_1 \dots \tau_k\sigma_k$ correspond to internal gates in M along this path. For $\ell < k$, $u = u_1 \dots u_\ell \in [2]^\ell$ and $v = v_1 \dots v_\ell \in [r]^\ell$, the node which corresponds to the word $v_1u_1 \dots v_\ell u_\ell$ is the $+$-gate of M computing $H_{u,v}$.

The polynomial H is easily seen to be set-multilinear with respect to the sets of variables $\{X_\sigma \mid \sigma \in [2]^k\}$ where $X_\sigma = \{x_{\sigma,\tau} \mid \tau \in [r]^k\}$. This means that each monomial has exactly one variable from each set X_σ.

▶ Remark 13. For $|u| = |v| = \ell \leqslant k$, $u \in [2]^\ell$ and $v \in [r]^\ell$, the polynomial $H^{(k,r)}_{u,v}$ is defined over the set of variables

$$X_{u,v} = \{x_{\sigma,\tau} \mid \sigma \in [2]^k,\ \tau \in [r]^k, u \sqsupset \sigma,\ v \sqsupset \tau\}$$

and is the polynomial $H^{(k-\ell,r)}$ (upto renaming of the variables). In particular, its degree is $d' = 2^{k-\ell}$, it has $r^{d'-1}$ monomials and it is set-multilinear with respect to $\{X_\sigma \mid \sigma \in [2]^k,\ u \sqsupset \sigma\}$.

Before proving hardness of H for small-depth monotone formulas, we need to show that the gates of a monotone formula computing H cannot compute too many monomials. This is proved in Lemma 15 below.

▶ **Proposition 14.** *Consider two variables $x_{\sigma,\tau}$ and $x_{\sigma',\tau'}$. If $x_{\sigma,\tau}x_{\sigma',\tau'}$ appears in a monomial of H, and if σ and σ' have a common prefix of length $\ell < k$, then τ and τ' have a common prefix of length $\ell + 1$.*

Proof. Observe that $x_{\sigma_1,\tau_1} \dots x_{\sigma_p,\tau_p}$ is a monomial of H if and only if these variables form the leaves of a parse tree of M. As observed above, the root-to-leaf path leading to the variable x_{σ_i,τ_i} in M is obtained by taking in turn the first letter of τ_i, the first letter of σ_i, the second letter of τ_i, etc.

If the product $x_{\sigma,\tau}x_{\sigma',\tau'}$ appears in a monomial of H, it must be possible to complete the union of the two paths, from the root to $x_{\sigma,\tau}$ and from the root to $x_{\sigma',\tau'}$, into a parse tree of the formula M.

If the longest common prefix of τ and τ' were of length at most ℓ, then the lowest common ancestor of $x_{\sigma,\tau}$ and $x_{\sigma',\tau'}$ in M would be a $+$-gate, which is not possible in a parse tree where each $+$-gate has a single child. ◀

▶ **Lemma 15.** *If F is a monotone formula which computes the polynomial H and if α is a gate of F of degree d_α, then the number of monomials of the polynomial computed at gate α is at most $r^{d_\alpha - 1}$.*

Proof. Since H has r^{d-1} monomials, the result is true when $d_\alpha = d$ by monotonicity. Assume now that $d_\alpha < d$.

Let

$$I = \{\sigma \in [2]^k \mid \text{some variable of } X_\sigma \text{ appears in } \alpha\}.$$

For $u \in [2]^{\leqslant k}$, let $I_u = \{\sigma \in [2]^k \mid u \sqsupset \sigma\}$. Let $\{u_1, \dots, u_p\}$ be the set of words w of minimal length such that $I_w \subseteq I$. Then I is the disjoint union $\bigcup_{\ell \in [p]} I_{u_\ell}$. Since $d_\alpha < d$, $I \neq [2]^k$ so no u_ℓ is the empty word.

Consider some $\ell \in [p]$. Let $\bar{u}_\ell$ be obtained by switching the last letter of u_ℓ. By minimality of the length of u_ℓ, we must have $I_{\bar{u}_\ell} \nsubseteq I$. Let σ be a word in $I_{\bar{u}_\ell} \backslash I$. Since F is monotone and computes a set-multilinear polynomial, it is a set-multilinear formula and therefore the polynomial computed at α must be multiplied by some variable $x_{\sigma,\tau}$. Let v_ℓ be the prefix of length $|u_\ell|$ of τ. Consider any variable $x_{\sigma',\tau'}$ appearing in α such that $u_\ell \sqsupset \sigma'$. Since the product $x_{\sigma,\tau}x_{\sigma',\tau'}$ must appear in a monomial of H by monotonicity, it must be that $v_\ell \sqsupset \tau'$ by Proposition 14.

Any monomial m in the polynomial computed in gate α can be written in a unique way $m = m_1 \cdots m_p$ with m_ℓ set-multilinear with respect to $\{X_\sigma \mid u_\ell \sqsupset \sigma\}$. By the above, m_ℓ is a monomial over the variables X_{u_ℓ,v_ℓ} of degree $|I_{u_\ell}|$. By monotonicity, it should be possible to complete leaves of M labelled with variables from m_ℓ into a parse tree of M appearing in H, which proves that m_ℓ is a monomial of H_{u_ℓ,v_ℓ}. There are at most $r^{|I_{u_\ell}|-1}$ such submonomials m_ℓ by Remark 13. It follows that the number of monomials of the polynomial computed in node α is at most

$$\prod_{\ell=1}^{p} r^{|I_{u_\ell}|-1} \leqslant r^{|I|-1} = r^{d_\alpha - 1}. \qquad ◀$$

We are ready to prove hardness of the polynomial H for monotone computation. We shall make use of the following "product lemma", which comes in different forms in e.g. [27, 10, 24].

▶ **Lemma 16.** *A degree-d homogeneous formula F of size s and product-depth Δ can be written as a sum of $O(s)$ polynomials, each of which is a product of $\Omega(\Delta d^{1/\Delta})$ many polynomials of positive degree. Moreover, each of these polynomials is computed by some gate in F.*

▶ **Proposition 17.** *If F is a monotone formula of product-depth $\Delta \leqslant \log d$ which computes H, then its size is at least $r^{\Omega(\Delta d^{1/\Delta})}$.*

Proof. Let $t = \Delta d^{1/\Delta}$. Since F is monotone, it is homogeneous and by Lemma 16 can be written as

$$F = \sum_{i=1}^{s'} \prod_{j=1}^{t_i} F_{i,j}$$

with $s' = O(s)$ and $t_i = \Omega(t)$ for all i, and $F_{i,j}$ is of degree at least 1 and computed by some gate in F.

Let $d_{i,j}$ be the degree of $F_{i,j}$. By Lemma 15, each $F_{i,j}$ computes at most $r^{d_{i,j}-1}$ monomials. The number of monomials of $\prod_{i=1}^{t_j} F_{i,j}$ is therefore bounded by

$$\prod_{j=1}^{t_i} r^{d_{i,j}-1} \leqslant r^{\sum_{j=1}^{t_i} d_{i,j} - t_i} \leqslant r^{d-t}$$

since $\sum_{j=1}^{t_i} d_{i,j} = d$. It follows that the number of monomials computed by F is at most $s \cdot r^{d-t}$. Since F computes H which has r^{d-1} monomials, we get $s = r^{\Omega(t)}$. ◀

We can now get Theorem 3 from the introduction, which is restated here for convenience.

▶ **Theorem 18.** *Let n and $d = d(n)$ be growing parameters such that $d(n) \leqslant \sqrt{n}$. Then there is a monotone algebraic formula F of size at most n and depth $O(\log d)$ computing a polynomial $P \in \mathbb{F}[x_1, \dots, x_n]$ of degree at most d such that any monotone formula F' of depth $o(\log d)$ computing P must have size $n^{\omega(1)}$.*

Proof of Theorem 3. Choose parameters $k(n)$ and $r(n)$ such that $P(n) := H^{(k,r)}$ has $\Theta(n)$ variables and degree $\Theta(d)$: let $k = \log d$ and $r = \frac{1}{2} n^{1/\log d}$. Condition $d(n) \leqslant \sqrt{n}$ ensures that $r \geqslant 2$. The polynomial P has a monotone formula of size $O(n)$ and depth $O(\log d)$. By Proposition 17, any monotone formula of product-depth $\Delta \leqslant \log d$ computing P has size

$$r^{\Omega(\Delta d^{1/\Delta})} = \left(\frac{1}{2} n^{1/\log d}\right)^{\Omega(\Delta d^{1/\Delta})} = (n/d)^{\left(\frac{\Delta d^{1/\Delta}}{\log d}\right)}$$

which is $n^{\Omega\left(\frac{\Delta d^{1/\Delta}}{\log d}\right)}$ using the hypothesis $d(n) \leqslant \sqrt{n}$. Since $\frac{\Delta d^{1/\Delta}}{\log d} \to +\infty$ when $\Delta = o(\log d)$ this bound is $n^{\omega(1)}$. ◀

5 Conclusion and Open questions

In this paper we investigated the possibility of reducing the depth of a formula of size s computing a polynomial of degree d to $O(\log d)$ while keeping the size $s^{O(1)}$.

We showed (Theorem 1) that we can do such a transformation when F is *homogeneous*. More generally, Theorem 7 states that we can achieve it as soon as the syntactic degree of F is polynomially bounded in d.

Structure inside VF

Let us consider a sequence of polynomials (f_n) whose number of variables and degree are bounded polynomially in n (such a family is usually called a p-family, see for example [6]). We can then consider three classes of such families:

- $\mathrm{homF}[s(n)] = \{(f_n) \mid f_n$ is computed by a homogeneous formula of size $\mathrm{poly}(s(n))\}$,
- $\mathrm{lowSynDegF}[s(n)] = \{(f_n) \mid f_n$ is computed by a formula of size $\mathrm{poly}(s(n))$ and of syntactic degree $\mathrm{poly}(\deg(f_n))\}$
- $\mathrm{lowDepthF}[s(n)] = \{(f_n) \mid f_n$ is computed by a formula of size $\mathrm{poly}(s(n))$ and of depth $O(\log \deg(f_n))\}$.

Clearly, we have the inclusion $\mathrm{homF}[s(n)] \subseteq \mathrm{lowSynDegF}[s(n)]$. Also, in this paper, we have shown the inclusion $\mathrm{lowSynDegF}[s(n)] \subseteq \mathrm{lowDepthF}[s(n)]$. Consequently,

$$\mathrm{homF}[\mathrm{poly}(n)] \subseteq \mathrm{lowSynDegF}[\mathrm{poly}(n)] \subseteq \mathrm{lowDepthF}[\mathrm{poly}(n)] \subseteq \mathrm{VF},$$

and we do not know if these inclusions are strict or not.

The complexity of the Elementary Symmetric Polynomials

A particularly interesting special case of the questions above comes from the example of the *Elementary Symmetric Polynomials*. Given parameters d, n with $d \leqslant n$, recall that the Elementary Symmetric polynomial $S_n^d(x_1, \ldots, x_n)$ is the sum of all the multilinear monomials of degree exactly d. A simple and elegant construction of Ben-Or (see [26]) shows that for any d, the polynomial S_n^d has an *inhomogeneous* formula of depth-3 and size $O(n^2)$. This puts this family of polynomials in the class $\mathrm{lowDepthF}[\mathrm{poly}(n)]$. Further, Shpilka and Wigderson [26, Theorem 5.3], showed that S_n^d has depth-6 formulas of syntactic degree at most $\mathrm{poly}(d)$, putting it in the class $\mathrm{lowSynDegF}[\mathrm{poly}(n)]$.

However, as far as we know, there are no known $\mathrm{poly}(n)$-sized homogeneous formulas for this family of polynomials.[13] In fact, under some further restrictions, Hrubeš and Yehudayoff showed [10] a superpolynomial homogeneous formula lower bound when $d = n/2$. Removing these restrictions would show a separation between $\mathrm{homF}[\mathrm{poly}(n)]$ and $\mathrm{lowSynDegF}[\mathrm{poly}(n)]$. On the other hand, if indeed the elementary symmetric polynomials have $\mathrm{poly}(n)$-sized homogeneous formulas, then this can be used to argue[14] the same for any polynomial computed by a depth-3 formula of polynomial size, hinting at a possible collapse between $\mathrm{homF}[\mathrm{poly}(n)]$ and $\mathrm{lowSynDegF}[\mathrm{poly}(n)]$.

[13] A strong form of this was conjectured by Nisan and Wigderson [18], which was subsequently refuted by Hrubeš and Yehudayoff [10]. However, this still does not yield polynomial-sized homogeneous formulas for all elementary symmetric polynomials.

[14] see e.g. [16, Section III] for the standard argument

Lower bounds for higher-depth formulas

Due to the recent lower bound results of [16], we know that there is an explicit homogeneous polynomial $P(X)$ of degree d on n variables that cannot be computed by any formula of size $\mathrm{poly}(n)$ and depth $\varepsilon \cdot \log\log d$, for some absolute constant $\varepsilon > 0$. It turns out that the polynomial P *is* computable by an *algebraic branching program* and therefore, lies in the complexity class called VBP.

It is known that VF is contained in VBP. However, we do not know whether this containment is strict or not. Our lower bound result helps us pose a refined version of this question. Specifically, it shows that if the lower bound from [16] can be improved from $\Omega(\log\log d)$ to $\omega(\log d)$, then we will have separated VF from VBP.

The fact that our depth reduction carries over to the non-commutative setting, makes a compelling case for revisiting the VF vs. VBP question in the non-commutative setting. Specifically, a recent result of [30] shows that there is an explicit non-commutative polynomial $P(X)$ of degree d on n variables that cannot be computed by any non-commutative formula of size $\mathrm{poly}(n)$ and depth $\varepsilon \cdot \sqrt{\log d}$. So, improving the lower bound in this case from $\Omega(\sqrt{\log d})$ to $\omega(\log d)$ would separate VF from VBP in the non-commutative setting.

References

1 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *proceedings of Foundations of Computer Science (FOCS)*, pages 67–75, 2008. doi:10.1109/FOCS.2008.32.

2 Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994. doi:10.1016/0020-0190(94)90093-0.

3 R. Brent, D. Kuck, and K. Maruyama. The parallel evaluation of arithmetic expressions without division. *IEEE Transactions on Computers*, C-22(5):532–534, 1973. doi:10.1109/T-C.1973.223757.

4 Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, April 1974. doi:10.1145/321812.321815.

5 Nader H. Bshouty, Richard Cleve, and Wayne Eberly. Size-depth tradeoffs for algebraic formulas. *SIAM J. Comput.*, 24(4):682–705, 1995. doi:10.1137/S0097539792232586.

6 Peter Bürgisser. Cook's versus Valiant's hypothesis. *Theoretical Computer Science*, 235(1):71–88, 2000.

7 Suryajith Chillara, Mrinal Kumar, Ramprasad Saptharishi, and V. Vinay. The chasm at depth four, and tensor rank: Old results, new insights. *CoRR*, abs/1606.04200, 2016. arXiv:1606.04200.

8 Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth multilinear formula lower bounds for iterated matrix multiplication, with applications. In *STACS*, volume 96 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018.

9 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM Journal of Computing*, 45(3):1064–1079, 2016. doi:10.1137/140957123.

10 Pavel Hrubeš and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Comput. Complexity*, 20(3):559–578, 2011. doi:10.1007/s00037-011-0007-3.

11 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 146–153. ACM, 2014. doi:10.1145/2591796.2591847.

12 Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theor. Comput. Sci.*, 448:56–65, 2012. doi:10.1016/j.tcs.2012.03.041.

13 S Rao Kosaraju. Parallel evaluation of division-free arithmetic equations. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 231–239, 1986.

14 Mrinal Kumar, Rafael Mendes de Oliveira, and Ramprasad Saptharishi. Towards optimal depth reductions for syntactically multilinear circuits. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 78:1–78:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.78.

15 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. *Electron. Colloquium Comput. Complex.*, TR21-081, 2021. URL: https://eccc.weizmann.ac.il/report/2021/081.

16 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 804–814. IEEE, 2021. doi:10.1109/FOCS52979.2021.00083.

17 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418, 1991.

18 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.

19 Franco P. Preparata and David E. Muller. The time required to evaluate division-free arithmetic expressions. *Inf. Process. Lett.*, 3(5):144–146, 1975. doi:10.1016/0020-0190(75)90028-9.

20 Franco P. Preparata and David E. Muller. Efficient parallel evaluation of boolean expression. *IEEE Trans. Computers*, 25(5):548–549, 1976. doi:10.1109/TC.1976.1674647.

21 Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(1):121–135, 2006. doi:10.4086/toc.2006.v002a006.

22 Ran Raz. Tensor-rank and lower bounds for arithmetic formulas. *Journal of the ACM*, 60(6):40:1–40:15, 2013. doi:10.1145/2535928.

23 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008. doi:10.1007/s00037-008-0254-0.

24 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/.

25 Eli Shamir and Marc Snir. On the depth complexity of formulas. *Math. Syst. Theory*, 13:301–322, 1980. doi:10.1007/BF01744302.

26 Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001.

27 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. doi:10.1561/0400000039.

28 Philip M. Spira. On time hardware complexity tradeoffs for Boolean functions. In Shu Lin, editor, *Proceedings of the Fourth Hawaii International Conference on System Sciences*, pages 525–527. Western Periodicals Company, North Hollywood, California, 1971.

29 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Information and Computation*, 240:2–11, 2015. doi:10.1016/j.ic.2014.09.004.

30 Sébastien Tavenas, Nutan Limaye, and Srikanth Srinivasan. Set-multilinear and non-commutative formula lower bounds for iterated matrix multiplication. In *STOC*, pages 416–425. ACM, 2022.

31 Leslie G. Valiant, Sven Skyum, Stuart J. Berkowitz, and Charles Rackoff. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM Journal of Computing*, 12(4):641–644, 1983. doi:10.1137/0212043.

32 SV Yablonskii and VP Kozyrev. Mathematical problems of cybernetics. *Information Materials of Scientific Council of Akad. Nauk SSSR on Complex Problem "Kibernetika"*, 19:3–15, 1968.

A The Bonet-Buss depth-reduction

In this section, we give a proof of Theorem 10. This is essentially the same proof as in the work of Bonet and Buss [2], but since the results in that paper are only stated for Boolean formulas, we give the proof here for completeness.

We show that for some large enough absolute constant $C > 0$, any polynomial P computed by a formula F of size s can also be computed by a formula F' of size at most $s^{1+\varepsilon}$, depth at most $2^{O(1/\varepsilon)} \cdot \log s$, and fan-in 2. Further, if F is homogeneous/monotone/non-commutative, then so is F'. This latter claim will follow directly from the construction.

We assume here that each gate in F has fan-in 2 to begin with. We can make this modification to F in the beginning at the expense of increasing the depth, and without increasing the size.

We prove the claim by induction on the size s of the formula. Let $T(s)$ and $D(s)$ denote the maximum size and depth (respectively) of F' thus obtained, assuming that F has size at most s. Let $k = 2^{C/\varepsilon}$ where $C > 0$ is an absolute constant we will choose below. We assume throughout that $\varepsilon < 1$, which is without loss of generality.

The base case of the formula is when $s \leqslant k$, in which case the claim is trivial, as any formula of size s has depth at most s, which in this case is at most $2^{C/\varepsilon}$.

Now, assume that $s > k$. In this case, we find a gate α such that the size of the subformula F_α rooted at α has size at least $s - s/k$, while the children β and γ of α do not satisfy this property. It is easy to observe that there is a unique α with this property. Let $*$ denote the operation (either $+$ or $\times$) labelling α.

We replace α by a fresh variable y in F to get a formula F_y. Note that F_y has size at most $s_1 = \frac{2s}{k}$. Since F_y has at most one occurrence of y, we can write

$$F_y = A \cdot y \cdot B + C$$

where A is the product of all subformulas that multiply y on the left in F along the path to the output, B is similarly the product of all subformulas that multiply y on the right, and C is the polynomial computed by F_y when we set y to 0. Clearly, A, B and C have formulas F_A, F_B and F_C of size at most s_1. Finally, we get the formula F' as in Figure 1.

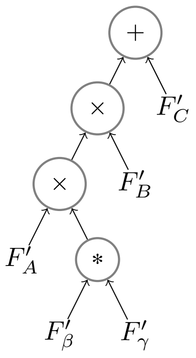

Figure 1 Constructing F'.

Here, $F'_A, F'_B, F'_C, F'_\beta, F'_\gamma$ are the formulas obtained by recursively applying the same procedure to $F_A, F_B, F_C, F_\beta, F_\gamma$.

We can bound the size and depth of the depth-reduced formula by induction. We have

$$D(s) \leqslant D(s - s/k) + O(1)$$

leading easily to an overall depth bound of $O(k \log s) = 2^{O(1/\varepsilon)} \cdot \log s$ for any choice of the constant C.

For the size, we have the following recursion.

$$T(s) \leqslant \max_{\substack{s_\beta, s_\gamma, s_1: \\ s_\beta, s_\gamma \leqslant s(1-1/k) \\ s_1 \leqslant 2s/k \\ s_\beta + s_\gamma + s_1 \leqslant s}} T(s_\beta) + T(s_\gamma) + 3T(s_1)$$

where s_β and s_γ are the sizes of F_β and F_γ respectively. We now use induction to bound $T(s)$ as follows. (We omit the conditions on s_β, s_γ and s_1 for notational simplicity.)

$$\begin{aligned} T(s) &\leqslant \max_{s_\beta, s_\gamma, s_1} s_\beta^{1+\varepsilon} + s_\gamma^{1+\varepsilon} + 3s_1^{1+\varepsilon} \\ &\leqslant \left(s\left(1 - \frac{1}{k}\right)\right)^{1+\varepsilon} + \left(\frac{s}{k}\right)^{1+\varepsilon} + 3\left(\frac{2s}{k}\right)^{1+\varepsilon} \end{aligned}$$

where we used the fact that $s_1 \leqslant 2s/k$ and the fact that, using the convexity of the map $x \mapsto x^{1+\varepsilon}$, the function $s_\beta^{1+\varepsilon} + s_\gamma^{1+\varepsilon}$ is maximized when $\max\{s_\beta, s_\gamma\} = s - s/k$, meaning that $\min\{s_\beta, s_\gamma\} \leqslant s/k$.

Continuing the computation, we get

$$T(s) \leqslant s^{1+\varepsilon}\left(\left(1 - \frac{1}{k}\right) + \frac{1}{k^{1+\varepsilon}} + 3\frac{4}{k^{1+\varepsilon}}\right) \leqslant s^{1+\varepsilon} \cdot \left(1 - \frac{1}{k} + \frac{C'}{k^{1+\varepsilon}}\right)$$

for some large enough absolute constant C'. Setting $k = 2^{C/\varepsilon}$ for a large enough absolute constant C gives us $T(s) \leqslant s^{1+\varepsilon}$, proving the inductive claim.