Electronic Colloquium on Computational Complexity, Report No. 122 (2023)

On Lifting Lower Bounds for Noncommutative Circuits using Automata

V. Arvind* Abhranil Chatterjee†

August 9, 2023

Abstract

We revisit the main result of Carmosino et al [CILM18] which shows that an $\Omega(n^{\omega/2+\epsilon})$ size noncommutative arithmetic circuit size lower bound (where ω is the matrix multiplication exponent) for a constant-degree n-variate polynomial family $(g_n)_n$, where each g_n is a noncommutative polynomial, can be "lifted" to an exponential size circuit size lower bound for another polynomial family (f_n) obtained from (g_n) by a lifting process. In this paper, we present a simpler and more conceptual automata-theoretic proof of their result.

1 Introduction

Algebraic Complexity concerns itself with the complexity of algebraic computations of multivariate polynomials. It starts with Strassen's work on matrix multiplication from the 1960's. In the 1970's, Valiant defined the algebraic complexity classes VP and VNP [Val79], which are analogues to P and NP, which brings to focus the problem of proving superpolynomial arithmetic circuit size lower bounds for an explicit polynomial family like the permanent Perm_n which is complete for VNP under projection reductions. This research area has a rich history, nicely described in the text by Burgisser et al [BCS97]. It is believed that separating VP from VNP is easier than the P vs NP problem. But the problem remains open despite intense research and highly nontrivial progress in recent years [LST21, KS18] and the $\Omega(n \log n)$ circuit size lower bound result of Baur and Strassen [BS83] remains the best known lower bound to this date.

Nisan [Nis91] initiated the study on the algebraic complexity of *noncommutative polynomials*. The noncommutative polynomial ring $\mathbb{F}\langle X\rangle$, where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of n free noncommuting variables, consists of noncommutative polynomials which are $\mathbb{F}$-linear combinations of words over X. Noncommutative arithmetic circuits computing polynomials in $\mathbb{F}\langle X\rangle$ are

*Institute of Mathematical Sciences (HBNI), and Chennai Mathematical Institute, Chennai, India. Email: arvind@imsc.res.in.

†Indian Statistical Institute, Kolkata, India. Email: abhneil@gmail.com. Research Supported by DST-INSPIRE Faculty Fellowship.

ISSN 1433-8092

defined like their commutative analogs. The only difference is that multiplication gates in the circuit are not commutative. The classes $\mathsf{VP}_{\mathsf{nc}}$ and $\mathsf{VNP}_{\mathsf{nc}}$, which are noncommutative analogs of VP and VNP, can be defined, as has been done by Hrubeš et al [sWY10]. In the same article, it is shown that Perm_n is $\mathsf{VNP}_{\mathsf{nc}}$-complete under projections. The main lower bound question is to separate $\mathsf{VP}_{\mathsf{nc}}$ and $\mathsf{VNP}_{\mathsf{nc}}$, i.e. whether the noncommutative permanent Perm_n requires superpolynomial size noncommutative arithmetic circuits. Arguably, this question should be easier in the noncommutative case. Indeed, Nisan [Nis91] has shown an exponential lower bound on the size of a noncommutative formula (more generally, a noncommutative algebraic branching program) computing the noncommutative Perm_n. However, it remains open for noncommutative circuits. Moreover, we do not have anything better than the $\Omega(n \log n)$ lower bound result of Baur and Strassen in the unrestricted setting. We note that, recently, Chatterjee and Hrubeš [CH23] have obtained a quadratic lower bound for *homogeneous* noncommutative circuits.

Why is it so difficult to obtain even a quadratic lower bound for unrestricted noncommutaive circuits? A few years ago, in 2018, Carmosino et al [CILM18] showed that an $\Omega(n^{\omega/2+\epsilon})$ circuit size lower bound[1] for a constant-degree n variate polynomial family (g_n) can be "lifted" to an exponential circuit size lower bound for a polynomial family (f_n) (which is obtained from (g_n) by the lifting process). The Carmosino et al lifting result partly explains the lack of success in showing even superlinear (in the number of variables) circuit size lower bounds for explicit polynomial families. The lifting result is reminiscent of Allender and Koucky's work in the Boolean circuit complexity setting [AK10], where the authors exploit the self-reducibility structure of some NC^1-complete problems to show that a superlinear TC^0 circuit size lower bound for them can be lifted to superpolynomial TC^0 circuit size lower bound.

Before we present the contribution of this paper, it is worth mentioning a similar result due to Hrubeš, Wigderson, and Yehudayoff [HWY10] which indeed predates [CILM18]. They show that a super-linear lower bound on the *width* of an explicit degree 4 polynomial can be lifted to an exponential circuit size lower bound for an explicit noncommutative polynomial.

This paper In this paper, we present a simple and a more structured automata-theoretic argument for the Carmosino et al result [CILM18] stated above. In their paper, the main idea is to use an encoding scheme that reduces the number of variables exponentially incurring only a polynomial blow-up in the degree. The core of the argument is to show the following:

Lemma 1 (Informal)**.** *A noncommutative circuit can be decoded efficiently.*

In this paper, we prove this using ideas from algebraic automata theory. The main two ingredients of our proof are to show (a) an efficient representation of a decoder using a weighted automaton, and (b) the use of the Hadamard product to construct the decoded circuit. Our proof is not only short and

[1] Here ω is the matrix multiplication exponent.

simple but also conceptually more satisfying. We highlight two consequences for different choices of parameters (details in Section 3.3):

- Let (g_N) be an explicit noncommutative p-family, where $\deg(g_N) = t$ for some constant t for each N, such that $\mathfrak{C}(g_N) \geq \Omega(N^{\omega/2+\epsilon})$, where $\epsilon > 0$ is a constant. Then there is an explicit p-family $(h_n)_n$ in $\mathsf{VNP_{nc}}$ such that (h_n) requires circuits of size $n^{\Omega(n)}$.
- Suppose (g_N) is an explicit noncommutative p-family, where each $\deg(g_N) = (\log N)^{O(1)}$, and g_N requires circuits of size $\omega(N^{\omega/2} \cdot \log N)$. Then there is an explicit p-family (h_n) in $\mathsf{VNP_{nc}}$ such that $\mathfrak{C}(h_n) = n^{\omega(1)}$.

2 Preliminaries

We recall some algebraic complexity definitions for noncommutative computation. Further details on these definitions and basic results can be found in Nisan's seminal paper [Nis91].

Definition 2 (Noncommutative Arithmetic Circuit)**.** Let $\mathbb{F}$ be a field. A *noncommutative arithmetic circuit* C over $\mathbb{F}$ and noncommuting indeterminates $x_1, x_2, \ldots, x_n$ is a directed acyclic graph (DAG) with each node of indegree zero labeled by a variable or a scalar constant from $\mathbb{F}$: the indegree 0 nodes are the input nodes of the circuit. Internal nodes are gates of the circuit, and are of indegree two. They are labeled either by a + or a $\times$ (indicating the gate type). Furthermore, the two inputs to each $\times$ gate are designated as left and right inputs prescribing the order of gate gate multiplication. Each internal gate computes a polynomial (by adding or multiplying its input polynomials), and the polynomial computed at an input node is just its label. A special gate of C is designated the *output*. The polynomial computed by the circuit C is the polynomial computed at its output gate. An arithmetic circuit is a *formula* if the fan-out of every gate is at most one. For a polynomial $f \in \mathbb{F}\langle X\rangle$ we denote by $\mathfrak{C}(f)$ its optimal circuit size.

We recall some more definitions from Burgisser's text [Bür00, sWY10, AJR18].

Definition 3 (p-family)**.** Let $\mathbb{F}$ be a field. A sequence of multivariate noncommutative polynomials (f_n) over $\mathbb{F}$ is called a *p-family* if there is a polynomial n^c that bounds both the degree and number of variables in f_n for each n. Suppose $f_n \in \mathbb{F}\langle X_n\rangle$ for each n. The p-family (f_n) is *explicit* if there is a polynomial-time algorithm that takes as input a monomial $m \in X_n^*$ and computes its coefficient in f_n, for all n, and in time polynomial in n. For example, the permament polynomial $(\mathrm{Perm}_n)_n$ is an explicit p-family.

Remark 4. In the definition of an explicit p-family, the running time of the algorithm that computes the coefficient of a monomial $m \in X_n^*$ is polynomial in the length of m encoded in some *fixed alphabet* like, for example, the binary alphabet. This point is important when we consider p-families –as indeed we will need to for the lower bound lifting result– $(g_n)_n$ of constant degree polynomials where $\deg(g_n) \leq t$ for t independent of n.

Some notation that we will use in this paper: for a polynomial $f \in \mathbb{F}\langle X\rangle$ its support $\text{supp}(f) = \{w \in X^* \mid \text{coefficient of } w \text{ is} \neq 0\}$ is the set of monomials with nonzero coefficient in f. Thus, letting f_w denote the coefficient of w in f, we can write $f = \sum_{w\in\text{supp}(f)} f_w w$.

Definition 5 (Formal Power Series)**.** Let X be a set of free noncommuting variables and $\mathbb{F}$ be any field. A *formal power series* is a function $f : X^* \to \mathbb{F}$, where X^* is the free monoid of all words (i.e. monomials) over X. We can equivalently denote the power series f by the formal infinite sum $\sum_{w\in X^*} f(w)w$. The set of formal power series form a ring $\mathbb{F}\langle\langle X\rangle\rangle$ over $\mathbb{F}$ known as the power series ring. Ring addition here is coefficient-wise and ring multiplication is the standard convolution product.

We recall the definition of a weighted automata [DK21] with some basic details. Let $\mathcal{A}$ be a finite state automaton with state set Q with designated start state s and final state t. Let R be any ring. Then $\mathcal{A}$ is an R-weighted automaton if the transition function

$$\delta : Q \times Y \times Q \to R$$

assigns to every transition (q_1, y, q_2) a weight $r_y \in R$. Consequently, every monomial $w = y_1y_2\cdots y_r \in Y^*$ along an s to t transition path P in the automaton $\mathcal{A}$ is assigned a weight $r_P \in R$ (which the product of the individual weights for each transition step). The actual weight r_w associated with monomial w is $r_w = \sum_P r_P$, where the sum is over all s to t transition paths P for the monomial w (and $r_w = 0$ if there are no such paths). We define the formal power series

$$\sum_{w\in Y^*} r_w w$$

to be the power series computed by the weighted automaton $\mathcal{A}$. Equivalently, for each variable $y \in Y$ we have its $|Q| \times |Q|$ state transition matrix $M_y \in \mathcal{M}_{|Q|}(R)$. The $(i,j)^{th}$ entry of M_y is the element $\delta(i, y, j) \in R$. Then, corresponding monomial $w = y_1y_2\cdots y_d \in Y^*$, the transition matrix is the matrix product

$$M_w = \prod_{j=1}^{d} M_{y_j},$$

and the coefficient r_w of monomial w in the power series computed by $\mathcal{A}$ is the $(s,t)^{th}$ coefficient $M_w[s,t]$ of M_w.

3 Lower Bounds via Efficient Decoding

The proof of the lower bound lifting result [CILM18] can be described quite simply using some automata theoretic arguments. It is based on a simple encoder and decoder which can be described using a weighted automata. We present the details in this section.

3.1 Hadamard Product Computation

The notion of Hadamard product is well-studied in algebraic automata theory [BR11, Theorem 5.5]. It has also been used for noncommutative polynomials to obtain some algebraic complexity results [AJS09, AMS10, AS10].

For the purpose of this paper, we define the Hadamard product of a noncommutative polynomial computed by a circuit and a formal series computed by a small automaton.

Definition 6. Let $f \in \mathbb{F}\langle X\rangle$ be a degree-d polynomial and S be a formal power series in $\mathbb{F}\langle\langle X\rangle\rangle$, where X is a finite set of free noncommuting variables. The *Hadamard product* of f and S is the noncommutative polynomial

$$f \circ S = \sum_{m \in X^{\leq d}} [m]f \cdot [m]S \cdot m,$$

where $[m]f$ and $[m]S$ denote the coefficients of the word m in f and in S, respectively.

We recall the following result showing efficient Hadamard product computation when the polynomial is computable by a small circuit and the series by a small automaton.

Theorem 7. [AS18] *Given a circuit C and an automaton B computing a homogeneous degree-k polynomial $f \in \mathbb{F}\langle X\rangle$ and a formal series $S \in \mathbb{F}\langle\langle X\rangle\rangle$ respectively, the Hadamard product polynomial $f \circ S$ can be evaluated at any point $(a_1, a_2, \ldots, a_n) \in \mathbb{F}^n$ by evaluating $C(a_1M_1, a_2M_2, \ldots, a_nM_n)$ where $M_1, M_2, \ldots, M_n$ are the transition matrices of B, and the dimension of each M_i is the size of B.*

If C is given by black-box access then $(f \circ S)(a_1, \ldots, a_n)$ for $a_i \in \mathbb{F}, 1 \leq i \leq n$ can be evaluated by evaluating C on matrices defined by the automaton B [AS18] as follows: For each $i \in [n]$, the transition matrix M_i in $\mathcal{M}_s(\mathbb{F})$ are computed from the automaton B (which is of size s) that encodes layers. We define $M_i[k, \ell] = [x_i]L_{k,\ell}$, where $L_{k,\ell}$ is the linear form on the edge (k, ℓ). Now to compute $(f \circ S)(a_1, a_2, \ldots, a_n)$ where $a_i \in \mathbb{F}$ for each $1 \leq i \leq n$, we compute $C(a_1M_1, a_2M_2, \ldots a_nM_n)$. The value $(f \circ S)(a_1, a_2, \ldots, a_n)$ is the $(1, s)^{th}$ entry of the matrix $f(a_1M_1, a_2M_2, \ldots, a_nM_n)$.

Theorem 7 can be used to efficiently compute a circuit for the Hadamard product polynomial $f \circ S$. Replace each x_i by y_ix_i in the automaton B. Let $M_1, \ldots, M_n$ in be the transition matrices where each entry is a linear form in Y variables. We can now compute $f \circ S$ by evaluating $C(M_1, \ldots, M_n)$ on the matrices $M_i, 1 \leq i \leq n$. In this evaluation each multiplication gate of the circuit C actually denotes matrix multiplication. Hence we have the following.

Theorem 8. *Given a noncommutative circuit of size s' computing a degree k polynomial $f \in \mathbb{F}\langle X\rangle$ and an automaton of size s computing a formal series $S \in \mathbb{F}\langle\langle X\rangle\rangle$, we can compute a noncommutative circuit of size $s's^{\omega}$ for the noncommutative polynomial $f \circ S$ in deterministic time $s's^{\omega} \cdot \mathsf{poly}(n, k)$, where ω denotes the matrix multiplication exponent.*[2]

[2]The current best algorithm for matrix multiplication, which is due to Alman and Williams [AW21], shows $\omega < 2.373$.

3.2 An Efficient Decoder using Weighted Automata

We first define the encoding scheme. Let $X = \{x_0, x_1, \ldots, x_{n-1}\}$, $Y = \{y_0, y_1, \ldots, y_{m-1}\}$ be disjoint sets of noncommuting variables and let X^* and Y^* denote the free monoids of words/monomials in X and Y, respectively.

A *monoid homomorphism* is a mapping

$$h : X^* \to Y^*$$

such that $h(\epsilon) = \epsilon$ and $h(ww') = h(w)h(w')$, where we denote the empty word universally by ϵ.

A mapping $h : X \to Y^*$ is *prefix-free*if for any $x, x' \in X$ $h(x)$ is not a proper prefix of $h(x')$. Any such prefix-free mapping h can be uniquely extended to an injective monoid homomorphism $h : X^* \to Y^*$, and we refer to it as an *encoder*. We will first consider the following simple encoder.

Definition 9 (Encoder)**.** Let $X = \{x_0, x_1, \ldots, x_{n-1}\}$, $Y = \{y_0, y_1, \ldots, y_{m-1}\}$ be disjoint sets of noncommuting variables where $n = m^3$. For each $i \in \{0, 1, \ldots, n-1\}$ let $j_i k_i \ell_i$ denote the base-m representation of i, where each $j_i, k_i, \ell_i \in \{0, 1, \ldots, m-1\}$. The encoder is the monoid homomorphism $\mathcal{E} : X^* \to Y^*$ that uniquely extends the substitution map $\mathcal{E}(x_i) = y_{j_i} y_{k_i} y_{\ell_i}$.

The encoder $\mathcal{E} : X^* \to Y^*$ of Definition 9 naturally extends by linearity to polynomials. Thus, $\mathcal{E} : \mathbb{F}\langle X\rangle \to \mathbb{F}\langle Y\rangle$ encodes noncommutative polynomials in X into noncommutative polynomials in Y.

The decoder automaton

A decoder $\mathcal{D} : Y^* \to X^*$ is a map such that $\mathcal{D}(\mathcal{E}(m)) = m$ for all monomials $m \in Y^*$. By linearity, for any polynomial $h \in \mathbb{F}\langle X\rangle$ we have $\mathcal{D}(\mathcal{E}(h)) = h$.

As summarized in the following lemma,, it is convenient to formally use weighted automata to describe the decoder corresponding to $\mathcal{E}$. Let the ring R be the free noncommutative polynomial ring $\mathbb{F}\langle X\rangle$. Assume that the elements of $\mathbb{F}\langle X\rangle$ commute with variables in Y. Then the formal series which defines the decoder $\mathcal{D}$ is $\sum_{u\in X^*} u\mathcal{E}(u)$. Notice that in this formal series, for $w = \mathcal{E}(u)$ we have $r_w = u$ and $r_w = 0$ for all $w \in Y^*$ not in the range of the encoder $\mathcal{E}$.

Lemma 10. *The series $S = \sum_{w\in X^*} w\mathcal{E}(w) \in \mathbb{F}\langle X\rangle\langle\langle Y\rangle\rangle$ is computable by an $\mathbb{F}\langle X\rangle$-weighted automaton of size $2(m+1)$, which is the decoder $\mathcal{D}$ corresponding to the encoder $\mathcal{E}$, and $m = |Y|$.*

Proof. As $xy = yx$ for all $x \in X$ and $y \in Y$, we observe that the power series $S = \sum_{u\in X^*} u\mathcal{E}(u)$ has the following simple expression:

$$S = \left(\sum_{i=1}^{n} x_i \mathcal{E}(x_i)\right)^* .$$

Now, consider the following automaton A of size $2m+2$ (see Figure 1).

We describe the automaton in some detail because in Section 4 we will discuss this further. The automaton has four layers. The initial layer has just

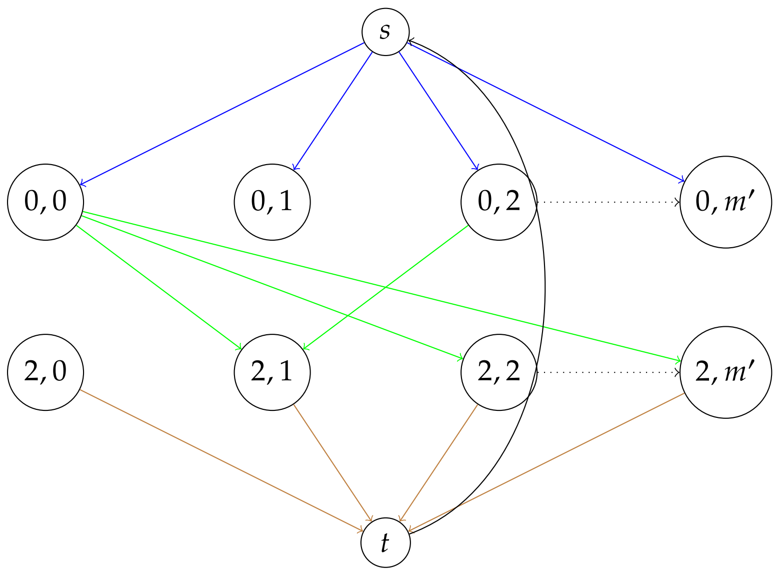

Figure 1: The transition diagram of the automaton A

the start state s. The second and third layers each have m states. The final layer has just the final state t from which the automaton loops back to the start state s on an ϵ-transition[3].

We now describe the role of the states in the second and third layers of the automaton.

Let $T = \{0, 1, \ldots, m-1\}$. For each $j \in T$, we define a transition from state s to state $(0, j)$ reading y_j (the state $(0, j)$ encodes the symbol y_j it has seen previously) and $(2, j)$ to t reading y_j (the state $(2, j)$ encodes the symbol y_j it will see next).

The transitions between the second and third layers is where the decoding actually happens. Between any pair of states $(0, i)$ in the second layer and $(2, j)$ in the third layer, $i, j \in T$, the automaton has a weighted transition on input $y_k, k \in T$ which has weight $x_{\sigma(i,j,k)}$, where $\sigma : \{0, 1, \ldots, m-1\}^3 \rightarrow \{0, 1, \ldots, n-1\}$ is the bijection

$$\sigma(i, j, k) = m^2 i + mk + j.$$

Notice that between $(0, i)$ and $(2, j)$ we have m transitions, one for each $y_k, k \in T$. The simple information-theoretic idea in this construction is that the states $(0, i)$, $(2, j)$ and the transition on y_k hold the complete information about the string $y_i y_j y_k$ which the decoder can substitute with $x_{\sigma(i,j,k)}$. □

Remark 11. We refer to the above encoder as the 1-to-3 encoder. In Section 4, where we discuss possibilities of improvements to the lower bound lifting result, we will consider the more general 1-to-r encoder.

[3]Strictly speaking we should remove the ϵ-transition and directly go to state $(0, j)$ in the second layer on reading y_j

3.3 The Lower Bound Lifting Result

We are now ready to present the automata-theoretic proof of the lower bound lifting result of [CILM18]: namely, that a circuit size lower bound of $\Omega(n^{\omega/2+\epsilon})$ for an explicit p-family (g_n) *of degree-t polynomials* can be "lifted" to obtain an exponential circuit size lower bound for an explicit p-family (h_n). Notice that the definition of explicit p-families applies to the constant-degree p-family (g_n) in the sense explained in Remark 4.

The result is an easy consequence of Theorem 8. In fact we will show stronger result, as the simple analysis in the proof goes through for the choice of $t = O(\log n)$ and $\epsilon = O(\log\log n/\log n)$.

We begin with showing that the decoder $\mathcal{D}$ preserves circuit size quite efficiently.

Lemma 12 (efficient decoding)**.** *For a noncommutative polynomial $h \in \mathbb{F}\langle X\rangle$ suppose its encoding $\mathcal{E}(h) \in \mathbb{F}\langle Y\rangle$ has a noncommutative circuit of size s. Then h has a noncommutative circuit of size bounded by $m^\omega \cdot s$, where $m = |Y|$. More precisely,*

$$\mathfrak{C}(h) \leq O(m^\omega)\cdot\mathfrak{C}(\mathcal{E}(h)).$$

Proof. The idea is to use the weighted automaton of Lemma 10 which defines the decoder $\mathcal{D}$ which computes the formal series S. We first observe the following easy claim, that the Hadamard product $\mathcal{E}(h)\circ S$ evaluated at $y_j = 1, 0 \leq j \leq m-1$ is precisely $h(X)$.

Claim 13. $h(X) = (\mathcal{E}(h)\circ S)(1,1,\ldots,1)$.

Writing $h = \sum_{w\in\mathrm{supp}(h)} h_w \cdot w$, notice that we have $\mathcal{E}(h) = \sum_{w\in\mathrm{supp}(h)} h_w \cdot \mathcal{E}(w)$. Thus we have

$$\mathcal{E}(h)\circ S = \sum_{w\in\mathrm{supp}(h)} h_w \cdot w \cdot \mathcal{E}(w),$$

noting that we are considering S as a formal series in the Y variables with coefficients as polynomials in the X variables. Thus, the evaluation of $\mathcal{E}(h)\circ S$ for Y variables substituted with 1 will yield $h = \sum_{w\in\mathrm{supp}(h)} h_w \cdot w$. This proves the claim.

As the size of the decoder automaton in Lemma 10 is $2m+2$, the proof of the lemma follows from Theorem 8 which gives the claimed bound on the circuit size of the Hadamard product of a circuit with a weighted automaton. ◻

Theorem 14. *Let $(g_n)_n$ be an explicit noncommutative p-family, where* $\deg(g_n) = t$ *for some constant t for each n, such that $\mathfrak{C}(g_n) \geq \Omega(n^{\omega/2+\epsilon})$, where $\epsilon > 0$ is a constant. Then there is an explicit p-family $(h_n)_n$ in $\mathsf{VNP_{nc}}$ where h_n is n-variate with* $\deg(h_n) = \mathsf{poly}(n)$ *such that $\mathfrak{C}(h_n) = n^{\Omega(n)}$.*

Proof. Set $d = \lceil\log_3 n\rceil$ and $N = n^{3^d}$. By assumption we have $\mathfrak{C}(g_N) = \Omega(N^{\omega/2+\epsilon})$, where $\deg(g_N) = t$. By a d-fold application of the encoder $\mathcal{E}$ to the polynomial g_N, we obtain the polynomial

$$h_n = \mathcal{E}^d(g_N),$$

where $h_n \in \mathbb{F}\langle Y_d \rangle$, letting Y_d denote the set of noncommuting variables in the output polynomial produced by d applications of the encoder $\mathcal{E}$.

In general, for $1 \leq k \leq d$ notice that $\mathcal{E}^k(g_N) \in \mathbb{F}\langle Y_k \rangle$, where Y_k is a set of $N_k = n^{3^{d-k}}$ noncommuting variables, and the degree of $\mathcal{E}^k(g_N)$ is $t \cdot 3^k$. Notice that $N_{k+1}^3 = N_k$ for each $k \geq 1$ and $|Y_d| = N_d = n$. Therefore, $h_n(Y_d)$ is an n-variate polynomial of degree precisely $t3^d = tn$.

Claim 15. $\mathfrak{C}(h_n) = n^{\Omega(n)}$.

We will prove the claim by an inductive argument. More precisely, note that $\mathcal{E}^0(g_N) = g_N$ and $\mathcal{E}^d(g_N) = h_n$. Let $n_k = \epsilon(N) \cdot 3^k, 0 \leq k \leq d$. By assumption, we have $\mathfrak{C}(\mathcal{E}^0(g_N)) = \mathfrak{C}(g_N) = \Omega(N^{\omega/2+\epsilon(N)}) = \Omega(N^{\omega/2+n_0})$.

Suppose, as induction hypothesis that $\mathfrak{C}(\mathcal{E}^k(g)) = \Omega(N_k^{\omega/2+n_k})$. Then, by Lemma 12 we have

$$\mathfrak{C}(\mathcal{E}^k(g_N)) \leq \alpha \cdot \mathfrak{C}(\mathcal{E}(\mathcal{E}^k(g_N))) \cdot N_{k+1}^{\omega},$$

for some constant $\alpha > 1$. That implies

$$\mathfrak{C}(\mathcal{E}^{k+1}(g_N)) \geq \frac{\alpha N_k^{\omega/2+n_k}}{N_{k+1}^{\omega}} = \frac{\alpha N_k^{\omega/2+n_k}}{N_k^{\omega/3}} = \alpha N_{k+1}^{\omega/2+n_{k+1}}.$$

Putting it together, therefore, $h_n = \mathcal{E}^d(g_N)$ is n-variate in the variables Y_d of degree $t3^d = t \cdot n = \mathsf{poly}(n)$ and

$$\mathfrak{C}(h_n) = \mathfrak{C}(\mathcal{E}^d(g_N)) = \Omega(n^{\omega/2+3^d\epsilon}) = n^{\Omega(n)}.$$

This completes the proof. □

In the above proof, if we let t be a function of N, notice that choosing $t(N) = (\log N)^c$ with other parameters remaining the same, still guarantees $(h_n)_n$ to be an explicit p-family with $\deg(h_n) = \mathsf{poly}(n)$ and the lower bound holds for $\mathfrak{C}(h_n)$ as well. Furthermore, suppose we allow ϵ to be a variable quantity and set $\epsilon = \omega\left(\frac{\log\log N}{\log N}\right)$.[4] Then the lower bound assumption becomes

$$\mathfrak{C}(g_N) = \Omega(N^{\omega/2+\epsilon(N)}) = \omega(N^{\omega/2} \cdot \log N),$$

where g_N is of degree $(\log N)^c$. In particular, this assumption is weaker than that of Theorem 14. Following the analysis in the proof of Theorem 14 we obtain the following

Corollary 16. *Let $(g_N)_N$ be an explicit noncommutative p-family, where $\deg(g_N) = (\log N)^c$ for constant $c > 0$ and each n, such that $\mathfrak{C}(g_N) = \omega(N^{\omega/2} \cdot \log N)$. Then there is an explicit p-family $(h_n)_n$ in $\mathsf{VNP}_{\mathsf{nc}}$ where h_n is n-variate with $\deg(h_n) = \mathsf{poly}(n)$ such that $\mathfrak{C}(h_n) = n^{\omega(1)}$.*

[4] Here $\omega(\cdot)$ is the standard asymptotic notation and not the matrix multiplication exponent.

4 Discussion

Can this lower lifting result be improved? As noted in [CILM18], the hardness assumption becomes $\mathfrak{C}(g_N) = N^{1+\epsilon}$ if the matrix multiplication exponent $\omega = 2$. Furthermore, the hardness assumption in Corollary 16 becomes $\omega(N \log N)$ for a degree $(\log N)^{O(1)}$ polynomial. Baur and Strassen's lower bound is $\Omega(N \log d)$ for an explicit degree-d N-variate polynomial. Compared to that the $\omega(N \log N)$ lower bound assumption translates to $\omega(Nd^{\alpha})$ for some $\alpha > 0$. Can the degree bound of $(\log N)^{O(1)}$ be relaxed in Corollary 16?

We crucially use the Hadamard product construction described in Lemma 8, for which the circuit upper bound is $O(s'^{\omega} s)$ where s' and s are the given automaton and circuit sizes respectively. Matrix multiplication is inherent here. For, suppose there was a Hadamard product construction with circuit upper bound $O(s'^{\alpha} s^{\beta})$. Now, we can easily reduce the multiplication of two $s' \times s'$ matrices to the Hadamard product of an automaton of size $O(s')$ and a circuit of size $s = O(1)$. Hence, it follows that $\alpha = \omega$.

Another place where there is arguably some room for improvement is in the choice of the encoder function and decoder automaton construction (Lemma 10). We note that the decoder automaton of size $2m + 2$ for the 1-to-3 decoder is already optimal to a constant factor. This is because we cannot have a $o(m)$ size automaton for $\mathcal{D}$ due to simple information-theoretic reasons. To see this, we observe that the decoder has to output a variable $x_{\sigma(i,j,k)} \in X$ on a single transition edge, call it $e = (s_1, s_2)$. But that means the information in the states s_1, s_2 and the input read on the transition must contain the complete information about the triple (i, j, k), where $i, j, k \in \{0, 1, \ldots, m-1\}$ which is impossible if there are only $o(m)$ many states as the number of edges need to be $\Omega(m^2)$.

The one-shot decoder and directly lifted lower bound Finally, we note that instead of using 1-to-3 decoder d times we can directly decode $\mathcal{E}^d$ which uniquely encodes each $x_i, 1 \le i \le N = n^{3^d}$ into a string in Y^{3^d}, where $Y = \{y_1, y_2, \ldots, y_n\}$. Let $\mathcal{D}^d$ denote the corresponding decoder. An automaton for $\mathcal{D}^d$ of size $2n^{(3^d-1)/2} + 2$ can be constructed exactly on the same lines as Lemma 10. The automaton has four layers. The first has the start state s and the last has the final state t. The second and the third layers have $n^{(3^d-1)/2}$ states each. From the start state the automaton reads a prefix of length $(3^d - 1)/2$ and remembers it in the state s_1 that it reaches in the second layer. Likewise, each state s_2 in the third layer corresponds to a suffix of length $(3^d - 1)/2$. The transition (s_1, s_2) reads the middle letter which, together with s_1 and s_2, describes the entire word over Y of length 3^d. This automaton has $M = 2n^{(3^d-1)/2} + 2$ states. Now, applying Lemma 8 we get

$$\mathfrak{C}(g_N) \le O(M^{\omega}) \cdot \mathfrak{C}(\mathcal{E}^d(g_N)) = O(M^{\omega}) \cdot \mathfrak{C}(h_n).$$

As $N = n^{3^d}$, by substituting we obtain $\mathfrak{C}(h_n) \ge n^{3^d \epsilon + \omega/2} = n^{\Omega(n)}$ for constant ϵ, which proves Theorem 14.

References

[AJR18] Vikraman Arvind, Pushkar S. Joglekar, and Gaurav Rattan. On the complexity of noncommutative polynomial factorization. *Inf. Comput.*, 262:22–39, 2018.

[AJS09] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic Circuits and the Hadamard Product of Polynomials. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 25–36, 2009.

[AK10] Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *J. ACM*, 57(3):14:1–14:36, 2010.

[AMS10] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New Results on Noncommutative and Commutative Polynomial Identity Testing. *Computational Complexity*, 19(4):521–558, 2010.

[AS10] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 677–686, 2010.

[AS18] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. *Computational Complexity*, 27(1):1–29, 2018.

[AW21] Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.

[BCS97] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997.

[BR11] J. Berstel and C. Reutenauer. *Noncommutative Rational Series with Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011.

[BS83] Walter Baur and Volker Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[CH23] Prerona Chatterjee and Pavel Hrubes. New Lower Bounds Against Homogeneous Non-Commutative Circuits. In *38th Computational Complexity Conference, CCC 2023, July 17-20, 2023, Warwick, UK*, volume 264 of *LIPIcs*, pages 13:1–13:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[Bür00] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*, volume 7 of *Algorithms and computation in mathematics*. Springer, 2000.

[CILM18] Marco L. Carmosino, Russell Impagliazzo, Shachar Lovett, and Ivan Mihajlin. Hardness Amplification for Non-Commutative Arithmetic Circuits. In *Proceedings of the 33rd Computational Complexity Conference*, CCC '18, Dagstuhl, DEU, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[DK21] Manfred Droste and Dietrich Kuske. Weighted automata. In *Handbook of Automata Theory*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.

[KS18] Neeraj Kayal and Chandan Saha. Guest Column: A Paradigm for Arithmetic Circuit Lower Bounds. *SIGACT News*, 49(1):55–65, 2018.

[LST21] 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.

[Nis91] Noam Nisan. Lower Bounds for Non-Commutative Computation (Extended Abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991.

[sWY10] Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Relationless Completeness and Separations. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, June 9-12, 2010*, pages 280–290, 2010.

[HWY10] Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Non-Commutative Circuits and the Sum-of-Squares Problem. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 667–676, New York, NY, USA, 2010. Association for Computing Machinery.

[Val79] Leslie G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261, 1979.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il