Hardness Amplification for Non-Commutative Arithmetic Circuits

Marco L. Carmosino[1]
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
marco@ntime.org

Russell Impagliazzo[2]
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
russell@cs.ucsd.edu

Shachar Lovett[3]
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
slovett@ucsd.edu

Ivan Mihajlin[4]
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
imikhail@cs.ucsd.edu

Abstract

We show that proving mildly super-linear lower bounds on non-commutative arithmetic circuits implies exponential lower bounds on non-commutative circuits. That is, non-commutative circuit complexity is a threshold phenomenon: an apparently weak lower bound actually suffices to show the strongest lower bounds we could desire.

This is part of a recent line of inquiry into why arithmetic circuit complexity, despite being a heavily restricted version of Boolean complexity, still cannot prove super-linear lower bounds on general devices. One can view our work as positive news (it suffices to prove weak lower bounds to get strong ones) or negative news (it is as hard to prove weak lower bounds as it is to prove strong ones). We leave it to the reader to determine their own level of optimism.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases arithmetic circuits, hardness amplification, circuit lower bounds, non-commutative computation

Digital Object Identifier 10.4230/LIPIcs.CCC.2018.12

1 Introduction

Arithmetic circuits are a natural computational model for computing polynomials, which has been extensively studied in complexity theory. Most of the research is focused on proving lower bounds. Namely, showing that certain "hard" polynomials (such as the permanent, which is complete for an arithmetic version of NP [17]) require large arithmetic circuits. Despite much research, strong lower bounds are only known for restricted families of circuits, such as circuits of fixed depth, multi-linear circuits, or monotone circuits. For general airthmetic circuits, the best lower bound known is still the classical result of Baur-Strassen [5]

[1] Supported by the Simons Foundation
[2] Supported by the Simons Foundation
[3] Supported by NSF CAREER award 1350481 and CCF award 1614023
[4] Supported by the Simons Foundation

33rd Computational Complexity Conference (CCC 2018).
Editor: Rocco A. Servedio; Article No. 12; pp. 12:1–12:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

who showed that to compute $X_1^n + \ldots + X_n^n$ one needs an arithmetic circuit of size $\Omega(n \log n)$. We refer to the recent survey [16] and the references within for details about these many works.

An interesting restriction of the arithmetic model, which is the focus on this paper, is that of *non-commutative polynomials* and correspondingly *non-commutative circuits*. A non-commutative polynomial over a field $\mathbb{F}$ in variables $X_1, \ldots, X_n$, is a linear combination of monomials, except that here monomials are defined as words over the variables. Otherwise put, variables do not commute, so the order of variables in a monomial is important. Despite this severe restriction, the non-commutative setting maintains complexity-theoretic structure: the permanent is complete for non-commutative arithmetic NP [10] (VNP), and natural polynomials are complete for non-commutative arithmetic P [3] (VP). The hope is that it will be easier to prove strong lower bounds against non-commutative circuits, as various cancellations that occur in standard (commutative) arithmetic circuits crucially depend on the commutativity of the variables. For example, the $n \times n$ determinant can be computed by a $O(n^3)$ arithmetic circuit, but to the best of our knowledge, there is no non-commutative arithmetic circuit for determinant of size $n^{o(n)}$. Moreover if determinant can be computed by polynomial size non-commutative circuits then $VP = VNP$ [4] [5].

If one restricts attention further to non-commutative *formulas*, then our understanding is dramatically better. A fundamental result in this area is a theorem of Nisan [14], who proved exponential lower bounds on non-commutative formulas. For example, his technique applied to the $n \times n$ permanent (or also the $n \times n$ determinant) shows that any non-commutative formula computing either of them requires size $\Omega(2^n)$. On the other hand, no lower bounds for non-commutative circuits are known which are better than these known for standard commutative circuits. This dichotomy leads to the main problem motivating this paper, posed by [11]:

> Why do we have exponential lower bounds for non-commutative formulas, but only marginally super-linear lower bounds for non-commutative circuits?

The main message of the this paper is that weak lower bounds for non-commutative circuits can be "amplified" to arbitrarily large polynomial, or even exponential, lower bounds for non-commutative circuits. One can view this as positive news (it suffices to prove weak lower bounds to get strong ones) or negative news (it is as hard to prove weak lower bounds as it is to prove strong ones). We leave it to the reader to make their own choice. Below, we state the formal versions of our main results.

We recall the standard notation that ω is the best known exponent for matrix multiplication, where the best known upper bounds on it are $\omega \leq 2.374$ due to [13]. Our first theorem is that polynomial lower bounds better than $\Omega(n^{\omega/2})$ for any non-commutative polynomial in n variables of polynomial degree can be lifted to arbitrary polynomial lower bounds.

▶ **Theorem 1.** *Let $\varepsilon > 0$. Assume that there exists an explicit non-commutative polynomial in n variables of degree poly(n), such that any non-commutative circuit computing it requires size $\Omega(n^{\omega/2+\varepsilon})$.*

Then, for any $c > 1$, there exists another explicit polynomial in m variables of degree poly(m), such that any non-commutative circuit computing it requires size $\Omega(m^c)$.

Some authors [11] had suggested that for non-commutative polynomials number of variables might be not the best parameter. In contrast with the commutative setting, one

[5] Formally, one needs to define a non-commutative determinant, by inducing some natural order on the variables in monomials of the standard commutative permanent.

can focus on polynomials with a constant number of variables, using the degree as a measure. The difference comes from the fact that there are 2^d different non-commutative monomials on 2 variables of degree d versus $d+1$ for the commutative case. For this regime, the best known lower bounds are of the form $\Omega(\log(d))$ where d is the degree. Theorem 1 states that if we have good enough lower bound to start with, we can give a family of polynomials of complexity $\Omega(d)$. We will, however, use number of variables as our measure, as we will be dealing with constant degree polynomials.

Our second theorem shows that proving lower bounds better than $\Omega(n^{\omega/2})$ for any *constant degree* non-commutative polynomial in n variables can be lifted to exponential lower bounds. This may help to explain why no super-linear lower bound for a constant-degree non-commutative polynomial is currently known. The polynomial that we start with in this case must be *explicit*, a notion of uniformity described in section 2.

▶ **Theorem 2.** *Let $\varepsilon > 0$. Assume that there exists an explicit non-commutative polynomial in n variables of constant degree, such that any non-commutative circuit computing it requires size $\Omega(n^{\omega/2+\varepsilon})$.*

Then, for some $c > 0$, there exists another explicit polynomial in m variables of degree poly(m), such that any non-commutative circuit computing it requires size $\exp(m^c)$.

Here is one way to interpret our results, which we find intriguing: proving any super-linear lower bound $\Omega(n^{1+\varepsilon})$ against non-commutative circuits would imply one of two things: (i) an arbitrarily large polynomial lower bound (or even better) against non-commutative circuits; or (ii) a proof that $\omega > 2$, namely, a super-linear lower bound for (standard, commutative) matrix multiplication.

1.1 Technique

Our main technical result is a lifting theorem, which allows us to amplify lower bounds against non-commutative circuits, by reducing the number of variables without hurting the lower bound too much.

Let f be a non-commutative polynomial over variables $X_1, \ldots, X_n$. Fix a constant integer $r \geq 1$ and assume that $n = m^r$. Define new variables $Y_{i,j}$ where $i = 0, \ldots, (r-1)$ and $j = 0, \ldots, (m-1)$. We will encode each X_i as a monomial $Y_{0,a_0} Y_{1,a_1} \ldots Y_{(r-1),a_{r-1}}$, where $a_0 \ldots a_{r-1}$ is the encoding of i in base-m. Let $E(f)$ denote the polynomial obtained by doing this replacement to each variable in f. Note that $E(f)$ is a polynomial over the rm variables $\{Y_{i,j}\}$ of degree $\deg(E(f)) = r \deg(f)$.

Our main technical lemma (lemma 4) shows that any non-commutative circuit C which computes $E(f)$ can be transformed to another non-commutative circuit C' which computes f. We think of this as "decoding" the circuit for the encoding $E(f)$ to a circuit for f. Moreover, the size of C' is not much larger than C. The optimal parameters are achieved by taking $r = 3$, using fast matrix multiplication; they give that $\text{size}(C') \leq \text{size}(C) \cdot n^{\omega/3}$.

Otherwise put, if f requires arithmetic circuits of size s, then $E(f)$ requires arithmetic circuits of size $s \cdot n^{-\omega/3}$. However, $E(f)$ has only marginally higher degree and many fewer variables $m = n^{1/3}$. Applying this idea iteratively, we make progress as long as $s \gg n^{\omega/2}$. This implies both of our main theorems (Theorem 1 and Theorem 2).

For our generic technique to go through, we need to "massage" non-commutative circuits for $E(f)$ so that they can be "decoded" into non-commutative circuits for f. Basically, we want all the gates in the circuit to compute polynomials over $\{Y_{i,j}\}$ that are encoding of polynomials over $\{X_i\}$. We accomplish that by several rounds of simplification of the structure of the circuit. This can be seen as an analog to the homogenization process performed on algebraic circuits, except that in our case, the process is more delicate.

1.2 Related Work

This work parallels that of Hrubeš, Wigderson and Yehudayoff [11]. They showed that if any *explicit* degree 4 polynomial has a strong enough super-linear lower bound on *width*, then this lower bound can be lifted to an exponential circuit lower bound for a non-commutative polynomial. We refer the reader to the original paper for the formal definition of width. To compare these results with ours, note that implicit in [11] is the relationship $\frac{s}{n^2} \leq w(P) \leq O(ns)$, where P is any degree 4 polynomial, w is the "width" of this polynomial, and s is the minimal size of a circuit computing P. Thus, [11] shows that any super-cubic circuit lower bound for an explicit polynomial of degree 4 implies exponential circuit lower bounds for some explicit polynomial.

We show that one can start from circuit lower bounds of the form $n^{\frac{\omega}{2}}$ against *any* constant degree polynomial and lift to exponential circuit lower bounds. Moreover, even lower bounds against higher degree polynomials can be lifted.

As in [11], we give new structural properties of non-commutative circuits computing restricted polynomials. The restrictions of [11] force polynomials to form monomials by selecting each variable from some sets of variables that always appear in a fixed order of some fixed length. Our restrictions allow the sets of variables to have a *periodic* ordering, according to $\mathbb{Z}/r$ for some r. This allows our structures to easily generalize to higher degrees.

An encoding of variables similar to our lifting was used previously in [2], as a step in randomized polynomial identity testing for *sparse* non-commutative circuits. The work of [3] uses a similar double-indexed "positional" encoding of monomials, to establish a transfer theorem from "f is complete for a non-commutative algebraic class" to "decoded(f) is complete for a commutative algebraic class."

There has been a great deal of recent interest in understanding why it is hard to prove lower bounds in the arithmetic setting, even though it is more restricted than the Boolean setting. Analogs of the Natural Proofs barrier of [15] have been proposed in [7] and [8], and an unconditional barrier for rank-based methods was just shown by [6]. Our result is most similar to the "chasm" family of results [1, 12, 9]: they show that one "only" needs to prove depth-3 lower bounds to prove general super-polynomial lower bounds. We show that, in the non-commutative case, one "only" needs to prove mildly super-linear lower bounds to prove super-polynomial lower bounds.

Organization

In section 2, we formally define lifting, state the key "circuit decoding" lemma, and show how the results follow. In section 3, we prove the decoding lemma by giving new structural results about non-commutative circuits.

2 Preliminaries

Polynomials and Circuits

Let $X = \{x_1, \ldots, x_n\}$ be a set of variables and let $\mathbb{F}$ be a field. We denote by $\mathbb{F}\langle X\rangle$ the set of non-commutative polynomials over X with coefficients in $\mathbb{F}$. These polynomials sum over monomials that are *words* over X, because multiplication of variables does not commute. We define circuits computing polynomials from $\mathbb{F}\langle X\rangle$ in the natural way: as directed acyclic graphs with internal nodes (gates) labeled by $+, \times$ and leaves labeled by $x \in X$ or field elements. Each $+, \times$ gate has two children, and each $\times$ gate has distinguished left and right children. Denote by $\mathrm{AC}(f)$ the arithmetic complexity of a non-commutative polynomial f, as the minimal number of gates in a non-commutative circuit computing f.

Explicitness

It is easy to prove that some polynomials require exponential size circuits. So we restrict ourselves to the set of explicit polynomials. A polynomial f is ***explicit*** if and only if each of its coefficients can be computed in polynomial time in the description length of a monomial. Thus, the coefficients of an explicit constant degree polynomial can be computed in polylogarithmic time.

3 Lifting Polynomials

We define polynomial lifting and give basic properties. Unless otherwise stated, all our polynomials and circuits are non-commutative. We consider both singly-indexed variables $X = \{x_i\}$ and doubly-indexed variables $Y = \{y_{i,j}\}$. To ease work over Y, define sets Y_i as $\{y_{i,j}\}_{j\in\mathbb{N}}$, the sets of all y variables with first index i. We use the notation $\mathbb{F}\langle X\rangle = \mathbb{F}\langle x_1, \ldots, x_n\rangle$ to denote non-commutative polynomials over the X variables, and analogously for polynomials over the Y variables.

Lifting takes a polynomial over the X variables to a polynomial over the Y variables. Starting with $f \in \mathbb{F}\langle x_1, \ldots, x_n\rangle$, we replace each x_i by a product of y variables that encodes i in base $n^{1/r}$, rounding up to ensure that $n^{1/r}$ is an integer. To simplify notations, we will always assume that $n = m^r$ for some integer m, so no rounding will be necessary. Since the y variables do not commute, the resulting polynomial can easily be mapped back to f by reading "sub-words" of monomials base $n^{1/r}$ to recover which x_i a string of y variables represents. To formalize this below, we use $\text{digit}(t, i, j)$ to refer to the jth digit of the base-t representation of i.

▶ **Definition 3** (Lifting). Let $f \in \mathbb{F}\langle X\rangle$. Define $L_r(f) \in \mathbb{F}\langle Y\rangle$ by applying the following map to each variable of f:

$$x_i \to \prod_{j=0}^{(r-1)} y_{j,\ell_j} \quad \text{where } \ell_j = \text{digit}(n^{1/r},\ i,\ j)$$

This means that $L_r(f)$ will be over $rn^{1/r}$ variables $(y_{0,1}, \ldots, y_{r-1,n^{1/r}})$. If the degree of f is d, then the degree of $L_r(f)$ is dr. So lifting shrinks the number of variables while increasing the degree.

Lifting preserves explicitness. Suppose we want to compute a coefficient of $L_r(f)$. Let's assume there is an algorithm that takes a description of a monomial of f and outputs the coefficient on it in time t. Then one can use the same algorithm to compute coefficients of $L_r(f)$, as the description of a monomial and it's lifted version is *exactly* the same.

Our main technical lemma, proved in Section 4, efficiently converts a circuit for the lifted polynomial $L_3(f)$ into a circuit for f. Setting $r = 3$ is easiest to present, and gives the best qualitative bounds that we know how to achieve with this technique. So we continue with this choice of r below.

▶ **Lemma 4** (Circuit Decoding). *If there exists an arithmetic circuit of size s computing $L_3(f)$, then there exists a circuit of size $O(n^{\omega/3}s)$ computing f.*

The lifting operation can be iterated. Take a polynomial $L_r(f) \in \mathbb{F}\langle Y\rangle$ and re-number the Y variables lexicographically to obtain new singly-indexed X variables, and lift the resulting polynomial again. The result of repeating this process k times on a polynomial f is denoted $L_r^k(f)$. Using the circuit-decoding Lemma 4 we have the following lower-bound amplification for iterated lifting.

▶ **Lemma 5** (Iterated Lifting Amplifies Hardness). *Let $k \leq \gamma \log(n)$ be a positive integer, where $\gamma > 0$ is a sufficiently small positive constant. Suppose f is a polynomial on $N = 3^{3/2}(3^{1/2}n)^{3^k}$ variables of degree d. Then $L_3^k(f)$ is a polynomial on $9n$ variables of degree $3^k d$ and the following holds:*

$$AC(L_3^k(f)) \geq \frac{AC(f)}{N^{\omega/2}}$$

If we have a small circuit for $L_3^k(f)$ then by applying Lemma 4 iteratively k times we will end up with a small circuit for f. We require N to be in a particular form to avoid dealing with rounding. The calculations appear below.

Proof of Lemma 5. Let N_k denote the number of variables of $L_3^k(f)$, where one can verify that $N_0 = N$ and $N_{i+1} = 3N_i^{\frac{1}{3}}$. Our choice for N guarantees that N_i is an integer for all $i = 0, \ldots, k$. Using Lemma 4 we get

$$\mathrm{AC}(L_3^{i+1}(f)) \geq \alpha \frac{\mathrm{AC}(L^i)}{N_i^{\omega/3}},$$

where $\alpha > 0$ is some absolute constant. Folding the recursion gives

$$\mathrm{AC}(L_3^k(f)) \geq \alpha^k \mathrm{AC}(f) \prod_{i=0}^{k-1} N_i^{-\omega/3}$$

We will need to use an explicit expression for $N_i = 3^{3/2}(3^{-3/2}N)^{\frac{1}{3^i}}$.

$$\begin{aligned}\mathrm{AC}(L_3^k(f)) &\geq \alpha^k \mathrm{AC}(f)(\prod_{i=0}^{k-1} 3^{3/2}(3^{-3/2}N)^{\frac{1}{3^i}})^{-\omega/3} \\ &= \alpha^k \mathrm{AC}(f)(3^{\frac{3}{4}(2k-3+3^{1-k})} N^{\frac{3}{2}(1-3^{-k})})^{-\omega/3}\end{aligned}$$

So:

$$\mathrm{AC}(L_3^k(f)) \geq \frac{\mathrm{AC}(f)}{N^{\omega/2}} \Big(\frac{\alpha^{\frac{3k}{\omega}} N^{\frac{3^{k-1}}{2}}}{3^{\frac{3}{4}(-3+3^{1-k}+2k)}} \Big)^{\omega/3}$$

If we recall that $k \leq \gamma \log(n)$ and choose γ small enough we can ensure that:

$$\alpha^{-\frac{3k}{\omega}} 3^{\frac{3}{4}(-3+3^{1-k}+2k)} < N^{\frac{3}{2}3^{-k}}$$

As left hand side is $2^{\theta(k)}$ and right hand side is $n^{\theta(1)}$. This immediately implies:

$$\mathrm{AC}(L_3^k(f)) \geq \frac{\mathrm{AC}(f)}{N^{\omega/2}}$$ ◀

3.1 Amplifying Lower Bounds via Lifting

Theorem 1 (amplification to any fixed polynomial hardness) is straightforward, by taking k to be some large constant in Lemma 5 above:

Proof of Theorem 1. Let $P = \{P_n\}$ be a family of explicit polynomials, where P_n is a polynomial on n variables, such that $\exists \alpha, \epsilon > 0$ such that $\forall n$: P_n is not computable by arithmetic circuits of size $\alpha n^{\frac{\omega}{2}+\epsilon}$. We will define family of polynomials $Q = \{Q_n\}$ to be

lifted version of P, where again Q_n is a polynomial on n variables. Formally, $Q_{9n} = L_3^k(P_N)$, where $N = N(n) = 3^{3/2}(3^{\frac{1}{2}}n)^{3^k}$. It is easy to verify that N is always an integer. For general n define $Q_n = Q_{9\lfloor n/9 \rfloor}$ by adding dummy variables. By Lemma 5:

$$AC(Q_n) \geq \frac{AC(P_N)}{N^{\frac{\omega}{2}}} \geq \alpha N^{\epsilon} \geq \alpha 3^{3/2\epsilon}(3^{1/2}n)^{\epsilon 3^k} = n^{\Omega(3^k)}$$

For any $c > 0$ we can take k to be sufficiently large constant and have $AC(Q_n) > n^c$. Furthermore, note that $\deg(Q_{9n}) = 3^k \deg(P_N)$. So if $\deg(P_N) = O(N^a)$ is polynomial in N, then $\deg(Q_n) = O(3^k n^{a3^k})$. In particular, for any fixed k, $\deg(Q_n) = \text{poly}(n)$ as claimed. Also Q is explicit as it is a lifted version of P. ◀

Proof of Theorem 2. The proof is identical to the proof of Theorem 1, except that we take $k = \gamma \log n$. Note that as we assume here that P_n all have a constant degree, then Q_n will have degree $\text{poly}(n)$ as claimed. As P is an explicit polynomial, Q is also explicit polynomial. ◀

4 Structuring Circuits

In this section we obtain a normal form for non-commutative circuits computing certain restricted types of polynomials. The idea is similar to homogenization: we classify monomials into "types" and efficiently re-write the circuit in terms of operations on those types. The proofs share a common structure: we define an operator that splits polynomials into well-typed monomials. We then pass this operator through the circuit C layer-by-layer, starting from the output gate. Each time we advance the operator-layer through C, we maintain:

- **(i)** The polynomial computed by C does not change;
- **(ii)** All gates above the operator-layer compute restricted polynomials;
- **(iii)** Not too much additional hardware is introduced;
- **(iv)** At leaf nodes, operators can be eliminated from C.

This process is like a glacial movement during the ice age. An operator slides over the circuit and then disappears, drastically changing the landscape behind it.

4.1 Monomial & Circuit Types

For non-commutative polynomials, monomials are just words over the variables. So all of our monomial types will be constraints on the ordering of variables, referring to the "place" part of a Y variable.

▶ **Definition 6** (Structured Monomials in Y)**.** For fixed $r \in \mathbb{N}$, we define the following subsets of all monomials over double-indexed variables Y.

***r*-pinned, $\widetilde{\mathcal{M}}^r_{i,j}$:** monomials m that start with $y \in Y_i$, end with $y' \in Y_j$, and obey $\mathbb{Z}/r$ ordering. That is, after each $y \in Y_k$ appearing in m the next variable is always some $y' \in Y_{(k+1) \bmod r}$.

***r*-aligned, $\widetilde{\mathcal{M}}^r$:** any $m \in \widetilde{\mathcal{M}}^r_{0,(r-1)}$.

We do not bound the lengths of pinned or aligned monomials. The counter k indexing sets of variables Y_k may circle around $\mathbb{Z}/r$ many times in going from i to j. We classify circuits and polynomials in the obvious way based on these sets of monomials.

▶ **Definition 7** (Structured Polynomials in Y)**.** A polynomial $p \in \mathbb{F}\langle Y \rangle$ is **r-pinned** if $\exists i, j$ such that every monomial of p is in $\widetilde{\mathcal{M}}^r_{i,j}$, or **r-aligned** if every monomial of p is in $\widetilde{\mathcal{M}}^r$.

When r is clear from the context, we shorthand $\widetilde{\mathcal{M}}_{i,j} = \widetilde{\mathcal{M}}^r_{i,j}$ and $\widetilde{\mathcal{M}} = \widetilde{\mathcal{M}}^r$.

▶ **Definition 8** (Structured Circuits in Y)**.** A circuit C is **r-pinned** if every gate of C computes an r-pinned polynomial. Note that each gate could have different start and end indices i, j. C is **r-aligned** if every gate of C computes an r-aligned polynomial. An r-aligned circuit has r-aligned monomials as inputs, not single variables.

Recall that our goal in this section is to build a circuit for f from a circuit for $L_r(f)$. If we have an r-aligned circuit of size s for $L_r(f)$, this is straightforward. The bottom layer of an r-aligned circuit is a set of monomials, not variables. Since these monomials are r-aligned, each one uniquely represents a sequence of natural numbers in base $n^{1/r}$. Simply replace each encoded number i with x_i and take their product. After this substitution, we have a circuit that computes f of size $O(s)$.

If some *general* circuit C computes an *aligned* polynomial f, we can obtain an *aligned* circuit C' for f of only slightly larger size. This construction proceeds in two stages: from general circuits to pinned circuits (lemma 9), then from pinned circuits to aligned circuits (lemmas 10 and 11).

The circuit decoding for lifted polynomials of Lemma 4 is then immediate, because $L(f)$ is always an aligned polynomial. We give two constructions: the first is elementary but inefficient, the second uses fast matrix multiplication to optimize storage of "type information" about polynomials. The first stage, from general to pinned circuits, is common to both proofs.

4.2 Operators on Polynomials

To efficiently store polynomials, we will sometimes need to "trim off" extraneous variables from the ends of each monomial. So we give two new operators on polynomials, $\div_{\mathcal{L}}$ and $\div_{\mathcal{R}}$, that "divide what they can and discard the remainder." These operators act on the left and right of f, respectively. Formally, $\div_{\mathcal{L}}$ and $\div_{\mathcal{R}}$ are defined in terms of two possible decompositions of a polynomial f:

- **Right division:** Let $f = Q \times \sigma + R$ where $Q \times \sigma$ sums over monomials of f *ending* with σ. Define: $f \div_{\mathcal{R}} \sigma = Q$.
- **Left division:** Let $f = \tau \times Q' + R$ where $\tau \times Q'$ sums over monomials of f *starting* with τ. Define: $\tau \div_{\mathcal{L}} f = Q'$.

Because our polynomials are non-commutative, these decompositions are unique. Notice that in left-division $\div_{\mathcal{L}}$, the monomial τ is *not* the object being operated on; it appears on the left to denote which side of the monomials of f is altered by the operation. Immediately, we have:

$$p \times q = \sum_{a \in Y} (p \div_{\mathcal{R}} a) \times (a \times q) = \sum_{a \in Y} (p \times a) \times (a \div_{\mathcal{L}} q)$$

Finally, we denote by $\mathcal{M}$ the set of all possible monomials, and by $\text{coeff}(f, m)$ the coefficient of f on monomial m. When expanding polynomials as sums over monomials, we write the monomial m as x^m or y^m, like so:

$$f(Y) = \sum_{m \in \mathcal{M}} \text{coeff}(f, m) \times y^m$$

4.3 Aligning Circuits

We begin the alignment process by taking a general circuit for a pinned polynomial, and constructing a pinned circuit. The is similar to homogenization using the more complex set of monomial types introduced above.

▶ **Lemma 9** (General to Pinned Circuits). *Let C be a **general** arithmetic circuit of size s computing an **r-pinned** polynomial $f(Y)$. Then there exists an **r-pinned** arithmetic circuit C' of size $r^3 s$ computing f.*

Proof of Lemma 9. Define $\Delta_{i,j}$ to transform $f(Y)$ into a r-pinned polynomial, by discarding any coefficients on monomials outside $\widetilde{\mathcal{M}}_{i,j}$:

$$\Delta_{i,j}(p) = \sum_{m \in \widetilde{\mathcal{M}}_{i,j}} \text{coeff}(p, m) \times y^m$$

Let g_o be the output gate of C. By assumption, g_o computes an r-pinned polynomial. From the definition, $\exists i, j$ such that $\Delta_{i,j}(g_o) = g_o$. This is our base case. Inductively, let $g \in C$ be such that $\exists i, j$ so $\Delta_{i,j}(g) = g$. We reason by cases on the type of g.

If $g = u + v$:

$$\begin{aligned}
\Delta_{i,j}(u+v) &= \Delta_{i,j}\left(\sum_{m \in \mathcal{M}} (\text{coeff}(m,u) + \text{coeff}(m,v))y^m\right) && \text{expand } u+v \\
&= \sum_{m \in \widetilde{\mathcal{M}}_{i,j}} (\text{coeff}(m,u) + \text{coeff}(m,v))y^m && \text{definition of } \Delta \\
&= \sum_{m \in \widetilde{\mathcal{M}}_{i,j}} \text{coeff}(m,u)y^m + \sum_{m \in \widetilde{\mathcal{M}}_{i,j}} \text{coeff}(m,v)y^m && \text{split the sum} \\
&= \Delta_{i,j}(u) + \Delta_{i,j}(v) && \text{definition of } \Delta
\end{aligned}$$

If $g = u \times v$:

$$\begin{aligned}
\Delta_{i,j}(u \times v) &= \Delta_{i,j}\left(\sum_{m_\ell \in \mathcal{M}} \text{coeff}(m_\ell, u)y^{m_\ell} \times \sum_{m_r \in \mathcal{M}} \text{coeff}(m_r, v)y^{m_r}\right) && \text{unroll} \\
&= \Delta_{i,j}\left(\sum_{\substack{m_\ell \in \mathcal{M} \\ m_r \in \mathcal{M}}} \text{coeff}(m_\ell, u)y^{m_\ell} \,\text{coeff}(m_r, v)y^{m_r}\right) && \text{distribute} \\
&= \Delta_{i,j}\left(\sum_{\substack{m_\ell \in \mathcal{M} \\ m_r \in \mathcal{M}}} \text{coeff}(m_\ell, u)\,\text{coeff}(m_r, v)y^{m_\ell m_r}\right) && \text{commute in } \mathbb{F} \\
&= \sum_{\substack{m_\ell, m_r \in \mathcal{M} \\ st.\ m_\ell m_r \in \widetilde{\mathcal{M}}_{i,j}}} \text{coeff}(m_\ell, u)\,\text{coeff}(m_r, v)y^{m_\ell m_r} && \text{definition of } \Delta
\end{aligned}$$

Because $m_\ell m_r$ is pinned, we know (1) that m_ℓ begins with some $y \in Y_i$ and m_r ends with some $y' \in Y_j$ and (2) that the transition from m_ℓ to m_r must respect ordering in $\mathbb{Z}/r$. Formally, we know that $\exists t$ such that $m_\ell \in Y_i \dots Y_t$ and $m_r \in Y_{(t+1) \bmod r} \dots Y_j$. So let's split the above summation on this index, which is bounded by r because we assumed the polynomial is r-pinned. To ease legibility below, all indexing arithmetic for monomial sets $\widetilde{\mathcal{M}}$ and for the operator Δ is *implicitly* carried out in $\mathbb{Z}/r$.

$$
\begin{aligned}
g &= \sum_{t\in\mathbb{Z}/r} \sum_{\substack{m_\ell\in\widetilde{\mathcal{M}}_{i,t}\\ m_r\in\widetilde{\mathcal{M}}_{t+1,j}}} \operatorname{coeff}(m_\ell,u)\operatorname{coeff}(m_r,v)y^{m_\ell m_r} \\
&= \sum_{t\in\mathbb{Z}/r} \left(\sum_{m_\ell\in\widetilde{\mathcal{M}}_{i,t}} \operatorname{coeff}(m_\ell,u)y^{m_\ell}\right)\left(\sum_{m_r\in\widetilde{\mathcal{M}}_{(t+1),j}} \operatorname{coeff}(m_r,v)y^{m_r}\right) && \text{distribute} \\
&= \sum_{t\in\mathbb{Z}/r} \Delta_{i,t}(u)\times\Delta_{t+1,j}(v) && \text{definition of } \Delta
\end{aligned}
$$

The circuit C' contains, for every gate g in C, the r^2 gates computing $\Delta_{i,j}(g)$ for all $i,j\in\mathbb{Z}/r$. Addition gates do not require additional gates; multiplication gates require an addition a factor of r more gates to compute. So in total if C has s gates then C' has at most $r^3 s$ gates. ◀

The pinning lemma proved above enforces an ordering on variables that respects $\mathbb{Z}/r$. But for circuit decoding, monomials that are *aligned* and thus represent complete numbers are required.

We partition pinned monomials into a prefix, body and suffix. The body of a monomial is the substring between the first variable from Y_0 and the last variable from $Y_{(r-1)}$ (it can be empty). By definition, the length of the body is a multiple of r. This means that the body uniquely represents a string of natural numbers, which can easily be mapped back to x-variables.

Then the prefix of a monomial is everything to the left of the body, and the suffix is everything to the right of the body. We also need to consider monomials of small length, for which the body is undefined. These parts of a monomial do not *yet* represent even a single natural number. But, because the circuit computes an aligned polynomial, we know that these monomials will eventually becomes part of the body via subsequent multiplication operations.

4.3.1 Simple Circuit Alignment

The construction below anticipates and brute-forces these possible "completions" of non-body monomials at each gate of the circuit.

▶ **Lemma 10** (Pinned to Aligned Circuits, Simply)**.** *Let C be an **r-pinned** arithmetic circuit of size s computing an **r-aligned** polynomial $f(Y)$. Then there exists a **r-aligned** arithmetic circuit C' of size $O(sn^{3r-2})$ computing f. If C was a monotone circuit, then C' is also monotone.*

Proof. First, we define the undesirable sets of monomials. These monomials are all possible obstructions to alignment that must be computed in terms of aligned polynomials.

$$
\begin{aligned}
&\text{Incomplete} : I = \{\rho \mid \rho\in\mathcal{M} \text{ of length } < r\} \\
&\text{Prefix} : P = \left(\cup_{i=1}^{r-1}\widetilde{\mathcal{M}}_{i,(r-1)}\right)\cap I \\
&\text{Suffix} : S = \left(\cup_{i=0}^{r-2}\widetilde{\mathcal{M}}_{0,i}\right)\cap I
\end{aligned}
$$

We use these monomial sets to separate the body of a monomial from the prefix and suffix, which are not perfectly aligned:

$$\mathcal{W}_{\sigma,\tau} = \{(w,m) \mid w = \sigma m \tau \text{ where } m \in \widetilde{\mathcal{M}},\ \sigma \in \text{Prefix},\ \text{and } \tau \in \text{Suffix}\}$$

We want all the polynomials computed by C' to be aligned, so we can only have monomials with empty prefix, suffix, and incomplete monomial sets at each gate. But we need the coefficients associated with these "flawed" polynomials to compute with. This suggests an operator Γ that will take only parts of the polynomial with a particular suffix and prefix, multiplying the coefficient on $\sigma m \tau$ by the monomial m only, where m is a body monomial. We will also need to recover coefficients on incomplete monomials, so we let a unary Γ extract specific coefficients: $\Gamma_\rho(f) = \text{coeff}(\rho, f)$. We could also use the "division" operators above to express Γ:

$$\begin{aligned}\Gamma_{\sigma,\tau}(f) &= \sum_{(w,m)\in\mathcal{W}_{\sigma,\tau}} \text{coeff}(w,f) \times y^m \\ &= \sigma \div_{\mathcal{L}} f \div_{\mathcal{R}} \tau\end{aligned}$$

If a polynomial f is aligned, then $\Gamma_{1,1}(f) = f$ and all other operators are 0. That means that if g_o is the output gate of the original circuit C, then $\Gamma_{1,1}(g_o) = g_o$. Inductively, let C be the pinned circuit computing an aligned polynomial f and suppose $g \in C$. We are going to show how to push Γ operators one level deeper into the circuit, reasoning by cases on the form of f.

Suppose $g = u + v$. Addition does not change the collection of monomials except by cancellation, so we have the following easy identities, which follow from the same kind of monomial partitioning used to prove the pinning Lemma 9 above:

$$\begin{aligned}&\forall a,b:\ \Gamma_{a,b}(g) = \Gamma_{a,b}(u) + \Gamma_{a,b}(v)\\ &\forall c:\ \Gamma_c(g) = \Gamma_c(u) + \Gamma_c(v)\end{aligned}$$

Now suppose $g = u \times v$. First consider how some incomplete monomial c could have a nonzero coefficient in g; it would have to be the case that two incomplete monomials of u and v were multiplied together to form c. Therefore:

$$\forall c,\ \Gamma_c(g) = \sum_{\{d,e\in I\ \mid\ de=c\}} \Gamma_d(u)\Gamma_e(v).$$

Similarly, we reason by cases on how the monomials of $\Gamma_{a,b}(g)$ could have been formed by multiplying the monomials of u and v:

$$\begin{aligned}\Gamma_{a,b}(g) = &\sum_{\{c\in S,\ d\in P\ :\ |cd|=r\}} \Gamma_{a,c}(u) y^{cd} \Gamma_{d,b}(v) && \text{// suffix}(u) \times \text{prefix}(v) \text{ becomes aligned}\\ &+ \sum_{\{c\in S,\ d\in I\ :\ cd=b\}} \Gamma_{a,c}(u)\Gamma_d(v) && \text{// suffix}(u) \times \text{incomplete}(v) \text{ becomes } b\\ &+ \sum_{\{c\in I,\ d\in P\ :\ cd=a\}} \Gamma_c(u)\Gamma_{d,b}(v) && \text{// incomplete}(u) \times \text{prefix}(v) \text{ becomes } a\end{aligned}$$

The above formula completely enumerates how the polynomials u and v could multiply to produce coefficients on monomials with prefix and suffix a, b in g, in terms of Γ applied to u and v. Thus we have successfully expressed $\Gamma_{a,b}(g)$ in terms of earlier gates.

Using the formulas above we can push the Γ-operators down one level. Clearly, all gates above the operator level compute aligned polynomials: we are keeping track of undesirable monomials in the labels on gates. Finally, observe that all the operators applied to a single variable or constant are constants. This means we can replace every operator applied to the input of the circuit by a constant. So after pushing the operators down to the leaves we get an aligned circuit that computes $\Gamma_{1,1}(g_o) = f$.

All that remains is to estimate the size of the resulting circuit. Each addition gate of the old circuit was substituted with a circuit of size n^{2r-2}. Each multiplication gate of the old circuit was substituted with a circuit of size n^{3r-2}. So the size of our aligned circuit computing f is at most $O(sn^{3r-2})$. ◀

4.3.2 Efficient Circuit Alignment

We can get smaller aligned circuits using a more sophisticated technique. Notice that the construction above enumerates all possible "completions" of non-aligned polynomials to aligned polynomials at each level. We assigned a gate to each such completion, which fails to exploit the fact that it is not possible to obtain non-aligned polynomials by arithmetic operations on aligned polynomials. The construction below *does* take advantage of these restrictions to do much less brute-force enumeration of intermediate non-aligned polynomials, by *implicitly* representing future completions at each gate. We use matrix multiplication to organize this more efficient combination of polynomial types, which is why ω appears in the complexity of the resulting circuit.

We restrict our attention from now on to $r = 3$. It will simplify the proof and it turns out that it gives almost optimal results.

▶ **Lemma 11** (Pinned to Aligned Circuits, Efficiently)**.** *If there exists a **3-pinned** arithmetic circuit C of size s computing a **3-aligned** polynomial $f(Y)$, then there exists a **3-aligned** circuit of size $O(sn^{\omega})$ computing $f(Y)$.*

The high level idea of the proof of Lemma 11 is as follows. Let f_M denote a matrix of size $n \times n$ that has f as it's $[1, 1]$ entry and 0 elsewhere. One can measure the arithmetic circuit complexity of f_M in a model where matrices are on the wires of the circuit instead of scalars. We use this observation to prove the above lemma in two steps:

1. Convert the circuit for f into a circuit for f_M over the ring of matrices. (Lemma 13)
2. Convert the circuit for f_M back into a circuit for f by replacing each gate with circuits for matrix addition and matrix multiplication. The resulting circuit is aligned, and has hardware proportional to the original number of gates times the cost of matrix multiplication.

The key step is converting a circuit for f into a circuit for f_M. As before, we introduce a mapping Φ to transform the original circuit layer-by-layer. This time, however, it is not an operator on polynomials: it maps polynomials to matrices. By propagating this Φ through C, we obtain a circuit for f_M. Lemma 12 below states the properties of Φ. We give the full proof of correctness for our efficient construction of aligned circuits (Lemma 11) at the end of this section, because it is straightforward once we have Φ.

▶ **Lemma 12** (Polynomial to Matrix)**.** *There exists a map Φ that takes a polynomial on $3n$ variables to an $n \times n$ matrix with polynomial entries satisfying the following conditions:*

- **(i)** *For all 3-pinned polynomials g all entries of $\Phi(g)$ are aligned polynomials.*
- **(ii)** *If g is a 3-aligned polynomial, then $\Phi(g)[1, 1] = g$ and all other entries of $\Phi(g)$ are zero.*
- **(iii)** *If g is a variable or a constant, then the degree of each entry of $\Phi(g)$ is at most 3.*

(iv) *For all 3-pinned polynomials g, u, v, and arithmetic $+, \times$ over the ring of matrices:*

$$g = u + v \Rightarrow \Phi(g) = \Phi(u) + \Phi(v)$$
$$g = u \times v \Rightarrow \Phi(g) = \Phi(u) \times \Phi(v)$$

One new trick that we are going to use is that we will sometimes not store the suffix or prefix of the monomial if it is too long. Instead we will store what it can become after we complete it to an aligned monomial. For example, consider the following polynomial: $y_{0,i}y_{2,j} + y_{1,i'}y_{2,j'}$. Instead of memorizing it this way, one can remember that it will become $y_{0,i}y_{2,j}y_{3,k} + y_{1,i'}y_{2,j'}y_{3,k}$ after we multiply it by $y_{3,k}$. By contrast, the simple alignment procedure stores these completions on *both* sides of the multiplication, duplicating information and wasting gates.

Proof of Lemma 12. We need only define the operator Φ for 3-pinned polynomials. Every 3-pinned polynomial $g(Y)$ is one of 9 types $(a, b) \in \{0, 1, 2\} \times \{0, 1, 2\}$, based on which Y-variables start and end all the monomials of f. Denote by $\mathbb{F}_{a,b}\langle Y\rangle$ the set of 3-pinned polynomials of type (a, b). Each entry $[i, j]$ of the matrix $\Phi(g)$ will be an arithmetic expression in terms of g that depends on the "pinning type" of g and the indices $[i, j]$. Below, we define functions λ and ρ which select how to transform g from the left and the right, respectively, in terms of pinning type of g and index of $\Phi(g)$. We use below the notation $\delta(i) = 1$ if $i = 1$ and $\delta(i) = 0$ otherwise.

For $g \in \mathbb{F}_{a,b}\langle Y\rangle$ define $\Phi(g)[i, j] = \lambda(a, i)\ g\ \rho(b, j)$ where:

$$\lambda(a,i) = \begin{cases} \delta(i)\times & \text{if } a = 0, \\ y_{0,i}\times & \text{if } a = 1, \\ y_{2,i}\div_{\mathcal{L}} & \text{if } a = 2 \end{cases} \qquad \text{and} \qquad \rho(b,j) = \begin{cases} \times\delta(j) & \text{if } b = 2, \\ \times y_{2,j} & \text{if } b = 1, \\ \div_{\mathcal{R}} y_{0,j} & \text{if } b = 0 \end{cases}$$

We expand the definition of Φ concretely below. This matrix is the outer product of the λ and ρ operation selection functions "around" g.

$$\Phi(g)[i,j] \leftarrow \text{entry } (a,b) \text{ of } \begin{bmatrix} \delta(i) \times g \div_{\mathcal{R}} y_{0,j} & \delta(i) \times g \times y_{2,j} & \delta(i) \times g \times \delta(j) \\ y_{0,i} \times g \div_{\mathcal{R}} y_{0,j} & y_{0,i} \times g \times y_{2,j} & y_{0,i} \times g \times \delta(j) \\ y_{2,i} \div_{\mathcal{L}} g \div_{\mathcal{R}} y_{0,j} & y_{2,i} \div_{\mathcal{L}} g \times y_{2,j} & y_{2,i} \div_{\mathcal{L}} g \times \delta(j) \end{bmatrix}$$

Inspecting the expansion above, properties (i), (ii), and (iii) claimed for Φ are clear. It remains to show property (iv): that Φ maps arithmetic over 3-pinned polynomials to arithmetic over the ring of matrices. Reasoning from the definitions of Φ and the division operators we have the following:

$$\forall c, d \in \{0, 1, 2\} \text{ such that } d = (c + 1) \bmod 3 \ : \ p \times q = \sum_{i \in [n]} p\ \rho(c, i)\ \times\ \lambda(d, i)\ q$$

If g, u, v are 3-pinned polynomials then, by additivity of the matrix ring, $g = u + v \Rightarrow \Phi(f) = \Phi(u) + \Phi(v)$. We also need $g = u \times v \Rightarrow \Phi(f) = \Phi(u) \times \Phi(v)$ which we prove directly. Let $a, b, c \in \mathbb{Z}/3$ be such that $u \in \mathbb{F}_{a,b}\langle Y\rangle$ and $v \in \mathbb{F}_{b+1,c}\langle Y\rangle$. These numbers must exist, since u

and v multiply to give the 3-pinned polynomial $g \in \mathbb{F}_{a,c}\langle Y \rangle$. So

$$\begin{aligned}(\Phi(u) \times \Phi(v))[i,j] &= \sum_k \Phi(u)[i,k]\Phi(v)[k,j] \\ &= \lambda(a,i)\left(\sum_k u\ \rho(b,k)\ \lambda(b+1,k)\ v\right)\rho(c,j) \\ &= \lambda(a,i)\ u \times v\ \rho(c,j) \\ &= \lambda(a,i)\ g\ \rho(c,j) = \Phi(g)[i,j]\end{aligned}$$

The key observation for the derivation above is that $\rho(b,k)\lambda(b+1,k)$ "cancels out" for any $b \in \mathbb{Z}/3$. This is what saves hardware compared to the simple construction: there is no "garbage" in the middle of the representation to enumerate over. ◀

We can now push Φ "down" through a pinned circuit to obtain an aligned circuit. We first need the following lemma, to convert a circuit for f into a circuit for f_M over the ring of matrices.

▶ **Lemma 13** (Pinned Circuit to Matrix Circuit)**.** *If there exists a* 3*-pinned arithmetic circuit of size s that computes a* 3*-pinned polynomial f, then there exists a circuit of size $O(s)$ that computes f_M. This circuit uses matrix addition and multiplication as gates, and has matrices with aligned monomials of degree at most* 3 *in entries as inputs.*

Proof. Suppose that we are given a 3-pinned circuit C for 3-aligned polynomial f. Then, as $\Phi(f) = f_M$ we can apply the operator Φ to the output of C and get a circuit for f_M. Recall the properties of Φ guaranteed in Lemma 12. We use property *(iv)* to push Φ down one level of C. We will measure the size of this circuit as the number of gates that perform arithmetic operations, both over polynomials and matrices, which is the same as counting all except Φ-gates. It is easy to see that when we apply rule *(iv)* we are not increasing size of the circuit measured this way.

Eventually we will sink all the Φ-gates to the very bottom. We will have a circuit with only matrix addition, matrix multiplication and Φ gates, and the last are only applied to the inputs. By property (*iii*) we know that Φ applied to the input computes a matrix whose entries are aligned polynomials of degree at most 3. That means that we can just claim the outputs of Φ as our new inputs – we are allowed to have matrices with degree 3 aligned polynomials as inputs in the model of matrix circuits. This removes all the Φ from C, and the only types of gates left are matrix multiplication and addition. Then our measure of size is now the same as the number of gates, so we have a new matrix circuit with size exactly matching that of C. ◀

Note that it is impossible to obtain non-aligned polynomials by arithmetic operations on aligned polynomials. Therefore, all matrices computed by the gates in such a circuit would have aligned polynomials in all entries. We conclude by mapping pinned circuits to aligned circuits, efficiently.

Proof of Lemma 11. Take a circuit for f, and construct a circuit for f_M, using Lemma 13. Replace each matrix with n^2 gates each representing one entry. Replace each matrix addition and multiplication gate with a circuit on $2n^2$ inputs that perform the same operations. This will leave us with a circuit of size $O(sn^\omega)$ over aligned monomials of degree at most 3 as inputs. ◀

References

1. Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 67–75. IEEE Computer Society, 2008.
2. Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial time identity testing for noncommutative circuits. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 831–841. ACM, 2017. doi:10.1145/3055399.3055442.
3. Vikraman Arvind, Pushkar S. Joglekar, and S. Raja. Noncommutative valiant's classes: Structure and complete problems. *TOCT*, 9(1):3:1–3:29, 2016. doi:10.1145/2956230.
4. Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 677–686. ACM, 2010.
5. Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical computer science*, 22(3):317–330, 1983.
6. Klim Efremenko, Ankit Garg, Rafael Mendes de Oliveira, and Avi Wigderson. Barriers for rank methods in arithmetic complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:27, 2017.
7. Michael A. Forbes, Amir Shpilka, and Ben Lee Volk. Succinct hitting sets and barriers to proving algebraic circuits lower bounds. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 653–664. ACM, 2017.
8. Joshua A. Grochow, Mrinal Kumar, Michael E. Saks, and Shubhangi Saraf. Towards an algebraic natural proofs barrier via polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:9, 2017.
9. Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM J. Comput.*, 45(3):1064–1079, 2016.
10. Pavel Hrubes, Avi Wigderson, and Amir Yehudayoff. Relationless completeness and separations. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:40, 2010. URL: http://eccc.hpi-web.de/report/2010/040.
11. Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Journal of the American Mathematical Society*, 24(3):871–898, 2011.
12. Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theor. Comput. Sci.*, 448:56–65, 2012.
13. François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014.
14. Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418. ACM, 1991.
15. Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.
16. Amir Shpilka, Amir Yehudayoff, et al. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010.
17. Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings*

of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.

A Infinitely often vs almost everywhere hardness

In this section we carry out our hardness amplification for polynomials that are sometimes hard, as opposed to hard everywhere. In the proof of 1 we used the assumption that:

$$\exists \alpha, \epsilon > 0 \text{ such that } \forall n: \ P_n \text{ is not computable by arithmetic circuits of size } \alpha n^{c+\epsilon}$$

A more natural way to say that some polynomial P requires circuits of size larger then n^c would be:

$$P \notin \text{ASize}[n^c],$$

where ASize$[f(n)]$ is set of all sequences of polynomials that can be computed by circuits of size $O(f(n))$. The difference between these two definitions is that the first means that the polynomial is not computable by small circuits everywhere, and the second means that the polynomial is not computable by small circuits for infinitely many n. While in the proof of Theorem 1 we used the first definition, we actually only needed that the polynomial is hard on infinitely many points of the form $3^{-3/2}(3^{\frac{1}{2}}n)^{3^k}$ for some n and fixed k. This motivates the notion of infinitely often hardness on a subset, described below:

▶ **Definition 14.** ASize$[f(n)]$ is set of all sequences of polynomials that can be computed by circuits of size $O(f(n))$

Now we will tweak this definition to describe hardness on subset of integers:

▶ **Definition 15.** Let S be a infinite size subset of natural numbers. ASize$_S[f(n)]$ is set of all sequences of polynomials that can be computed by circuits of size $O(f(n))$ for all $n \in S$.

▶ **Lemma 16.** *Let A be a subset of even natural numbers, such that $\frac{\log A_{n+1}}{\log A_n} = 1 + o(1)$, where A_n is n-th smallest element of A is $\leq 2^{n^\gamma}$ for some γ and P is an explicit sequence of polynomials that is i.o. n^c hard for some c. Then for every $\epsilon > 0$ there is an explicit sequence of polynomials Q, such that Q is i.o. $n^{c-\epsilon}$ hard on A.*

Proof. We construct Q as by setting:

$$Q_{2n+1} = Q_{2n} = \sum_{k=1}^{n} x_{n+k} P_i(x_1, x_2, \ldots, x_k)$$

It is easy to see that $Q_{2n}(x_1, x_2 \ldots, x_n, 0, 0, \ldots, 0, 1, 0, \ldots, 0) = P_k(x_1, \ldots x_k)$ if 1 is set in the $n+k$-th position. This means that:

$$\forall n: \ AC(Q_{2n+1}) = AC(Q_{2n}) \geq max_{k \in [n]} AC(P_k)$$

Then suppose that $AC(P_n) > \alpha n^c$ and let i be the smallest number, such that A_i is bigger than $2n$. Then $AC(Q_{A_i}) > AC(P_n)$. This implies that $AC(Q_{A_i}) > \alpha n^c > \alpha A_{i-1}^c > \alpha A_i^{c\frac{\log A_{i-1}}{\log A_i}} > \alpha A_i^{c-o(1)}$. This means that for any $\epsilon > 0$ there would be infinitely many $n \in A$, such that $AC(Q_n) > \alpha n^{c-\epsilon}$ ◀

Now we just need to observe that the set $A = \{x | \exists n: \ x = 3^{-3/2}(3^{\frac{1}{2}}n)^{3^k}\}$ satisfies the property $\frac{\log A_{n+1}}{\log A_n} = 1 + o(1)$. It is true even if we allow k to be a monotone function of n if $k = O(\log(n))$, which covers all the range of parameters that we are currently using.