Synopsis of the proposed PhD thesis titled

Hardness and Independence of Polynomials

submitted by

Prerona Chatterjee

under the guidance of

Dr. Ramprasad Saptharishi

to the

Computer and Systems Sciences Subject Board

Tata Institute of Fundamental Research, Mumbai

1 Introduction

The focus of this thesis is on problems of the form "How hard is it to perform a computational task Y on a given algebraic object X?" In particular, the question can be as simple as

How hard is it to compute a given polynomial for a given computational model?

Indeed, this is one of the questions we will be focussing on. We are also interested in the following two questions and how they are related to each other. Firstly,

How hard is it to check whether a given set of polynomials are *algebraically independent*?

and secondly,

How hard is it to check if a given computational model computes the zero polynomial?

These are central questions in the field of Algebraic Complexity Theory, and we describe them in greater detail later. But before that, let us define the models of computation we study in this thesis.

An algebraic circuit is a very natural (and the most general) algebraic model of computation. It is a rooted directed acyclic graph in which the leaves are labelled by indeterminates or field constants and the internal nodes are labelled by addition ($+$) or multiplication ($\times$). Thus every node in the circuit naturally computes a polynomial and the polynomial computed at the root is said to be the polynomial computed by the circuit. The size of a circuit is defined to be the number of operations the circuit performs. If the underlying graph is only allowed to be a tree, then the model of computation is that of an algebraic formula.

Another important model of computation is that of an algebraic branching program (ABP). They are layered graphs in which the first and the last layer only contain a single vertex each called s and t respectively. Edges in an ABP are allowed to be between consecutive layers and each edge is labelled by an affine linear polynomial over the underlying variables. The polynomial computed by any $s - t$ path is the product of the edge labels on that path and the polynomial computed by the ABP is the sum of the polynomials computed by all the $s - t$ paths.

1.1 Lower Bounds in Algebraic Circuit Complexity

Algebraic Circuit Complexity is an integral sub-area of algebraic complexity theory which deals with classifying polynomials depending on how hard it is to compute them. Polynomials over n variables of degree $\text{poly}(n)$ that can be computed by algebraic circuits of size $\text{poly}(n)$ is said to belong to the class VP — the class of efficiently computable polynomials. Similarly, polynomials that can be computed by efficient algebraic formulas and ABPs are said to belong to the class VF and VBP respectively. In the same landmark paper that defined efficiently computable

polynomials [Val79], Valiant also defined the the class of efficiently definable polynomials, which we now call VNP. The classes VP and VNP can be thought of as algebraic analogues of P and NP and it is known that VP $\subseteq$ VNP. The central question in algebraic circuit complexity is whether VP $\neq$ VNP. In other words, is there an efficiently definable (or explicit) polynomial that is not efficiently computable. Similarly, it also known that VF $\subseteq$ VBP $\subseteq$ VNP and it is believed that each of the inclusions are tight. Showing that they are indeed so are also important open problems.

We now mention the contributions of the thesis with respect to lower bounds.

1.1.1 Lower Bounds Against General Models

In a major result, Baur and Strassen [Str73, BS83] gave an explicit n-variate polynomial which requires circuits of size at least $\Omega(n \cdot \log n)$. Such super-linear lower bounds are not known against the analogous model of *boolean* circuits. However the aim is to prove super-polynomial lower bounds and despite decades of work, this lower bound has not been improved for general circuits.

For algebraic formulas, however, slightly better bounds are know due to Kalorkoti [Kal85], who gave an $\Omega(n^{3/2})$ lower bound for the n-variate determinant polynomial (that is the $\sqrt{n} \times \sqrt{n}$ determinant). In fact these ideas can be used to improve the bound to $\Omega(n^2/\log n)$ for an explicit n-variate multilinear polynomial (for example, see [SY10]). Note that, with respect to algebraic formulas, $\Omega(nd)$ is a trivial lower bound for any n-variate polynomial in which every variable has individual degree at least d. Therefore the regime where the explicit hard polynomial is multilinear is particularly interesting. It turns out that in general, Kalorkoti's technique *cannot* prove a lower bound which is asymptotically better than $(n^2/\log n)$ for multilinear polynomials.

With respect to algebraic branching programs, Kumar [Kum19] gave a quadratic lower bound in the restricted setting of homogeneous ABPs. For general ABPs however, the result of Baur and Strassen [Str73, BS83] remained the best known lower bound prior to our work.

Our Contribution In a joint work with Mrinal Kumar (IIT Bombay), Adrian She (University of Toronto) and Ben Lee Volk (IDC Herzliya) [CKSV19], we show that any algebraic branching program computing the polynomial $\sum_{i=1}^{n} x_i^n$ has at least $\Omega(n^2)$ vertices. This improves the previous best lower bound of $\Omega(n \log n)$. We also use our proof techniques to show an $\Omega(n^2)$ lower bound against algebraic formulas computing the elementary symmetric polynomial of degree $0.1n$ over n variables. Note that this lower bound is asymptotically better than $(n^2/\log n)$, the strongest lower bound for multilinear polynomials that can be proved using previous known methods.

1.1.2 Lower Bounds in the Non-Commutative Setting

A natural restriction that is studied is non-commutativity of multiplication [Hya77, Nis91]. That is, in this setting $XY \neq YX$ for indeterminates X and Y. For example if X, Y were replaced by

matrices, this is indeed the case. Eventhough the best lower bound known for circuits in this setting is still the one by Baur and Strassen [Str73, BS83], exponential lower bounds are known against non-commutative formulas [Nis91]. In fact the polynomial Nisan gave the lower bound for is efficiently computable by non-commutative circuits. Therefore his work gave a separation between the powers of formulas and circuits in the non-commutative world.

A point to note however is that Nisan's proof actually holds for ABPs as well. In fact Nisan's work gives a characterisation for the size of the smallest ABP computing a given non-commutative polynomial. The exponential lower bound against formulas is essentially due to this charcterisation and hence is also an exponential lower bound against ABPs, leading to an exponential separation between the powers of ABPs and circuits in the non-commutative setting.

In the commutative setting, ABPs are believed to be computationally more powerful than formulas. A natural question therefore, posed by Nisan [Nis91] himself, is whether one can show a super-polynomial separation between ABPs and formulas in the non-commutative setting. In a remarkable recent result, Limaye, Srinivasan and Tavenas [LST21b] showed that there is indeed a super-polynomial separation between *homogeneous* formulas and ABPs in the non-commutative setting. The general question, however, continues to remain open.

Our Contribution We provide a new approach towards resolving this problem by studying certain structured formulas that we call *abecedarian* formulas [Cha21]. Firstly, we show a super-polynomial separation between non-commutative ABPs and abecedarian formulas. We then go on to show that, for certain settings of parameters, proving lower bounds against abecedarian formulas is enough to prove lower bounds against general formulas. Abecedarian formulas are not necessarily homogeneous, and hence this work is incomparable to the work of [LST21b].

1.2 Algebraic Independence Testing and Constructing Faithful Homomorphisms

We now look at a different question. A set of polynomials $\{f_1, \ldots, f_m\}$ is said to be *algebraically dependent* if there is some non-zero polynomial combination of them that is zero. For example, if $f_1 = x$, $f_2 = y$ and $f_3 = x^2 + y^2$, then $A = z_1^2 + z_2^2 - z_3$ is an *annihilator*. We note that the underlying field is very important in this case. For example, $x + y$ and $x^p + y^p$ are algebraically independent when considered as polynomials over $\mathbb{R}$ or $\mathbb{C}$, but they are algebraically dependent over $\mathbb{F}_p$ since $(x + y)^p = x^p + y^p$. The obvious question at this point is thus the following.

Is there an efficient *algorithm to check whether a given set of polynomials are algebraically independent*?

Surprisingly, this is not known in general. Over fields of characteristic zero, a classical result of Jacobi [Jac41] leads to a randomized polynomial time algorithm for algebraic independence testing. However, the algorithm fails over fields of finite characteristic.

Over such fields, Pandey, Saxena and Sinhababu [PSS18] characterised the extent of failure of the Jacobian criterion using the notion of *inseparable degree* of the given set of polynomials. They presented a randomised algorithm to solve the problem in general. Unfortunately, the algorithm is efficient only when the inseparable degree is bounded by a constant. Later, Guo, Saxena and Sinhababu [GSS19] showed that the problem of algebraic independence testing in $\mathsf{AM} \cap \mathsf{co\text{-}AM}$ via a very different approach that does not depend on the inseparable degree.

Application to Polynomial Identity Testing Given an algebraic circuit as input, the Polynomial Identity Testing (PIT) problem asks whether the polynomial computed by it is identically zero. It is well known that this problem has an efficient randomised polynomial-time algorithm. A central algorithmic question in Algebraic Complexity Theory is whether this can be derandomised.

Beecken, Mittman and Saxena [BMS13] showed that the concept of algebraic independence has a connection with the question of PIT via the notion of *faithful homomorphisms* (or faithful maps). They used the results in [Jac41] to construct such maps over fields of zero characteristics, which were then used by them and Agrawal, Saha, Saptharishi and Saxena [ASSS16] to design identity tests for several classes of algebraic circuits. These results however do not carry over to fields of finite characteristic, since the Jacobian criterion fails over such fields.

Our Contribution In a joint work with Ramprasad Saptharishi (TIFR) [CS18], we addressed the question of constructing faithful maps over fields of finite characteristic in several restricted cases. Using the characterisation given by Pandey, Saxena and Sinhababu [PSS18], we provide a recipe for constructing faithful maps for polynomials coming from certain circuit classes (such as sum of sparse polynomials among others). The techniques used are similar to those in [BMS13] and [ASSS16], and lead to polynomial identity tests for these classes in fields of small characteristic.

1.3 Organisation

In Section 2 we discuss our lower bounds against ABPs and formulas in greater detail. We then move on to discussing our lower bounds against structured non-commutative formulas in Section 3. In Section 4 we talk about testing algebraic independence of polynomials and constructing faithful homomorphisms over fields of finite characteristic. Finally, we conclude with an outline of the thesis in Section 5 and a list of publications from teh thesis in Section 6.

2 Lower Bounds against General Models

The question we focus on in this section is whether there exist explicit polynomials that cannot be computed by efficient ABPs and formulas. Using counting arguments, it is easy to check that

there are n-variate, degree d polynomials that cannot be computed by ABPs and formulas of size $\text{poly}(n, d)$. Therefore the question is just whether any of these polynomials are explicit.

Unfortunately, we are very far from this goal. The best lower bound we know against both ABPs and formulas is just quadratic in the number of underlying variables.

2.1 Algebraic Branching Programs

We start by formally defining an algebraic branching program.

Definition 2.1 (Algebraic Branching Programs)**.** *An Algebraic Branching Program (ABP) is a layered graph in which the first and the last layer only contain a single vertex each, called the* start *(or s) and* terminal *(or t) respectively. Edges in an ABP are only allowed to be between consecutive layers and each edge is labelled by an affine linear polynomial over the underlying variables.*

The polynomial computed by any s-t path is the product of the edge labels on that path and the polynomial computed by the ABP is the sum of the polynomials computed by all the s-t paths.

The size of an ABP is the number of vertices in it. ◊

While Definition 2.1 is quite standard, there are some small variants of it in the literature which we now discuss. These distinctions make no difference as far as super-polynomial lower bounds are concerned, since it can be easily seen that each variant can be simulated by the other to within polynomial factors, and thus the issues described here are usually left unaddressed. However, while proving super-linear lower bounds, these issues are quite relevant.

Layered vs. Unlayered. In Definition 2.1, we have required the graph to be layered. We also consider ABPs whose underlying graphs are unlayered, which we call *unlayered ABPs*. We are able to prove super-linear (but weaker) lower bounds for this model as well.

One motivation for considering layered graph as the "standard" model is given by the following interpretation. From the definition, it can be observed that any polynomial computable by an ABP with d layers and ℓ_i vertices in the i-th layer can be written as the (only) entry of the 1×1 matrix given by the product $M := \prod_{i=0}^{d} M_i$, where M_i is an $\ell_i \times \ell_{i+1}$ matrix with affine forms as entries. One natural complexity measure of such a representation is the total number of non-zero entries in those matrices, which is the number of edges in the ABP. Another natural measure, which can only be smaller, is the sums of dimensions of the matrices involved in the product, which is the same as the number of vertices in the underlying graph.

Branching programs are also prevalent in boolean complexity theory and in particular in the context of derandomizing the class RL. In this setting again, it only makes sense to talk about layered graphs. Unlayered ABPs can also be thought of as (a slight generalization of) *skew circuits*. These are circuits in which on every multiplication gate, at least one of the operands is a variable (or more generally, a linear function).

Edge labels. In Definition 2.1 we have allowed each edge to be labeled by an arbitrary affine linear polynomial in the underlying variables. This is again quite standard, perhaps inspired by Nisan's characterization of the ABP complexity of a non-commutative polynomial as the rank of an associated coefficients matrix [Nis91], which requires this freedom. A more restrictive definition would only allow each edge to be labeled by a linear function in 1 variable. On the other hand, an even more general definition, which we sometimes adopt, is to allow every edge to be labeled by an *arbitrary* polynomial of degree at most Δ. In this case we refer to the model as an ABP with edge labels of degree at most Δ. Thus, the common case is $\Delta = 1$, but our results are meaningful even when $\Delta = \omega(1)$. Note that this is quite a powerful model, which is allowed to use polynomials with super-polynomial standard circuit complexity "for free".

We will recall some of these distinctions in Section 2.4, where we discuss previous results, some of which apply to several of the variants discussed here.

2.2 Lower Bounds for Algebraic Branching Programs.

Our first result is a quadratic lower bound on the size of any algebraic branching program computing some explicit polynomial.

Theorem 2.2. *Let $\mathbb{F}$ be a field and $n \in \mathbb{N}$ such that $\mathrm{char}(\mathbb{F}) \nmid n$. Then any algebraic branching program over $\mathbb{F}$ computing the polynomial $\sum_{i=1}^{n} x_i^n$ is of size $\Omega(n^2)$.*

When the ABP's edge labels are allowed to be polynomials of degree at most Δ, the size is $\Omega(n^2/\Delta)$.

Note that there also exists an algebraic branching program for $\sum_{i=1}^{n} x_i^n$ of size $O(n^2/\Delta)$. A rough sketch of the construction is as follows. The ABP essentially consists of n parallel paths from the source vertex to the target vertex, with the i^{th} path computing x_i^n. If the labels on the edges are allowed to have degree at most Δ, then each path consists of n/Δ edges, with all the edges on the i^{th} path being labelled by x_i^{Δ} (except possibly the last edge, which is labelled by $x_i^{n-\Delta((n/\Delta)-1)}$). This shows that the bound we give here is in fact tight.

For the unlayered case, we prove a weaker (but still superlinear) lower bound.

Theorem 2.3. *Let $\mathbb{F}$ be a field and $n \in \mathbb{N}$ such that $\mathrm{char}(\mathbb{F}) \nmid n$. Then any unlayered algebraic branching program over $\mathbb{F}$ with edge labels of degree at most Δ computing the polynomial $\sum_{i=1}^{n} x_i^n$ has at least $\Omega(n \log n / (\log\log n + \log \Delta))$ edges.*

2.3 Lower bounds for algebraic formulas.

Recall that an algebraic formula is an algebraic circuit where the underlying graph is a tree. For polynomials like $\sum_{i=1}^{n} x_i^n$, a quadratic lower bound on the formula size is immediate. This is because the degree of the polynomial in every variable is n, and hence there must be at least n leaves

which are labeled by x_i for every i. Therefore, for a super-linear formula lower bound to be non-trivial it should hold for a polynomial with bounded individual degrees (we discuss this further in Section 2.4).

As our next result, we show an $\Omega(n^2)$ lower bound for a multilinear polynomial.

Theorem 2.4. *Let $n \in \mathbb{N}$ and let $\mathbb{F}$ be a field of characteristic greater than $0.1n$. Then any algebraic formula over $\mathbb{F}$ computing the elementary symmetric polynomial*

$$\mathrm{ESYM}_{n,0.1n}(\mathbf{x}) = \sum_{S \subseteq [n], |S| = 0.1n} \prod_{j \in S} x_j \,,$$

is of size at least $\Omega(n^2)$.

As we describe in Section 2.5, even though formulas can be simulated by ABPs with little overhead and despite the fact that both proofs use similar ideas, this theorem is not an immediate corollary of Theorem 2.2.

2.4 Previous Work

ABP lower bounds. The best lower bound known for ABPs prior to this work is a lower bound of $\Omega(n \log n)$ on the number of edges for the same polynomial $\sum_{i=1}^{n} x_i^n$. This follows from the classical lower bound of $\Omega(n \log n)$ by Baur and Strassen [Str73, BS83] on the number of multiplication gates in any algebraic circuit computing the polynomial $\sum_{i=1}^{n} x_i^n$ and the observation that when converting an ABP to an algebraic circuit, the number of product gates in the resulting circuit is at most the number of edges in the ABP. Theorem 2.2 improves upon this bound quantitatively and also qualitatively, since the lower bound is on the number of vertices in the ABP.

For the case of homogeneous ABPs,[1] a quadratic lower bound for the polynomial $\sum_{i=1}^{n} x_i^n$ was shown by Kumar [Kum19] and our proofs paper build on the ideas in [Kum19]. In a nutshell, the result in [Kum19] is equivalent to a quadratic lower bound for ABPs computing the polynomial $\sum_{i=1}^{n} x_i^n$ when the number of layers in the ABP is at most n. We generalize this to proving essentially the same lower bound for ABPs with an unbounded number of layers.

In general, an ABP computing an n-variate homogeneous polynomial of degree $\mathrm{poly}(n)$ can be homogenized with a polynomial blow-up in size. This is proved in a similar manner to the standard classical result which shows this statement for algebraic circuits [Str73]. Thus much like the discussion following Definition 2.1, homogeneity in the context of super-polynomial lower bounds. But it becomes relevant when proving polynomial lower bounds.

For *unlayered* ABPs, the situation is more complex. If the edge labels are only functions of one variable, then (for the same reasons discussed in Section 2.3) we need to only consider multilinear

[1] An ABP is *homogeneous* if the polynomial computed between the start vertex and any other vertex is a homogeneous polynomial. This condition is essentially equivalent to assuming that the number of layers in the ABP is upper bounded by the degree of the output polynomial.

polynomials for the lower bound to be non-trivial. It is possible to adapt Nechiporuk's method [Nec66] in order to obtain a lower bound of $\tilde{\Omega}(n^{3/2})$ for a multilinear polynomial in this model.

As mentioned above, the lower bound of Baur-Strassen does hold in the unlayered case, assuming the edge labels are linear functions in the variables. However, when we allow edge labels of degree at most Δ for some $\Delta \geq 2$, their technique does not seem to carry over. Indeed, even if we equip the circuit with the ability to compute such low-degree polynomials "for free", a key step in the Baur-Strassen proof is the claim that if a polynomial f has a circuit of size τ, then there is a circuit of size $O(\tau)$ which computes all its first order partial derivatives, and this statement does not seem to hold in this new model. It is possible to get an $\Omega(n \log n / \log \Delta)$ lower bound for this model, for a different polynomial, by suitably extending the techniques of Ben-Or [Ben83, Ben94]. Our lower bounds are weaker by at most a doubly-logarithmic factor; however, the techniques are completely different. Ben-Or's proofs rely as a black-box on strong modern results in algebraic geometry, whereas our proofs are much more elementary.

Formula lower bounds. Before discussing prior results on formula lower bounds, we recall that it is easy to see that any algebraic formula computing an n-variate polynomial with individual degree at least d has at least nd leaves. Therefore, for a lower bound against algebraic formulas to be interesting and non-trivial, it must be asymptotically larger than the sum of the individual degrees of the variables. One particularly interesting regime of parameters is when the target hard polynomial has constant individual degree (which can be taken to be 1, essentially without loss of generality).

As mentioned earlier, the first non-trivial lower bound for algebraic formulas was shown by Kalorkoti [Kal85] who proved a lower bound of $\Omega(n^{3/2})$ for an n-variate multilinear polynomial (in fact, for the $\sqrt{n} \times \sqrt{n}$ determinant). Kalorkoti's proof is essentially an algebraic version of Nechiporuk's proof [Nec66] of $\Omega(n^2/\log n)$ lower bound on the size of Boolean formula in the full binary basis (gates are allowed to be all Boolean functions on 2 variables). Using very similar arguments, it is possible to show an $\Omega(n^2/\log n)$ lower bound for a different multilinear polynomial (the construction we describe here is based on a non-multilinear construction given in [SY10]): let $\left\{ x_i^{(j)} : i \in [\log n], j \in [n/\log n] \right\}$ and $\{y_k : k \in [n]\}$ be two sets of n variables each, and fix a bijection $\iota : 2^{[\log n]} \to [n]$. The hard multilinear polynomial will be

$$\sum_{j=1}^{n/\log n} \left(\sum_{S \subseteq [\log n]} y_{\iota(S)} \cdot \prod_{i \in S} x_i^{(j)} \right).$$

It turns out that, in general, this technique of Nechipurok and Kalorkoti *cannot* prove a lower bound which is asymptotically better than $\left(n^2/\log n\right)$ (see Remark 6.18 in [Juk12]).

In Theorem 2.4, we adapt the ideas used in Theorem 2.2 to prove an $\Omega(n^2)$ lower bound for an explicit n variate multilinear polynomial. Thus, while the quantitative improvement over the

previous lower bound is by a mere $\log n$ factor, it is interesting to note that Theorem 2.4 gives an asymptotically better lower bound than the best bound that can be proved using the classical, well known techniques of Nechiporuk and Kalorkoti.

Our lower bound is for one of the elementary symmetric polynomials which, due to a well known observation of Ben-Or, are also known to be computable by a formula (in fact, a depth-3 formula) of size $O(n^2)$ over all large enough fields. For a large regime of parameters, Ben-Or's construction was shown to be tight for depth-3 formulas (even depth-3 circuits) by Shpilka and Wigderson [SW01]. Theorem 2.4 shows that elementary symmetric polynomials do not have formulas of size $o(n^2)$ regardless of their depth.

Prior to this work, the only super linear lower bound on the general formula complexity of elementary symmetric polynomials we are aware of is the $\Omega(n \log n)$ lower bound of Baur and Strassen [BS83], which as we mentioned earlier is in fact a lower bound on the stronger notion of *circuit* complexity. As far as we understand, Kalorkoti's proof directly does not seem to give any better lower bounds on the formula complexity of these polynomials.

As another point, we note that Baur and Strassen [BS83] also proved an *upper bound* of $O(n \log n)$ on the circuit complexity of the elementary symmetric polynomials. Hence, our lower bound implies a super-linear separation between the formula complexity and the circuit complexity of an explicit polynomial family.

2.5 Proof Overview

ABP lower bounds. The first part in the proof of Theorem 2.2 is an extension of the lower bound proved in [Kum19] for ABPs with at most n layers. This straightforward but conceptually important adaptation shows that a similar lower bound holds for any polynomial of the form

$$\sum_{i=1}^{n} x_i^n + \varepsilon(\mathbf{x}) \,,$$

where the suggestively named $\varepsilon(\mathbf{x})$ should be thought of as an "error term" which is "negligible" as far as the proof of [Kum19] is concerned. The exact structure we require is that $\varepsilon(\mathbf{x})$ is of the form $\sum_{i=1}^{r} P_i Q_i + R$, where P_i, Q_i are polynomials with no constant term and $\deg(R) \leq n-1$. The parameter r measures the "size" of the error, which we want to keep small and the lower bound holds if, for example, $r \leq n/10$. We call this the base case.

To argue about ABPs with d layers, with $d > n$, we use a notion of depth reduction which is reminiscent of similar statements in the context of matrix rigidity. We show that unless the size τ of the ABP is too large to begin with (in which case there is nothing to prove), it is possible to find a small set of vertices (of size about $\eta = \tau/d$) whose removal adds a small error term $\varepsilon(\mathbf{x})$ as above with at most η summands, but also reduces the depth of the ABP by a constant factor. Repeatedly applying this operation $O(\log n)$ times eventually gives an ABP of depth at most n while ensuring

that we have not accumulated too much "error",[2] so that we can apply the base case.

The proof of Theorem 2.3 follows the same strategy, although the main impediment is that general undirected graphs can have much more complex structure then layered graphs. One of the main ingredients in our proof is (a small variant of) a famous lemma of Valiant [Val77], which shows that for every graph of depth 2^k with m edges, it is possible to find a set of edges, of size at most m/k, whose removal reduces the depth of the graph to 2^{k-1}. This lemma helps us identify a small set of vertices which can reduce the depth of the graph by a constant factor while again accumulating small error terms.

Formula lower bounds. The proof of Theorem 2.4 follows along the lines of the proof of Theorem 2.2 but needs a few more ideas. A natural attempt at recovering formula lower bounds from ABP lower bounds is to convert the formula to an ABP and then invoke the lower bound in Theorem 2.2. However while it is easy to see that a formula can be transformed into an ABP with essentially no blow up in size, the ABP obtained at the end of this transformation is not necessarily layered. Therefore, we cannot directly invoke Theorem 2.2. We can still invoke Theorem 2.3, but the lower bound thus obtained is barely super-linear and in particular weaker than the known lower bounds [Kal85, SY10].

For the proof of Theorem 2.4, we apply the ideas in the proof of Theorem 2.2 in a non black-box manner. The proof is again in two steps. In the first step we show a lower bound of $\Omega(n^2)$ for polynomials of the form $\mathrm{ESYM}_{n,0.1n}(\mathbf{x}) + \varepsilon(\mathbf{x})$ for formulas of formal degree at most $0.1n$. Here $\varepsilon(\mathbf{x})$ should again be thought of as an error term as in the outline of the proof of Theorem 2.2.

For formulas of formal degree greater than $0.1n$, we describe a simple structural procedure to reduce the formal degree of a formula while maintaining its size. However, the complexity of error terms increases in the process. We argue that starting from a formula of size at most n^2 (since otherwise, we are already done) computing $\mathrm{ESYM}_{n,0.1n}(\mathbf{x})$, we can repeatedly apply this procedure to obtain a formula of formal degree at most $0.1n$ which computes the polynomial $\mathrm{ESYM}_{n,0.1n}(\mathbf{x}) + \varepsilon(\mathbf{x})$, where the error terms is not too complex. Combining this with the robust lower bound for formulas of formal degree at most $0.1n$ completes the proof of Theorem 2.4.

Elementary symmetric vs power symmetric polynomials. We remark that the lower bound in Theorem 2.2 also holds for elementary symmetric polynomials of degree $0.1n$ on n variables. However, the proof is a bit simpler and more instructive for the case of power symmetric polynomials and so we state and prove the theorem for these polynomials. As discussed in Section 2.4, for the case of lower bounds for formulas, we are forced to work with multilinear polynomials and hence Theorem 2.4 is stated and proved for elementary symmetric polynomials. In general, these lower

[2] It takes some care in showing that the total number of error terms accumulated is at most $n/10$ as opposed to the obvious upper bound of $O(n \log n)$. In particular, we observe that the number of error terms can be upper bounded by a geometric progression with first term roughly τ/n and common ratio being a constant less than 1.

bounds hold for any family of polynomials of high enough degree whose zeroes of multiplicity at least two lie in a *low dimensional variety*.

3 Lower Bounds in the Non-Commutative Models

In this section we are interested in polynomials that come from the non-commutative polynomial ring $\mathbb{F}\langle x_1, \ldots, x_n \rangle$, where the indeterminates do not commute with each other (that is, $xy \neq yx$ for indeterminates x, y). As a consequence, any monomial in a non-commutative polynomial $f \in \mathbb{F}\langle x_1, \ldots, x_n \rangle$ is essentially a string over the alphabet $\{x_1, \ldots, x_n\}$. This is a natural restriction and there has been a long line of work that studies non-commutative computation beginning with the works of Hyafil and Nisan [Hya77, Nis91][3].

One motivation for studying non-commutative computation is that it is possibly easier to prove strong lower bounds in this setting as compared to the usual commutative setting. At least intuitively, it seems harder to *cancel* monomials once they have been calculated when commutativity is not allowed amongst the variables. For example, the $n \times n$ determinant can be computed by an $O(n^3)$ algebraic circuit, but to the best of our knowledge there is no circuit for the non-commutative determinant of size $2^{o(n)}$. In fact, it was shown by Arvind and Srinivasan [AS18] that if the non-commutative determinant has a poly-sized circuit, then every efficiently definable polynomial is also efficiently computable in the non-commutative setting.

Even though a super-polynomial lower bound is not known for the non-commutative determinant against circuits, Nisan [Nis91] gave an exponential lower bound on the number of gates in any formula computing it. A point to note however, is that Nisan's proof actually works against algebraic branching programs itself. In fact, Nisan [Nis91] gave an exact characterisation for the size of any ABP computing a non-commutative polynomial.

The question we address here is whether there is a separation between the powers of ABPs and formulas in the non-commutative setting. Let us denote the class of non-commutative polynomials over n variables that can be computed by $\mathrm{poly}(n)$-sized ABPs by $\mathsf{VBP}_{\mathrm{nc}}$. Similarly, let $\mathsf{VF}_{\mathrm{nc}}$ denote the class of non-commutative polynomials over n variables that can be computed by $\mathrm{poly}(n)$-sized formulas. The question is essentially whether $\mathsf{VBP}_{\mathrm{nc}}$ is contained in $\mathsf{VF}_{\mathrm{nc}}$.

This question had been posed by Nisan [Nis91] and at the time when this work was done, the only work we were aware of that made some progress with respect to this question was the one by Lagarde, Limaye and Srinivasan [LLS19], where they show that certain syntactically restricted non-commutative formulas (called Unique Parse Tree formulas) cannot compute the iterated matrix multiplication polynomial (or $\mathrm{IMM}_{n,n}(\mathbf{x})$) unless they have size $n^{\Omega(\log n)}$. However very recently, in a break through work, Limaye, Srinivasan and Tavenas [LST21a] showed that there is indeed a super-polynomial separation between the powers of formulas and ABPs in the

[3]The main result in [Hya77] is unfortuantely false as shown in [Nis91]

homogeneous non-commutative setting. We consider restricted formulas of a different nature.

3.1 Abecedarian Polynomials and Models That Compute Them

Hrubeš, Wigderson and Yehudayoff [HWY11] defined the notion of *ordered* polynomials. A homogeneous polynomial of degree d is said to be ordered if the set of variables it depends on can be partitioned into d buckets such that variables occuring in position k only come from the k-th bucket. We generalise this notion by making the bucket indices *position independent*. That is, a variables in position k need not necessarily come from the k-th bucket as long as the variables appear in non-decreasing order of their bucket indices. We call such polynomials abecedarian since, in English, an abecedarian word is one in which all its letters are arranged alphabetically [MW19].

The difference between ordered polynomials and abecedarian ones can be explained succintly using the notion of *regular expressions* from Automata Theory. For a non-commutative polynomial $f \in \mathbb{F}\langle x_1, \ldots, x_n \rangle$, suppose the variables can been partitioned into buckets $\{X_1, \ldots, X_m\}$. Then f is said to be *ordered* with respect to $\{X_1, \ldots, X_m\}$ if every monomial in it is a word that can be generated using the *regular expression* $X_1 \cdots X_m$. Note that this is equivalent to set-multilinear polynomials in the commutative setting. On the other hand, f is abecedarian if the monomials are words that satisfy the regular expression $X_1^* \cdots X_m^*$.

Abecedarian Models of Computation

Hrubeš, Wigderson and Yehudayoff [HWY11] have defined *ordered circuits*, a model naturally suited to compute ordered polynomials. We generalise this notion to define circuits that naturally compute abecedarian polynomials. A circuit is said to be abecedarian if every gate v in it computes an abecedarian polynomial.

Going a step further, we also define abecedarian formulas to be those in which the polynomial computed at each of its vertices is abecedarian. Similarly, an ABP is said to be abecedarian if the polynomial computed between any two of its vertices computes an abecedarian polynomial.

We also define syntactically abecedarian formulas which additionally have its nodes labelled by a pair of bucket indices. The labels denote which bucket the first and last variable, in any monomial being computed by it, might belong. We can similarly define syntactically abecedarian circuits and ABPs. However it is not very hard to check that any abecedarian circuit or ABP can be converted into a syntactically abecedarian circuit or ABP without much blow-up in size. For formulas, on the other hand, it is not very clear if such a statement holds.

3.2 Our Results

Our first main result is a tight super-polynomial separation between the powers of syntactically abecedarian formulas and algebraic branching programs.

Theorem 3.1. *Let*

$$\text{linked\_CHSYM}_{n,d}(\mathbf{x}) = \sum_{i_0=1}^{n} \left(\sum_{i_0 \le i_1 \le \ldots \le i_d \le n} x_{i_0,i_1} \cdot x_{i_1,i_2} \cdots x_{i_{d-1},i_d} \right)$$

be the linked *complete homogeneous polynomial over n-variables of degree d.*

The polynomial $\text{linked\_CHSYM}_{n,d}(\mathbf{x})$ *is* abecedarian *with respect to the partition* $\{X_i : i \in [n]\}$ *where* $X_i = \{x_{i,j} : i \le j \le n\}$*. With respect to this partition,*

1. $\text{linked\_CHSYM}_{n,d}(\mathbf{x})$ *has an* abecedarian *ABP of size* $O(nd)$*;*
2. *any* abecedarian *formula computing* $\text{linked\_CHSYM}_{n/2,\log n}(\mathbf{x})$ *has size* $n^{\Omega(\log\log n)}$*.*

That is, there is a super-polynomial separation between abecedarian *ABPs and* abecedarian *formulas.*

Our second main result shows that in certain settings, formulas computing abecedarian polynomials can be assumed to be abecedarian without loss of generality.

Theorem 3.2. *Let* f *be an* abecedarian *polynomial with respect to a partition of size* m *and* $\mathcal{F}$ *be a formula computing* f *of size* s*. If* $m = O(\log s)$*, then there is an* abecedarian *formula computing* f *of size* $\text{poly}(s)$*.*

In other words, an $n^{\omega(1)}$ lower bound against abecedarian formulas computing any polynomial that is abecedarian with respect to a partition of size $O(\log n)$, would result in a super-polynomial lower bound against general non-commutative formulas.

These statements suggest a new approach towards resolving the general VF_{nc} vs VBP_{nc} question.

Connections to the general VF_{nc} vs VBP_{nc} question

Theorem 3.1 gives a separation between abecedarian formulas and ABPs. On the other hand, Theorem 3.2 shows that if we are given a formula that computes a polynomial that is abecedarian with respect to a partition of *small* size, then we can assume that the formula is abecedarian without loss of generality. Unfortunately the partition with respect to which our *hard polynomial* from Theorem 3.1 is abecedarian, is *not small* in size. Thus, the general question of whether VBP_{nc} is contained in VF_{nc} or not still remains open.

However, there are two natural questions that arise at this point.

1. Can any formula computing an abecedarian polynomial be converted to an abecedarian formula without much blow-up in size, irrespective of the size of the partition?
2. Is there a polynomial f which is abecedarian with respect to a partition that has *small* size such that f witnesses a separation between abecedarian formulas and ABPs?

Clearly, a positive answer to either of these questions would imply that $\mathsf{VBP}_{\text{nc}} \neq \mathsf{VF}_{\text{nc}}$.

Proof Overview

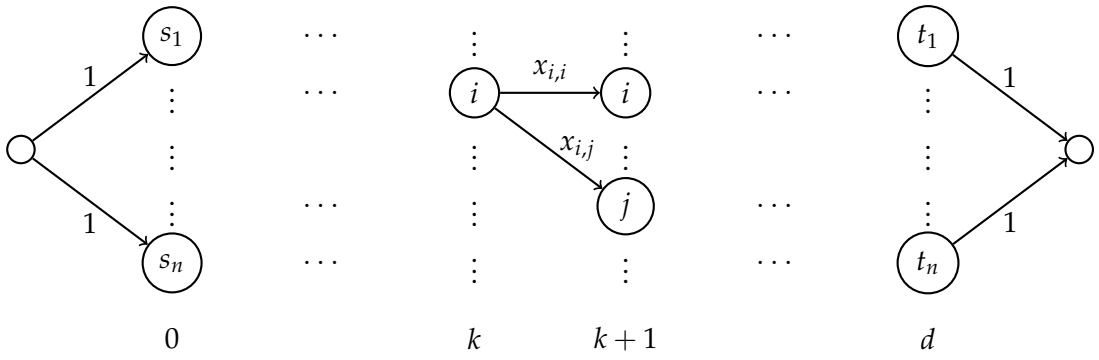
We now give a proof overview of our main theorems.

Let us first consider Theorem 3.2, since the proof is simpler. In order to prove the statement, we first convert the given formula $\mathcal{F}$ into an abecedarian circuit $\mathcal{C}$, and then unravel $\mathcal{C}$ to get an abecedarian formula $\mathcal{F}'$ computing the same polynomial.

The first step is fairly straightforward. The proof is along the same lines as that for homogenising circuits. The only difference is that we keep track of bucket indices of the variables on either ends of the monomials being computed at the gates, instead of their degrees.

In the second step, we convert $\mathcal{C}$ into a formula $\mathcal{F}'$. In order to do that, we need to recompute vertices every time it is reused. Thus, to give an upper bound on the size of $\mathcal{F}'$, we need to find an upper bound on the number of distinct paths from any vertex in $\mathcal{C}$ to the root. This analysis is done in a way similar to the one by Raz [Raz13] in his proof of the fact that formulas computing low degree polynomials can be homogenised without much blow-up in size. The requirement of the size of the partition being small also arises because of this analysis. The only additional point for the proof to go through is that, similar to the commutative setting non-commutative formulas can be depth reduced as well.

Next we go over the proof idea of Theorem 3.1.
A *small* abecedarian ABP computing linked_CHSYM$_{n,d}(\mathbf{x})$ essentially has the following structure.



For the lower bound, assume that we have been given a *small* abecedarian formula computing the polynomial. First, we show that any abecedarian formula can be converted to a syntactically abecedarian formula. We then keep modifying this formula till we get a *small* homogeneous multilinear formula computing the *elementary symmetric polynomial* of degree $n/2$. Finally, we use the known lower bound against homogeneous multilinear formulas for this polynomial (shown by Hrubeš and Yehudayoff [HY11]), to get a contradiction.

Let us spell out the proof in some more detail.

Step 1: Assume that we are given an abecedarian formula computing linked_CHSYM$_{n/2,\log n}(\mathbf{x})$ of size $O(n^{\varepsilon \log\log n})$. Since the degree of the polynomial being computed is *small*, we can assume

that there is in fact a *homogeneous* abecedarian formula computing linked_CHSYM$_{n/2,\log n}(\mathbf{x})$ of size $O(n^{c\cdot\varepsilon\log\log n})$ for some constant c independent of ε.

Step 2: Using the homogeneous abecedarian formula from Step 1, we obtain a *structured* homogeneous abecedarian formula, of size $O(n^{c\cdot\varepsilon\log\log n})$, computing linked_CHSYM$_{n/2,\log n}(\mathbf{x})$.

Step 3: We consider the complete homogeneous polynomial over n variables of degree d

$$\text{CHSYM}_{n,d}(\mathbf{x}) = \sum_{1\le i_1\le\ldots\le i_d\le n} x_{i_1}\cdots x_{i_d},$$

and show that there is a homogeneous abecedarian formula of size poly(n) that computes CHSYM$_{n/2,\log n}(\mathbf{x})$.

Step 4: If the formula in Step 2 has size s and that in Step 3 has size s', then we show that there is a homogeneous abecedarian formula of size $(s\cdot s')$ computing CHSYM$_{n/2,\log^2 n}(\mathbf{x})$.

Step 5: Next, we show that Step 4 can be used repeatedly $O(\log n/\log\log n)$ times, to obtain a homogeneous abecedarian formula computing the polynomial CHSYM$_{n/2,n/2}(\mathbf{x})$, of size $O(n^{c\cdot\varepsilon\log n})$.

Step 6: Using the formula obtained in Step 5, we get a homogeneous multilinear formula computing the elementary symmetric polynomial of degree $n/2$, of size $O(n^{c\cdot\varepsilon\log n})$.

Step 7: Finally, we choose an ε that contradicts the theorem in [HY11].

The crucial observation that makes this proof work, is that the polynomial we are working with is structured enough for us to be able to amplify its degree in a systematic way (without blowing up the size by much). This is Step 4 in the description above.

4 Algebraic Independence and Faithful Homomorphisms

In this section we will be focussing on questions on algebraic independence.

Recall that a set of polynomials $\mathbf{f} = \{f_1,\ldots,f_m\} \subset \mathbb{F}[\mathbf{x}]$ is said to be *algebraically dependent* if and only if there is some nonzero polynomial combination of $\{f_1,\ldots,f_m\}$ that is zero. Such a nonzero polynomial $A(z_1,\ldots,z_m) \in \mathbb{F}[\mathbf{z}]$, if one exist, for which $A(f_1,\ldots,f_m) = 0$ is called the *annihilating polynomial* for the set $\{f_1,\ldots,f_m\}$. As mentioned earlier, the underlying field is very important. For example the polynomials $x+y$ and x^p+y^p are algebraically dependent over $\mathbb{F}_p$ but algebraically independent over a characteristic zero field like $\mathbb{R}$ or $\mathbb{C}$.

Algebraic independence is very well-studied and it is known that algebraically independent subsets of a given set of polynomials form a *matroid* (see [Oxl92]). Hence the size of the maximum

algebraically independent subset of $\mathbf{f}$ is well-defined and is called the *algebraic rank* or *transcendence degree* of $\mathbf{f}$. We denote it by $\mathsf{algrank}(\mathbf{f}) = \mathsf{algrank}(f_1, \ldots, f_m)$.

Several computational questions arise from the above definition. For instance given a set of polynomials $\mathbf{f} = \{f_1, \ldots, f_m\}$, each f_i given in its dense representation, can we compute the algebraic rank of this set efficiently? What if the f_i's are provided as algebraic circuits?

Furthermore, in instances when $\mathsf{algrank}(\mathbf{f}) = m - 1$, the smallest degree annihilating polynomial is unique ([Kay09]). There could be various questions about the minimal degree annihilator in this case. For instance, can we compute it efficiently? Kayal [Kay09] showed that even checking if the constant term of the annihilator is zero is NP-hard, and evaluating the annihilator at a given point is #P-hard. In fact, recently Guo, Saxena, Sinhababu [GSS19] showed that even in the general case, checking if the constant term of every annihilator is zero is NP-hard. This effectively rules out any attempt to compute the algebraic rank via properties of the annihilating polynomials.

Despite this, over fields of characteristic zero, algebraic rank has an alternate characterisation via the Jacobian criterion. Jacobi [Jac41] showed that the algebraic rank of a set of polynomials $\mathbf{f}(\subseteq \mathbb{F}[\mathbf{x}])$ is given by the linear rank (over the rational function field $\mathbb{F}(\mathbf{x})$) of the Jacobian of these polynomials. This immediately yields a randomized polynomial time algorithm to compute the algebraic rank of a given set of polynomials by computing the rank of the Jacobian evaluated at a random point [Ore22, Sch80, Zip79, DL78].

4.1 Faithful Homomorphisms and Polynomial Identity Testing

Algebraic independence shares a lot of similarities with linear independence due to the matroid structure. One natural task is to find a *rank-preserving transformation* in this setting. This is defined by what are called *faithful homomorphisms*.

Definition 4.1 ([BMS13]). *Let $\mathbf{f} = \{f_1, \ldots, f_m\} \subseteq \mathbb{F}[\mathbf{x}]$ be a set of polynomials. If $\mathbb{K}$ is an extension field of $\mathbb{F}$, a homomorphism $\Phi : \mathbb{F}[\mathbf{x}] \to \mathbb{K}[\mathbf{y}]$ is said to be an* $\mathbb{F}$-faithful *homomorphism for $\{f_1, \ldots, f_m\}$ if*

$$\mathsf{algrank}_{\mathbb{F}} \{f_1, \ldots, f_m\} = \mathsf{algrank}_{\mathbb{F}} \{\Phi(f_1), \ldots, \Phi(f_m)\}. \qquad \Diamond$$

Ideally, we would like a faithful homomorphism with $|\mathbf{y}| \approx \mathsf{algrank}\{\mathbf{f}\}$ and $\mathbb{K} = \mathbb{F}$. Beecken, Mittmann and Saxena [BMS13] showed that a *generic* $\mathbb{F}$-linear homomorphism to $\mathsf{algrank}(\mathbf{f})$ many variables would be an $\mathbb{F}$-faithful homomorphism with high probability.

One important consequence of faithful homomorphisms is that they preserve nonzeroness of any polynomial composition of $f_1, \ldots, f_m$.

Lemma 4.2 ([BMS13, ASSS16]). *Suppose $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ and Φ is an $\mathbb{F}$-faithful homomorphism for $\{f_1, \ldots, f_m\}$. Then, for any circuit $C(z_1, \ldots, z_m) \in \mathbb{F}[z_1, \ldots, z_m]$, we have*

$$C(f_1, \ldots, f_m) = 0 \Leftrightarrow C(\Phi(f_1), \ldots, \Phi(f_m)) = 0.$$

Thus constructing explicit faithful homomorphisms can also be used for polynomial identity testing (PIT), which is the task of checking if a given algebraic circuit C computes the identically zero polynomial. For PIT, the goal is to design a deterministic algorithm that runs in time polynomial in the size of the circuit. There are two types of PIT algorithms, *whitebox* and *blackbox* — in the blackbox setting, we are only provided evaluation access to the circuit and some of its parameters (such as degree, number of variables, size etc.). Thus blackbox PIT algorithms for a class $\mathcal{C}$ is equivalent to constructing a *hitting set*, which is a small list of points in $S \subset \mathbb{F}^n$ such that any nonzero polynomial $f \in \mathcal{C}$ is guaranteed to evaluate to a nonzero value on some $\mathbf{a} \in S$.

It follows from Lemma 4.2 that if we can construct explicit $\mathbb{F}$-faithful homomorphisms for a set $\{f_1, \ldots, f_m\}$ whose algebraic rank is $k \ll n$, then we have a *variable reduction* that preserves the nonzeroness of any composition $C(f_1, \ldots, f_m)$. This approach was used by Beecken, Mittmann and Saxena [BMS13] and Agrawal, Saha, Saptharishi, Saxena [ASSS16], in the characteristic zero setting, to design identity tests for several subclasses by constructing faithful maps for $\{f_1, \ldots, f_m\}$ with algebraic rank at most $k = O(1)$, when

- each f_i is a sparse polynomial,
- each f_i is a product of multilinear, variable disjoint, sparse polynomials,
- each f_i is a product of linear polynomials,

and further generalisations.

All the above constructions crucially depend on the fact that the rank of the Jacobian captures algebraic independence. However, this fact is true only over fields of characteristic zero[4] and hence all the above results no longer hold over fields of positive characteristic.

Algebraic Independence over Finite Characteristic

A standard example to exhibit the failure of the Jacobian criterion over fields of finite characteristic, is $\left\{x^{p-1}y, y^{p-1}x\right\}$ — these polynomials are algebraically independent over $\mathbb{F}_p$ but the Jacobian is *not* full-rank over $\mathbb{F}_p$. Pandey, Saxena and Sinhababu [PSS18] characterised the extent of failure of the Jacobian criterion for $\{f_1, \ldots, f_m\}$ by a notion called the *inseparable degree* associated with this set. Over characteristic zero fields, this is always 1 but over fields of characteristic p this is a power of p. In their work, Pandey, Saxena and Sinhababu presented a Jacobian-like criterion to capture algebraic independence. Informally, each row of the *generalized Jacobian matrix* is obtained by taking the Taylor expansion of $f_i(\mathbf{x} + \mathbf{z})$ about a generic point, and truncating to just the terms of degree up to the *inseparable degree*[5]. The formal statement is as follows.

[4]Except if the polynomials are all quadratics and the underlying field is not of characteristic two (Unpublished joint work with Abhibhav Garg, Nitin Saxena and Ramprasad Saptharishi while Ramprasad and I were visiting IIT Kanpur).

[5]Over characteristic zero, the inseparable degree is 1 and this is just the vector of first order partial derivatives

Theorem 4.3. [PSS18] *Let $\{f_1, \ldots f_k\}$ be a set of n-variate polynomials over a field $\mathbb{F}$ with inseparable degree t. Also, for a generic point $\mathbf{z}$, let $\mathcal{H}_t(f_i) = \deg_{\leq t}(f_i(\mathbf{x}+\mathbf{z}) - f_i(\mathbf{z}))$. Then, they are algebraically dependent if and only if*

$$\exists(\alpha_1, \ldots, \alpha_k)(\neq \mathbf{0}) \in \mathbb{F}(\mathbf{z})^k \text{ s.t. } \sum_{i=1}^{k} \alpha_i \cdot \mathcal{H}_t(f_i) = 0 \mod \langle \mathcal{H}_t(f_1), \ldots, \mathcal{H}_t(f_k) \rangle^{\geq 2}_{\mathbb{F}(\mathbf{z})} + \langle \mathbf{x} \rangle^{t+1}.$$

In the setting when the *inseparable degree* is constant, this characterisation yields a randomized polynomial time algorithm to compute the algebraic rank. Thus, a natural question is whether this criterion can be used to construct faithful homomorphisms for similar classes of polynomials as studied by Beecken, Mittman, Saxena [BMS13] and Agrawal, Saha, Saptharishi, Saxena [ASSS16].

Following up on the criterion of Pandey, Saxena and Sinhababu [PSS18] for algebraic independence over finite characteristic, we extend the results of Beecken, Mittm, Saxena [BMS13] and Agrawal, Saha, Saptharishi, Saxena [ASSS16] to construct faithful homomorphisms for some restricted settings.

Theorem 4.4. *Let $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ be such that $\mathsf{algrank}\{f_1, \ldots, f_m\} = k$ and the inseparable degree is t. If t and k are bounded by a constant, then we can construct a polynomial (in the input length) sized list of homomorphisms of the form $\Phi : \mathbb{F}[\mathbf{x}] \to \mathbb{F}(s)[y_0, y_1, \ldots, y_k]$ such that at least one of them is guaranteed to be to $\mathbb{F}$-faithful for the set $\{f_1, \ldots, f_m\}$, in the following two settings:*

- *When each of the f_i's are sparse polynomials,*
- *When each of the f_i's are products of variable disjoint, multilinear, sparse polynomials.*

Prior to this, construction of faithful homomorphisms over finite fields was known only in the setting when each f_i has small individual degree [BMS13]. Over characteristic zero fields, the inseparable degree is always 1 and hence the faithful maps constructed in [BMS13], [ASSS16] over such fields can be viewed as special cases of our constructions.

The above theorem also holds for a few other models studied by Agrawal, Saha, Saptharishi and Saxena [ASSS16] (for instance, occur-k products of sparse polynomials). We mention the above two models just as an illustration of lifting the recipe for faithful maps from [BMS13, ASSS16] to the finite characteristic setting. As corollaries, we get efficient PIT algorithms for these models.

4.2 Proof Overview

The general structure of our proof follows that of Agrawal, Saha, Saptharishi and Saxena [ASSS16], where they constructed faithful homomorphisms in the characteristic zero setting. Roughly, their construction can be described in the following steps:

Step 1 : For a *generic linear map* $\Phi : \mathbf{x} \to \mathbb{F}(s)[y_1, \ldots, y_k]$, write the Jacobian of the set of polynomials $\{f_1 \circ \Phi, \cdots, f_k \circ \Phi\}$. Thus can be described succinctly as a matrix product of the form

$$J_{\mathbf{y}}(f \circ \Phi) = \Phi(J_{\mathbf{x}}(\mathbf{f})) \cdot J_{\mathbf{y}}(\Phi(\mathbf{x})).$$

Step 2 : We know that $J_{\mathbf{x}}(\mathbf{f})$ is full rank. Ensure that $\Phi(J_{\mathbf{x}}(\mathbf{f}))$ (where Φ is applied to every entry of the matrix $J_{\mathbf{x}}(\mathbf{f})$) remains full rank. This can be done if $\mathbf{f}$'s are some structured polynomials such as sparse polynomials, or variable-disjoint products of sparse polynomials etc.

Step 3 : Choose the map Φ so as to ensure that

$$\operatorname{rank}(\Phi(J_{\mathbf{x}}(\mathbf{f})) \cdot J_{\mathbf{y}}(\Phi(\mathbf{x}))) = \operatorname{rank}(\Phi(J_{\mathbf{x}}(\mathbf{f}))).$$

This is typically achieved by choosing Φ so as to make $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ a *rank-extractor*. It was shown by Gabizon and Raz [GR08] that a parametrized Vandermonde matrix has this property and this allows us to work with a homomorphism of the form (loosely speaking)

$$\Phi : x_i \mapsto \sum_{j=1}^{k} s^{ij} y_j.$$

We would like to execute essentially the same sketch over fields of finite characteristic but we encounter some immediate difficulties. The criterion of Pandey, Saxena and Sinhababu [PSS18] over finite characteristic is more involved but it is reasonably straightforward to execute the first two steps in the above sketch using the chain rule of (Hasse) derivatives.

The primary issue is in executing Step 3 and this is for two very different reasons. Firstly the analogue of the matrix $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ in the finite characteristic setting has many correlated entries. In the characteristic zero setting, we have complete freedom to choose Φ so that $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ can be any matrix that we want. In the finite characteristic setting however, even though we only have $n \cdot k$ parameters to define Φ, the analogue of $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ is much larger. Fortunately, there is just about enough structure in the matrix to show that it continues to have some rank-preserving properties.

The second hurdle comes from the subspace that we need to work with in the modified criterion. The *rank-extractor* is essentially parametrized by the variable s. In order to show that it preserves the rank of $\Phi(J_{\mathbf{x}}(\mathbf{f}))$ under right multiplication, we would like ensure that the variable s effectively does not appear in this matrix. In the characteristic zero setting, this is done by suitable restriction on other variables to remove any dependencies on s in $\Phi(J_{\mathbf{x}}(\mathbf{f}))$. Unfortunately, in the criterion of Pandey, Saxena and Sinhababu [PSS18], we have to work modulo some suitable subspace and these elements introduce other dependencies on s that appear to be hard to remove. Due to this hurdle, we are unable to construct $\mathbb{F}(s)$-faithful homomorphisms even in restricted

settings.

However, we observe that for the PIT applications, we are merely required to ensure that $\{f_1 \circ \Phi, \ldots, f_k \circ \Phi\}$ remain $\mathbb{F}$-algebraically independent instead of $\mathbb{F}(s)$-algebraically independent. With this weaker requirement, we can obtain a little more structure in the subspace involved and that lets us effectively execute Step 3.

5 Outline of the thesis

The thesis begins with introducing the algebraic models of computation formally, after which it is divided into two parts. In the first part, we focus on proving lower bounds for explicit polynomials. We begin with describing a proof of the quadratic lower bounds against ABPs and formulas, following which we study lower bounds in the non-commutative setting. The second part of the thesis deals with the question of testing algebraic independence of a given set of polynomials and how it leads to efficient PIT algorithms via constructing faithful homomorphisms.

6 Publications from the thesis

Quadratic Lower Bounds For Algebraic Branching Programs and Formulas [CKSV19]
To appear in Computational Complexity
Subsumes the paper titled "A Quadratic Lower Bound For Algebraic Branching Programs" that appeared in the proceedings of CCC 2020 [CKSV20].

Separating ABPs and Some Structured Formulas in the Non-Commutative Setting
Appeared in the Proceedings of CCC 2021 [Cha21].

Constructing Faithful Homomorphisms over Fields of Finite Characteristic [CS18]
Full version currently under review
Preliminary version appeared in the proceedings of FSTTCS 2019 [CS19].

References

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

[ASSS16] Manindra Agrawal, Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. Jacobian Hits Circuits: Hitting Sets, Lower Bounds for Depth-D Occur-k Formulas and Depth-3 Transcendence Degree-k Circuits. *SIAM J. Comput.*, 45(4):1533–1562, 2016.

[Ben83] Michael Ben-Or. Lower Bounds for Algebraic Computation Trees (Preliminary Report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 80–86. ACM, 1983.

[Ben94] Michael Ben-Or. Algebraic Computation Trees in Characteristi p>0 (Extended Abstract). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 534–539. IEEE Computer Society, 1994.

[BMS13] Malte Beecken, Johannes Mittmann, and Nitin Saxena. Algebraic independence and blackbox identity testing. *Inf. Comput.*, 222:2–19, 2013.

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

[Cha21] Prerona Chatterjee. Separating ABPs and Some Structured Formulas in the Non-Commutative Setting. In *36th Computational Complexity Conference, CCC 2021*, volume 200 of *LIPIcs*, pages 7:1–7:24, 2021.

[CKSV19] Prerona Chatterjee, Mrinal Kumar, Adrian She, and Ben Lee Volk. A Quadratic Lower Bound for Algebraic Branching Programs and Formulas. *CoRR*, abs/1911.11793, 2019.

[CKSV20] Prerona Chatterjee, Mrinal Kumar, Adrian She, and Ben Lee Volk. A Quadratic Lower Bound for Algebraic Branching Programs. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[CS18] Prerona Chatterjee and Ramprasad Saptharishi. Constructing Faithful Homomorphisms over Fields of Finite Characteristic. *CoRR*, abs/1812.10249, 2018. Pre-print available at `arXiv:1812.10249`.

[CS19] Prerona Chatterjee and Ramprasad Saptharishi. Constructing Faithful Homomorphisms over Fields of Finite Characteristic. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 150 of *LIPIcs*, pages 11:1–11:14, 2019.

[DL78] Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[GR08] Ariel Gabizon and Ran Raz. Deterministic extractors for affine sources over large fields. *Comb.*, 28(4):415–440, 2008.

[GSS19] Zeyu Guo, Nitin Saxena, and Amit Sinhababu. Algebraic Dependencies and PSPACE Algorithms in Approximative Complexity over Any Field. *Theory Comput.*, 15:1–30, 2019.

[HWY11] 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.

[HY11] Pavel Hrubes and Amir Yehudayoff. Homogeneous Formulas and Symmetric Polynomials. *Comput. Complex.*, 20(3):559–578, 2011.

[Hya77] L. Hyafil. The power of commutativity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 171–174, 1977.

[Jac41] C.G.J. Jacobi. De Determinantibus functionalibus. *Journal für die reine und angewandte Mathematik*, 22:319–359, 1841.

[Juk12] Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.

[Kal85] Kyriakos Kalorkoti. A Lower Bound for the Formula Size of Rational Functions. *SIAM Journal on Computing*, 14(3):678–687, 1985.

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

[Kum19] Mrinal Kumar. A quadratic lower bound for homogeneous algebraic branching programs. *Computational Complexity*, 28(3):409–435, 2019.

[LLS19] Guillaume Lagarde, Nutan Limaye, and Srikanth Srinivasan. Lower Bounds and PIT for Non-commutative Arithmetic Circuits with Restricted Parse Trees. *Comput. Complex.*, 28(3):471–542, 2019.

[LST21a] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. New Non-FPT Lower Bounds for Some Arithmetic Formulas. *Electron. Colloquium Comput. Complex.*, 28:94, 2021.

[LST21b] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial Lower Bounds Against Low-Depth Algebraic Circuits. *Electron. Colloquium Comput. Complex.*, 28:81, 2021.

[MW19] Merriam and Webster. Definition of Abecedarian. Word of the Day at www.merriam-webster.com, 2019.

[Nec66] Eduard Ivanovich Nechiporuk. On a Boolean function. *Dokl. Akad. Nauk SSSR*, 169:765–766, 1966.

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

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

[Oxl92] James G. Oxley. *Matroid theory*. Oxford University Press, 1992.

[PSS18] Anurag Pandey, Nitin Saxena, and Amit Sinhababu. Algebraic independence over positive characteristic: New criterion and applications to locally low-algebraic-rank circuits. *Comput. Complex.*, 27(4):617–670, 2018.

[Raz13] Ran Raz. Tensor-Rank and Lower Bounds for Arithmetic Formulas. *J. ACM*, 60(6):40:1–40:15, 2013.

[Sch80] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

[Str73] Volker Strassen. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973.

[SW01] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Comput. Complex.*, 10(1):1–27, 2001.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010.

[Val77] Leslie G. Valiant. Graph-Theoretic Arguments in Low-Level Complexity. In Jozef Gruska, editor, *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977.

[Val79] Leslie G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing*, pages 249–261. ACM, 1979.

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